Django Ledger

Miguel Sanda

Apr 18, 2024

CONTENTS:

sphinx-quickstart on Mon Jan 6 19:38:59 2020. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

CHAPTER

ONE

AN ACCOUNTING, BOOKKEEPING & FINANCIAL ANALYSIS ENGINE FOR THE DJANGO FRAMEWORK.

Introducing **Django Ledger**, a powerful double entry accounting system designed for financially driven applications using the Django Web Framework. Developed by lead developer Miguel Sanda, this system offers a simplified, high-level API, making it easier for users to navigate the complexities of accounting. If you have prior experience with Django, you'll find this software even more effective. And, for those interested in contributing, consider joining our new discord channel for further collaboration and discussions.

CHAPTER

TWO

QUESTIONS? JOIN OUR DISCORD CHANNEL HERE

CHAPTER

THREE

DOCUMENTATION

Access the latest documentation and QuickStart guide here. Also, you may download the QuickStart Jupyter Notebook here.

CHAPTER

FOUR

MAIN FEATURES

- High Level API.
- Double entry accounting system.
- Multiple Hierarchical Chart of Accounts.
- Financial Statements (Income Statement, Balance Sheet & Cash Flow Statement).
- Purchase Orders, Sales Orders (Estimates), Bills and Invoices.
- Automatic financial ratio & insight calculations.
- Multi tenancy (multiple companies/users/clients).
- Self-contained Ledgers, Journal Entries & Transactions.
- Basic OFX & QFX file import.
- Closing Entries.
- Items, lists & inventory management.
- Unit of Measures.
- Bank Accounts Information.
- Django Admin Classes.
- Built In Entity Management UI.

4.1 Need a new feature or report a bug?

Feel free to initiate an Issue describing your new feature request.

CHAPTER

FIVE

WANT TO CONTRIBUTE?

Finance and Accounting is a complicated subject. Django Ledger stands out from other Django projects due to its focus on providing a developer-friendly accounting engine and a reliable, extensible API for financially driven applications. The project requires expertise in Python, Django programming, finance, and accounting. In essence, the project is seeking assistance from individuals with the specific skill set needed to contribute effectively. So, it's clear that they are in need of support from individuals with the right expertise.

The project is actively seeking contributors with financial and/or accounting experience. Prior accounting experience is a big plus for potential contributors. If you have the relevant experience and want to contribute, feel free to reach out to me. You can find the contribution guidelines at the specified link. The project welcomes anyone interested in making a contribution.

See **contribution guidelines**.

CHAPTER

SIX

INSTALLATION

Django Ledger is a Django application. If you haven't, you need working knowledge of Django and a working Django project before you can use Django Ledger. A good place to start is here.

Make sure you refer to the django version you are using.

The easiest way to start is to use the zero-config Django Ledger starter template. See details here. Otherwise, you may create your project from scratch.

To create a new Django Ledger project:

- Make sure you have the latest version of python here (recommended).
- Install Django:

```
pip install django
```

- Install Python Pipenv (python package manager):

```
pip install pipenv
```

- Go to your desired development folder and create a new django project:

```
django-admin startproject django_ledger_project && cd django_ledger_project
```

- Install Django on you virtual environment.

```
pipenv install django
```

- Install Django Ledger

```
pipenv install django-ledger[graphql,pdf]
```

- Activate your new virtual environment:

```
pipenv shell
```

- Add django_ledger to INSTALLED_APPS in you new Django Project.

```
INSTALLED_APPS = [
    ...,
    'django_ledger',
    ...,
]
```

- Perform database migrations:

```
python manage.py migrate
```

- Add Django SuperUser and follow the prompts.

```
python manage.py createsuperuser
```

- Add URLs to your project's **urls.py**:

```
from django.urls import include, path

urlpatterns = [
    ...,
    path('ledger/', include('django_ledger.urls', namespace='django_ledger')),
    ...,
]
```

- Run your project:

```
python manage.py runserver
```

- Navigate to Django Ledger root view assigned in your project urlpatterns setting (typically http://127.0.0.1:8000/ledger if you followed this installation guide).
- Use your superuser credentials to login.

CHAPTER

SEVEN

HOW TO SET UP DJANGO LEDGER FOR DEVELOPMENT

Django Ledger comes with a basic development environment already configured under **dev_env/** folder not to be used for production environments. If you want to contribute to the project perform the following steps:

1. Navigate to your projects directory.
2. Clone the repo from github and CD into project.

```
git clone https://github.com/arrobalytics/django-ledger.git && cd django-ledger
```

3. Install PipEnv, if not already installed:

```
pip install -U pipenv
```

4. Create virtual environment.

```
pipenv install
```

If using a specific version of Python you may specify the path.

```
pipenv install --python PATH_TO_INTERPRETER
```

5. Activate environment.

```
pipenv shell
```

6. Apply migrations.

```
python manage.py migrate
```

7. Create a Development Django user.

```
python manage.py createsuperuser
```

8. Run development server.

```
python manage.py runserver
```

CHAPTER

EIGHT

HOW TO SET UP DJANGO LEDGER FOR DEVELOPMENT USING DOCKER

1. Navigate to your projects directory.
2. Give executable permissions to entrypoint.sh

```
sudo chmod +x entrypoint.sh
```

3. Add host '0.0.0.0' into ALLOWED_HOSTS in settings.py.
4. Build the image and run the container.

```
docker compose up --build
```

5. Add Django Superuser by running command in seprate terminal

```
docker ps
```

Select container id of running container and execute following command

```
docker exec -it containerId /bin/sh
```

```
python manage.py createsuperuser
```

6. Navigate to http://0.0.0.0:8000/ on browser.

CHAPTER

NINE

RUN TEST SUITE

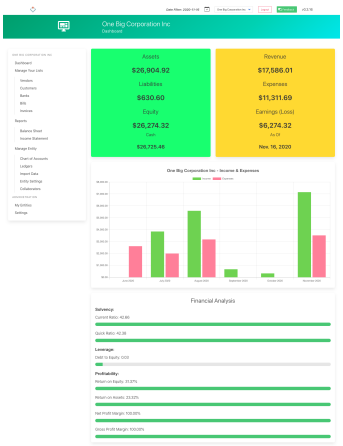
After setting up your development environment you may run tests.

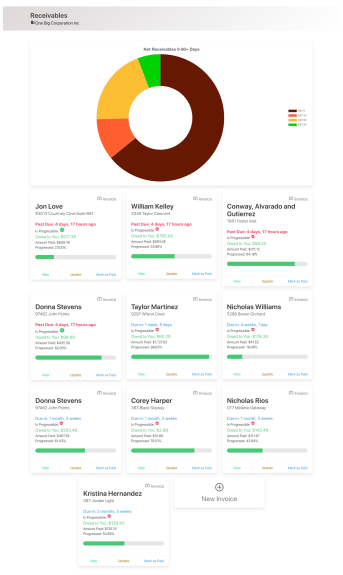
```
python manage.py test django_ledger
```

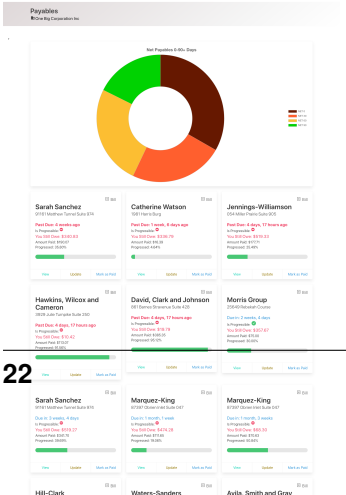
CHAPTER

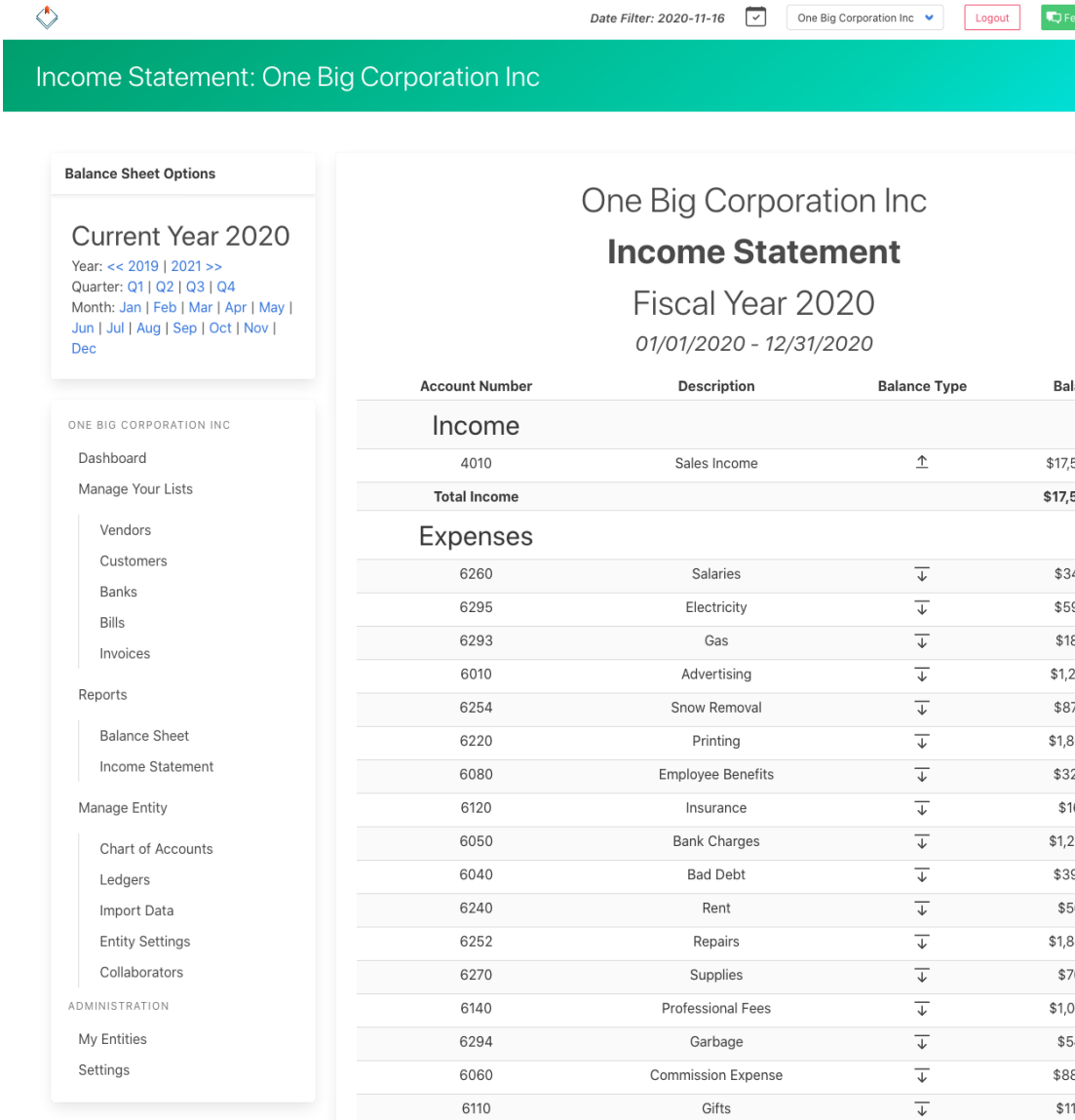
TEN

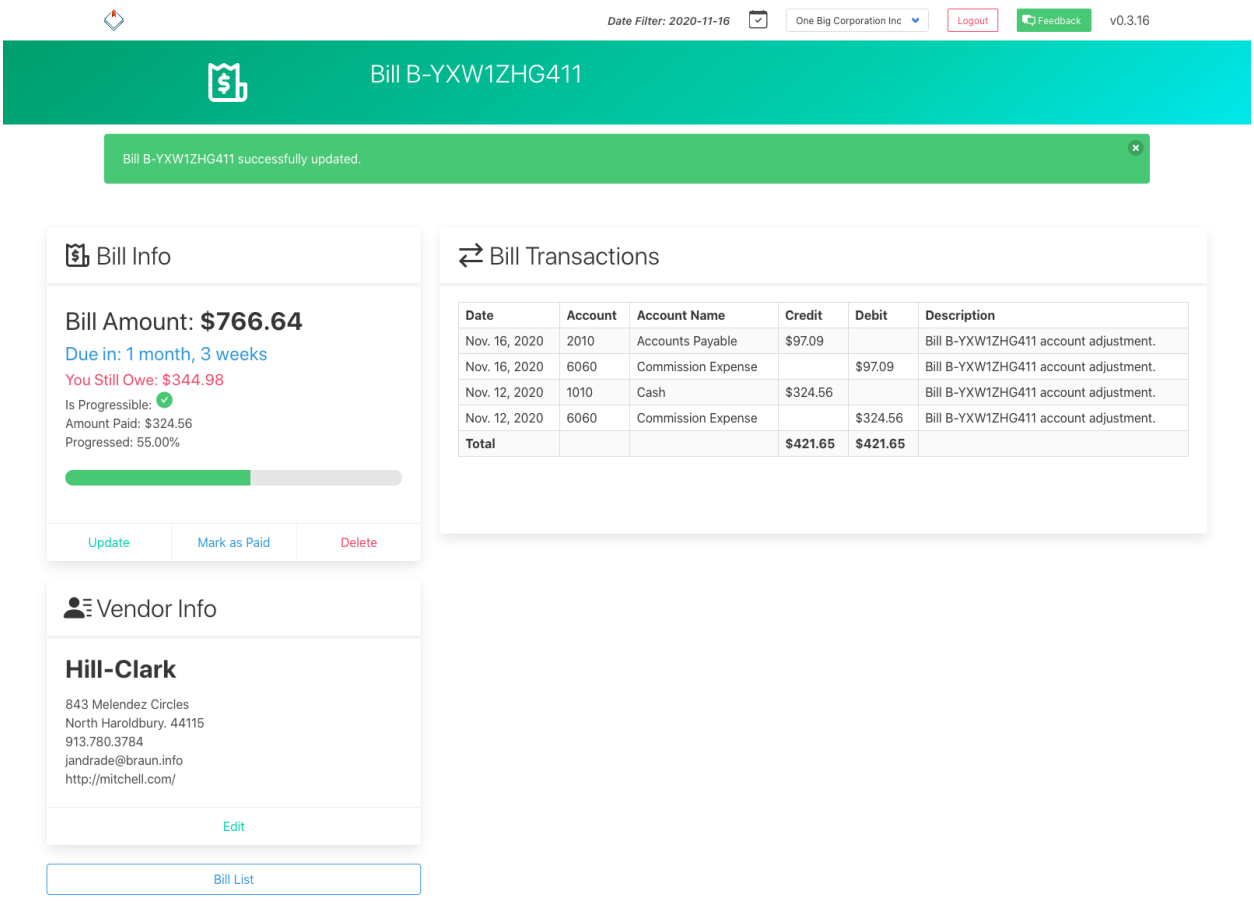
SCREENSHOTS

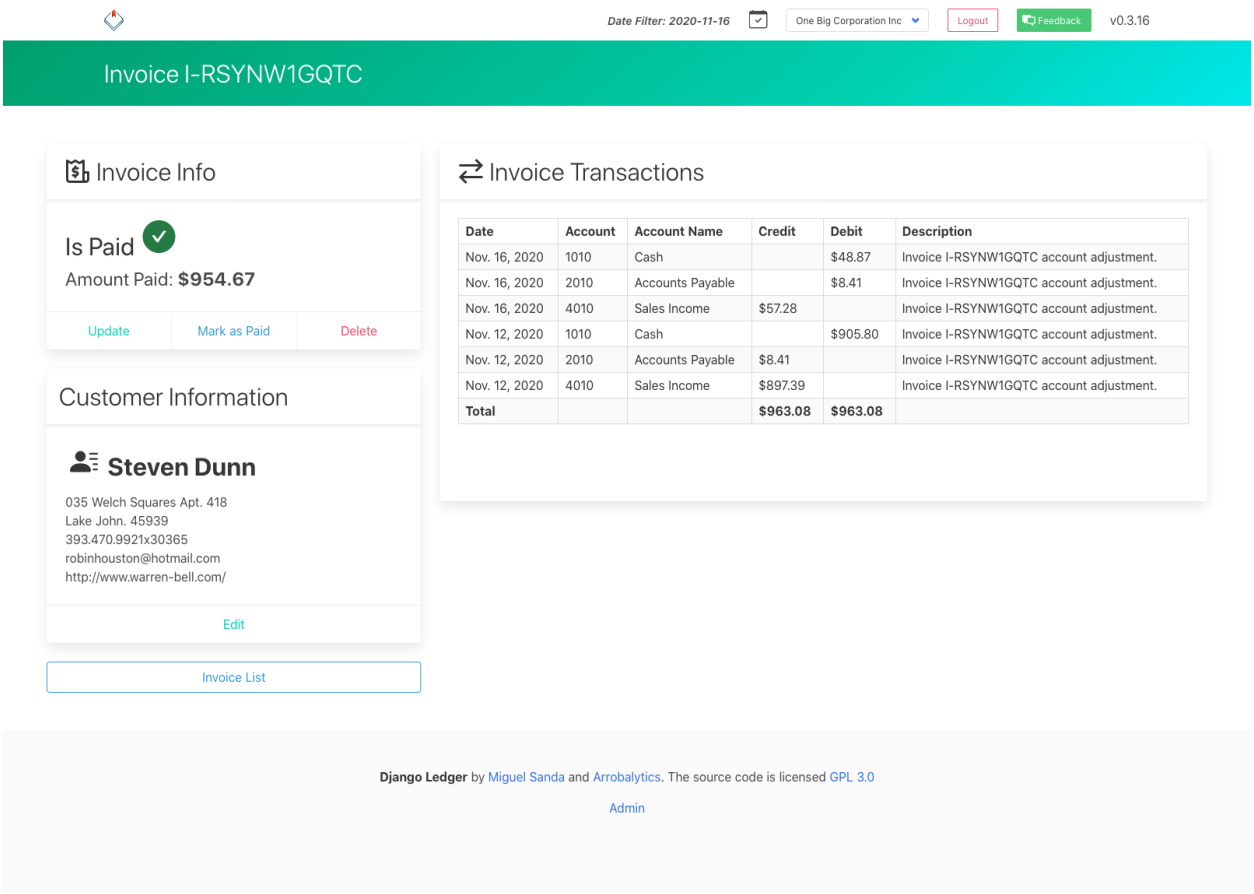









Date Filter: 2020-11-16
One Big Corporation Inc
Logout
Feedback
v0.3.16
Bill B-YXW1ZHG411
Bill B-YXW1ZHG411 successfully updated.
Bill Info
Bill Amount: $766.64
Due in: 1 month, 3 weeks
You Still Owe: $344.98
Is Progressible:
Amount Paid: $324.56
Progressed: 55.00%
Update
Mark as Paid
Delete
Vendor Info
Hill-Clark
843 Melendez Circles
North Haroldbury. 44115
913.780.3784
jandrade@braun.info
http://mitchell.com/
Edit
Bill List
Bill Transactions
Date | Account | Account Name | Credit | Debit | Description
Nov. 16, 2020 | 2010 | Accounts Payable | $97.09 | | Bill B-YXW1ZHG411 account adjustment.
Nov. 16, 2020 | 6060 | Commission Expense | | $97.09 | Bill B-YXW1ZHG411 account adjustment.
Nov. 12, 2020 | 1010 | Cash | $324.56 | | Bill B-YXW1ZHG411 account adjustment.
Nov. 12, 2020 | 6060 | Commission Expense | | $324.56 | Bill B-YXW1ZHG411 account adjustment.
Total | | | $421.65 | $421.65 |
Django Ledger by Miguel Sanda and Arrobalytics. The source code is licensed GPL 3.0
Admin


Date Filter: 2020-11-16
One Big Corporation Inc
Logout
Feedback
v0.3.16
Invoice I-RSYNW1GQTC
Invoice Info
Is Paid
Amount Paid: $954.67
Update
Mark as Paid
Delete
Customer Information
Steven Dunn
035 Welch Squares Apt. 418
Lake John. 45939
393.470.9921x30365
robinhouston@hotmail.com
http://www.warren-bell.com/
Edit
Invoice List
Invoice Transactions
Date | Account | Account Name | Credit | Debit | Description
Nov. 16, 2020 | 1010 | Cash | | $48.87 | Invoice I-RSYNW1GQTC account adjustment.
Nov. 16, 2020 | 2010 | Accounts Payable | | $8.41 | Invoice I-RSYNW1GQTC account adjustment.
Nov. 16, 2020 | 4010 | Sales Income | $57.28 | | Invoice I-RSYNW1GQTC account adjustment.
Nov. 12, 2020 | 1010 | Cash | | $905.80 | Invoice I-RSYNW1GQTC account adjustment.
Nov. 12, 2020 | 2010 | Accounts Payable | $8.41 | | Invoice I-RSYNW1GQTC account adjustment.
Nov. 12, 2020 | 4010 | Sales Income | $897.39 | | Invoice I-RSYNW1GQTC account adjustment.
Total | | | $963.08 | $963.08 |
Django Ledger by Miguel Sanda and Arrobalytics. The source code is licensed GPL 3.0
Admin

CHAPTER

ELEVEN

FINANCIAL STATEMENTS SCREENSHOTS

Balance Sheet Statement

One Big Company, LLC

Through December 31, 2022

Account Code	Account Name	Balance Type	Balance ($)	Total ($)
Assets				
Current Asset				
1010	Cash	debit	$64,753.47	
Total Current Asset				**$64,753.47**
Receivables				
1100	Accounts Receivable	debit	$1,621.34	
Total Receivables				**$1,621.34**
Inventory				
1200	Inventory	debit	$10,897.92	
Total Inventory				**$10,897.92**
				Total Assets: $77,272.73
Liabilities				
Accounts Payable				
2010	Accounts Payable	credit	$0.00	
Total Accounts Payable				**$0.00**
				Total Liabilities: $0.00
Equity				
Capital				
3030	Capital Account 3	credit	$50,000.00	
Total Capital				**$50,000.00**
Retained Earnings				**$27,272.73**
				Total Equity: $77,272.73
				Total Liabilities + Equity: $77,272.73

Cash Flow Statement

One Big Company, LLC

From January 01, 2022 through December 31, 2022

	Total ($)
Net Income as of January 01, 2022	**$27,272.73**
Cash from Operating Activities	
Non-cash Charges to Non-current Accounts	
Depreciation & Amortization of Assets	**$0.00**
Gain/Loss Sale of Assets	**$0.00**
Non-cash Charges to Current Accounts	
Accounts Receivable	**$-1,621.34**
Inventories	**$-10,897.92**
Accounts Payable	**$0.00**
Other Current Assets	**$0.00**
Other Current Liabilities	**$0.00**
Net Cash Provided by Operating Activities	**$14,753.47**
Cash from Financing Activities	
Common Stock, Preferred Stock and Capital Raised	**$50,000.00**
Dividends Payed Out to Shareholders	**$0.00**
Increase/Reduction of Short-Term Debt Principal	**$0.00**
Increase/Reduction of Long-Term Debt Principal	**$0.00**
Net Cash Provided by Financing Activities	**$50,000.00**
Cash from Investing Activities	
Purchase, Maturity and Sales of Investments & Securities	**$0.00**
Addition and Disposition of Property, Plant & Equipment	**$0.00**
Net Cash Provided by Investing Activities	**$0.00**
Net Cash Flow	
Net Cash Flow from January 01, 2022 through December 31, 2022	**$64,753.47**

CHAPTER

TWELVE

API QUICKSTART

```
import os
from datetime import date, datetime
from decimal import Decimal
from random import randint, choices, random
from zoneinfo import ZoneInfo

import django
# for easier visualization it is recommended to use pandas to render data...
# if pandas is not installed, you may install it with this command: pip install -U pandas
# pandas is not a dependecy of django_ledger...
import pandas as pd
from django.core.exceptions import ObjectDoesNotExist

# Set your django settings module if needed...
os.environ['DJANGO_SETTINGS_MODULE'] = 'dev_env.settings'

# if using jupyter notebook need to set DJANGO_ALLOW_ASYNC_UNSAFE as "true"
os.environ['DJANGO_ALLOW_ASYNC_UNSAFE'] = 'true'

# change your working directory as needed...
os.chdir('../')

django.setup()

from django_ledger.models.entity import EntityModel
from django_ledger.models.items import ItemModel
from django_ledger.models.invoice import InvoiceModel
from django_ledger.models.bill import BillModel
from django_ledger.models.estimate import EstimateModel
from django.contrib.auth import get_user_model
from django_ledger.io import roles, DEBIT, CREDIT
from django_ledger.io.io_library import IOBluePrint, IOLibrary
```

12.1 Get Your Entity Administrator UserModel

```
# change this to your preferred django username...
MY_USERNAME = 'ceo_user'
MY_PASSWORD = 'NeverUseMe|VeryInsecure!'
UserModel = get_user_model()

try:
    user_model = UserModel.objects.get(username__exact=MY_USERNAME)
except:
    user_model = UserModel(username=MY_USERNAME)
    user_model.set_password(MY_PASSWORD)
    user_model.save()
```

12.2 Get or Create an Entity Model

```
ENTITY_NAME = 'One Big Company, LLC'

entity_model = EntityModel.create_entity(
    name=ENTITY_NAME,
    admin=user_model,
    use_accrual_method=True,
    fy_start_month=1
)

entity_model
```

12.3 Chart of Accounts

12.3.1 Create a Default Chart of Accounts

- Newly created EntityModel do not have a default Code of Accounts yet.

```
entity_model.has_default_coa()
```

```
default_coa_model = entity_model.create_chart_of_accounts(
    assign_as_default=True,
    commit=True,
    coa_name='My QuickStart CoA'
)
```

```
default_coa_model
```

```
entity_model.default_coa == default_coa_model
```

12.4 Populate Entity with Random Data (Optional)

12.4.1 Define a Start Date for Transactions

```
START_DTTM = datetime(year=2022, month=10, day=1, tzinfo=ZoneInfo('UTC'))
```

- This action will populate the EntityModel with random data.
- It will populate a Code of Accounts using a default pre-defined list.
- This approach is for illustration, educational and testing purposes, not encouraged for new production entities.

```
entity_model.populate_random_data(start_date=START_DTTM)
```

12.4.2 EntityModel has now a Default Chart of Accounts

```
entity_model.has_default_coa()
```

```
default_coa_model = entity_model.get_default_coa()
default_coa_model
```

12.5 Chart of Accounts (CoA)

- A Chart of Accounts is a user-defined list of accounts.
- Each Entity Model must have at least one default Chart of Accounts.

12.5.1 Django Ledger support multiple chart of accounts.

```
another_coa_model = entity_model.create_chart_of_accounts(
    assign_as_default=False,
    commit=True,
    coa_name='My Empty Chart of Accounts'
)
```

```
another_coa_model
```

12.6 Accounts

12.6.1 Default CoA Accounts

```
default_coa_accounts_qs = entity_model.get_default_coa_accounts()
pd.DataFrame(default_coa_accounts_qs)
```

12.6.2 Get CoA Accounts by CoA Model

```
coa_accounts_by_coa_model_qs = entity_model.get_coa_accounts(coa_model=default_coa_model)
pd.DataFrame(coa_accounts_by_coa_model_qs)
```

No Accounts yet on this CoA...

```
coa_accounts_by_coa_model_qs = entity_model.get_coa_accounts(coa_model=another_coa_model)
pd.DataFrame(coa_accounts_by_coa_model_qs)
```

12.6.3 Get CoA Accounts by CoA Model UUID

- May pass UUID instance instead of ChartOF AccountsModel...

```
coa_accounts_by_coa_uuid_qs = entity_model.get_coa_accounts(coa_model=default_coa_model.
↪uuid)
pd.DataFrame(coa_accounts_by_coa_uuid_qs)
```

12.6.4 Get CoA Accounts by CoA Model Slug

- If string is passed, will lookup by slug...

```
coa_accounts_by_coa_slug_qs = entity_model.get_coa_accounts(coa_model=default_coa_model.
↪slug)
pd.DataFrame(coa_accounts_by_coa_slug_qs)
```

12.6.5 Get Accounts With Codes and CoA Model

- Assumes default CoA if no coa_model is passed...

```
coa_accounts_by_codes_qs = entity_model.get_accounts_with_codes(code_list=['1010', '1050
↪'])
pd.DataFrame(coa_accounts_by_codes_qs)
```

Empty ChartOfAccountModel...

```
coa_accounts_by_codes_qs = entity_model.get_accounts_with_codes(
    code_list=['1010', '1050'],
    coa_model=another_coa_model
)
pd.DataFrame(coa_accounts_by_codes_qs)
```

Get All Accounts at Once

```
coa_qs, coa_map = entity_model.get_all_coa_accounts()
```

A dictionary, CoA Model -> Account List.

```
coa_map
```

```
pd.DataFrame(coa_map[default_coa_model])
```

```
pd.DataFrame(coa_map[another_coa_model])
```

12.6.6 Create Account Model

- Creating AccountModel into empty "another_coa_model"...

```
account_model = entity_model.create_account(
    coa_model=another_coa_model,
    code=f'1{str(randint(10000, 99999))}',
    role=roles.ASSET_CA_INVENTORY,
    name='A cool account created from the EntityModel API!',
    balance_type=roles.DEBIT,
    active=True)
```

```
account_model
```

```
another_coa_accounts_qs = entity_model.get_coa_accounts(coa_model=another_coa_model)
pd.DataFrame(another_coa_accounts_qs)
```

12.7 Basic Django Ledger Usage

- The LedgerModel name is whatever your heart desires.
- Examples:
  - A month.
  - A customer.
  - A vendor.
  - A project.
- The more ledgers are created, the more segregation and control over transactions is possible.

```
ledger_model = entity_model.create_ledger(name='My October 2023 Ledger', posted=True)
```

12.7.1 Create a Library

```
library = IOLibrary(name='quickstart-library')
```

12.7.2 Create and Register a BluePrint

```
@library.register
def sale_blueprint(
        sale_amount,
        contribution_margin_percent: float,
        description: str = None
) -> IOBluePrint:
    blueprint = IOBluePrint()
    cogs_amount = (1 - contribution_margin_percent) * sale_amount
    blueprint.debit(account_code='1010', amount=sale_amount, description=description)
    blueprint.credit(account_code='4010', amount=sale_amount, description=description)
    blueprint.credit(account_code='1200', amount=cogs_amount, description=description)
    blueprint.debit(account_code='5010', amount=cogs_amount, description=description)
    return blueprint
```

12.7.3 Get a Cursor

```
cursor = library.get_cursor(entity_model=entity_model, user_model=user_model)
```

12.7.4 Dispatch Some Instructions

```
cursor.dispatch('sale_blueprint',
                ledger_model='ledger-test-1',
                sale_amount=120.345,
                contribution_margin_percent=0.25,
                description='so cool')
cursor.dispatch('sale_blueprint',
                ledger_model='ledger-test-1',
                sale_amount=12.345,
                contribution_margin_percent=0.2,
                description='so cool')
cursor.dispatch('sale_blueprint',
                ledger_model=ledger_model,
                sale_amount=34.455,
                contribution_margin_percent=0.13,
                description='so cool')
cursor.dispatch('sale_blueprint',
                ledger_model='ledger-test-12',
                sale_amount=90.43,
                contribution_margin_percent=0.17,
                description='so cool')
```

12.7.5 Commit Your Instructions

Not recommended to post both ledger and journal entries. Posted transactions will immediately hit the books. **result** contains resulting ledger models, journal entries and transactions fro the committed

```
result = cursor.commit(
    post_new_ledgers=True,
    post_journal_entries=True,
    je_timestamp='2023-12-02 12:00'
)
```

```
# result
```

Get Financial Statement Report Data fro Ledger Model

Balance Sheet

```
bs_data = ledger_model.digest_balance_sheet(
    to_date=date(2023, 12, 31),
    entity_slug=entity_model
)

bs_data.get_balance_sheet_data()
```

Income Statement

```
is_data = ledger_model.digest_income_statement(
    from_date=date(2023, 1, 1),
    to_date=date(2023, 12, 31),
    entity_slug=entity_model
)
is_data.get_income_statement_data()
```

Cash Flow Statement

```
cfs_data = ledger_model.digest_cash_flow_statement(
    from_date=date(2023, 1, 1),
    to_date=date(2023, 12, 31),
    entity_slug=entity_model
)

cfs_data.get_cash_flow_statement_data()
```

All Statements in a Single Call

```
fin_digest = ledger_model.digest_financial_statements(
    from_date=date(2023, 1, 1),
    to_date=date(2023, 12, 31),
    entity_slug=entity_model
)

statement_data = fin_digest.get_financial_statements_data()
```

```
statement_data['balance_sheet']
```

```
statement_data['income_statement']
```

```
statement_data['cash_flow_statement']
```

12.8 Customers

12.8.1 Get Customers

```
customer_qs = entity_model.get_customers()
pd.DataFrame(customer_qs.values())
```

12.8.2 Create Customers

```
customer_model = entity_model.create_customer(customer_model_kwargs={
    'customer_name': 'Mr. Big',
    'description': 'A great paying customer!',
})
```

12.9 Vendors

12.9.1 Get Vendors

```
vendor_qs = entity_model.get_vendors()
pd.DataFrame(vendor_qs.values())
```

12.9.2 Create Vendor

```
vendor_model = entity_model.create_vendor(vendor_model_kwargs={
    'vendor_name': 'ACME LLC',
    'description': 'A Reliable Vendor!'
})
```

12.10 Invoices

12.10.1 Get Invoices

```
invoices_qs = entity_model.get_invoices()
pd.DataFrame(invoices_qs.values())
```

12.10.2 Create Invoice

```
invoice_model = entity_model.create_invoice(
    customer_model='C-0000000006',
    terms=InvoiceModel.TERMS_NET_30
)
```

```
invoice_model
```

12.10.3 Add Items to Invoices

```
invoices_item_models = invoice_model.get_item_model_qs()

# K= number of items...
K = 6

invoice_itemtxs = {
    im.item_number: {
        'unit_cost': round(random() * 10, 2),
        'quantity': round(random() * 100, 2),
        'total_amount': None
    } for im in choices(invoices_item_models, k=K)
}

# Choose operation ITEMIZE_APPEND to append itemtxs...
invoice_itemtxs = invoice_model.migrate_itemtxs(itemtxs=invoice_itemtxs,
                                                commit=True,
                                                operation=InvoiceModel.ITEMIZE_REPLACE)
invoice_itemtxs
```

```
invoice_model.amount_due
```

12.11 Bills

12.11.1 Get Bills

```
bills_qs = entity_model.get_bills()
pd.DataFrame(bills_qs.values())
```

12.11.2 Create Bill

```
bill_model = entity_model.create_bill(
    vendor_model='V-0000000002',
    terms=BillModel.TERMS_NET_60
)
```

```
bill_model
```

12.11.3 Add Items to Bills

```
bill_item_models = bill_model.get_item_model_qs()

K = 6

bill_itemtxs = {
    im.item_number: {
        'unit_cost': round(random() * 10, 2),
        'quantity': round(random() * 100, 2),
        'total_amount': None
    } for im in choices(bill_item_models, k=K)
}

# Choose operation ITEMIZE_APPEND to append itemtxs...
bill_itemtxs = bill_model.migrate_itemtxs(itemtxs=bill_itemtxs,
                                          commit=True,
                                          operation=BillModel.ITEMIZE_REPLACE)

bill_itemtxs
```

```
bill_model.amount_due
```

12.12 Purchase Orders

12.12.1 Get Purchase Orders

```
purchase_orders_qs = entity_model.get_purchase_orders()
pd.DataFrame(purchase_orders_qs.values())
```

12.12.2 Create Purchase Order

```
po_model = entity_model.create_purchase_order()
```

12.12.3 Add Items to Purchase Orders

```
po_item_models = po_model.get_item_model_qs()

K = 6

po_itemtxs = {
    im.item_number: {
        'unit_cost': round(random() * 10, 2),
        'quantity': round(random() * 100, 2),
        'total_amount': None
    } for im in choices(po_item_models, k=K)
}

# Choose operation ITEMIZE_APPEND to append itemtxs...
po_itemtxs = po_model.migrate_itemtxs(itemtxs=po_itemtxs,
                                      commit=True,
                                      operation=EstimateModel.ITEMIZE_REPLACE)

po_itemtxs
```

```
po_model.po_amount
```

12.13 Estimates/Contracts

12.13.1 Get Estimates/Contracts

```
estimates_qs = entity_model.get_estimates()
pd.DataFrame(estimates_qs.values())
```

12.13.2 Create Estimate

```
estimate_model = entity_model.create_estimate(
    estimate_title='A quote for new potential customer!',
    customer_model='C-0000000009',
    contract_terms=EstimateModel.CONTRACT_TERMS_FIXED
)
```

12.13.3 Add Items to Estimates

```
estimate_item_models = estimate_model.get_item_model_qs()

K = 6

estimate_itemtxs = {
    im.item_number: {
        'unit_cost': round(random() * 10, 2),
        'unit_revenue': round(random() * 20, 2),
        'quantity': round(random() * 100, 2),
        'total_amount': None
    } for im in choices(estimate_item_models, k=K)
}

# Choose operation ITEMIZE_APPEND to append itemtxs...
estimate_itemtxs = estimate_model.migrate_itemtxs(itemtxs=estimate_itemtxs,
                                                  commit=True,
                                                  operation=EstimateModel.ITEMIZE_
↪REPLACE)

estimate_itemtxs
```

```
estimate_model.get_cost_estimate()
```

```
estimate_model.get_revenue_estimate()
```

```
estimate_model.get_profit_estimate()
```

```
estimate_model.get_gross_margin_estimate(as_percent=True)
```

12.14 Bank Accounts

12.14.1 Get Bank Accounts

```
bank_accounts_qs = entity_model.get_bank_accounts()
pd.DataFrame(bank_accounts_qs.values())
```

12.14.2 Create Bank Account

```
bank_account_model = entity_model.create_bank_account(name='A big bank account!',
                                                      account_type='checking')
```

12.15 Items

12.15.1 Unit of Measures

Get Unit of Measures

```
uom_qs = entity_model.get_uom_all()
pd.DataFrame(uom_qs.values())
```

Create a UOM

```
uom_model_ft = entity_model.create_uom(
    name='Linear Feet',
    unit_abbr='lin-ft'
)
```

Get Some UoMs

```
uom_model_unit = uom_qs.get(unit_abbr__exact='unit')
uom_model_man_hr = uom_qs.get(unit_abbr__exact='man-hour')
```

12.15.2 Expenses

Get Expense Items

```
expenses_qs = entity_model.get_items_expenses()
pd.DataFrame(expenses_qs.values())
```

Create Expense Item

```
expense_item_model = entity_model.create_item_expense(
    name='Premium Pencils',
    uom_model=uom_model_unit,
    expense_type=ItemModel.ITEM_TYPE_MATERIAL
)
```

```
expense_item_model.is_expense()
```

12.15.3 Services

Get Service Items

```
services_qs = entity_model.get_items_services()
pd.DataFrame(services_qs.values())
```

Create Service Item

```
service_model = entity_model.create_item_service(
    name='Yoga Class',
    uom_model=uom_model_man_hr
)
```

```
service_model.is_service()
```

12.15.4 Products

Get Product Items

```
products_qs = entity_model.get_items_products()
pd.DataFrame(products_qs.values())
```

Create Product Items

```
product_model = entity_model.create_item_product(
    name='1/2" Premium PVC Pipe',
    uom_model=uom_model_ft,
    item_type=ItemModel.ITEM_TYPE_MATERIAL
)
```

```
product_model.is_product()
```

12.15.5 Inventory

Get Inventory Items

```
inventory_qs = entity_model.get_items_inventory()
pd.DataFrame(inventory_qs.values())
```

Create Inventory Items

```
inventory_model = entity_model.create_item_inventory(
    name='A Home to Flip!',
    uom_model=uom_model_unit,
    item_type=ItemModel.ITEM_TYPE_LUMP_SUM
)
```

```
inventory_model.is_inventory()
```

12.16 Financial Statement PDF Reports

12.16.1 Set Up

- Must enable PDF support by installing dependencies via *pipenv*.
  - pipenv install –categories pdf

12.16.2 Balance Sheet

```
bs_report = entity_model.get_balance_sheet_statement(
    to_date=date(2022, 12, 31),
    save_pdf=True,
    filepath='./'
)
# save_pdf=True saves the PDF report in the project's BASE_DIR.
# filename and filepath may also be specified...
# will raise not implemented error if PDF support is not enabled...
```

Balance Sheet Statement Raw Data

```
bs_report.get_report_data()
```

12.16.3 Income Statement

```
ic_report = entity_model.get_income_statement(
    from_date=date(2022, 1, 1),
    to_date=date(2022, 12, 31),
    save_pdf=True,
    filepath='./'
)
# save_pdf=True saves the PDF report in the project's BASE_DIR.
# filename and filepath may also be specified...
# will raise not implemented error if PDF support is not enabled...
```

Income Statement Raw Data

```
ic_report.get_report_data()
```

12.16.4 Cash Flow Statement

```
cf_report = entity_model.get_cash_flow_statement(
    from_date=date(2022, 1, 1),
    to_date=date(2022, 12, 31),
    save_pdf=True,
    filepath='./'
)
# save_pdf=True saves the PDF report in the project's BASE_DIR.
# filename and filepath may also be specified...
```

Cash Flow Statement Raw Data

```
cf_report.get_report_data()
```

12.16.5 All Financial Statements Data in a single Call

```
reports = entity_model.get_financial_statements(
    user_model=user_model,
    from_date=date(2022, 1, 1),
    to_date=date(2022, 12, 31),
    save_pdf=True,
    filepath='./'
)
# save_pdf=True saves the PDF report in the project's BASE_DIR.
# filename and filepath may also be specified...
```

```
reports.balance_sheet_statement.get_report_data()
```

```
reports.income_statement.get_report_data()
```

```
reports.cash_flow_statement.get_report_data()
```

CHAPTER

THIRTEEN

IO ENGINE

13.1 Random Data Generation

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module: Miguel Sanda <msanda@arrobalytics.com>

This is a random data generator module used during the testing of the API and for Educational purposes.

The class EntityDataGenerator will only work on new entities that contain no Transactions. This is with the intention of avoiding unintentional commingling with an actual EntityModel with production data and the data generated randomly.

This class will conveniently create a Chart of Accounts and populate the database will Bills, Invoices and various other Transactions. The user will be able to immediately browse the Entity data by clicking on the newly created entity's details page.

All data generated is random and fake, not related to any other entity data.

class django_ledger.io.io_generator.**EntityDataGenerator**(*user_model*, *entity_model: Union[EntityModel, str]*, *start_dttm: datetime*, *capital_contribution: Decimal*, *days_forward: int*, *tx_quantity: int = 25*)

A random data generator for Entity Models. Requires a user to me the entity model administrator.

user_model

The Django user model that administers the entity.

Type
UserModel

entity_model

The Entity model to populate.

Type
EntityModel

start_dttm

The start datetime for new transactions. All transactions will be posted no earlier than this date.

Type
datetime

capital_contribution

The initial capital contribution amount for the Entity Model. This will help fund the entity.

Type
Decimal

days_forward

The number of days to span from the start_dttm for new transactions.

Type
int

exception django_ledger.io.io_generator.**EntityModelValidationError**(*message*, *code=None*, *params=None*)

django_ledger.io.io_generator.**random**() → x in the interval [0, 1).

13.2 IO MixIn

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

This module provides the building block interface for Django Ledger. The classes and functions contained in this module provide an interface to Django Ledger to create and manage Transactions into the Database. It also provides an optimized interface to push as much work as possible to the database without having to pull transactions from the database into the Python memory.

The database records the individual transactions associated with each Journal Entry. However, this interface aggregates transactions during the digest method based on a specific request. The Python interpreter is responsible for applying accounting rules to the transactions associated with each Journal Entry so the appropriate account balances are computed.

class django_ledger.io.io_core.**IODatabaseMixIn**

The main entry point to query DB for transactions. The database_digest method pushes as much load as possible to the Database so transactions are aggregated at the database layer and are not pulled into memory. This is important por performance purposes since Entities may have a large amount of transactions to be aggregated.

The python_digest method aggregates and processes the raw data stored in the database and applies accounting rules to stored transactions.

This method also makes use of Closing Entries whenever possible to minimize the amount of data to aggregate during a specific call. Closing Entries can be considered "checkpoints", which create materialized aggregation of transactions for commonly used dates. (i.e. Fiscal Year End, Month End, Quarter End, etc.). This approach helps minimize the number of transactions to aggregate for a given request.

database_digest(*entity_slug: Optional[str] = None*, *unit_slug: Optional[str] = None*, *user_model: Optional[User] = None*, *from_date: Optional[Union[datetime, date]] = None*, *to_date: Optional[Union[datetime, date]] = None*, *by_activity: bool = False*, *by_tx_type: bool = False*, *by_period: bool = False*, *by_unit: bool = False*, *activity: Optional[str] = None*, *role: str = typing.Optional[str]*, *accounts: Optional[Union[str, List[str], Set[str]]] = None*, *posted: bool = True*, *exclude_zero_bal: bool = True*, *use_closing_entries: bool = False*, ***kwargs*) → *IOResult*

Performs the appropriate database aggregation query for a given request.

Parameters

- **entity_slug** (*str*) – EntityModel slug to use. If not provided it will be derived from the EntityModel instance. Will be validated against current EntityModel instance for safety. Defaults to None.
- **unit_slug** (*str*) – EntityUnitModel used to query transactions. If provided will be validated against current EntityModelUnit instance. Defaults to None.
- **user_model** (*UserModel*) – The django UserModel to validate against transaction ownership and permissions (i.e. Admin and Manager). Defaults to None.
- **from_date** (*date or datetime*) – Stating date or datetime to query from (inclusive).
- **to_date** (*date or datetime*) – End date or datetime to query to (inclusive).
- **activity** (*str*) – Filters transactions to match specific activity. Defaults to None.
- **role** (*str*) – Filters transactions to match specific role. Defaults to None.
- **accounts** (*str or List[str] ot Set[str]*) – Optional list of accounts to query. Defaults to None (all).
- **posted** (*bool*) – Consider only posted transactions. Defaults to True.
- **exclude_zero_bal** (*bool*) – Excludes transactions with zero balance, if any.
- **by_activity** (*bool*) – Returns results aggregated by activity if needed. Defaults to False.
- **by_tx_type** (*bool*) – Returns results aggregated by DEBIT/CREDIT if needed. Defaults to False.
- **by_period** (*bool*) – Returns results aggregated by accounting if needed. Defaults to False.
- **by_unit** (*bool*) – Returns results aggregated by unit if needed. Defaults to False.
- **use_closing_entry** (*bool*) – Overrides the DJANGO_LEDGER_USE_CLOSING_ENTRIES setting.

Return type
IOResult

python_digest(*user_model: Optional[User] = None*, *entity_slug: Optional[str] = None*, *unit_slug: Optional[str] = None*, *to_date: Optional[Union[date, datetime, str]] = None*, *from_date: Optional[Union[date, datetime, str]] = None*, *equity_only: bool = False*, *activity: Optional[str] = None*, *role: Optional[Union[Set[str], List[str]]] = None*, *accounts: Optional[Union[Set[str], List[str]]] = None*, *signs: bool = True*, *by_unit: bool = False*, *by_activity: bool = False*, *by_tx_type: bool = False*, *by_period: bool = False*, *use_closing_entries: bool = False*, *force_queryset_sorting: bool = False*, ***kwargs*) → *IOResult*

Performs the appropriate transaction post-processing after DB aggregation..

Parameters

- **entity_slug** (*str*) – EntityModel slug to use. If not provided it will be derived from the EntityModel instance. Will be validated against current EntityModel instance for safety. Defaults to None.
- **unit_slug** (*str*) – EntityUnitModel used to query transactions. If provided will be validated against current EntityModelUnit instance. Defaults to None.
- **user_model** (*UserModel*) – The django UserModel to validate against transaction ownership and permissions (i.e. Admin and Manager). Defaults to None.
- **from_date** (*date or datetime*) – Stating date or datetime to query from (inclusive).
- **to_date** (*date or datetime*) – End date or datetime to query to (inclusive).

- **activity** (*str*) – Filters transactions to match specific activity. Defaults to None.
- **role** (*str*) – Filters transactions to match specific role. Defaults to None.
- **accounts** (*str or List[str] ot Set[str]*) – Optional list of accounts to query. Defaults to None (all).
- **by_activity** (*bool*) – Returns results aggregated by activity if needed. Defaults to False.
- **by_tx_type** (*bool*) – Returns results aggregated by DEBIT/CREDIT if needed. Defaults to False.
- **by_period** (*bool*) – Returns results aggregated by accounting if needed. Defaults to False.
- **by_unit** (*bool*) – Returns results aggregated by unit if needed. Defaults to False.
- **equity_only** (*bool*) – Performs aggregation only on accounts that impact equity only (i.e. Income Statement Generation). Avoids unnecessary inclusion of accounts not relevant to what's needed.
- **signs** (*bool*) – Changes the balance of an account to negative if it represents a "negative" for display purposes. (i.e. Expense accounts will show balance as negative and Income accounts as positive.)
- **force_closing_entry_use** (*bool*) – Forces the use of closing entries if DJANGO_LEDGER_USE_CLOSING_ENTRIES setting is set to False.
- **force_queryset_sorting** (*bool*) – Forces sorting of the TransactionModelQuerySet before aggregation balances. Defaults to false.

Return type
IOResult

class django_ledger.io.io_core.**IOMixIn**

class django_ledger.io.io_core.**IOResult**(*db_from_date: Optional[date] = None*, *db_to_date: Optional[date] = None*, *ce_match: bool = False*, *ce_from_date: Optional[date] = None*, *ce_to_date: Optional[date] = None*, *accounts_digest: Optional[List[Dict]] = None*)

A carrier class to store IO digest information during the digest call.

exception django_ledger.io.io_core.**IOValidationError**(*message*, *code=None*, *params=None*)

django_ledger.io.io_core.**get_localdate**() → date

Convenience function to retrieve the local date of the current system based on the USE_TZ django setting.

Return type
date

django_ledger.io.io_core.**get_localtime**(*tz=None*) → datetime

Convenience function to retrieve the localtime of the current system based on the USE_TZ django setting.

Parameters
tz (*ZoneInfo*) – Optional timezone to use, otherwise the system timezone is used.

Return type
datetime

13.3 IO Context

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

13.4 IO Library

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

This module contains classes and functions used to document, dispatch and commit new transaction into the database.

class django_ledger.io.io_library.**IOBluePrint**(*name: Optional[str] = None*, *precision_decimals: int = 2*)

This class documents instructions required to assemble and dispatch transactions into the ledger.

Parameters

- **name** (*str, optional*) – The human-readable name of the IOBluePrint instance.
- **precision_decimals** (*int*) – The number of decimals to use when balancing transactions. Defaults to 2.

commit(*entity_model: EntityModel*, *user_model*, *ledger_model: Optional[Union[str, LedgerModel, UUID]] = None*, *je_timestamp: Optional[Union[date, datetime, str]] = None*, *post_new_ledgers: bool = False*, *post_journal_entries: bool = False*, ***kwargs*) → Dict

Commits the blueprint transactions to the database.

Parameters

- **entity_model** (*EntityModel*) – The entity model instance where transactions will be committed.
- **user_model** (*UserModel*) – The user model instance executing transactions to check for permissions.
- **ledger_model** (*Optional[Union[str, LedgerModel, UUID]]*) – The ledger model instance identifier to be used for the transactions. If none, a new ledger will be created.
- **je_timestamp** (*date or datetime or str, optional*) – The date and/or time to be used for the transactions. If none, localtime will be used.
- **post_new_ledgers** (*bool*) – If True, newly created ledgers will be posted. Defaults to False.
- **post_journal_entries** (*bool*) – If True, newly created journal entries will be posted. Defaults to False.
- **kwargs** – Keyword arguments passed to the IO Library.

Returns
A dictionary containing the resulting models of the transactions.

Return type
Dict

credit(*account_code: str*, *amount: Union[float, Decimal]*, *description: Optional[str] = None*)

Registers a CREDIT to the specified account..

Parameters

- **account_code** (*str*) – The account code to use for the transaction.
- **amount** (*float or Decimal*) – The amount of the transaction.
- **description** (*str*) – Description of the transaction.

debit(*account_code: str*, *amount: Union[float, Decimal]*, *description: Optional[str] = None*)

Registers a DEBIT to the specified account.

Parameters

- **account_code** (*str*) – The account code to use for the transaction.
- **amount** (*float or Decimal*) – The amount of the transaction.
- **description** (*str*) – Description of the transaction.

get_name(*entity_model:* EntityModel) → str

Determines the name of the blueprint if none provided.

Parameters
entity_model (EntityModel) – The EntityModel instance where the resulting blueprint transactions will be stored.

Returns
The name of the blueprint.

Return type
str

exception django_ledger.io.io_library.**IOBluePrintValidationError**(*message*, *code=None*, *params=None*)

class django_ledger.io.io_library.**IOCursor**(*io_library*, *entity_model:* EntityModel, *user_model*, *coa_model: Optional[Union[*ChartOfAccountModel*, UUID, str]] = None*)

Represents a Django Ledger cursor capable of dispatching transactions to the database. The Cursor class is responsible for coordinating the creation of new ledgers, journal entries and transactions It is a low level interface to the IOBlueprint and IOLibrary classes.

Parameters

- **io_library** (IOLibrary) – The IOLibrary class that contains all the necessary instructions to dispatch the transactions.
- **entity_model** (EntityModel) – The EntityModel instance that will be used for the new transactions.
- **user_model** (*UserModel*) – The UserModel instance that will be used for the new transactions. Used for read permissions of QuerySets.

- **coa_model** (*ChartOfAccountModel or UUID or str.*) – The ChartOfAccountModel instance that contains the accounts to be used for transactions. Instance, UUID or slug can be sued to retrieve the model.

commit(*je_timestamp: Optional[Union[date, datetime, str]] = None*, *post_new_ledgers: bool = False*, *post_journal_entries: bool = False*, ***kwargs*)

Commits the compiled blueprint transactions into the database. This action is irreversible and if journal entries are posted, the books will be immediately impacted by the new transactions. It is encouraged NOT to post anything until the transaction is reviews for accuracy.

Parameters

- **je_timestamp** (*Optional[Union[datetime, date, str]]*) – The date or timestamp used for the committed journal entries. If none, localtime will be used.
- **post_new_ledgers** (*bool*) – If a new ledger is created, the ledger model will be posted to the database.
- **post_journal_entries** (*bool*) – If new journal entries are created, the journal entry models will be posted to the database.
- **kwargs** – Additional keyword arguments passed to the IO commit_txs function.

compile_instructions() → Dict

Compiles the blueprint instructions into Journal Entries and Transactions to be committed to the ledger.

Returns
A dictionary containing the compiled instructions.

Return type
Dict

dispatch(*name*, *ledger_model: Optional[Union[str, LedgerModel, UUID]] = None*, ***kwargs*)

Stages the instructions to be processed by the IOCursor class. This method does not commit the transactions into the database.

Parameters

- **name** (*str*) – The registered blueprint name to be staged.
- **ledger_model** (*Optional[Union[str, LedgerModel, UUID]]*) – Optional ledger model identifier to house the transactions associated with the blueprint. If none is provided, a new ledger model will be created.
- **kwargs** – The keyword arguments to be passed to the blueprint function.

get_account_model_qs() → *AccountModelQuerySet*

Determines the AccountModelQuerySet associated with the Chart of Accounts specified.

Return type
AccountModelQuerySet

get_ledger_model_qs() → *LedgerModelQuerySet*

Determines the ledger model queryset associated with the entity model and user model provided.

Return type
LedgerModelQuerySet

is_committed() → bool

Determines if the IOCursor instance has committed the transactions into the database. A cursor can only commit transactions once.

Returns
True if committed, False otherwise.

Return type
bool

resolve_account_model_qs(*codes: List[str]*) → *AccountModelQuerySet*

Resolves the final AccountModelQuerySet associated with the given account codes used by the blueprint.

Parameters
codes (*List[str]*) – List of codes used during the execution of the blueprint.

Returns
The resolved AccountModelQuerySet associated with the given codes.

Return type
AccountModelQuerySet

resolve_ledger_model_qs() → *LedgerModelQuerySet*

Resolves the final LedgerModelQuerySet associated with the provided ledger model identifiers used by the blueprints.

Returns
The resolved LedgerModelQuerySet associated with the given ledger model identifiers.

Return type
LedgerModelQuerySet

exception django_ledger.io.io_library.**IOCursorValidationError**(*message*, *code=None*, *params=None*)

class django_ledger.io.io_library.**IOLibrary**(*name: str*)

The IO Library is a centralized interface for documenting commonly used operations. The library will register and document the blueprints and their instructions so that they can be dispatched from anywhere in the application.

Parameters
name (*str*) – The human-readable name of the library (i.e. PayRoll, Expenses, Rentals, etc…)

get_blueprint(*name: str*) → Callable

Retrieves a blueprint by name.

Parameters
name (*str*) – The name of the blueprint to retrieve.

Return type
Callable

get_cursor(*entity_model:* EntityModel, *user_model*, *coa_model: Optional[Union[*ChartOfAccountModel, *UUID, str]] = None*) → *IOCursor*

Creates a cursor instance for associated with the library, entity and user model.

Parameters

- **entity_model** (*EntityModel*) – The entity model instance where transactions will be committed.
- **user_model** (*UserModel*) – The user model instance executing the transactions.
- **coa_model** (*ChartOfAccountModel or UUID or str, optional*) – The ChartOfAccountsModel instance or identifier used to determine the AccountModelQuerySet used for the transactions.

Return type
IOCursor

exception django_ledger.io.io_library.**IOLibraryError**(*message*, *code=None*, *params=None*)

class django_ledger.io.io_library.**TransactionInstructionItem**(*account_code: str*, *amount: Union[Decimal, float]*, *tx_type: str*, *description: Optional[str]*, *account_model: Optional[AccountModel] = None*)

A class to represent a transaction instruction used during the development of transaction blueprints.

account_code

The account code of the AccountModel as a String.

Type
str

amount

The transaction amount as a Decimal value. Will be rounded to the nearest decimal place.

Type
Decimal

tx_type

A choice of 'debit' or 'credit' transaction.

Type
str

description

Description of the transaction.

Type
str

account_model

The resolved account model for the transaction. Not to be modified. Defaults to None.

Type
AccountModel

13.5 IO Digest

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

exception django_ledger.io.io_digest.**IODigestValidationError**(*message*, *code=None*, *params=None*)

13.6 Account Roles

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

`django_ledger.io.roles.`**`validate_roles`**(*roles: Union[str, List[str]]*, *raise_exception: bool = True*) → Set[str]

Validates a given role identifier against the valid role available. :param roles: The role or list of roles to validate. :type roles: str or list :param raise_exception: Raises InvalidRoleError if any of the roles provided if not valid. :type raise_exception: bool

Returns
A set of the valid roles.

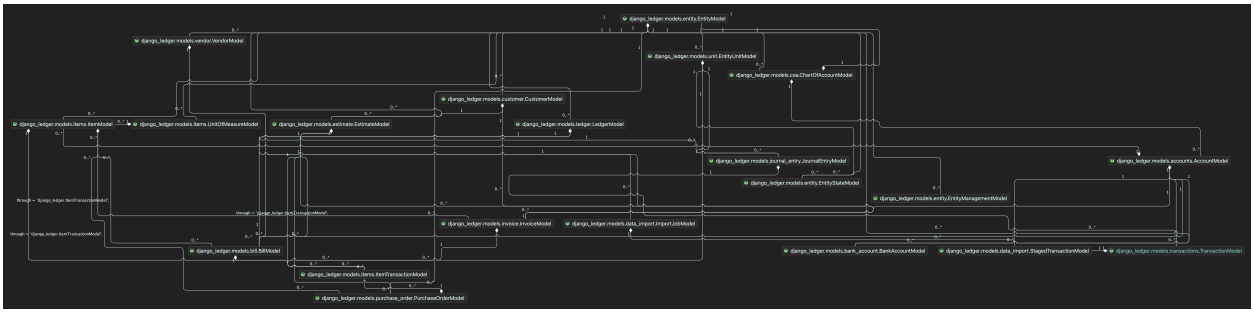
Return type
set

CHAPTER

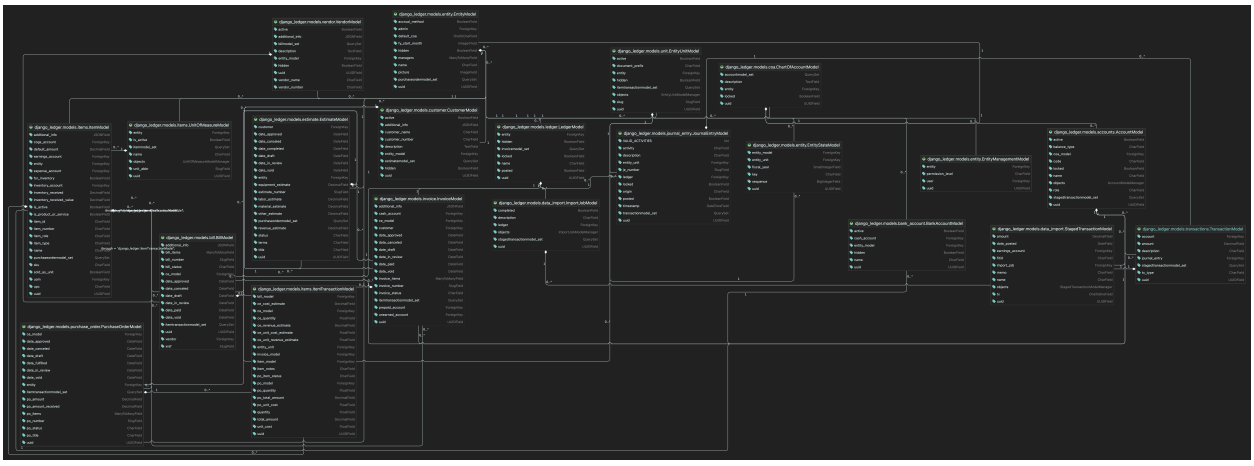
FOURTEEN

MODELS

14.1 Model Dependency Diagram



14.2 Database Fields



14.3 Entity Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan ptulshyan77@gmail.com<>

The EntityModel represents the Company, Corporation, Legal Entity, Enterprise or Person that engage and operate as a business. EntityModels can be created as part of a parent/child model structure to accommodate complex corporate structures where certain entities may be owned by other entities and may also generate consolidated financial statements. Another use case of parent/child model structures is the coordination and authorization of inter-company transactions across multiple related entities. The EntityModel encapsulates all LedgerModel, JournalEntryModel and TransactionModel which is the core structure of Django Ledger in order to track and produce all financials.

The EntityModel must be assigned an Administrator at creation, and may have optional Managers that will have the ability to operate on such EntityModel.

EntityModels may also have different financial reporting periods, (also known as fiscal year), which start month is specified at the time of creation. All key functionality around the Fiscal Year is encapsulated in the EntityReportMixIn.

class django_ledger.models.entity.**EntityManagementModel**(**args*, ***kwargs*)

EntityManagement Model Base Class From Abstract

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.entity.**EntityManagementModelAbstract**(**args*, ***kwargs*)

Entity Management Model responsible for manager permissions to read/write.

class django_ledger.models.entity.**EntityModel**(**args*, ***kwargs*)

Entity Model Base Class From Abstract

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.entity.**EntityModelAbstract**(**args*, ***kwargs*)

The base implementation of the EntityModel. The EntityModel represents the Company, Corporation, Legal Entity, Enterprise or Person that engage and operate as a business. The base model inherit from the Materialized Path Node of the Django Treebeard library. This allows for complex parent/child relationships between Entities to be tracked and managed properly.

The EntityModel also inherits functionality from the following MixIns:

1. *SlugNameMixIn*
2. *PaymentTermsMixIn*
3. *ContactInfoMixIn*
4. *CreateUpdateMixIn*
5. EntityReportMixIn
6. IOMixIn

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

name

The name of the Company, Enterprise, Person, etc. used to identify the Entity.

Type
str

admin

The Django UserModel that will be assigned as the administrator of the EntityModel.

Type
UserModel

default_coa

The default Chart of Accounts Model of the Entity. EntityModel can have multiple Chart of Accounts but only one can be assigned as default.

Type
ChartOfAccounts

managers

The Django UserModels that will be assigned as the managers of the EntityModel by the admin.

Type
UserModel

hidden

A flag used to hide the EntityModel from QuerySets. Defaults to False.

Type
bool

accrual_method

A flag used to define which method of accounting will be used to produce financial statements.

- If False, Cash Method of Accounting will be used.
- If True, Accrual Method of Accounting will be used.

Type
bool

fy_start_month

An integer that specifies the month that the Fiscal Year starts.

Type
int

picture

The image or logo used to identify the company on reports or UI/UX.

class Meta

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

create_account(*code: str*, *role: str*, *name: str*, *balance_type: str*, *active: bool = False*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *raise_exception: bool = True*) → *AccountModel*

Creates a new AccountModel for the EntityModel.

Parameters

- **code** (*str*) – The AccountModel code of the new Account.
- **role** (*str*) – The choice of role that the new Account belongs to.
- **name** (*str*) – The name of the new Account.
- **balance_type** (*str*) – The balance type of the new account. Possible values are 'debit' or 'credit'.
- **active** (*bool*) – Active status of the new account.
- **coa_model** (*ChartOfAccountModel*, *UUID*, *str*) – The ChartOfAccountsModel UUID, model instance or slug to pull accounts from. Uses default Coa if not provided.
- **raise_exception** (*bool*) – Raises EntityModelValidationError if ChartOfAccountsModel is not valid for the EntityModel instance.

Returns

The ChartOfAccountModel and AccountModel instance just created.

Return type

A tuple of *ChartOfAccountModel*, *AccountModel*

create_account_by_kwargs(*account_model_kwargs: Dict*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *raise_exception: bool = True*) → Tuple[*ChartOfAccountModel*, *AccountModel*]

Creates a new AccountModel for the EntityModel by passing AccountModel KWARGS. This is a legacy method for creating a new AccountModel for the EntityModel. Will be deprecated in favor of create_account().

Parameters

- **coa_model** (*ChartOfAccountModel*, *UUID*, *str*) – The ChartOfAccountsModel UUID, model instance or slug to pull accounts from. Uses default Coa if not provided.
- **account_model_kwargs** (*dict*) – A dictionary of kwargs to be used to create the new AccountModel instance.
- **raise_exception** (*bool*) – Raises EntityModelValidationError if ChartOfAccountsModel is not valid for the EntityModel instance.

Returns

The ChartOfAccountModel and AccountModel instance just created.

Return type

A tuple of *ChartOfAccountModel*, *AccountModel*

create_bank_account(*name: str*, *account_type: str*, *active=False*, *cash_account: Optional[AccountModel] = None*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *bank_account_model_kwargs: Optional[Dict] = None*, *commit: bool = True*)

Creates a new BankAccountModel for the EntityModel instance. Estimate will have DRAFT status.

Parameters

- **name** (*str*) – A user defined name for the BankAccountModel.
- **account_type** (*date*) – A choice of BankAccountModel.VALID_ACCOUNT_TYPES.
- **active** (*bool*) – Marks the account as active.
- **cash_account** (AccountModel) – Optional CASH AccountModel associated with the new InvoiceModel. Defaults to CASH default AccountModel role.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels.
- **commit** (*bool*) – If True, commits the new BankAccountModel in the Database. Defaults to True.
- **bank_account_model_kwargs** (*Dict*) – Additional kwargs for the new BankAccountModel instance.

Returns

The newly created PurchaseOrderModel in DRAFT state.

Return type

PurchaseOrderModel

create_bill(*vendor_model: Union[VendorModel, UUID, str]*, *terms: str*, *date_draft: Optional[Union[datetime, date]] = None*, *xref: Optional[str] = None*, *cash_account: Optional[AccountModel] = None*, *prepaid_account: Optional[AccountModel] = None*, *payable_account: Optional[AccountModel] = None*, *additional_info: Optional[Dict] = None*, *ledger_name: Optional[str] = None*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *commit: bool = True*)

Creates a new BillModel for the EntityModel instance. Bill will have DRAFT status.

Parameters

- **vendor_model** (*VendorModel or UUID or str*) – The VendorModel, VendorModel UUID or VendorModel Number
- **terms** (*str*) – Payment terms of the new BillModel. A choice of BillModel.TERM_CHOICES_VALID
- **date_draft** (*date or datetime*) – Date to use as draft date for the new BillModel.
- **xref** (*str*) – Optional External Reference for the Bill (i.e. Vendor invoice number.)
- **cash_account** (AccountModel) – Optional CASH AccountModel associated with the new BillModel. Defaults to CASH default AccountModel role.
- **prepaid_account** (AccountModel) – Optional PREPAID AccountModel associated with the new BillModel for accruing purposes. Defaults to PREPAID default AccountModel role.
- **payable_account** (AccountModel) – Optional PAYABLE AccountModel associated with the new BillModel for accruing purposes. Defaults to ACCOUNTS PAYABLE default AccountModel role.
- **additional_info** (*Dict*) – Additional user-defined information stored as JSON in the Database.

- **ledger_name** (*str*) – Optional LedgerModel name to be assigned to the BillModel instance.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels
- **commit** (*bool*) – If True, commits the new BillModel in the Database.

Returns
The newly created BillModel in DRAFT state.

Return type
BillModel

create_chart_of_accounts(*assign_as_default: bool = False*, *coa_name: Optional[str] = None*, *commit: bool = False*) → *ChartOfAccountModel*

Creates a Chart of Accounts for the Entity Model and optionally assign it as the default Chart of Accounts. EntityModel must have a default Chart of Accounts before being able to transact.

Parameters

- **coa_name** (*str*) – The new CoA name. If not provided will be auto generated based on the EntityModel name.
- **commit** (*bool*) – Commits the transaction into the DB. A ChartOfAccountModel will
- **assign_as_default** (*bool*) – Assigns the newly created ChartOfAccountModel as the EntityModel default_coa.

Returns
The newly created chart of accounts model.

Return type
ChartOfAccountModel

create_customer(*customer_model_kwargs: Dict*, *commit: bool = True*) → *CustomerModel*

Creates a new CustomerModel associated with the EntityModel instance.

Parameters

- **customer_model_kwargs** (*dict*) – The kwargs to be used for the new CustomerModel.
- **commit** (*bool*) – Saves the CustomerModel instance in the Database.

Return type
CustomerModel

classmethod create_entity(*name: str*, *use_accrual_method: bool*, *admin: User*, *fy_start_month: int*, *parent_entity=None*)

Convenience Method to Create a new Entity Model. This is the preferred method to create new Entities in order to properly handle potential parent/child relationships between EntityModels.

Parameters

- **name** (*str*) – The name of the new Entity.
- **use_accrual_method** (*bool*) – If True, accrual method of accounting will be used, otherwise Cash Method of accounting will be used.
- **fy_start_month** (*int*) – The month which represents the start of a new fiscal year. 1 represents January, 12 represents December.
- **admin** (*UserModel*) – The administrator of the new EntityModel.

- **parent_entity** (EntityModel) – The parent Entity Model of the newly created Entity. If provided, the admin user must also be admin of the parent company.

create_estimate(*estimate_title: str*, *contract_terms: str*, *customer_model: Union[CustomerModel, UUID, str]*, *date_draft: Optional[date] = None*, *commit: bool = True*)

Creates a new EstimateModel for the EntityModel instance. Estimate will have DRAFT status.

Parameters

- **estimate_title** (*str*) – A user defined title for the Estimate.
- **date_draft** (*date*) – Optional date to use as Draft Date. Defaults to localdate() if None.
- **customer_model** (CustomerModel *or UUID or str*) – The CustomerModel, CustomerModel UUID or CustomerModel Number
- **contract_terms** (*str*) – A choice of EstimateModel.CONTRACT_TERMS_CHOICES_VALID
- **commit** (*bool*) – If True, commits the new PO in the Database. Defaults to True.

Returns
The newly created PurchaseOrderModel in DRAFT state.

Return type
PurchaseOrderModel

create_invoice(*customer_model: Union[VendorModel, UUID, str]*, *terms: str*, *cash_account: Optional[AccountModel] = None*, *prepaid_account: Optional[AccountModel] = None*, *payable_account: Optional[AccountModel] = None*, *additional_info: Optional[Dict] = None*, *ledger_name: Optional[str] = None*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *date_draft: Optional[date] = None*, *commit: bool = True*)

Creates a new InvoiceModel for the EntityModel instance. Invoice will have DRAFT status.

Parameters

- **customer_model** (CustomerModel *or UUID or str*) – The CustomerModel, CustomerModel UUID or CustomerModel Number
- **terms** (*str*) – A choice of InvoiceModel.TERM_CHOICES_VALID
- **cash_account** (AccountModel) – Optional CASH AccountModel associated with the new InvoiceModel. Defaults to CASH default AccountModel role.
- **prepaid_account** (AccountModel) – Optional PREPAID AccountModel associated with the new InvoiceModel for accruing purposes. Defaults to PREPAID default AccountModel role.
- **payable_account** (AccountModel) – Optional PAYABLE AccountModel associated with the new InvoiceModel for accruing purposes. Defaults to ACCOUNTS PAYABLE default AccountModel role.
- **additional_info** (*Dict*) – Additional user-defined information stored as JSON in the Database.
- **ledger_name** (*str*) – Optional LedgerModel name to be assigned to the InvoiceModel instance.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels
- **date_draft** (*date*) – Optional date to use as Draft Date. Defaults to localdate() if None.

- **commit** (*bool*) – If True, commits the new BillModel in the Database.

Returns
The newly created InvoiceModel in DRAFT state.

Return type
InvoiceModel

create_item_expense(*name: str*, *expense_type: str*, *uom_model: Union[UUID, UnitOfMeasureModel]*, *expense_account: Optional[Union[UUID, AccountModel]] = None*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *commit: bool = True*) → *ItemModel*

Creates a new items of type EXPENSE.

Parameters

- **name** (*str*) – The name of the new service.
- **expense_type** (*str*) – The type of expense. A choice of ItemModel.ITEM_TYPE_CHOICES
- **uom_model** – The UOM UUID or UnitOfMeasureModel of the Service. Will be validated if provided.
- **expense_account** (AccountModel) – Optional EXPENSE_OPERATIONAL AccountModel associated with the new Expense Item. Defaults to EXPENSE_OPERATIONAL default AccountModel role.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels.
- **commit** (*bool*) – Commits the ItemModel in the DB. Defaults to True.

Return type
ItemModel

create_item_inventory(*name: str*, *uom_model: Union[UUID, UnitOfMeasureModel]*, *item_type: str*, *inventory_account: Optional[Union[UUID, AccountModel]] = None*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *commit: bool = True*)

Creates a new items of type INVENTORY.

Parameters

- **name** (*str*) – The name of the new service.
- **item_type** (*str*) – The type of expense. A choice of ItemModel.ITEM_TYPE_CHOICES
- **uom_model** – The UOM UUID or UnitOfMeasureModel of the Service. Will be validated if provided.
- **inventory_account** (AccountModel) – Optional ASSET_CA_INVENTORY AccountModel associated with the new Expense Item. Defaults to ASSET_CA_INVENTORY default AccountModel role.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels.
- **commit** (*bool*) – Commits the ItemModel in the DB. Defaults to True.

Return type
ItemModel

create_item_product(*name: str*, *item_type: str*, *uom_model: Union[UUID, UnitOfMeasureModel]*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *commit: bool = True*) → *ItemModel*

Creates a new items of type PRODUCT.

Parameters

- **name** (*str*) – Name of the new service.
- **item_type** (*str*) – The type of product. A choice of ItemModel.ITEM_TYPE_CHOICES
- **uom_model** – The UOM UUID or UnitOfMeasureModel of the Service. Will be validated if provided.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels.
- **commit** (*bool*) – Commits the ItemModel in the DB. Defaults to True.

Returns
The created Product.

Return type
ItemModel

create_item_service(*name: str*, *uom_model: Union[UUID, UnitOfMeasureModel]*, *coa_model: Optional[Union[ChartOfAccountModel, UUID, str]] = None*, *commit: bool = True*) → *ItemModel*

Creates a new items of type SERVICE.

Parameters

- **name** (*str*) – Name of the new service.
- **uom_model** – The UOM UUID or UnitOfMeasureModel of the Service. Will be validated if provided.
- **coa_model** (ChartOfAccountModel) – Optional ChartOfAccountsModel to use when fetching default role AccountModels.
- **commit** (*bool*) – Commits the ItemModel in the DB. Defaults to True.

Returns
The created Service.

Return type
ItemModel

create_purchase_order(*po_title: Optional[str] = None*, *estimate_model=None*, *date_draft: Optional[date] = None*, *commit: bool = True*)

Creates a new PurchaseOrderModel for the EntityModel instance. PO will have DRAFT status.

Parameters

- **po_title** (*str*) – The user defined title for the new Purchase Order Model.
- **date_draft** (*date*) – Optional date to use as Draft Date. Defaults to localdate() if None.
- **estimate_model** (EstimateModel) – The EstimateModel to associate the PO for tracking.
- **commit** (*bool*) – If True, commits the new PO in the Database. Defaults to True.

Returns
The newly created PurchaseOrderModel in DRAFT state.

Return type
PurchaseOrderModel

create_uom(*name: str*, *unit_abbr: str*, *active: bool = True*, *commit: bool = True*) → *UnitOfMeasureModel*

Creates a new Unit of Measure Model associated with the EntityModel instance

Parameters

- **name** (*str*) – The user defined name of the new Unit of Measure Model instance.
- **unit_abbr** (*str*) – The unique slug abbreviation of the UoM model. Will be indexed.
- **active** (*bool*) – Mark this UoM as active.
- **commit** (*bool*) – Saves the model in the DB if True. Defaults to True

Return type
UnitOfMeasureModel

create_vendor(*vendor_model_kwargs: Dict*, *commit: bool = True*) → *VendorModel*

Creates a new VendorModel associated with the EntityModel instance.

Parameters

- **vendor_model_kwargs** (*dict*) – The kwargs to be used for the new VendorModel.
- **commit** (*bool*) – Saves the VendorModel instance in the Database.

Return type
VendorModel

generate_slug(*commit: bool = False*, *raise_exception: bool = True*, *force_update: bool = False*) → str

Convenience method to create the EntityModel slug.

Parameters

- **force_update** (*bool*) – If True, will update the EntityModel slug.
- **raise_exception** (*bool*) – Raises ValidationError if EntityModel already has a slug.
- **commit** (*bool*) – If True,

static generate_slug_from_name(*name: str*) → str

Uses Django's slugify function to create a valid slug from any given string.

Parameters
name (*str*) – The name or string to slugify.

Return type
The slug as a String.

get_accounts_url() → str

The EntityModel Code of Accounts llist import URL.

Returns
EntityModel Code of Accounts llist import URL as a string.

Return type
str

get_accounts_with_codes(*code_list: Union[str, List[str], Set[str]]*, *coa_model: Optional[Union[*ChartOfAccountModel*, UUID, str]] = None*) → *AccountModelQuerySet*

Fetches the AccountModelQuerySet with provided code list.

Parameters

- **coa_model** (*ChartOfAccountModel, UUID, str*) – The ChartOfAccountsModel UUID, model instance or slug to pull accounts from. Uses default Coa if not provided.
- **code_list** (*list or str*) – Code or list of codes to fetch.

Returns
The requested AccountModelQuerySet with applied code filter.

Return type
AccountModelQuerySet

get_all_accounts(*active: bool = True*, *order_by: Optional[Tuple[str]] = ('code',)*) → *AccountModelQuerySet*

Fetches all AccountModelQuerySet associated with the EntityModel.

Parameters

- **active** (*bool*) – Selects only active accounts.
- **order_by** (*list of strings.*) – Optional list of fields passed to the order_by QuerySet method.

Returns
The AccountModelQuerySet of the assigned default CoA.

Return type
AccountModelQuerySet

get_all_coa_accounts(*order_by: Optional[Tuple[str]] = ('code',)*, *active: bool = True*) → Tuple[*ChartOfAccountModelQuerySet*, Dict[*ChartOfAccountModel*, *AccountModelQuerySet*]]

Fetches all the AccountModels associated with the EntityModel grouped by ChartOfAccountModel.

Parameters

- **active** (*bool*) – Selects only active accounts.
- **order_by** (*list of strings.*) – Optional list of fields passed to the order_by QuerySet method.

Returns
Tuple – The ChartOfAccountModelQuerySet and a grouping of AccountModels by ChartOfAccountModel as keys.

Return type
Tuple[*ChartOfAccountModelQuerySet*, Dict[*ChartOfAccountModel*, *AccountModelQuerySet*]

get_balance_sheet_url() → str

The EntityModel Balance Sheet Statement URL.

Returns
EntityModel Balance Sheet Statement URL as a string.

Return type
str

get_bank_accounts(*active: bool = True*) → *BankAccountModelQuerySet*

Fetches a QuerySet of BankAccountModels associated with the EntityModel instance.

Parameters
active (*bool*) – If True, returns only active Bank Accounts. Defaults to True.

Return type
BankAccountModelQuerySet

get_banks_url() → str

The EntityModel bank account list URL.

Returns
EntityModel bank account list URL as a string.

Return type
str

get_bills()

Fetches a QuerySet of BillModels associated with the EntityModel instance.

Return type
BillModelQuerySet

get_bills_url() → str

The EntityModel bill list URL.

Returns
EntityModel bill list URL as a string.

Return type
str

get_cashflow_statement_url() → str

The EntityModel Cashflow Statement URL.

Returns
EntityModel Cashflow Statement URL as a string.

Return type
str

get_coa_accounts(*coa_model: Optional[Union[*ChartOfAccountModel*, UUID, str]] = None*, *active: bool = True*, *order_by: Optional[Tuple] = ('code',)*) → *AccountModelQuerySet*

Fetches the AccountModelQuerySet for a specific ChartOfAccountModel.

Parameters

- **coa_model** (*ChartOfAccountModel*, *UUID*, *str*) – The ChartOfAccountsModel UUID, model instance or slug to pull accounts from. If None, will use default CoA.
- **active** (*bool*) – Selects only active accounts.
- **order_by** (*list of strings.*) – Optional list of fields passed to the order_by QuerySet method.

Returns
The AccountModelQuerySet of the assigned default CoA.

Return type
AccountModelQuerySet

get_coa_model_qs(*active: bool = True*)

Fetches the current Entity Model instance Chart of Accounts Model Queryset.

Parameters
active (*bool*) – Returns only active Chart of Account Models. Defaults to True.

Return type
ChartOfAccountModelQuerySet

get_customers(*active: bool = True*) → *CustomerModelQueryset*

Fetches the CustomerModel associated with the EntityModel instance.

Parameters
active (*bool*) – Active CustomerModel only. Defaults to True.

Returns
The EntityModel instance CustomerModelQueryset with applied filters.

Return type
CustomerModelQueryset

get_customers_url() → str

The EntityModel customers list URL.

Returns
EntityModel customers list URL as a string.

Return type
str

get_dashboard_url() → str

The EntityModel Dashboard URL.

Returns
EntityModel dashboard URL as a string.

Return type
str

get_data_import_url() → str

The EntityModel transaction import URL.

Returns
EntityModel transaction import URL as a string.

Return type
str

get_default_account_for_role(*role: str*, *coa_model: Optional[ChartOfAccountModel] = None*) → *AccountModel*

Gets the given role default AccountModel from the provided CoA. CoA will be validated against the EntityModel instance.

Parameters

- **role** (*str*) – The CoA role to fetch the corresponding default Account Model.
- **coa_model** (ChartOfAccountModel) – The CoA Model to pull default account from. If not provided, will use EntityModel default CoA.

Returns
The default account model for the specified CoA role.

Return type
AccountModel

get_default_coa(*raise_exception: bool = True*) → Optional[*ChartOfAccountModel*]

Fetches the EntityModel default Chart of Account.

Parameters
raise_exception (*bool*) – Raises exception if no default CoA has been assigned.

Returns
The EntityModel default ChartOfAccount.

Return type
ChartOfAccountModel

get_default_coa_accounts(*active: bool = True*, *order_by: Optional[Tuple[str]] = ('code',)*, *raise_exception: bool = True*) → Optional[*AccountModelQuerySet*]

Fetches the default AccountModelQuerySet.

Parameters

- **active** (*bool*) – Selects only active accounts.
- **order_by** (*list of strings.*) – Optional list of fields passed to the order_by QuerySet method.
- **raise_exception** (*bool*) – Raises EntityModelValidationError if no default_coa found.

Returns
The AccountModelQuerySet of the assigned default CoA.

Return type
AccountModelQuerySet

get_delete_url() → str

The EntityModel delete URL.

Returns
EntityModel delete URL as a string.

Return type
str

get_estimates()

Fetches a QuerySet of EstimateModels associated with the EntityModel instance.

Return type
EstimateModelQuerySet

get_income_statement_url() → str

The EntityModel Income Statement URL.

Returns
EntityModel Income Statement URL as a string.

Return type
str

get_invoices()

Fetches a QuerySet of InvoiceModels associated with the EntityModel instance.

Return type
InvoiceModelQuerySet

get_invoices_url() → str

The EntityModel invoice list URL.

Returns
EntityModel invoice list URL as a string.

Return type
str

get_items_all(*active: bool = True*) → *ItemModelQuerySet*

Fetches all EntityModel instance ItemModel's. QuerySet selects relevant related fields to avoid additional DB queries for most use cases.

Parameters
active (*bool*) – Filters the QuerySet to active accounts only. Defaults to True.

Return type
ItemModelQuerySet

get_items_expenses(*active: bool = True*) → *ItemModelQuerySet*

Fetches all EntityModel instance ItemModel's that qualify as Products. QuerySet selects relevant related fields to avoid additional DB queries for most use cases.

Parameters
active (*bool*) – Filters the QuerySet to active accounts only. Defaults to True.

Return type
ItemModelQuerySet

get_items_inventory(*active: bool = True*)

Fetches all EntityModel instance ItemModel's that qualify as inventory. QuerySet selects relevant related fields to avoid additional DB queries for most use cases.

Parameters
active (*bool*) – Filters the QuerySet to active accounts only. Defaults to True.

Return type
ItemModelQuerySet

get_items_inventory_wip(*active: bool = True*)

Fetches all EntityModel instance ItemModel's that qualify as work in progress inventory. QuerySet selects relevant related fields to avoid additional DB queries for most use cases.

Parameters
active (*bool*) – Filters the QuerySet to active accounts only. Defaults to True.

Return type
ItemModelQuerySet

get_items_products(*active: bool = True*) → *ItemModelQuerySet*

Fetches all EntityModel instance ItemModel's that qualify as Products. QuerySet selects relevant related fields to avoid additional DB queries for most use cases.

Parameters
active (*bool*) – Filters the QuerySet to active accounts only. Defaults to True.

Return type
ItemModelQuerySet

get_items_services(*active: bool = True*) → *ItemModelQuerySet*

Fetches all EntityModel instance ItemModel's that qualify as Services. QuerySet selects relevant related fields to avoid additional DB queries for most use cases.

Parameters
active (*bool*) – Filters the QuerySet to active accounts only. Defaults to True.

Return type
ItemModelQuerySet

get_ledgers_url() → str

The EntityModel Ledger List URL.

Returns
EntityModel ledger list URL as a string.

Return type
str

get_manage_url() → str

The EntityModel Manage URL.

Returns
EntityModel manage URL as a string.

Return type
str

get_purchase_orders()

Fetches a QuerySet of PurchaseOrderModels associated with the EntityModel instance.

Return type
PurchaseOrderModelQuerySet

get_uom_all() → *UnitOfMeasureModelQuerySet*

Fetches the EntityModel instance Unit of Measures QuerySet.

Return type
UnitOfMeasureModelQuerySet

get_vendors(*active: bool = True*) → *VendorModelQuerySet*

Fetches the VendorModels associated with the EntityModel instance.

Parameters
active (*bool*) – Active VendorModels only. Defaults to True.

Returns
The EntityModel instance VendorModelQuerySet with applied filters.

Return type
VendorModelQuerySet

get_vendors_url() → str

The EntityModel vendors list URL.

Returns
EntityModel vendors list URL as a string.

Return type
str

has_default_coa() → bool

Determines if the EntityModel instance has a Default CoA.

Returns
True if EntityModel instance has a Default CoA.

Return type
bool

static inventory_adjustment(*counted_qs*, *recorded_qs*) → defaultdict

Computes the necessary inventory adjustment to update balance sheet.

Parameters

- **counted_qs** (ItemTransactionModelQuerySet) – Inventory recount queryset from Purchase Order received inventory. See ItemTransactionModelManager.inventory_count. Expects ItemTransactionModelQuerySet to be formatted "as values".
- **recorded_qs** (ItemModelQuerySet) – Inventory received currently recorded for each inventory item. See ItemTransactionModelManager.inventory_count Expects ItemModelQuerySet to be formatted "as values".

Returns

A dictionary with necessary adjustments with keys as tuple:

0. item_model_id
1. item_model__name
2. item_model__uom__name

Return type
defaultdict

populate_default_coa(*activate_accounts: bool = False*, *force: bool = False*, *ignore_if_default_coa: bool = True*, *coa_model: Optional[ChartOfAccountModel] = None*, *commit: bool = True*)

Populates the EntityModel default CoA with the default Chart of Account list provided by Django Ledger or user defined. See DJANGO_LEDGER_DEFAULT_COA setting.

Parameters

- **activate_accounts** (*bool*) – Activates all AccountModels for immediate use. Defaults to False.
- **force** (*bool*) – Forces the creation of accounts even if other accounts are present. Defaults to False.
- **ignore_if_default_coa** (*bool*) – Raises exception if EntityModel already has a default CoA. Defaults to True.
- **coa_model** (ChartOfAccountModel) – Optional CoA Model to populate. Will be validated against EntityModel if provided.
- **commit** (*bool*) –
- **True.** (*' Commits the newly created CoA into the Database. Defaults to*) –

recorded_inventory(*item_qs: Optional[*ItemModelQuerySet*] = None*, *as_values: bool = True*) → *ItemModelQuerySet*

Recorded inventory on the books marked as received. PurchaseOrderModel drives the ordering and receiving of inventory. Once inventory is marked as "received" recorded inventory of each item is updated by calling *update_inventory*. This function returns relevant values of the recoded inventory, including Unit of Measures.

Parameters

- **item_qs** (*ItemModelQuerySet*) – Pre fetched ItemModelQuerySet. Avoids additional DB Query.
- **as_values** (*bool*) – Returns a list of dictionaries by calling the Django values() QuerySet function.

Returns
The ItemModelQuerySet containing inventory ItemModels with additional Unit of Measure information.

Return type
ItemModelQuerySet

update_inventory(*commit: bool = False*) → Tuple[defaultdict, *ItemTransactionModelQuerySet*, *ItemModelQuerySet*]

Triggers an inventory recount with optional commitment of transaction.

Parameters
commit – Updates all inventory ItemModels with the new inventory count.

Returns

Return a tuple as follows:

0. All necessary inventory adjustments as a dictionary.
1. The recounted inventory.
2. The recorded inventory on Balance Sheet.

Return type
Tuple[defaultdict, *ItemTransactionModelQuerySet*, *ItemModelQuerySet*]

validate_account_model_for_coa(*account_model: AccountModel*, *coa_model: ChartOfAccountModel*, *raise_exception: bool = True*) → bool

Validates that the AccountModel provided belongs to the CoA Model provided.

Parameters

- **account_model** (*AccountModel*) – The AccountModel to validate.
- **coa_model** (*ChartOfAccountModel*) – The ChartOfAccountModel to validate against.
- **raise_exception** (*bool*) – Raises EntityModelValidationError if AccountModel is invalid for the EntityModel and CoA instance.

Returns
True if valid, else False.

Return type
bool

validate_chart_of_accounts_for_entity(*coa_model: ChartOfAccountModel*, *raise_exception: bool = True*) → bool

Validates the CoA Model against the EntityModel instance.

Parameters

- **coa_model** (ChartOfAccountModel) – The CoA Model to validate.
- **raise_exception** (*bool*) – Raises EntityModelValidationError if CoA Model is not valid for the EntityModel instance.

Returns
True if valid, else False.

Return type
bool

validate_item_qs(*item_qs: ItemModelQuerySet*, *raise_exception: bool = True*) → bool

Validates the given ItemModelQuerySet against the EntityModel instance. :param item_qs: The ItemModelQuerySet to validate. :type item_qs: ItemModelQuerySet :param raise_exception: Raises EntityModelValidationError if ItemModelQuerySet is not valid. :type raise_exception: bool

Returns
True if valid, else False.

Return type
bool

class django_ledger.models.entity.**EntityModelFiscalPeriodMixIn**

This class encapsulates the functionality needed to determine the start and end of all financial periods of an EntityModel. At the moment of creation, an EntityModel must be assigned a calendar month which is going to determine the start of the Fiscal Year.

get_fiscal_quarter_dates(*year: int*, *quarter: int*, *fy_start_month: Optional[int] = None*) → Tuple[date, date]

Convenience method to get in one shot both, fiscal year quarter start and end dates.

Parameters

- **year** (*int*) – The fiscal year associated with the requested start and end date.
- **quarter** (*int*) – The quarter number associated with the requested start and end date.
- **fy_start_month** (*int*) – Optional fiscal year month start. If passed, it will override the EntityModel setting.

Returns
Both, the date when the requested EntityModel fiscal year quarter start and end date as a tuple. The start date will be first.

Return type
tuple

get_fiscal_year_dates(*year: int*, *fy_start_month: Optional[int] = None*) → Tuple[date, date]

Convenience method to get in one shot both, fiscal year start and end dates.

Parameters

- **year** (*int*) – The fiscal year associated with the requested start and end date.
- **fy_start_month** (*int*) – Optional fiscal year month start. If passed, it will override the EntityModel setting.

Returns
Both, the date when the requested EntityModel fiscal year start and end date as a tuple. The start date will be first.

Return type
tuple

`get_fy_end`(*year: int*, *fy_start_month: Optional[int] = None*) → date

The fiscal year ending date of the EntityModel, according to its settings.

Parameters

- **year** (*int*) – The fiscal year associated with the requested end date.
- **fy_start_month** (*int*) – Optional fiscal year month start. If passed, it will override the EntityModel setting.

Returns
The date when the requested EntityModel fiscal year ends.

Return type
date

`get_fy_for_date`(*dt: Union[date, datetime]*, *as_str: bool = False*) → Union[str, int]

Given a known date, returns the EntityModel fiscal year associated with the given date.

Parameters

- **dt** (*date*) – Date to evaluate.
- **as_str** (*bool*) – If True, return date as a string.

Returns
Fiscal year as an integer or string, depending on as_str parameter.

Return type
str or date

`get_fy_start`(*year: int*, *fy_start_month: Optional[int] = None*) → date

The fiscal year start date of the EntityModel, according to its settings.

Parameters

- **year** (*int*) – The fiscal year associated with the requested start date.
- **fy_start_month** (*int*) – Optional fiscal year month start. If passed, it will override the EntityModel setting.

Returns
The date when the requested EntityModel fiscal year starts.

Return type
date

`get_fy_start_month`() → int

The fiscal year start month represents the month (as an integer) when the assigned fiscal year of the EntityModel starts.

Returns
An integer representing the month that the fiscal year starts.

Return type
int

Examples

- 1 -> January.
- 4 -> April.
- 9 -> September.

get_quarter_end(*year: int*, *quarter: int*, *fy_start_month: Optional[int] = None*) → date

The fiscal year quarter ending date of the EntityModel, according to its settings.

Parameters

- **year** (*int*) – The fiscal year associated with the requested end date.
- **quarter** (*int*) – The quarter number associated with the requested end date.
- **fy_start_month** (*int*) – Optional fiscal year month start. If passed, it will override the EntityModel setting.

Returns
The date when the requested EntityModel quarter ends.

Return type
date

get_quarter_start(*year: int*, *quarter: int*, *fy_start_month: Optional[int] = None*) → date

The fiscal year quarter starting date of the EntityModel, according to its settings.

Parameters

- **year** (*int*) – The fiscal year associated with the requested start date.
- **quarter** (*int*) – The quarter number associated with the requested start date.
- **fy_start_month** (*int*) – Optional fiscal year month start. If passed, it will override the EntityModel setting.

Returns
The date when the requested EntityModel quarter starts.

Return type
date

validate_month(*month: int*)

Validates the month as a valid parameter for other functions. Makes sure that only integers between 1 and 12 are used to refer to a particular month. Prevents injection of invalid values from views into the IOMixIn.

Parameters
month (*int*) – The month number to validate.

Raises
ValidationError – If month is not valid.

validate_quarter(*quarter: int*)

Validates the quarter as a valid parameter for other functions. Makes sure that only integers 1,2,3, or 4 are used to refer to a particular Quarter. Prevents injection of invalid values from views into the IOMixIn.

Parameters
quarter (*int*) – The quarter number to validate.

Raises
ValidationError – If quarter is not valid.

class django_ledger.models.entity.**EntityModelManager**(**args*, ***kwargs*)

A custom defined EntityModel Manager. This ModelManager uses the custom defined EntityModelQuerySet as default. Inherits from the Materialized Path Node Manager to include the necessary methods to manage tree-like models. This Model Manager keeps track and maintains a root/parent/child relationship between Entities for the purposes of producing consolidated financial statements.

Examples

```
>>> user = request.user
>>> entity_model_qs = EntityModel.objects.for_user(user_model=user)
```

for_user(*user_model*)

This QuerySet guarantees that Users do not access or operate on EntityModels that don't have access to. This is the recommended initial QuerySet.

Parameters
user_model – The Django User Model making the request.

Returns

A filtered QuerySet of EntityModels that the user has access. The user has access to an Entity if:

1. Is the Administrator.
2. Is a manager.

Return type
EntityModelQuerySet

get_queryset()

Sets the custom queryset as the default.

class django_ledger.models.entity.**EntityModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined EntityModel QuerySet. Inherits from the Materialized Path Node QuerySet Class from Django Treebeard.

hidden()

A QuerySet of all hidden EntityModel.

Returns
A filtered QuerySet of hidden EntityModels only.

Return type
EntityModelQuerySet

visible()

A Queryset of all visible EntityModel.

Returns
A filtered QuerySet of visible EntityModels only.

Return type
EntityModelQuerySet

exception django_ledger.models.entity.**EntityModelValidationError**(*message*, *code=None*, *params=None*)

class django_ledger.models.entity.**EntityStateModel**(**args*, ***kwargs*)

Entity State Model Base Class from Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.entity.**EntityStateModelAbstract**(**args*, ***kwargs*)

14.4 Entity Unit Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

An EntityUnit is a logical, user-defined grouping which is assigned to JournalEntryModels to help segregate business operations into separate components. Examples of business units may include Departments (i.e. Human Resources, IT, etc.) office locations, a real estate property, or any other label relevant to the business.

An EntityUnit is self contained. Meaning that double entry accounting rules apply to all transactions associated within them. When An Invoice or Bill is updated, the migration process generates the appropriate Journal Entries associated with each unit, if any. This means that an invoice or bill can split items into different units and the migration process will allocate costs to each unit accordingly.

The main advantages of EntityUnits are:

1. Entity units can generate their own financial statements which can give additional insight to specific operations of the business.
2. Entity units can be assigned to specific items on Bills and Invoices, providing additional flexibility to track inventory, expenses or income attributable to specific units of the business.

class django_ledger.models.unit.**EntityUnitModel**(**args*, ***kwargs*)

Base Model Class for EntityUnitModel

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.unit.**EntityUnitModelAbstract**(**args*, ***kwargs*)

Base implementation of the EntityUnitModel.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

slug

A unique, indexed identifier for the EntityUnitModel instance used in URLs and queries.

Type
str

entity

The EntityModel associated with this EntityUnitModel.

Type
EntityModel

document_prefix

A predefined prefix automatically incorporated into JournalEntryModel document numbers. Max Length 3. May be user defined. Must be unique for the EntityModel.

Type
str

active

Active EntityUnits may transact. Inactive units are considered archived. Defaults to True.

Type
bool

hidden

Hidden Units will not show on drop down menus on the UI. Defaults to False.

Type
bool

class Meta

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

create_entity_unit_slug(*name: Optional[str] = None*, *force: bool = False*, *add_suffix: bool = True*, *k: int = 5*) → str

Automatically generates a EntityUnitModel slug. If slug is present, will not be replaced. Called during the clean() method.

Parameters

- **force** (*bool*) – Forces generation of new slug if already present.
- **name** (*str*) – The name used to create slug. If none, the unit name will be used.
- **add_suffix** (*bool*) – Adds a random suffix to the slug. Defaults to True.
- **k** (*int*) – Length of the suffix if add_suffix is True. Defaults to 5.

Returns
The EntityUnitModel slug, regardless if generated or not.

Return type
str

get_dashboard_url() → str

The dashboard URL of the EntityModelUnit.

Returns
The EntityModelUnit instance dashboard URL.

Return type
str

class django_ledger.models.unit.**EntityUnitModelManager**(**args*, ***kwargs*)

for_entity(*entity_slug: str*, *user_model*)

Fetches a QuerySet of EntityUnitModels associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> bill_model_qs = EntityUnitModel.objects.for_entity(user_model=request_user,␣
↪entity_slug=slug)
```

Returns

Returns a EntityUnitModelQuerySet with applied filters.

Return type

EntityUnitModelQuerySet

class django_ledger.models.unit.**EntityUnitModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined EntityUnitModel Queryset.

exception django_ledger.models.unit.**EntityUnitModelValidationError**(*message*, *code=None*, *params=None*)

14.5 Account Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

The AccountModel groups and sorts transactions involving the company's assets, liabilities and equities. Per accounting principles, an Account must be either a DEBIT-type balance account or a CREDIT-type balance account, depending on its purpose.

The AccountModel plays a major role when creating Journal Entries in a double entry accounting systems where a DEBIT to a DEBIT-type AccountModel will increase its balance, and a CREDIT to a DEBIT-type AccountModel will reduce its balance. Conversely, a CREDIT to a CREDIT-type AccountModel will increase its balance, and a DEBIT to a CREDIT-type AccountModel will reduce its balance.

It is entirely up to the user to adopt the chart of accounts that best suits the EntityModel. The user may choose to use the default Chart of Accounts provided by Django Ledger when creating a new EntityModel.

In Django Ledger, all account models must be assigned a role from ACCOUNT_ROLES. Roles are a way to group accounts to a common namespace, regardless of its user-defined fields. Roles are an integral part to Django Ledger since they are critical when requesting and producing financial statements and financial ratio calculations.

AccountModels may also contain parent/child relationships as implemented by the Django Treebeard functionality.

class django_ledger.models.accounts.**AccountModel**(**args*, ***kwargs*)

Base Account Model from Account Model Abstract Class

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.accounts.**AccountModelAbstract**(**args*, ***kwargs*)

Django Ledger Base Account Model Abstract. This is the main abstract class which the Account Model database will inherit, and it contains the fields/columns/attributes which the said ledger table will have. In addition to the attributes mentioned below, it also has the fields/columns/attributes mentioned in the ParentChileMixin & the CreateUpdateMixIn. Read about these mixin here.

Below are the fields specific to the accounts model.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

code

Each account will have its own alphanumeric code. For example: * Cash Account -> Code 1010. * Inventory -> 1200. * Maximum Length allowed is 10.

Type
str

name

This is the user defined name of the Account. the maximum length for Name of the ledger allowed is 100

Type
str

role

Each Account needs to be assigned a certain Role. The exhaustive list of ROLES is defined in io.roles.

Type
str

balance_type

Each account will have a default Account type i.e. Either Debit or Credit. For example: * Assets like Cash, Inventory, Accounts Receivable or Expenses like Rent, Salary will have balance_type=DEBIT. * Liabilities, Equities and Income like Payables, Loans, Income, Sales, Reserves will have balance_type=CREDIT.

Type
str

locked

This determines whether any transactions can be added in the account. Before making any update to the account, the account needs to be unlocked. Default value is set to False i.e. Unlocked.

Type
bool

active

Determines whether the concerned account is active. Any Account can be used only when it is unlocked and Active. Default value is set to True.

Type
bool

coa_model

Each Accounts must be assigned a ChartOfAccountsModel. By default, one CoA will be created for each entity. However, the creating of a new AccountModel must have an explicit assignment of a ChartOfAccountModel.

Type
ChartOfAccountsModel

class Meta

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod create_account(*name: str*, *role: bool*, *balance_type: str*, *is_role_default: bool = False*, *locked: bool = False*, *active: bool = False*, ***kwargs*)

Convenience Method to Create a new Account Model. This is the preferred method to create new Accounts in order to properly handle parent/child relationships between models.

Parameters

- **name** (*str*) – The name of the new Entity.
- **role** (*str*) – Account role.
- **balance_type** (*str*) – Account Balance Type. Must be 'debit' or 'credit'.
- **is_role_default** (*bool*) – If True, assigns account as default for role. Only once default account per role is permitted.
- **locked** (*bool*) – Marks account as Locked. Defaults to False.
- **active** (*bool*) – Marks account as Active. Defaults to True.

Returns
The newly created AccountModel instance.

Return type
AccountModel

is_credit()

Checks if the account has a CREDIT balance. :returns: True if account has a CREDIT balance, else False. :rtype: bool

is_debit() → bool

Checks if the account has a DEBIT balance. :returns: True if account has a DEBIT balance, else False. :rtype: bool

property role_bs: str

The principal role of the account on the balance sheet. Options are: * asset * liability * equity

Returns
A String representing the principal role of the account on the balance sheet.

Return type
str

class django_ledger.models.accounts.**AccountModelManager**(**args*, ***kwargs*)

This Model Manager will be used as interface through which the database query operations can be provided to the Account Model. It uses the custom defined AccountModelQuerySet and hence overrides the normal get_queryset function which return all rows of a model.

coa_roots(*user_model*, *entity_slug*, *coa_slug*) → *AccountModelQuerySet*

Fetches the Code of Account Root Accounts.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **coa_slug** (*str*) – Explicitly specify which chart of accounts to use. If None, will pull default Chart of Accounts. Discussed in detail in the CoA Model CoA slug, basically helps in identifying the complete Chart of Accounts for a particular EntityModel.
- **user_model** – The Django User Model making the request to check for permissions.

for_bill(*user_model*, *entity_slug*, *coa_slug: Optional[str] = None*) → *AccountModelQuerySet*

Convenience method to pull only available and unlocked AccountModels for a specific EntityModel relevant only for creating and management of Bills. See GROUP_BILL.

Roles in GROUP_BILL: ASSET_CA_CASH, ASSET_CA_PREPAID, LIABILITY_CL_ACC_PAYABLE.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **coa_slug** (*str*) – Explicitly specify which chart of accounts to use. If None, will pull default Chart of Accounts. Discussed in detail in the CoA Model CoA slug, basically helps in identifying the complete Chart of Accounts for a particular EntityModel.
- **user_model** – The Django User Model making the request to check for permissions.

Returns
A QuerySet of all requested EntityModel Chart of Accounts.

Return type
AccountModelQuerySet

for_entity(*user_model*, *entity_slug*, *coa_slug: Optional[str] = None*, *select_coa_model: bool = True*) → *AccountModelQuerySet*

Ensures that only accounts associated with the given EntityModel are returned.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **coa_slug** (*str*) – Explicitly specify which chart of accounts to use. If None, will pull default Chart of Accounts. Discussed in detail in the CoA Model CoA slug, basically helps in identifying the complete Chart of Accounts for a particular EntityModel.

- **user_model** – The Django User Model making the request to check for permissions.
- **select_coa_model** (*bool*) – Pre fetches the CoA Model information in the QuerySet. Defaults to True.

Returns
A QuerySet of all requested EntityModel Chart of Accounts.

Return type
AccountModelQuerySet

for_entity_available(*user_model*, *entity_slug*, *coa_slug: Optional[str] = None*) → *AccountModelQuerySet*

Convenience method to pull only available and unlocked AccountModels for a specific EntityModel.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **coa_slug** (*str*) – Explicitly specify which chart of accounts to use. If None, will pull default Chart of Accounts. Discussed in detail in the CoA Model CoA slug, basically helps in identifying the complete Chart of Accounts for a particular EntityModel.
- **user_model** – The Django User Model making the request to check for permissions.

Returns
A QuerySet of all requested EntityModel Chart of Accounts.

Return type
AccountModelQuerySet

for_invoice(*user_model*, *entity_slug: str*, *coa_slug: Optional[str] = None*) → *AccountModelQuerySet*

Convenience method to pull only available and unlocked AccountModels for a specific EntityModel relevant only for creating and management of Invoices. See GROUP_INVOICE.

Roles in GROUP_INVOICE: ASSET_CA_CASH, ASSET_CA_RECEIVABLES, LIABILITY_CL_DEFERRED_REVENUE.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **coa_slug** (*str*) – Explicitly specify which chart of accounts to use. If None, will pull default Chart of Accounts. Discussed in detail in the CoA Model CoA slug, basically helps in identifying the complete Chart of Accounts for a particular EntityModel.
- **user_model** – The Django User Model making the request to check for permissions.

Returns
A QuerySet of all requested EntityModel Chart of Accounts.

Return type
AccountModelQuerySet

get_queryset() → *AccountModelQuerySet*

Sets the custom queryset as the default.

with_roles(*roles: Union[list, str]*, *entity_slug*, *user_model*) → *AccountModelQuerySet*

This method is used to make query of accounts with a certain role. For instance, the fixed assets like Buildings have all been assigned the role of "asset_ppe_build" role is basically an aggregation of the accounts under a similar category. So, to query the list of all accounts under the role "asset_ppe_build", we can use this function.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **user_model** – The Django User Model making the request to check for permissions.
- **roles** (*list or str*) – Function accepts a single str instance of a role or a list of roles. For a list of roles , refer io.roles.py

Returns
Returns a QuerySet filtered by user-provided list of Roles.

Return type
AccountModelQuerySet

with_roles_available(*roles: Union[list, str]*, *entity_slug*, *user_model*, *coa_slug: Optional[str]*) → *AccountModelQuerySet*

Convenience method to pull only available and unlocked AccountModels for a specific EntityModel and for a specific list of roles.

Parameters

- **entity_slug** (*EntityModel or str*) – The EntityModel or EntityModel slug to pull accounts from. If slug is passed and coa_slug is None will result in an additional Database query to determine the default code of accounts.
- **coa_slug** (*str*) – Explicitly specify which chart of accounts to use. If None, will pull default Chart of Accounts. Discussed in detail in the CoA Model CoA slug, basically helps in identifying the complete Chart of Accounts for a particular EntityModel.
- **user_model** – The Django User Model making the request to check for permissions.
- **roles** (*list or str*) – Function accepts a single str instance of a role or a list of roles. For a list of roles , refer io.roles.py

Returns
A QuerySet of all requested EntityModel Chart of Accounts.

Return type
AccountModelQuerySet

class django_ledger.models.accounts.**AccountModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined QuerySet, which inherits from the Materialized Path Tree implementation of Django Treebeard for tree-like model implementation.

active()

Active accounts which can be used to create new transactions that show on drop-down menus and forms.

Returns
A filtered AccountModelQuerySet of active accounts.

Return type
AccountModelQuerySet

inactive()

Inactive accounts cannot be used to create new transactions and don't show on drop-down menus and forms.

Returns
A filtered AccountModelQuerySet of inactive accounts.

Return type
AccountModelQuerySet

with_roles(*roles: Union[List, str]*)

This method is used to make query of accounts with a certain role. For instance, the fixed assets like Buildings have all been assigned the role of "asset_ppe_build" role is basically an aggregation of the accounts under a similar category. So, to query the list of all accounts under the role "asset_ppe_build", we can use this function.

Parameters
roles (*list or str*) – Function accepts a single str instance of a role or a list of roles. For a list of roles , refer io.roles.py

Returns
Returns a QuerySet filtered by user-provided list of Roles.

Return type
AccountModelQuerySet

exception django_ledger.models.accounts.**AccountModelValidationError**(*message*, *code=None*, *params=None*)

django_ledger.models.accounts.**CREDIT = 'credit'**

A constant, identifying a CREDIT Account or CREDIT transaction in the respective database fields

django_ledger.models.accounts.**DEBIT = 'debit'**

A constant, identifying a DEBIT Account or DEBIT transaction in the respective database fields

14.6 Ledger Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

The LedgerModel is the heart of Django Ledger. It is a self-contained unit of accounting that implements a double-entry accounting system capable of creating and managing transactions into the ledger and producing any financial statements. In essence, an EntityModel is made of a collection of LedgerModels that drive the whole bookkeeping process. Each LedgerModel is independent and they can operate as an individual or as a group.

Each LedgerModel encapsulates a collection of JournalEntryModels, which in turn hold a collection of TransactionModels. LedgerModels can be used to represent any part of the EntityModel and can be extended to add additional functionality and custom logic that drives how transactions are recorded into the books. One example of this is the LedgerWrapperMixIn (see django_ledger.models.mixins.LedgerWrapperMixIn), which is the foundation of LedgerModel abstractions such as the BillModel, InvoiceModel, PurchaseOrderModel and EstimateModel. Extending the LedgerModel can add additional functionality necessary to implement industry-specific functionality to almost anything you can think of. Examples: Farming Equipment, Real Estate, Investment Portfolio, etc.

Also, the LedgerModel inherits functionality from the all mighty IOMixIn (see django_ledger.io.io_mixin.IOMixIn), which is the class responsible for making accounting queries to the Database in an efficient and performing way. The

digest() method executes all necessary aggregations and optimizations in order to push as much work to the Database layer as possible in order to minimize the amount of data being pulled for analysis into the Python memory.

The Django Ledger core model follows the following structure:

EntityModel -< LedgerModel -< JournalEntryModel -< TransactionModel

class django_ledger.models.ledger.**LedgerModel**(**args*, ***kwargs*)

Base LedgerModel from Abstract.

exception **DoesNotExist**

exception **MultipleObjectsReturned**

class django_ledger.models.ledger.**LedgerModelAbstract**(**args*, ***kwargs*)

Base implmentation of the LedgerModel.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

name

Human-readable name of the LedgerModel. Maximum 150 characters.

Type
str

ledger_xid

A unique user-defined identifier for the LedgerModel. Unique for the Entity Model.

Type
str

entity

The EntityModel associated with the LedgerModel instance.

Type
EntityModel

posted

Determines if the LedgerModel is posted. Defaults to False. Mandatory.

Type
bool

locked

Determines if the LedgerModel is locked. Defaults to False. Mandatory.

Type
bool

hidden

Determines if the LedgerModel is hidden. Defaults to False. Mandatory.

Type
bool

can_hide() → bool

Determines if the LedgerModel can be hidden.

Returns
True if can be hidden, else False.

Return type
bool

can_lock() → bool

Determines if the LedgerModel can be locked.

Returns
True if can be locked, else False.

Return type
bool

can_post() → bool

Determines if the LedgerModel can be marked as posted.

Returns
True if can be posted, else False.

Return type
bool

can_unhide() → bool

Determines if the LedgerModel can be un-hidden.

Returns
True if can be un-hidden, else False.

Return type
bool

can_unlock() → bool

Determines if the LedgerModel can be un-locked.

Returns
True if can be un-locked, else False.

Return type
bool

can_unpost() → bool

Determines if the LedgerModel can be un-posted.

Returns
True if can be un-posted, else False.

Return type
bool

get_absolute_url() → str

Determines the absolute URL of the LedgerModel instance. Results in additional Database query if entity field is not selected in QuerySet.

Returns
URL as a string.

Return type
str

get_create_url() → str

Determines the update URL of the LedgerModel instance. Results in additional Database query if entity field is not selected in QuerySet.

Returns
URL as a string.

Return type
str

get_list_url() → str

Determines the list URL of the LedgerModel instances. Results in additional Database query if entity field is not selected in QuerySet.

Returns
URL as a string.

Return type
str

get_update_url() → str

Determines the update URL of the LedgerModel instance. Results in additional Database query if entity field is not selected in QuerySet.

Returns
URL as a string.

Return type
str

is_hidden() → bool

Determines if the LedgerModel instance is hidden.

Returns
True if hidden, else False.

Return type
bool

is_locked() → bool

Determines if the LedgerModel instance is locked.

Returns
True if locked, else False.

Return type
bool

is_posted() → bool

Determines if the LedgerModel instance is posted.

Returns
True if posted, else False.

Return type
bool

lock(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Locks the LedgerModel.

Parameters

- **commit** (*bool*) – If True, saves the LedgerModel instance instantly. Defaults to False.
- **raise_exception** (*bool*) – Raises LedgerModelValidationError if locking not allowed.

post(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Posts the LedgerModel.

Parameters

- **commit** (*bool*) – If True, saves the LedgerModel instance instantly. Defaults to False.
- **raise_exception** (*bool*) – Raises LedgerModelValidationError if posting not allowed.

unlock(*commit: bool = False*, ***kwargs*)

Un-locks the LedgerModel.

Parameters
commit (*bool*) – If True, saves the LedgerModel instance instantly. Defaults to False.

unpost(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Un-posts the LedgerModel.

Parameters

- **commit** (*bool*) – If True, saves the LedgerModel instance instantly. Defaults to False.
- **raise_exception** (*bool*) – Raises LedgerModelValidationError if un-posting not allowed.

class django_ledger.models.ledger.**LedgerModelManager**(**args*, ***kwargs*)

A custom-defined LedgerModelManager that implements custom QuerySet methods related to the LedgerModel.

for_entity(*entity_slug*, *user_model*)

Returns a QuerySet of LedgerModels associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered LedgerModelQuerySet.

Return type
LedgerModelQuerySet

get_queryset()

Return a new QuerySet object. Subclasses can override this method to customize the behavior of the Manager.

class django_ledger.models.ledger.**LedgerModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Custom defined LedgerModel QuerySet.

`locked()`

Filters the QuerySet to only locked LedgerModel.

Returns
A QuerySet with applied filters.

Return type
LedgerModelQuerySet

`posted()`

Filters the QuerySet to only posted LedgerModel.

Returns
A QuerySet with applied filters.

Return type
LedgerModelQuerySet

`unlocked()`

Filters the QuerySet to only un-locked LedgerModel.

Returns
A QuerySet with applied filters.

Return type
LedgerModelQuerySet

`unposted()`

Filters the QuerySet to only un-posted LedgerModel.

Returns
A QuerySet with applied filters.

Return type
LedgerModelQuerySet

exception django_ledger.models.ledger.**LedgerModelValidationError**(*message*, *code=None*, *params=None*)

14.7 Transaction Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

The TransactionModel is the lowest accounting level where financial information is recorded. Every transaction with a financial implication must be part of a JournalEntryModel, which encapsulates a collection of TransactionModels. Transaction models cannot exist without being part of a validated JournalEntryModel. Orphan TransactionModels are not allowed, and this is enforced by the database.

A transaction must perform a CREDIT or a DEBIT to the underlying AccountModel. The IOMixIn is crucial for the production of financial statements and sets its foundation in the TransactionModel API. It allows for effective querying and aggregating transactions at the Database layer without pulling all TransactionModels into memory. This approach streamlines the production of financial statements. The IOMixIn in the TransactionModel API is essential for efficient and effective financial statement generation.

class django_ledger.models.transactions.**TransactionModel**(**args*, ***kwargs*)

Base Transaction Model From Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.transactions.**TransactionModelAbstract**(**args*, ***kwargs*)

An abstract class that represents a transaction in the ledger system.

- **CREDIT**

  Type
  A constant representing a credit transaction.

- **DEBIT**

  Type
  A constant representing a debit transaction.

- **TX_TYPE**

  Type
  A list of tuples representing the transaction type choices.

- **uuid**

  This field is automatically generated and is not editable.

  Type
  A UUIDField representing the unique identifier of the transaction.

- **tx_type**

  It has a maximum length of 10 characters and accepts choices from the TX_TYPE list.

  Type
  A CharField representing the type of the transaction.

- **journal_entry**

  It references the 'django_ledger.JournalEntryModel' model.

  Type
  A ForeignKey representing the journal entry associated with the transaction.

- **account**

  It references the 'django_ledger.AccountModel' model.

  Type
  A ForeignKey representing the account associated with the transaction.

- **amount**

  It has a maximum of 2 decimal places and a maximum of 20 digits. It defaults to 0.00 and accepts a minimum value of 0.

  Type
  A DecimalField representing the amount of the transaction.

- **description**

  It has a maximum length of 100 characters and is optional.

  Type
  A CharField representing the description of the transaction.

- objects

Type
An instance of the TransactionModelAdmin class.

- clean(): Performs validation on the transaction instance.

Raises a TransactionModelValidationError if the account is a root account.

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

class django_ledger.models.transactions.**TransactionModelAdmin**(**args*, ***kwargs*)

A manager class for the TransactionModel.

for_bill(*user_model*, *entity_slug: str*, *bill_model: Union[BillModel, str, UUID]*)

Parameters

- **user_model** (*Type*) – An instance of user model.
- **entity_slug** (*str*) – The slug of the entity.
- **bill_model** (*Union[BillModel, str, UUID]*) – An instance of bill model or a string/UUID representing the UUID of the bill model.

Returns
A filtered queryset based on the user model, entity slug, and bill model.

Return type
FilterQuerySet

for_entity(*entity_slug: Union[EntityModel, str, UUID]*, *user_model: Optional[User] = None*) → *TransactionModelQuerySet*

Parameters

- **entity_slug** (*Union[EntityModel, str, UUID]*) – The entity slug or ID for which to retrieve transactions. Can be an instance of EntityModel, a string representing the slug, or a UUID.
- **user_model** (*Optional[UserModel], optional*) – The user model for which to filter transactions. If provided, only transactions associated with the specified user will be returned. Defaults to None.

Returns
A QuerySet of TransactionModel instances filtered by the provided parameters.

Return type
TransactionModelQuerySet

for_invoice(*user_model*, *entity_slug: str*, *invoice_model: Union[InvoiceModel, str, UUID]*)

Parameters

- **user_model** (*[type]*) – The user model used for filtering entities.
- **entity_slug** (*str*) – The slug of the entity used for filtering.
- **invoice_model** (*Union[InvoiceModel, str, UUID]*) – The invoice model or its identifier used for filtering.

Returns
The filtered queryset based on the specified parameters.

Return type
QuerySet

for_journal_entry(*entity_slug: Union[EntityModel, str]*, *ledger_model: Union[LedgerModel, str, UUID]*, *je_model*, *user_model: Optional[User] = None*)

Parameters

- **entity_slug** (*Union[EntityModel, str]*) – The entity slug or instance of EntityModel representing the entity for which the journal entry is requested.
- **ledger_model** (*Union[LedgerModel, str, UUID]*) – The ledger model or its identifier (str or UUID) representing the ledger for which the journal entry is requested.
- **je_model** (*Type[JournalEntryModel]*) – The journal entry model or its identifier (str or UUID) representing the journal entry to filter by.
- **user_model** (*Optional[UserModel], default=None*) – An optional user model instance representing the user for whom the journal entry is requested.

Returns
The filtered queryset of journal entries.

Return type
QuerySet

for_ledger(*entity_slug: Union[EntityModel, str]*, *ledger_model: Union[LedgerModel, UUID]*, *user_model: Optional[User] = None*)

Parameters

- **entity_slug** (*Union[EntityModel, str]*) – The slug or instance of the entity for which to filter the ledger.
- **ledger_model** (*Union[LedgerModel, UUID]*) – The ledger model or UUID of the ledger for which to filter the journal entries.
- **user_model** (*Optional[UserModel], optional*) – The user model associated with the entity. Default is None.

Returns
The filtered QuerySet containing the journal entries for the specified entity and ledger.

Return type
QuerySet

for_unit(*entity_slug: Union[EntityModel, str]*, *unit_slug: str = typing.Union[django_ledger.models.unit.EntityUnitModel, str]*, *user_model: Optional[User] = None*)

Returns the queryset filtered for the specified entity unit.

Parameters

- **entity_slug** (*Union[EntityModel, str]*) – The entity model or slug used to filter the queryset.
- **unit_slug** (*Union[EntityUnitModel, str]*) – The entity unit model or slug used to filter the queryset.
- **user_model** (*Optional[UserModel], optional*) – The user model to consider for filtering the queryset, by default None.

Returns
The filtered queryset based on the specified entity unit.

Return type
QuerySet

Notes

- If *unit_slug* is an instance of *EntityUnitModel*, the queryset is filtered using *journal_entry__entity_unit=unit_slug*.
- If *unit_slug* is a string, the queryset is filtered using *journal_entry__entity_unit__slug__exact=unit_slug*.

for_user(*user_model*) → *TransactionModelQuerySet*

Parameters
user_model (*User model object*) – The user model object representing the user for whom to filter the transactions.

Returns
A queryset of transaction models filtered based on the user's permissions.

Return type
TransactionModelQuerySet

Raises

- **None** –
- **Description** –
- **----------** –
- **This method filters the transactions based on the user's permissions.** –
- **If the user is a superuser, all transactions are returned. Otherwise, the transactions are filtered based on** –
- **the user's relationship to the entities associated with the transactions. Specifically, the transactions are** –
- **filtered to include only those where either the user is an admin of the entity associated with the transaction's** –
- **ledger or the user is one of the managers of the entity associated with the transaction's ledger.** –

get_queryset() → *TransactionModelQuerySet*

Return a new QuerySet object. Subclasses can override this method to customize the behavior of the Manager.

class django_ledger.models.transactions.**TransactionModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom QuerySet class for TransactionModels implementing methods to effectively and safely read TransactionModels from the database.

posted() → TransactionModelQuerySet:

Fetches a QuerySet of posted transactions only.

for_accounts(*account_list: List[str or AccountModel]*) → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels which AccountModel has a specific role.

for_roles(*role_list: Union[str, List[str]]*) → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels which AccountModel has a specific role.

for_unit(*unit_slug: Union[str, EntityUnitModel]*) → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels associated with a specific EntityUnitModel.

for_activity(*activity_list: Union[str, List[str]]*) → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels associated with a specific activity or list of activities.

to_date(*to_date: Union[str, date, datetime]*) → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels associated with a maximum date or timestamp filter.

from_date(*from_date: Union[str, date, datetime]*) → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels associated with a minimum date or timestamp filter.

not_closing_entry() → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels that are not part of a closing entry.

is_closing_entry() → TransactionModelQuerySet:
Fetches a QuerySet of TransactionModels that are part of a closing entry.

for_accounts(*account_list: List[str]*)
Fetches a QuerySet of TransactionModels which AccountModel has a specific role.

Parameters
account_list (*list*) – A string or list of strings representing the roles to be used as filter.

Returns
Returns a TransactionModelQuerySet with applied filters.

Return type
TransactionModelQuerySet

for_activity(*activity_list: Union[str, List[str]]*)
Fetches a QuerySet of TransactionModels associated with a specific activity or list of activities.

Parameters
activity_list (*str or list*) – A string or list of strings representing the activity or activities used to filter the QuerySet.

Returns
Returns a TransactionModelQuerySet with applied filters.

Return type
TransactionModelQuerySet

for_roles(*role_list: Union[str, List[str]]*)
Fetches a QuerySet of TransactionModels which AccountModel has a specific role.

Parameters
role_list (*str or list*) – A string or list of strings representing the roles to be used as filter.

Returns
Returns a TransactionModelQuerySet with applied filters.

Return type
TransactionModelQuerySet

for_unit(*unit_slug: Union[str, EntityUnitModel]*)

Fetches a QuerySet of TransactionModels associated with a specific EntityUnitModel.

Parameters
unit_slug (*str or EntityUnitModel*) – A string representing the unit slug used to filter the QuerySet.

Returns
Returns a TransactionModelQuerySet with applied filters.

Return type
TransactionModelQuerySet

from_date(*from_date: Union[str, date, datetime]*)

Fetches a QuerySet of TransactionModels associated with a minimum date or timestamp filter. May pass aware or naive date or timestamps. If naive is passed, it is assumed to be in localtime based on Django Settings.

Parameters
from_date (*str or date or datetime*) – A string, date or datetime representing the minimum point in time used to filter the QuerySet. If date is used, dates are inclusive. (i.e 12/20/2022 will also include the 20th day).

Returns
Returns a TransactionModelQuerySet with applied filters.

Return type
TransactionModelQuerySet

is_closing_entry()

Filter the Transactions based on whether they are closing entries or not.

Returns
A filtered QuerySet of entries where the journal_entry__is_closing_entry field is True.

Return type
QuerySet

not_closing_entry()

Filter the Transactions based on whether they are closing entries or not.

Returns
A filtered QuerySet of entries where the journal_entry__is_closing_entry field is False.

Return type
QuerySet

posted() → QuerySet

Fetches a QuerySet of posted transactions only. Posted transactions are must meet the following criteria:

- Be bart of a *posted* JournalEntryModel.
- The associated JournalEntryModel must be part of a *posted* LedgerModel.

Returns
A QuerySet with applied filters.

Return type
TransactionModelQuerySet

to_date(*to_date: Union[str, date, datetime]*)

Fetches a QuerySet of TransactionModels associated with a maximum date or timestamp filter. May pass aware or naive date or timestamps. If naive is passed, it is assumed to be in localtime based on Django Settings.

Parameters

to_date (*str or date or datetime*) – A string, date or datetime representing the maximum point in time used to filter the QuerySet. If date is used, dates are inclusive. (i.e 12/20/2022 will also include the 20th day).

Returns

Returns a TransactionModelQuerySet with applied filters.

Return type

TransactionModelQuerySet

exception django_ledger.models.transactions.**TransactionModelValidationError**(*message*, *code=None*, *params=None*)

django_ledger.models.transactions.**transactionmodel_presave**(*instance:* TransactionModel, ***kwargs*)

Parameters

- **instance** (TransactionModel) – The transaction model instance that is being saved.
- **kwargs** (*dict*) – Additional keyword arguments.

Notes

This method is called before saving a transaction model instance. It performs some checks before allowing the save operation.

Raises

TransactionModelValidationError – If one of the following conditions is met: - *bypass_account_state* is False and the *can_transact* method of the associated account model returns False. - The journal entry associated with the transaction is locked.

14.8 Journal Entry Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

A Journal Entry (JE) is the foundation of all double entry accounting and financial data of any EntityModel. A JE encapsulates a collection of TransactionModel, which must contain two transactions at a minimum. Each transaction must perform a DEBIT or a CREDIT to an AccountModel. The JE Model performs additional validation to make sure that the sum of all DEBITs and the sum of all CREDITs are equal to keep the books balanced.

A JE by default will be un-posted, which means that simply creating a JE will have no effect on the EntityModel books. This behavior allows for constant refinement and persistence of JEs in the database without any impact on the books. Only Journal Entries contained within a *POSTED* LedgerModel (see LedgerModel for documentation) will have an impact in the EntityModel finances.

The JournalEntryModel also carries an optional EntityUnitModel, which are logical user-defined labels which help segregate the different financial statements into different business operations (see EntityUnitModel for documentation). Examples of EntityModelUnits are offices, departments, divisions, etc. *The user may request financial statements by unit*.

All JEs automatically generate a sequential Journal Entry Number, which takes into consideration the Fiscal Year of the JournalEntryModel instance. This functionality enables a human-readable tracking mechanism which helps with audits. It is also searchable and indexed to support quick searches and queries.

The JournalEntryModel is also responsible for validating the Financial Activity involved in the operations of the business. Whenever an account with ASSET_CA_CASH role is involved in a Journal Entry (see roles for more details), the JE is responsible for programmatically determine the kind of operation for the JE (Operating, Financing, Investing).

class django_ledger.models.journal_entry.**ActivityEnum**(*value*)

The database string representation of each accounting activity prefix in the database.

OPERATING

The database representation prefix of a Journal Entry that is an Operating Activity.

Type
str

INVESTING

The database representation prefix of a Journal Entry that is an Investing Activity.

Type
str

FINANCING

The database representation prefix of a Journal Entry that is an Financing Activity.

Type
str

class django_ledger.models.journal_entry.**JournalEntryModel**(**args*, ***kwargs*)

Journal Entry Model Base Class From Abstract

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.journal_entry.**JournalEntryModelAbstract**(**args*, ***kwargs*)

The base implementation of the JournalEntryModel.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

je_number

A unique, sequential, human-readable alphanumeric Journal Entry Number (a.k.a Voucher or Document Number in other commercial bookkeeping software). Contains the fiscal year under which the JE takes place within the EntityModel as a prefix.

Type
str

timestamp

The date of the JournalEntryModel. This date is applied to all TransactionModels contained within the JE, and drives the financial statements of the EntityModel.

Type
datetime

description

A user defined description for the JournalEntryModel.

Type
str

entity_unit

A logical, self-contained, user defined class or structure defined withing the EntityModel. See EntityUnitModel documentation for more details.

Type
EntityUnitModel

activity

Programmatically determined based on the JE transactions and must be a value from ACTIVITIES. Gives additional insight of the nature of the JournalEntryModel in order to produce the Statement of Cash Flows for the EntityModel.

Type
str

origin

A string giving additional information behind the origin or trigger of the JournalEntryModel. For example: reconciliations, migrations, auto-generated, etc. Any string value is valid. Max 30 characters.

Type
str

posted

Determines if the JournalLedgerModel is posted, which means is affecting the books. Defaults to False.

Type
bool

locked

Determines if the JournalEntryModel is locked, which the creation or updates of new transactions are not allowed.

Type
bool

ledger

The LedgerModel associated with this JournalEntryModel. Cannot be null.

Type
LedgerModel

can_generate_je_number() → bool

Checks if the JournalEntryModel instance can generate its own JE number. Conditions are: * The JournalEntryModel must have a LedgerModel instance assigned. * The JournalEntryModel instance must not have a pre-existing JE number.

Returns
True if JournalEntryModel needs a JE number, otherwise False.

Return type
bool

`can_lock`() → bool

Determines if a JournalEntryModel can be locked. Locked JournalEntryModels cannot be modified.

Returns
True if JournalEntryModel can be locked, otherwise False.

Return type
bool

`can_post`(*ignore_verify: bool = True*) → bool

Determines if a JournalEntryModel can be posted.

Parameters
ignore_verify (*bool*) – Skips JournalEntryModel verification if True. Defaults to False.

Returns
True if JournalEntryModel can be posted, otherwise False.

Return type
bool

`can_unlock`() → bool

Determines if a JournalEntryModel can be un-locked. Locked transactions cannot be modified.

Returns
True if JournalEntryModel can be un-locked, otherwise False.

Return type
bool

`can_unpost`() → bool

Determines if a JournalEntryModel can be un-posted.

Returns
True if JournalEntryModel can be un-posted, otherwise False.

Return type
bool

`clean`(*verify: bool = False*, *raise_exception: bool = True*, *txs_qs: Optional[TransactionModelQuerySet] = None*) → Tuple[*TransactionModelQuerySet*, bool]

Customized JournalEntryModel clean method. Generates a JE number if needed. Optional verification hook on clean.

Parameters

- **raise_exception** (*bool*) – Raises exception if JE could not be verified. Defaults to True.
- **verify** (*bool*) – Attempts to verify the JournalEntryModel during cleaning.
- **txs_qs** (TransactionModelQuerySet) – Prefetched TransactionModelQuerySet. If provided avoids additional DB query. Will be verified against JournalEntryModel instance.

Returns
tuple – The TransactionModelQuerySet of the JournalEntryModel instance, verification result as True/False.

Return type
TransactionModelQuerySet, bool

generate_je_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next Journal Entry document number available. The operation will result in two additional queries if the Journal Entry LedgerModel & EntityUnitModel are not cached in QuerySet via select_related('ledger', 'entity_unit').

Parameters
commit (*bool*) – Commits transaction into JournalEntryModel when function is called.

Returns
A String, representing the new or current JournalEntryModel instance Document Number.

Return type
str

get_activity_name() → Optional[str]

Returns a human-readable, GAAP string representing the JournalEntryModel activity.

Returns
Representing the JournalEntryModel activity in the statement of cash flows.

Return type
str or None

get_detail_txs_url() → str

Determines the update URL of the LedgerModel instance. Results in additional Database query if entity field is not selected in QuerySet.

Returns
URL as a string.

Return type
str

get_detail_url() → str

Determines the update URL of the LedgerModel instance. Results in additional Database query if entity field is not selected in QuerySet.

Returns
URL as a string.

Return type
str

get_transaction_queryset(*select_accounts: bool = True*) → *TransactionModelQuerySet*

Fetches the TransactionModelQuerySet associated with the JournalEntryModel instance.

Parameters
select_accounts (*bool*) – Fetches the associated AccountModel of each transaction. Defaults to True.

Returns
The TransactionModelQuerySet associated with the current JournalEntryModel instance.

Return type
TransactionModelQuerySet

get_txs_balances(*txs_qs: Optional[*TransactionModelQuerySet*] = None*, *as_dict: bool = False*) → Union[*TransactionModelQuerySet*, Dict]

Fetches the sum total of CREDITs and DEBITs associated with the JournalEntryModel instance. This method performs a reduction/aggregation at the database level and fetches exactly two records. Optionally, may pass an existing TransactionModelQuerySet if previously fetched. Additional validation occurs to

ensure that all TransactionModels in QuerySet are of the JE instance. Due to JournalEntryModel pre-save validation and basic rules of accounting, CREDITs and DEBITS will always match.

Parameters

- **txs_qs** (TransactionModelQuerySet) – The JE TransactionModelQuerySet to use if previously fetched. Will be validated to make sure all TransactionModel in QuerySet belong to the JournalEntryModel instance.
- **as_dict** (*bool*) – If True, returns the result as a dictionary, with exactly two keys: 'credit' and 'debit'. The values will be the total CREDIT or DEBIT amount as Decimal.

Examples

```
>>> je_model: JournalEntryModel = je_qs.first() # any existing␣
↪JournalEntryModel QuerySet...
>>> balances = je_model.get_txs_balances()
>>> balances
Returns exactly two records:
<TransactionModelQuerySet [{'tx_type': 'credit', 'amount__sum': Decimal('2301.5
↪')},
{'tx_type': 'debit', 'amount__sum': Decimal('2301.5')}]>
```

Examples

```
>>> balances = je_model.get_txs_balances(as_dict=True)
>>> balances
Returns a dictionary:
{'credit': Decimal('2301.5'), 'debit': Decimal('2301.5')}
```

Raises

JournalEntryValidationError – If JE is not valid or TransactionModelQuerySet provided does not belong to JE instance.

Returns

An aggregated queryset containing exactly two records. The total CREDIT or DEBIT amount as Decimal.

Return type

TransactionModelQuerySet or dict

get_txs_roles(*txs_qs: Optional[*TransactionModelQuerySet*] = None*, *exclude_cash_role: bool = False*) → Set[str]

Determines the list of account roles involved in the JournalEntryModel instance. It reaches into the AccountModel associated with each TransactionModel of the JE to determine a Set of all roles involved in transactions. This method is important in determining the nature of the

Parameters

- **txs_qs** (TransactionModelQuerySet) – Prefetched TransactionModelQuerySet. Will be validated if provided. Avoids additional DB query if provided.
- **exclude_cash_role** (*bool*) – Removes CASH role from the Set if present. Useful in some cases where cash role must be excluded for additional validation.

Returns
The set of account roles as strings associated with the JournalEntryModel instance.

Return type
set

is_balance_valid(*txs_qs: Optional[*TransactionModelQuerySet*] = None*) → bool

Checks if CREDITs and DEBITs are equal.

Parameters
txs_qs (TransactionModelQuerySet) – Optional pre-fetched JE instance TransactionModelQuerySet. Will be validated if provided.

Returns
True if JE balances are valid (i.e. are equal).

Return type
bool

is_txs_qs_valid(*txs_qs:* TransactionModelQuerySet, *raise_exception: bool = True*) → bool

Validates a given TransactionModelQuerySet against the JournalEntryModel instance.

Parameters

- **txs_qs** (TransactionModelQuerySet) – The queryset to validate.
- **raise_exception** (*bool*) – Raises JournalEntryValidationError if TransactionModelQuerySet is not valid.

Raises
JournalEntryValidationError if JE model is invalid and raise_exception is True. –

Returns
True if valid, otherwise False.

Return type
bool

is_verified() → bool

Determines if the JournalEntryModel is verified.

Returns
True if is verified, otherwise False.

Return type
bool

mark_as_locked(*commit: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Locked JournalEntryModels do not allow transactions to be edited.

Parameters

- **commit** (*bool*) – Commits changes into the Database, Defaults to False.
- **raise_exception** (*bool*) – Raises JournalEntryValidationError if cannot lock. Defaults to False.
- **kwargs** (*dict*) – Additional keyword arguments.

mark_as_posted(*commit: bool = False*, *verify: bool = True*, *force_lock: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Posted transactions show on the EntityModel ledger and financial statements.

Parameters

- **commit** (*bool*) – Commits changes into the Database, Defaults to False.
- **verify** (*bool*) – Verifies JournalEntryModel before marking as posted. Defaults to False.
- **force_lock** (*bool*) – Forces to lock the JournalEntry before is posted.
- **raise_exception** (*bool*) – Raises JournalEntryValidationError if cannot post. Defaults to False.
- **kwargs** (*dict*) – Additional keyword arguments.

mark_as_unlocked(*commit: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Unlocked JournalEntryModels allow transactions to be edited.

Parameters

- **commit** (*bool*) – Commits changes into the Database, Defaults to False.
- **raise_exception** (*bool*) – Raises JournalEntryValidationError if cannot lock. Defaults to False.
- **kwargs** (*dict*) – Additional keyword arguments.

mark_as_unposted(*commit: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Un-posted JournalEntryModels do not show on the EntityModel ledger and financial statements.

Parameters

- **commit** (*bool*) – Commits changes into the Database, Defaults to False.
- **raise_exception** (*bool*) – Raises JournalEntryValidationError if cannot post. Defaults to False.
- **kwargs** (*dict*) – Additional keyword arguments.

save(*verify: bool = True*, *post_on_verify: bool = False*, **args*, ***kwargs*)

Custom JournalEntryModel instance save method. Additional options are added to attempt to verify JournalEntryModel before saving into database.

Parameters

- **verify** (*bool*) – If True, verifies JournalEntryModel transactions before saving. Defaults to True.
- **post_on_verify** (*bool*) – Posts JournalEntryModel if verification is successful and can_post() is True.

Returns
The saved instance.

Return type
JournalEntryModel

verify(*txs_qs: Optional[*TransactionModelQuerySet*] = None*, *force_verify: bool = False*, *raise_exception: bool = True*, ***kwargs*) → Tuple[*TransactionModelQuerySet*, bool]

Verifies the JournalEntryModel. The JE Model is verified when:

- All TransactionModels associated with the JE instance are in balance (i.e. the sum of CREDITs and DEBITs are equal).
- If the JournalEntryModel is using cash, a cash flow activity is assigned.

Parameters

- **txs_qs** (TransactionModelQuerySet) – Prefetched TransactionModelQuerySet. If provided avoids additional DB query. Will be verified against JournalEntryModel instance.
- **force_verify** (*bool*) – If True, forces new verification of JournalEntryModel if previously verified. Defaults to False.
- **raise_exception** (*bool*) – If True, will raise JournalEntryValidationError if verification fails.
- **kwargs** (*dict*) – Additional function key-word args.

Raises

JournalEntryValidationError if JE instance could not be verified. –

Returns

tuple – The TransactionModelQuerySet of the JournalEntryModel instance, verification result as True/False.

Return type

TransactionModelQuerySet, bool

class django_ledger.models.journal_entry.**JournalEntryModelManager**(**args*, ***kwargs*)

A custom defined Journal Entry Model Manager that supports additional complex initial Queries based on the EntityModel and authenticated UserModel.

for_entity(*entity_slug*, *user_model*)

Fetches a QuerySet of JournalEntryModels associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> journal_entry_qs = JournalEntryModel.objects.for_entity(user_model=request_
↪user, entity_slug=slug)
```

Returns

Returns a JournalEntryModelQuerySet with applied filters.

Return type

JournalEntryModelQuerySet

for_ledger(*ledger_pk: Union[str, UUID]*, *entity_slug*, *user_model*)

Fetches a QuerySet of JournalEntryModels associated with a specific EntityModel & UserModel & LedgerModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.

- **user_model** – Logged in and authenticated django UserModel instance.
- **ledger_pk** (*str or UUID*) – The LedgerModel uuid as a string or UUID.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> ledger_pk = kwargs['ledger_pk'] # may come from request kwargs
>>> journal_entry_qs = JournalEntryModel.objects.for_ledger(ledger_pk=ledger_pk,
↪ user_model=request_user, entity_slug=slug)
```

Returns
Returns a JournalEntryModelQuerySet with applied filters.

Return type
JournalEntryModelQuerySet

class django_ledger.models.journal_entry.**JournalEntryModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Custom defined JournalEntryQuerySet.

create(*verify_on_save: bool = False*, *force_create: bool = False*, ***kwargs*)

Overrides the standard Django QuerySet create() method to avoid the creation of POSTED Journal Entries without proper business logic validation. New JEs using the create() method don't have any transactions to validate. therefore, it is not necessary to query DB to balance TXS

Parameters

- **verify_on_save** (*bool*) – Executes a Journal Entry verification hook before saving. Avoids additional queries to validate the Journal Entry
- **force_create** (*bool*) – If True, will create return a new JournalEntryModel even if Posted at time of creation. Use only if you know what you are doing.

Returns
The newly created Journal Entry Model.

Return type
JournalEntryModel

locked()

Filters the QuerySet to only locked Journal Entries.

Returns
A QuerySet with applied filters.

Return type
JournalEntryModelQuerySet

posted()

Filters the QuerySet to only posted Journal Entries.

Returns
A QuerySet with applied filters.

Return type
JournalEntryModelQuerySet

exception django_ledger.models.journal_entry.**JournalEntryValidationError**(*message*, *code=None*, *params=None*)

14.9 Bank Account Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

A Bank Account refers to the financial institution which holds financial assets for the EntityModel. A bank account usually holds cash, which is a Current Asset. Transactions may be imported using the open financial format specification OFX into a staging area for final disposition into the EntityModel ledger.

class django_ledger.models.bank_account.**BackAccountModelAbstract**(**args*, ***kwargs*)

This is the main abstract class which the BankAccountModel database will inherit from. The BankAccountModel inherits functionality from the following MixIns:

1. *BankAccountInfoMixIn*
2. *CreateUpdateMixIn*

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

name

A user defined name for the bank account as a String.

Type
str

entity_model

The EntityModel associated with the BankAccountModel instance.

Type
EntityModel

cash_account

The AccountModel associated with the BankAccountModel instance. Must be an account with role ASSET_CA_CASH.

Type
AccountModel

active

Determines whether the BackAccountModel instance bank account is active. Defaults to True.

Type
bool

hidden

Determines whether the BackAccountModel instance bank account is hidden. Defaults to False.

Type
bool

class django_ledger.models.bank_account.**BankAccountModel**(**args*, ***kwargs*)

Base Bank Account Model Implementation

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.bank_account.**BankAccountModelManager**(**args*, ***kwargs*)

Custom defined Model Manager for the BankAccountModel.

for_entity(*entity_slug*, *user_model*) → *BankAccountModelQuerySet*

Allows only the authorized user to query the BankAccountModel for a given EntityModel. This is the recommended initial QuerySet.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

class django_ledger.models.bank_account.**BankAccountModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined QuerySet for the BankAccountModel.

active() → QuerySet

Active bank accounts which can be used to create new transactions.

Returns
A filtered BankAccountModelQuerySet of active accounts.

Return type
BankAccountModelQuerySet

hidden() → QuerySet

Hidden bank accounts which can be used to create new transactions. but will not show in drop down menus in the UI.

Returns
A filtered BankAccountModelQuerySet of active accounts.

Return type
BankAccountModelQuerySet

exception django_ledger.models.bank_account.**BankAccountValidationError**(*message*, *code=None*, *params=None*)

14.10 Chart of Accounts Model

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

14.10.1 Chart Of Accounts

A Chart of Accounts (CoA) is a crucial collection of logically grouped accounts within a ChartOfAccountModel, forming the backbone of financial statements. The CoA includes various account roles such as cash, accounts receivable, expenses, liabilities, and income. For example, the Balance Sheet may have a Fixed Assets heading consisting of Tangible and Intangible Assets with multiple accounts like Building, Plant & Equipments, and Machinery under tangible assets. Aggregation of individual account balances based on the Chart of Accounts and AccountModel roles is essential for preparing Financial Statements.

All EntityModel must have a default CoA to create any type of transaction. When no explicit CoA is specified, the default behavior is to use the EntityModel default CoA. Only ONE Chart of Accounts can be used when creating Journal Entries. No commingling between CoAs is allowed to preserve the integrity of the Journal Entry.

class django_ledger.models.coa.**ChartOfAccountModel**(**args*, ***kwargs*)

Base ChartOfAccounts Model

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.coa.**ChartOfAccountModelAbstract**(**args*, ***kwargs*)

Abstract base class for the Chart of Account model.

uuid

UUID field for the chart of account model (primary key).

Type
UUIDField

entity

ForeignKey to the EntityModel.

Type
ForeignKey

active

BooleanField indicating whether the chart of account is active or not.

Type
BooleanField

description

TextField storing the description of the chart of account.

Type
TextField

objects

Manager for the ChartOfAccountModel.

Type
ChartOfAccountModelManager

can_activate() → bool

Check if the ChartOffAccountModel instance can be activated.

Return type
True if the object can be activated, False otherwise.

can_deactivate() → bool

Check if the ChartOffAccountModel instance can be deactivated.

Return type
True if the object can be deactivated, False otherwise.

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

configure(*raise_exception: bool = True*)

A method that properly configures the ChartOfAccounts model and creates the appropriate hierarchy boilerplate to support the insertion of new accounts into the chart of account model tree. This method must be called every time the ChartOfAccounts model is created.

Parameters
raise_exception (*bool, optional*) – Whether to raise an exception if root nodes already exist in the Chart of Accounts (default is True). This indicates that the ChartOfAccountModel instance is already configured.

create_account(*code: str*, *role: str*, *name: str*, *balance_type: str*, *active: bool*, *root_account_qs: Optional[AccountModelQuerySet] = None*)

Proper method for inserting a new Account Model into a CoA. Use this in liu of the direct instantiation of the AccountModel of using the django related manager.

Parameters

- **code** (*str*) – The code of the account to be created.
- **role** (*str*) – The role of the account. This can be a user-defined value.
- **name** (*str*) – The name of the account.
- **balance_type** (*str*) – The balance type of the account. This can be a user-defined value.
- **active** (*bool*) – Specifies whether the account is active or not.
- **root_account_qs** (*Optional[AccountModelQuerySet], optional*) – The query set of root accounts to which the created account should be linked. Defaults to None.

Returns
The created account model instance.

Return type
AccountModel

generate_slug(*raise_exception: bool = False*) → str

Generates and assigns a slug based on the ChartOfAccounts model instance EntityModel information.

Parameters
raise_exception (*bool, optional*) – If set to True, it will raise a ChartOfAccountsModelValidationError if the *self.slug* is already set.

Returns
The generated slug for the Chart of Accounts.

Return type
str

Raises
ChartOfAccountsModelValidationError – If *raise_exception* is set to True and *self.slug* is already set.

get_account_root_node(*account_model:* AccountModel, *root_account_qs:* *Optional[*AccountModelQuerySet*]* = *None*, *as_queryset: bool* = *False*) → *AccountModel*

Fetches the root node of the ChartOfAccountModel instance. The root node is the highest level of the CoA hierarchy. It can be used to traverse the hierarchy of the CoA structure downstream.

Parameters

- **account_model** (AccountModel) – The account model for which to find the root node.
- **root_account_qs** (*Optional[*AccountModelQuerySet*], optional*) – The queryset of root accounts. If not provided, it will be retrieved using *get_coa_root_accounts_qs* method.
- **as_queryset** (*bool, optional*) – If True, return the root account queryset instead of a single root account. Default is False.

Returns
If *as_queryset* is True, returns the root account queryset. Otherwise, returns a single root account.

Return type
Union[*AccountModelQuerySet*, *AccountModel*]

Raises
ChartOfAccountsModelValidationError – If the account model is not part of the chart of accounts.

get_coa_account_tree() → Dict

Performs a bulk dump of the ChartOfAccounts model instance accounts to a dictionary. The method invokes the`dump_bulk` method on the ChartOfAccount model instance root node. See Django Tree Beard documentation for more information.

Returns
A dictionary containing all accounts from the chart of accounts in a nested structure.

Return type
Dict

get_coa_accounts(*active_only: bool* = *True*) → *AccountModelQuerySet*

Returns the AccountModelQuerySet associated with the ChartOfAccounts model instance.

Parameters
active_only (*bool, optional*) – Flag to indicate whether to retrieve only active accounts or all accounts. Default is True.

Returns
A queryset containing accounts from the chart of accounts.

Return type
AccountModelQuerySet

`get_coa_root_accounts_qs()` → *AccountModelQuerySet*

Retrieves the root accounts in the chart of accounts.

Returns
A queryset containing the root accounts in the chart of accounts.

Return type
AccountModelQuerySet

`get_coa_root_node()` → *AccountModel*

Retrieves the root node of the chart of accounts.

Returns
The root node of the chart of accounts.

Return type
AccountModel

`get_non_root_coa_accounts_qs()` → *AccountModelQuerySet*

Returns a query set of non-root accounts in the chart of accounts.

Returns
A query set of non-root accounts in the chart of accounts.

Return type
AccountModelQuerySet

`insert_account`(*account_model:* AccountModel, *root_account_qs: Optional[*AccountModelQuerySet*] = None*)

This method inserts the given account model into the chart of accounts (COA) instance. It first verifies if the account model's COA model ID matches the COA's UUID. If not, it raises a *ChartOfAccountsModelValidationError*. If the *root_account_qs* is not provided, it retrieves the root account query set using the *get_coa_root_accounts_qs* method. Providing a pre-fetched *root_account_qs* avoids unnecessary retrieval of the root account query set every an account model is inserted into the CoA.

Next, it validates the provided *root_account_qs* if it is not None. Then, it obtains the root node for the account model using the *get_account_root_node* method and assigns it to *account_root_node*.

Finally, it adds the account model as a child to the *account_root_node* and retrieves the updated COA accounts query set using the *get_non_root_coa_accounts_qs* method. It returns the inserted account model found in the COA accounts query set.

Parameters

- **account_model** (AccountModel) – The account model to be inserted into the chart of accounts.
- **root_account_qs** (*Optional[AccountModelQuerySet], default=None*) – The root account query set. If not provided, it will be obtained using the *get_coa_root_accounts_qs* method.

Returns
The inserted account model.

Return type
AccountModel

Raises
ChartOfAccountsModelValidationError – If the provided account model has an invalid COA model ID for the current COA.

is_active() → bool

Check if the ChartOfAccountModel instance is active.

Returns
True if the ChartOfAccountModel instance is active, False otherwise.

Return type
bool

is_default() → bool

Check if the ChartOfAccountModel instance is set as the default for the EntityModel.

Returns
True if the ChartOfAccountModel instance is set as the default for the EntityModel. Else, False.

Return type
bool

mark_as_active(*commit: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Marks the current Chart of Accounts as Active.

Parameters

- **commit** (*bool*) – Commit the action into the Database. Default is False.
- **raise_exception** (*bool*) – Raises exception if Chart of Account model instance is already active. Default is False.

mark_as_active_url() → str

Returns the URL to mark the current Chart of Accounts instances as active.

Returns
The URL as a String.

Return type
str

mark_as_default(*commit: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Marks the current Chart of Accounts instances as default for the EntityModel.

Parameters

- **commit** (*bool*) – Commit the action into the Database. Default is False.
- **raise_exception** (*bool*) – Raises exception if Chart of Account model instance is already marked as default.

mark_as_default_url() → str

Returns the URL to mark the current Chart of Accounts instances as Default for the EntityModel.

Returns
The URL as a String.

Return type
str

mark_as_inactive(*commit: bool = False*, *raise_exception: bool = False*, ***kwargs*)

Marks the current Chart of Accounts as Active.

Parameters

- **commit** (*bool*) – Commit the action into the Database. Default is False.
- **raise_exception** (*bool*) – Raises exception if Chart of Account model instance is already active. Default is False.

mark_as_inactive_url() → str

Returns the URL to mark the current Chart of Accounts instances as inactive.

Returns
The URL as a String.

Return type
str

validate_account_model_qs(*account_model_qs:* AccountModelQuerySet)

Validates the given AccountModelQuerySet for the ChartOfAccountsModel.

Parameters
account_model_qs (AccountModelQuerySet) – The AccountModelQuerySet to validate.

Raises
ChartOfAccountsModelValidationError – If the account_model_qs is not an instance of AccountModelQuerySet or if it contains an account model with a different coa_model_id than the current CoA model.

class django_ledger.models.coa.**ChartOfAccountModelManager**(**args*, ***kwargs*)

A custom defined ChartOfAccountModelManager that will act as an interface to handling the initial DB queries to the ChartOfAccountModel.

for_entity(*entity_slug*, *user_model*) → *ChartOfAccountModelQuerySet*

Fetches a QuerySet of ChartOfAccountsModel associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Returns
Returns a ChartOfAccountQuerySet with applied filters.

Return type
ChartOfAccountQuerySet

for_user(*user_model*) → *ChartOfAccountModelQuerySet*

Fetches a QuerySet of ChartOfAccountModel that the UserModel as access to. May include ChartOfAccountModel from multiple Entities. The user has access to bills if: 1. Is listed as Manager of Entity. 2. Is the Admin of the Entity.

Parameters
user_model – Logged in and authenticated django UserModel instance.

Returns
Returns a ChartOfAccountQuerySet with applied filters.

Return type
ChartOfAccountQuerySet

class django_ledger.models.coa.**ChartOfAccountModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

active()
QuerySet method to retrieve active items.

exception django_ledger.models.coa.**ChartOfAccountsModelValidationError**(*message*, *code=None*, *params=None*)

14.11 Default Chart of Accounts

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

This is the base Chart of Accounts that has all the possible accounts that are useful for the preparation of the Financial Statements. A user may choose to use the default CoA at the creation of each EntityModel but it is not required. The default CoA is intended to provide a QuickStart solution for most use cases.

The Chart of Accounts is broadly bifurcated into 5 different Sections:

1. Assets:
2. Liabilities
3. Shareholder's Equity
4. Expenses
5. Revenue

The Django Ledger Default Chart of Accounts must include the following fields:

- Code - String
- Role - A choice from any of the possible account roles (see django_ledger.roles module).
- Balance Type - A CREDIT or DEBIT balance account setting.
- Name - A human readable name.
- Parent - The parent account of the AccountModel instance.

If the DEFAULT_CHART_OF_ACCOUNTS setting is present, the default CoA will be replace by such setting.

14.11.1 Default Chart of Accounts Table

code	role	balance_type	name	parent	root_g
1910	asset_adjustment	debit	Securities Unrealized Gains/Losses		root_as
1920	asset_adjustment	debit	PPE Unrealized Gains/Losses		root_as
1010	asset_ca_cash	debit	Cash		root_as
1200	asset_ca_inv	debit	Inventory		root_as
1050	asset_ca_mkt_sec	debit	Short Term Investments		root_as
1300	asset_ca_prepaid	debit	Prepaid Expenses		root_as
1100	asset_ca_recv	debit	Accounts Receivable		root_as
1110	asset_ca_uncoll	credit	Uncollectibles		root_as
1810	asset_ia	debit	Goodwill		root_as
1820	asset_ia	debit	Intellectual Property		root_as
1830	asset_ia_accum_amort	credit	Less: Intangible Assets Accumulated Amortization		root_as
1520	asset_lti_land	debit	Land		root_as
1510	asset_lti_notes	debit	Notes Receivable		root_as
1530	asset_lti_sec	debit	Securities		root_as
1610	asset_ppe_build	debit	Buildings		root_as
1611	asset_ppe_build_accum_depr	credit	Less: Buildings Accumulated Depreciation		root_as
1630	asset_ppe_equip	debit	Equipment		root_as
1631	asset_ppe_equip_accum_depr	credit	Less: Equipment Accumulated Depreciation		root_as
1620	asset_ppe_plant	debit	Plant		root_as
1640	asset_ppe_plant	debit	Vehicles		root_as
1650	asset_ppe_plant	debit	Furniture & Fixtures		root_as
1621	asset_ppe_plant_depr	credit	Less: Plant Accumulated Depreciation		root_as
1641	asset_ppe_plant_depr	credit	Less: Vehicles Accumulated Depreciation		root_as
1651	asset_ppe_plant_depr	credit	Less: Furniture & Fixtures Accumulated Depreciation		root_as
3910	eq_adjustment	credit	Available for Sale		root_ca
3920	eq_adjustment	credit	PPE Unrealized Gains/Losses		root_ca
3010	eq_capital	credit	Capital Account 1		root_ca
3020	eq_capital	credit	Capital Account 2		root_ca
3030	eq_capital	credit	Capital Account 3		root_ca
3930	eq_dividends	debit	Dividends & Distributions		root_ca
3110	eq_stock_common	credit	Common Stock		root_ca
3120	eq_stock_preferred	credit	Preferred Stock		root_ca
5010	cogs_regular	debit	Cost of Goods Sold		root_co
6075	ex_amortization	debit	Amortization Expense		root_ex
6070	ex_depreciation	debit	Depreciation Expense		root_ex
6130	ex_interest	debit	Interest Expense		root_ex
6500	ex_other	debit	Misc. Expense		root_ex
6010	ex_regular	debit	Advertising		root_ex
6020	ex_regular	debit	Amortization		root_ex
6030	ex_regular	debit	Auto Expense		root_ex
6040	ex_regular	debit	Bad Debt		root_ex
6050	ex_regular	debit	Bank Charges		root_ex
6060	ex_regular	debit	Commission Expense		root_ex
6080	ex_regular	debit	Employee Benefits		root_ex
6090	ex_regular	debit	Freight		root_ex
6110	ex_regular	debit	Gifts		root_ex
6120	ex_regular	debit	Insurance		root_ex

continues on nex

Table 1 – continued from previous page

code	role	balance_type	name	parent	root_g
6140	ex_regular	debit	Professional Fees		root_ex
6150	ex_regular	debit	License Expense		root_ex
6170	ex_regular	debit	Maintenance Expense		root_ex
6180	ex_regular	debit	Meals & Entertainment		root_ex
6190	ex_regular	debit	Office Expense		root_ex
6220	ex_regular	debit	Printing		root_ex
6230	ex_regular	debit	Postage		root_ex
6240	ex_regular	debit	Rent		root_ex
6250	ex_regular	debit	Maintenance & Repairs		root_ex
6251	ex_regular	debit	Maintenance		root_ex
6252	ex_regular	debit	Repairs		root_ex
6253	ex_regular	debit	HOA		root_ex
6254	ex_regular	debit	Snow Removal		root_ex
6255	ex_regular	debit	Lawn Care		root_ex
6260	ex_regular	debit	Salaries		root_ex
6270	ex_regular	debit	Supplies		root_ex
6290	ex_regular	debit	Utilities		root_ex
6292	ex_regular	debit	Sewer		root_ex
6293	ex_regular	debit	Gas		root_ex
6294	ex_regular	debit	Garbage		root_ex
6295	ex_regular	debit	Electricity		root_ex
6300	ex_regular	debit	Property Management		root_ex
6400	ex_regular	debit	Vacancy		root_ex
6210	ex_taxes	debit	Payroll Taxes		root_ex
6280	ex_taxes	debit	Taxes		root_ex
4040	in_gain_loss	credit	Capital Gain/Loss Income		root_in
4030	in_interest	credit	Interest Income		root_in
4010	in_operational	credit	Sales Income		root_in
4050	in_other	credit	Other Income		root_in
4020	in_passive	credit	Investing Income		root_in
2010	lia_cl_acc_payable	credit	Accounts Payable		root_li
2060	lia_cl_def_rev	credit	Deferred Revenues		root_li
2030	lia_cl_int_payable	credit	Interest Payable		root_li
2050	lia_cl_ltd_mat	credit	Current Maturities LT Debt		root_li
2070	lia_cl_other	credit	Other Payables		root_li
2040	lia_cl_st_notes_payable	credit	Short-Term Notes Payable		root_li
2020	lia_cl_wages_payable	credit	Wages Payable		root_li
2120	lia_ltl_bonds	credit	Bonds Payable		root_li
2130	lia_ltl_mortgage	credit	Mortgage Payable		root_li
2110	lia_ltl_notes	credit	Long Term Notes Payable		root_li

django_ledger.models.coa_default.**get_default_coa_rst**(*default_coa: Optional[Dict] = None*) → str

Converts the provided Chart of Account into restructuredText format. :param default_coa: A dictionary of chart of accounts. Must follow the same keys as CHART_OF_ACCOUNTS.

Returns
The table in RestructuredText format.

Return type
str

django_ledger.models.coa_default.**verify_unique_code**()

A function that verifies that there are no duplicate code in the Default CoA during the development and launch.

14.12 Item Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

The Items refer to the additional detail provided to Bills, Invoices, Purchase Orders and Estimates for the purposes of documenting a breakdown of materials, labor, equipment, and other resources used for the purposes of the business operations.

The items associated with any of the aforementioned models are responsible for calculating the different amounts that ultimately drive the behavior of Journal Entries onto the company books.

Each item must be assigned a UnitOfMeasureModel which is the way or method used to quantify such resource. Examples are Pounds, Gallons, Man Hours, etc used to measure how resources are quantified when associated with a specific ItemTransactionModel. If many unit of measures are used for the same item, it would constitute a different item hence a new record must be created.

ItemsTransactionModels constitute the way multiple items and used resources are associated with Bills, Invoices, Purchase Orders and Estimates. Each transaction will record the unit of measure and quantity of each resource. Totals will be calculated and associated with the containing model at the time of update.

class django_ledger.models.items.**ItemModel**(**args*, ***kwargs*)

Base ItemModel from Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.items.**ItemModelAbstract**(**args*, ***kwargs*)

Base implementation of the ItemModel.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

name

Human readable name of the ItemModel instance. Maximum of 100 characters.

Type
str

item_role

A choice of ITEM_ROLE_CHOICES that determines whether the ItemModel should be treated as an expense, inventory, service or product.

Type
str

item_type

A choice of ITEM_TYPE_CHOICES that determines whether the ItemModel should be treated as labor, material, equipment, lump sum or other.

Type
str

uom

The assigned UnitOfMeasureModel of the ItemModel instance. Mandatory.

Type
UnitOfMeasureModel

sku

The SKU number associated with the ItemModel instance. Maximum 50 characters.

Type
str

upc

The UPC number associated with the ItemModel instance. Maximum 50 characters.

Type
str

item_id

EntityModel specific id associated with the ItemModel instance. Maximum 50 characters.

Type
str

item_number

Auto generated human-readable item number.

Type
str

is_active

Determines if the ItemModel instance is considered active. Defaults to True.

Type
bool

default_amount

The default, prepopulated monetary amount of the ItemModel instance .

Type
Decimal

for_inventory

Legacy field used to determine if the ItemModel instance is considered an inventory item. Mandatory. Superseded by item_role field. Will be deprecated.

Type
bool

is_product_or_service

Legacy field used to determine if the ItemModel instance is considered a product or service item. Mandatory. Superseded by item_role field. Will be deprecated.

Type
bool

`sold_as_unit`

Determines if only whole numbers can be used when specifying the quantity on ItemTransactionModels.

Type
bool

`inventory_account`

Inventory account associated with the ItemModel instance. Enforced if ItemModel instance is_inventory() is True.

Type
AccountModel

`inventory_received`

Holds the total quantity of the inventory received for the whole EntityModel instance.

Type
Decimal

`inventory_received_value`

Holds the total monetary value of the inventory received for the whole EntityModel instance.

Type
Decimal

`cogs_account`

COGS account associated with the ItemModel instance. Enforced if ItemModel instance is_inventory() is True.

Type
AccountModel

`earnings_account`

Earnings account associated with the ItemModel instance. Enforced if ItemModel instance is_product() or
is_service() is True.

Type
AccountModel

`expense_account`

Expense account associated with the ItemModel instance. Enforced if ItemModel instance is_expense() is True.

Type
AccountModel

`additional_info`

Additional user defined information stored as JSON document in the Database.

Type
dict

`entity`

The EntityModel associated with the ItemModel instance.

Type
EntityModel

`clean()`

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`generate_item_number`(*commit: bool = False*) → str

Atomic Transaction. Generates the next Vendor Number available. @param commit: Commit transaction into VendorModel. @return: A String, representing the current InvoiceModel instance Document Number.

`save`(***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`class` django_ledger.models.items.`ItemModelManager`(**args*, ***kwargs*)

A custom defined ItemModelManager that implement custom QuerySet methods related to the ItemModel

`for_bill`(*entity_slug*, *user_model*)

Returns a QuerySet of ItemModels that can only be used for BillModels for a specific EntityModel & UserModel. These types of items qualify as expenses or inventory purchases. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

`for_entity`(*entity_slug*, *user_model*)

Returns a QuerySet of ItemModel associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

`for_entity_active`(*entity_slug*, *user_model*)

Returns a QuerySet of Active ItemModel associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

for_estimate(*entity_slug: str*, *user_model*)

Returns a QuerySet of ItemModels that can only be used for EstimateModels for a specific EntityModel & UserModel. These types of items qualify as products. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

for_invoice(*entity_slug*, *user_model*)

Returns a QuerySet of ItemModels that can only be used for InvoiceModels for a specific EntityModel & UserModel. These types of items qualify as products or services sold. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

for_po(*entity_slug*, *user_model*)

Returns a QuerySet of ItemModels that can only be used for PurchaseOrders for a specific EntityModel & UserModel. These types of items qualify as inventory purchases. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

class django_ledger.models.items.**ItemModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom-defined ItemModelQuerySet that implements custom QuerySet methods related to the ItemModel.

active()

Filters the QuerySet to only active Item Models.

Returns
A QuerySet with applied filters.

Return type
ItemModelQuerySet

bills()

Filters the QuerySet to ItemModels that are eligible only for bills..

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

expenses()

Filters the QuerySet to ItemModels that only qualify as expenses.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

inventory_all()

Filters the QuerySet to ItemModels that only qualify as inventory. These types of items may be finished or unfinished.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

inventory_wip()

Filters the QuerySet to ItemModels that only qualify as inventory. These types of items cannot be sold as they are not considered a finished product.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

products()

Filters the QuerySet to ItemModels that only qualify as products.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

services()

Filters the QuerySet to ItemModels that only qualify as services.

Returns
A Filtered ItemModelQuerySet.

Return type
ItemModelQuerySet

exception django_ledger.models.items.**ItemModelValidationError**(*message*, *code=None*, *params=None*)

class django_ledger.models.items.**ItemTransactionModel**(**args*, ***kwargs*)

Base ItemTransactionModel from Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.items.**ItemTransactionModelAbstract**(**args*, ***kwargs*)

can_create_bill() → bool

Determines if the ItemModel instance can be associated with a BillModel. :returns: True, if instance can be associated with a BillModel, else False. :rtype: bool

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

get_status_css_class() → str

Determines the CSS Class used to represent the ItemModel instance in the UI based on its status.

Returns
The CSS class as a String.

Return type
str

has_bill()

Determines if the ItemModel instance is associated with a BillModel.

Returns
True if associated with an BillModel, else False.

Return type
bool

has_estimate() → bool

Determines if the ItemModel instance is associated with an EstimateModel.

Returns
True if associated with an EstimateModel, else False.

Return type
bool

has_invoice()

Determines if the ItemModel instance is associated with a InvoiceModel.

Returns
True if associated with an InvoiceModel, else False.

Return type
bool

`has_po()` → bool

Determines if the ItemModel instance is associated with a PurchaseOrderModel.

Returns
True if associated with an PurchaseOrderModel, else False.

Return type
bool

`html_id()` → str

Unique ItemModel instance HTML ID.

Returns
HTML ID as a String.

Return type
str

`html_id_quantity()` → str

Unique ItemModel instance quantity field HTML ID.

Returns
HTML ID as a String.

Return type
str

`html_id_unit_cost()` → str

Unique ItemModel instance unit cost field HTML ID.

Returns
HTML ID as a String.

Return type
str

`is_canceled()`

Determines if the ItemModel instance is canceled. ItemModel status is only relevant for ItemModels associated with PurchaseOrderModels.

Returns
True if canceled, else False.

Return type
bool

`is_ordered()` → bool

Determines if the ItemModel instance is ordered. ItemModel status is only relevant for ItemModels associated with PurchaseOrderModels.

Returns
True if received, else False.

Return type
bool

is_received() → bool

Determines if the ItemModel instance is received. ItemModel status is only relevant for ItemModels associated with PurchaseOrderModels.

Returns
True if received, else False.

Return type
bool

update_cost_estimate()

Hook that updates and checks the ItemModel instance cost estimate fields according to its associations. Calculates and updates ce_cost_estimate accordingly. Called on every clean() call.

update_po_total_amount()

Hook that updates and checks the ItemModel instance purchase order fields according to its associations. Calculates and updates po_total_amount accordingly. Called on every clean() call.

update_revenue_estimate()

Hook that updates and checks the ItemModel instance revenue estimate fields according to its associations. Calculates and updates ce_revenue_estimate accordingly. Called on every clean() call.

update_total_amount()

Hook that updates and checks the ItemModel instance fields according to its associations. Calculates and updates total_amount accordingly. Called on every clean() call.

class django_ledger.models.items.**ItemTransactionModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

class django_ledger.models.items.**UnitOfMeasureModel**(**args*, ***kwargs*)

Base UnitOfMeasureModel from Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.items.**UnitOfMeasureModelAbstract**(**args*, ***kwargs*)

Base implementation of a Unit of Measure assigned to each Item Transaction.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

name

The name of the unit of measure. Maximum of 50 characters.

Type
str

unit_abbr

An abbreviation of the unit of measure used as an identifier or slug for URLs and queries.

Type
str

is_active

A boolean representing of the UnitOfMeasureModel instance is active to be used on new transactions.

Type
bool

entity

The EntityModel associated with the UnitOfMeasureModel instance.

Type
EntityModel

class django_ledger.models.items.**UnitOfMeasureModelManager**(**args*, ***kwargs*)

A custom defined QuerySet Manager for the UnitOfMeasureModel.

for_entity(*entity_slug: str*, *user_model*) → QuerySet

Fetches the UnitOfMeasureModels associated with the provided EntityModel and UserModel.

Parameters

- **entity_slug** (*str or EntityModel*) – The EntityModel slug or EntityModel used to filter the QuerySet.
- **user_model** (*UserModel*) – The Django UserModel to check permissions.

Returns
A QuerySet with applied filters.

Return type
QuerySet

for_entity_active(*entity_slug: str*, *user_model*)

Fetches the Active UnitOfMeasureModels associated with the provided EntityModel and UserModel.

Parameters

- **entity_slug** (*str or EntityModel*) – The EntityModel slug or EntityModel used to filter the QuerySet.
- **user_model** (*UserModel*) – The Django UserModel to check permissions.

Returns
A QuerySet with applied filters.

Return type
QuerySet

class django_ledger.models.items.**UnitOfMeasureModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

14.13 Bill Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

This module implements the BillModel, which represents an Invoice received from a Supplier/Vendor, on which the Vendor states the amount owed by the recipient for the purposes of supplying goods and/or services. In addition to tracking the bill amount, it tracks the paid and due amount.

Examples

```
>>> user_model = request.user  # django UserModel
>>> entity_slug = kwargs['entity_slug'] # may come from view kwargs
>>> bill_model = BillModel()
>>> ledger_model, bill_model = bill_model.configure(entity_slug=entity_slug, user_
↪model=user_model)
>>> bill_model.save()
```

class django_ledger.models.bill.**BillModel**(**args*, ***kwargs*)

Base BillModel from Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.bill.**BillModelAbstract**(**args*, ***kwargs*)

This is the main abstract class which the BillModel database will inherit from. The BillModel inherits functionality from the following MixIns:

1. LedgerWrapperMixIn
2. *PaymentTermsMixIn*
3. *MarkdownNotesMixIn*
4. *CreateUpdateMixIn*

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

bill_number

Auto assigned number at creation by generate_bill_number() function. Prefix be customized with DJANGO_LEDGER_BILL_NUMBER_PREFIX setting. Includes a reference to the Fiscal Year and a sequence number. Max Length is 20.

Type
str

bill_status

Current status of the BillModel. Must be one of the choices as mentioned under "BILL_STATUS". By default , the status will be "Draft". Options are: Draft, In Review, Approved, Paid, Void or Canceled.

Type
str

xref

This is the filed for capturing of any External reference number like the PO number of the buyer. Any other reference number like the Vendor code in buyer books may also be captured.

Type
str

vendor

This is the foreign key reference to the VendorModel from whom the purchase has been made.

Type
`VendorModel`

additional_info

Any additional metadata about the BillModel may be stored here as a dictionary object. The data is serialized and stored as a JSON document in the Database.

Type
dict

bill_items

A foreign key reference to the list of ItemTransactionModel that make the bill amount.

ce_model

A foreign key to the BillModel associated EstimateModel for overall Job/Contract tracking.

Type
EstimateModel

date_draft

The draft date represents the date when the BillModel was first created. Defaults to `localdate`.

Type
date

date_in_review

The in review date represents the date when the BillModel was marked as In Review status. Will be null if BillModel is canceled during draft status. Defaults to `localdate`.

Type
date

date_approved

The approved date represents the date when the BillModel was approved. Will be null if BillModel is canceled. Defaults to `localdate`.

Type
date

date_paid

The paid date represents the date when the BillModel was paid and amount_due equals amount_paid. Will be null if BillModel is canceled. Defaults to `localdate`.

Type
date

date_void

The void date represents the date when the BillModel was void, if applicable. Will be null unless BillModel is void. Defaults to `localdate`.

Type
date

date_canceled

The canceled date represents the date when the BillModel was canceled, if applicable. Will be null unless BillModel is canceled. Defaults to `localdate`.

Type
date

BILL_STATUS = [('draft', 'Draft'), ('in_review', 'In Review'), ('approved', 'Approved'), ('paid', 'Paid'), ('canceled', 'Canceled'), ('void', 'Void')]

The different bill status options and their representation in the Database.

bind_estimate(*estimate_model*, *commit: bool = False*, *raise_exception: bool = True*)

Binds BillModel to a given EstimateModel. Raises ValueError if EstimateModel cannot be bound.

Parameters

- **estimate_model** (EstimateModel) – EstimateModel to bind.
- **raise_exception** (*bool*) – Raises BillModelValidationError if unable to bind EstimateModel.
- **commit** (*bool*) – Commits transaction into current BillModel.

can_approve() → bool

Checks if the BillModel can be marked as Approved.

Returns
True if BillModel can be marked as approved, else False.

Return type
bool

can_bind_estimate(*estimate_model*, *raise_exception: bool = False*) → bool

Checks if the BillModel can be bound to a given EstimateModel.

Parameters

- **estimate_model** (EstimateModel) – EstimateModel to check against.
- **raise_exception** (*bool*) – If True, raises BillModelValidationError if unable to bind. Else, returns False.

Returns
True if can bind provided EstimateModel, else False.

Return type
bool

can_bind_po(*po_model*, *raise_exception: bool = False*) → bool

Checks if the BillModel can be bound to a given PurchaseOrderModel.

Parameters

- **po_model** (PurchaseOrderModel) – The PurchaseOrderModel to check against.
- **raise_exception** (*bool*) – If True, raises BillModelValidationError if unable to bind, else False.

Returns
True if can bind provided PurchaseOderModel, else False.

Return type
bool

can_cancel() → bool

Checks if the BillModel can be marked as Canceled status.

Returns
True if BillModel can be marked as canceled, else False.

Return type
bool

`can_delete()` → bool

Checks if the BillModel can be deleted.

Returns
True if BillModel can be deleted, else False.

Return type
bool

`can_draft()` → bool

Checks if the BillModel can be marked as Draft.

Returns
True if BillModel can be marked as draft, else False.

Return type
bool

`can_edit_items()` → bool

Checks if the BillModel item transactions can be edited.

Returns
True if BillModel items can be edited, else False.

Return type
bool

`can_generate_bill_number()` → bool

Checks if BillModel can generate its Document Number.

Returns
True if BillModel can generate its bill_number, else False.

Return type
bool

`can_make_payment()` → bool

Checks if the BillModel can accept a payment.

Returns
True if can bind provided PurchaseOderModel, else False.

Return type
bool

`can_migrate()` → bool

Checks if the BillModel can be migrated.

Returns
True if BillModel can be migrated, else False.

Return type
bool

can_migrate_itemtxs() → bool

Checks if item transaction list can be migrated.

Return type
bool

can_pay() → bool

Checks if the BillModel can be marked as Paid.

Returns
True if BillModel can be marked as paid, else False.

Return type
bool

can_review() → bool

Checks if the BillModel can be marked as In Review.

Returns
True if BillModel can be marked as in review, else False.

Return type
bool

can_void() → bool

Checks if the BillModel can be marked as Void status.

Returns
True if BillModel can be marked as void, else False.

Return type
bool

clean(*commit: bool = True*)

Clean method for BillModel. Results in a DB query if bill number has not been generated and the BillModel is eligible to generate a bill_number.

Parameters
commit (*bool*) – If True, commits into DB the generated BillModel number if generated.

configure(*entity_slug: Union[str, EntityModel]*, *user_model: Optional[User] = None*, *date_draft: Optional[Union[datetime, date]] = None*, *ledger_posted: bool = False*, *ledger_name: Optional[str] = None*, *commit: bool = False*, *commit_ledger: bool = False*)

A configuration hook which executes all initial BillModel setup on to the LedgerModel and all initial values of the BillModel. Can only call this method once in the lifetime of a BillModel.

Parameters

- **date_draft** (*date*) – Optional date to use as Draft Date. Defaults to localdate() if None.
- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel to associate the Bill with.
- **user_model** (*UserModel*) – The UserModel making the request to check for QuerySet permissions.
- **ledger_posted** (*bool*) – An option to mark the BillModel Ledger as posted at the time of configuration. Defaults to False.
- **ledger_name** (*str*) – Optional additional InvoiceModel ledger name or description.
- **commit** (*bool*) – Saves the current BillModel after being configured.

- **commit_ledger** (*bool*) – Saves the BillModel's LedgerModel while being configured.

Return type
A tuple of *LedgerModel*, *BillModel*

generate_bill_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next BillModel document number available. The operation will result in two additional queries if the BillModel & LedgerModel is not cached in QuerySet via select_related('ledger').

Parameters
commit (*bool*) – Commits transaction into BillModel.

Returns
A String, representing the generated BillModel instance Document Number.

Return type
str

get_document_id() → Optional[str]

Human-readable document number. Defaults to bill_number.

Returns
Document Number as a String.

Return type
str

get_html_amount_due_id() → str

Unique amount due HTML ID.

Returns
HTML ID as a String.

Return type
str

get_html_amount_paid_id() → str

Unique amount paid HTML ID

Returns
HTML ID as a String.

Return type
str

get_html_form_id() → str

Unique BillModel Form HTML ID.

Returns
HTML ID as a String.

Return type
str

get_html_id() → str

Unique BillNumber HTML ID.

Returns
HTML ID as a String.

Return type
str

get_item_model_qs() → *ItemModelQuerySet*

Fetches the ItemModelQuerySet eligible to itemize.

Return type
ItemModelQuerySet

get_itemtxs_data(*queryset: Optional[*ItemTransactionModelQuerySet*] = None*, *aggregate_on_db: bool = False*, *lazy_agg: bool = False*) → Tuple[*ItemTransactionModelQuerySet*, Dict]

Fetches the BillModel Items and aggregates the QuerySet.

Parameters

- **queryset** – Optional pre-fetched ItemModelQueryset to use. Avoids additional DB query if provided.
- **aggregate_on_db** (*bool*) – If True, performs aggregation of ItemsTransactions in the DB resulting in one additional DB query.

Returns
A tuple

Return type
ItemTransactionModelQuerySet, dict

get_mark_as_approved_html_id() → str

BillModel Mark as Approved HTML ID.

Returns
HTML ID as a String.

Return type
str

get_mark_as_approved_message() → str

Internationalized confirmation message with Bill Number.

Returns
Mark-as-Approved BillModel confirmation message as a String.

Return type
str

get_mark_as_approved_url(*entity_slug: Optional[str] = None*) → str

BillModel Mark-as-Approved action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
BillModel mark-as-approved action URL.

Return type
str

get_mark_as_canceled_html_id() → str

BillModel Mark as Canceled HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_canceled_message() → str

Internationalized confirmation message with Bill Number.

Returns
Mark-as-Canceled BillModel confirmation message as a String.

Return type
str

get_mark_as_canceled_url(*entity_slug: Optional[str] = None*) → str

BillModel Mark-as-Canceled action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
BillModel mark-as-canceled action URL.

Return type
str

get_mark_as_draft_html_id() → str

BillModel Mark as Draft HTML ID Tag.

Returns
HTML ID as a String

Return type
str

get_mark_as_draft_message() → str

Internationalized confirmation message with Bill Number.

Returns
Mark-as-Draft BillModel confirmation message as a String.

Return type
str

get_mark_as_draft_url(*entity_slug: Optional[str] = None*) → str

BillModel Mark-as-Draft action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
HTML ID as a String

Return type
str

get_mark_as_paid_html_id() → str

BillModel Mark as Paid HTML ID Tag.

Returns
HTML ID as a String

Return type
str

get_mark_as_paid_message() → str

Internationalized confirmation message with Bill Number.

Returns
Mark-as-Paid BillModel confirmation message as a String.

Return type
str

get_mark_as_paid_url(*entity_slug: Optional[str] = None*) → str

BillModel Mark-as-Paid action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
BillModel mark-as-paid action URL.

Return type
str

get_mark_as_review_html_id() → str

BillModel Mark as In Review HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_review_message() → str

Internationalized confirmation message with Bill Number.

Returns
Mark-as-Review BillModel confirmation message as a String.

Return type
str

get_mark_as_review_url(*entity_slug: Optional[str] = None*) → str

BillModel Mark-as-Review action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
BillModel mark-as-review action URL.

Return type
str

get_mark_as_void_html_id() → str

BillModel Mark as Void HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_void_message() → str

Internationalized confirmation message with Bill Number.

Returns
Mark-as-Void BillModel confirmation message as a String.

Return type
str

get_mark_as_void_url(*entity_slug: Optional[str] = None*) → str

BillModel Mark-as-Void action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Return type
BillModel mark-as-void action URL.

get_migrate_state_desc() → str

Description used when migrating transactions into the LedgerModel.

Returns
Description as a string.

Return type
str

get_migration_data(*queryset: Optional[ItemTransactionModelQuerySet] = None*) → *ItemTransactionModelQuerySet*

Fetches necessary item transaction data to perform a migration into the LedgerModel.

Parameters
queryset (ItemTransactionModelQuerySet) – Optional pre-fetched ItemModelTransactionQueryset to use. Avoids additional DB query if provided.

get_status_action_date() → date

Current status action date.

Returns
A date. i.e. If status is Approved, return date_approved. If Paid, return date_paid.

Return type
date

get_terms_start_date() → Optional[date]

Date where BillModel term start to apply.

Returns
A date which represents the start of BillModel terms.

Return type
date

is_active()

Checks if the BillModel has the potential to impact the books and produce financial statements status.

Returns
True if BillModel is Active, else False.

Return type
bool

is_approved() → bool

Checks if the BillModel is in Approved status.

Returns
True if BillModel is Approved, else False.

Return type
bool

is_canceled() → bool

Checks if the BillModel is in Canceled status.

Returns
True if BillModel is Canceled, else False.

Return type
bool

is_configured() → bool

Determines if the accruable financial instrument is properly configured.

Returns
True if configured, else False.

Return type
bool

is_draft() → bool

Checks if the BillModel is in Draft status.

Returns
True if BillModel is Draft, else False.

Return type
bool

is_paid() → bool

Checks if the BillModel is in Paid status.

Returns
True if BillModel is Paid, else False.

Return type
bool

is_past_due() → bool

Checks if the BillModel is past due.

Returns
True if BillModel is past due, else False.

Return type
bool

is_review() → bool

Checks if the BillModel is In Review status.

Returns
True if BillModel is in Review, else False.

Return type
bool

is_void() → bool

Checks if the BillModel is in Void status.

Returns
True if BillModel is Void, else False.

Return type
bool

make_payment(*payment_amount: Union[Decimal, float, int]*, *payment_date: Optional[Union[datetime, date]] = None*, *commit: bool = False*, *raise_exception: bool = True*)

Makes a payment to the BillModel.

Parameters

- **payment_amount** (*Decimal ot float*) – The payment amount to process.
- **payment_date** (*datetime or date.*) – Date or timestamp of the payment being applied.
- **commit** (*bool*) – If True, commits the transaction into the DB. Defaults to False.
- **raise_exception** (*bool*) – If True, raises BillModelValidationError if payment exceeds amount due, else False.

Returns
True if can make payment, else False.

Return type
bool

mark_as_approved(*user_model*, *entity_slug: Optional[str] = None*, *date_approved: Optional[Union[datetime, date]] = None*, *commit: bool = False*, *force_migrate: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Marks BillModel as Approved.

Parameters

- **entity_slug** – Entity slug associated with the BillModel. Avoids additional DB query if passed.
- **user_model** – UserModel associated with request.
- **date_approved** (*date*) – BillModel approved date. Defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.
- **force_migrate** (*bool*) – Forces migration. True if Accounting Method is Accrual.

mark_as_canceled(*date_canceled: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Mark BillModel as Canceled.

Parameters

- **date_canceled** (*date*) – BillModel canceled date. Defaults to localdate() if None.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_draft(*date_draft: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks BillModel as Draft.

Parameters

- **date_draft** (*date*) – Draft date. If None, defaults to localdate().

- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_paid(*user_model*, *entity_slug: Optional[str] = None*, *date_paid: Optional[Union[datetime, date]] = None*, *itemtxs_qs: Optional[*ItemTransactionModelQuerySet*] = None*, *commit: bool = False*, ***kwargs*)

Marks BillModel as Paid.

Parameters

- **entity_slug** (*str*) – Entity slug associated with the BillModel. Avoids additional DB query if passed.
- **user_model** – UserModel associated with request.
- **date_paid** (*date*) – BillModel paid date. Defaults to localdate() if None.
- **itemtxs_qs** (ItemTransactionModelQuerySet) – Pre-fetched ItemTransactionModelQuerySet. Avoids additional DB query. Validated if passed.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_review(*commit: bool = False*, *itemtxs_qs: Optional[*ItemTransactionModelQuerySet*] = None*, *date_in_review: Optional[date] = None*, *raise_exception: bool = True*, ***kwargs*)

Marks BillModel as In Review.

Parameters

- **date_in_review** (*date*) – BillModel in review date. Defaults to localdate() if None.
- **itemtxs_qs** (ItemTransactionModelQuerySet) – Pre fetched ItemTransactionModelQuerySet to use. Avoids additional DB Query if previously fetched.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.
- **raise_exception** (*bool*) – Raises BillModelValidationError if BillModel cannot be marked as in review. Defaults to True.

mark_as_void(*user_model*, *entity_slug: Optional[str] = None*, *date_void: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks BillModel as Void. When mark as void, all transactions associated with BillModel are reversed as of the void date.

Parameters

- **entity_slug** (*str*) – Entity slug associated with the BillModel. Avoids additional DB query if passed.
- **user_model** – UserModel associated with request.
- **date_void** (*date*) – BillModel void date. Defaults to localdate() if None.
- **commit** (*bool*) – Commits transaction into DB. Defaults to False.

migrate_itemtxs(*itemtxs: Dict*, *operation: str*, *commit: bool = False*)

Migrates a predefined item transaction list.

Parameters

- **itemtxs** (*dict*) – A dictionary where keys are the document number (invoice/bill number, etc) and values are a dictionary:
- **operation** (*str*) – A choice of ITEMIZE_REPLACE, ITEMIZE_APPEND, ITEMIZE_UPDATE
- **commit** (*bool*) – If True, commits transaction into the DB. Default to False

Returns
A list of ItemTransactionModel appended or created.

Return type
list

update_amount_due(*itemtxs_qs: Optional[Union[*ItemTransactionModelQuerySet, *List[*ItemTransactionModel*]]] = None*) → *ItemTransactionModelQuerySet*

Updates the BillModel amount due.

Parameters
itemtxs_qs (*ItemTransactionModelQuerySet or list of ItemTransactionModel*) – Optional pre-fetched ItemTransactionModelQuerySet. Avoids additional DB if provided. Queryset is validated if provided.

Returns
Newly fetched of previously fetched ItemTransactionModelQuerySet if provided.

Return type
ItemTransactionModelQuerySet

validate_itemtxs_qs(*queryset: Union[*ItemTransactionModelQuerySet, *List[*ItemTransactionModel*]]*)

Validates that the entire ItemTransactionModelQuerySet is bound to the BillModel.

Parameters
queryset (*ItemTransactionModelQuerySet or list of ItemTransactionModel.*) – ItemTransactionModelQuerySet to validate.

class django_ledger.models.bill.**BillModelManager**(**args*, ***kwargs*)

A custom defined BillModelManager that will act as an interface to handling the initial DB queries to the BillModel. The default "get_queryset" has been overridden to refer the custom defined "BillModelQuerySet".

for_entity(*entity_slug*, *user_model*) → *BillModelQuerySet*

Fetches a QuerySet of BillModels associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> bill_model_qs = BillModel.objects.for_entity(user_model=request_user,␣
↪entity_slug=slug)
```

Returns
Returns a BillModelQuerySet with applied filters.

Return type
BillModelQuerySet

for_user(*user_model*) → *BillModelQuerySet*

Fetches a QuerySet of BillModels that the UserModel as access to. May include BillModels from multiple Entities.

The user has access to bills if:

1. Is listed as Manager of Entity.
2. Is the Admin of the Entity.

Parameters
user_model – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> bill_model_qs = BillModel.objects.for_user(user_model=request_user)
```

Returns
Returns a BillModelQuerySet with applied filters.

Return type
BillModelQuerySet

get_queryset()

Return a new QuerySet object. Subclasses can override this method to customize the behavior of the Manager.

class django_ledger.models.bill.**BillModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined QuerySet for the BillModel. This implements multiple methods or queries needed to get a filtered QuerySet based on the BillModel status. For example, we might want to have list of bills which are paid, unpaid, due ,overdue, approved or in draft stage. All these separate functions will assist in making such queries and building customized reports.

active()

Active bills are those that are approved or paid, which have impacted or have the potential to impact the Entity's Ledgers.

Returns
Returns a QuerySet of active bills only.

Return type
BillModelQuerySet

approved()

Approved bills are those that have been reviewed and are expected to be paid before the due date.

Returns
Returns a QuerySet of approved bills only.

Return type
BillModelQuerySet

canceled()

Canceled bills are those that are discarded during the draft or in review status. These bills never had an impact on the books.

Returns
Returns a QuerySet of canceled bills only.

Return type
BillModelQuerySet

`draft()`

Default status of any bill that is created. Draft bills do not impact the Ledger.

Returns
Returns a QuerySet of draft bills only.

Return type
BillModelQuerySet

`in_review()`

In review bills are those that need additional review or approvals before being approved. In review bills do not impact the Ledger.

Returns
Returns a QuerySet of bills in review only.

Return type
BillModelQuerySet

`overdue()`

Overdue bills are those which due date is in the past.

Returns
Returns a QuerySet of overdue bills only.

Return type
BillModelQuerySet

`paid()`

Paid bills are those that have received 100% of the amount due.

Returns
Returns a QuerySet of paid bills only.

Return type
BillModelQuerySet

`unpaid()`

Unpaid bills are those that are approved but have not received 100% of the amount due. Equivalent to approved().

Returns
Returns a QuerySet of paid bills only.

Return type
BillModelQuerySet

`void()`

Void bills are those that where rolled back after being approved. Void bills rollback all transactions by creating a new set of transactions posted on the date_void.

Returns
Returns a QuerySet of void bills only.

Return type
BillModelQuerySet

exception django_ledger.models.bill.**BillModelValidationError**(*message*, *code=None*, *params=None*)

14.14 Estimate Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

The EstimateModel provides the means to estimate customer requests, jobs or quotes that may ultimately be considered contracts, if approved. The EstimateModels will estimate revenues and costs associated with a specific scope of work which is documented using ItemTransactionModels.

Once approved, the user may initiate purchase orders, bills and invoices that will be associated with the EstimateModel for tracking purposes. It is however not required to always have an EstimateModel, but recommended in order to be able to produce more specific financial reports associated with a specific scope of work.

class django_ledger.models.estimate.**EstimateModel**(**args*, ***kwargs*)

Base EstimateModel Class.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.estimate.**EstimateModelAbstract**(**args*, ***kwargs*)

This is the main abstract class which the EstimateModel database will inherit from. The EstimateModel inherits functionality from the following MixIns:

1. *MarkdownNotesMixIn*
2. *CreateUpdateMixIn*

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

estimate_number

Auto assigned number at creation by generate_estimate_number() function. Prefix be customized with DJANGO_LEDGER_ESTIMATE_NUMBER_PREFIX setting. Includes a reference to the Fiscal Year and a sequence number. Max Length is 20.

Type
str

entity

The EntityModel associated with te EntityModel instance.

Type
EntityModel

customer

The CustomerModel associated with the EstimateModel instance.

Type
CustomerModel

`title`

A string representing the name or title of the EstimateModel instance.

Type
str

`status`

The status of the EstimateModel instance. Must be one of Draft, In Review, Approved, Completed Void or Canceled.

Type
str

`terms`

The contract terms that will be associated with this EstimateModel instance. Choices are Fixed Price, Target Price, Time & Materials and Other.

Type
str

`date_draft`

The draft date represents the date when the EstimateModel was first created. Defaults to `localdate`.

Type
date

`date_in_review`

The in review date represents the date when the EstimateModel was marked as In Review status. Will be null if EstimateModel is canceled during draft status. Defaults to `localdate`.

Type
date

`date_approved`

The approved date represents the date when the EstimateModel was approved. Will be null if EstimateModel is canceled. Defaults to `localdate`.

Type
date

`date_completed`

The paid date represents the date when the EstimateModel was completed and fulfilled. Will be null if EstimateModel is canceled. Defaults to `localdate`.

Type
date

`date_void`

The void date represents the date when the EstimateModel was void, if applicable. Will be null unless EstimateModel is void. Defaults to `localdate`.

Type
date

`date_canceled`

The canceled date represents the date when the EstimateModel was canceled, if applicable. Will be null unless EstimateModel is canceled. Defaults to `localdate`.

Type
date

`revenue_estimate`

The total estimated revenue of the EstimateModel instance.

Type
Decimal

`labor_estimate`

The total labor costs estimate of the EstimateModel instance.

Type
Decimal

`material_estimate`

The total material costs estimate of the EstimateModel instance.

Type
Decimal

`equipment_estimate`

The total equipment costs estimate of the EstimateModel instance.

Type
Decimal

`other_estimate`

the total miscellaneous costs estimate of the EstimateModel instance.

Type
Decimal

`can_approve()`

Determines if the EstimateModel can be marked as approved.

Returns
True if EstimateModel can be marked as approved, else False.

Return type
bool

`can_bind()`

Determines if the EstimateModel can be bound to a set of POs, Bills or Invoices.

Returns
True if EstimateModel can be bound, else False.

Return type
bool

`can_cancel()`

Determines if the EstimateModel can be marked as canceled.

Returns
True if EstimateModel can be marked as canceled, else False.

Return type
bool

can_complete()

Determines if the EstimateModel can be marked as completed.

Returns
True if EstimateModel can be marked as completed, else False.

Return type
bool

can_draft() → bool

Determines if the EstimateModel can be marked as Draft.

Returns
True if EstimateModel can be marked as draft, else False.

Return type
bool

can_generate_estimate_number()

Determines if the EstimateModel can generate its own estimate number..

Returns
True if EstimateModel can generate the estimate number, else False.

Return type
bool

can_migrate_itemtxs() → bool

Checks if item transaction list can be migrated.

Return type
bool

can_review()

Determines if the EstimateModel can be marked as In Review.

Returns
True if EstimateModel can be marked as In Review, else False.

Return type
bool

can_update_items()

Determines if the EstimateModel item transactions can be edited or changed.

Returns
True if EstimateModel item transactions can be edited or changed, else False.

Return type
bool

can_void()

Determines if the EstimateModel can be marked as void.

Returns
True if EstimateModel can be marked as void, else False.

Return type
bool

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

configure(*entity_slug: Union[EntityModel, UUID, str]*, *customer_model:* CustomerModel, *user_model: Optional[User] = None*, *date_draft: Optional[date] = None*, *estimate_title: Optional[str] = None*, *commit: bool = False*, *raise_exception: bool = True*)

A configuration hook which executes all initial EstimateModel setup. Can only call this method once in the lifetime of a EstimateModel.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel to associate the Bill with.
- **user_model** – The UserModel making the request to check for QuerySet permissions.
- **date_draft** (*date*) – The draft date to use. If None defaults to localdate().
- **customer_model** (CustomerModel) – The CustomerModel to be associated with this EstimateModel instance.
- **commit** (*bool*) – Saves the current EstimateModel after being configured.
- **estimate_title** (*str*) – Optional EstimateModel title.
- **raise_exception** (*bool*) – If True, raises EstimateModelValidationError when model is already configured.

Returns
The configured EstimateModel instance.

Return type
EstimateModel

generate_estimate_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next EstimateModel document number available. The operation will result in two additional queries if the EntityModel is not cached in QuerySet via select_related('entity').

Parameters
commit (*bool*) – Commits transaction into BillModel.

Returns
A String, representing the generated BillModel instance Document Number.

Return type
str

get_contract_summary(*po_qs: Optional[PurchaseOrderModelQuerySet] = None*, *bill_qs: Optional[BillModelQuerySet] = None*, *invoice_qs: Optional[InvoiceModelQuerySet] = None*) → dict

Computes an aggregate of all related ItemTransactionModels summarizing original contract amounts, amounts authorized, amounts billed and amount invoiced.

Parameters

- **po_qs** (PurchaseOrderModelQuerySet) – Prefetched PurchaseOrderModelQuerySet. If provided will be validated.
- **bill_qs** (BillModelQuerySet) – Prefetched BillModelQuerySet. If provided will be validated.

- **invoice_qs** (InvoiceModelQuerySet) – Prefetched InvoiceModelQuerySet. If provided will be validated.

Returns
A dictionary of aggregated values.

Return type
dict

get_cost_estimate(*as_float: bool = False*) → Union[float, Decimal]

Calculates the total EstimateModel cost by summing all individual cost components: Labor, Material, Equipment & Other costs.

Parameters
as_float (*bool*) – If True, will return value as float, else as Decimal.

Returns
The total EstimateModel cost.

Return type
Decimal or float

get_gross_margin_estimate(*as_percent: bool = False*, *raise_exception: bool = False*) → float

Computes the EstimateModel gross margin.

Parameters

- **as_percent** (*bool*) – If True, computes gross margin as percent.
- **raise_exception** (*bool*) –

Returns
The EstimateModel gross margin.

Return type
float

get_item_model_qs() → *ItemModelQuerySet*

Fetches the ItemModelQuerySet eligible to itemize.

Return type
ItemModelQuerySet

get_itemtxs_annotation(*itemtxs_qs: Optional[*ItemTransactionModelQuerySet*] = None*)

Gets an annotated ItemTransactionModelQuerySet with additional average unit cost & revenue.

Parameters
itemtxs_qs (ItemTransactionModelQuerySet) – Prefetched ItemTransactionModelQuerySet if any. If None a new queryset will be evaluated. Will be validated if provided.

Returns
The original ItemTransactionModelQuerySet and the annotated ItemTransactionModelQuerySet.

Return type
tuple

get_itemtxs_data(*queryset: Optional[Union[*ItemTransactionModelQuerySet*, List[*ItemTransactionModel*]]] = None*, *aggregate_on_db: bool = False*, *lazy_agg: bool = False*)

Returns all ItemTransactionModels associated with the EstimateModel and a total aggregate.

Parameters
queryset (ItemTransactionModelQuerySet) – ItemTransactionModelQuerySet to use. Avoids additional DB query if provided. Validated if provided.

Return type
ItemTransactionModelQuerySet

`get_mark_as_approved_html_id()`

EstimateModel Mark as Approved HTML ID Tag.

Returns
HTML ID as a String

Return type
str

`get_mark_as_approved_message()`

EstimateModel Mark as Approved Message.

Returns
Message as a String

Return type
str

`get_mark_as_approved_url()`

EstimateModel Mark as Approved URL.

Returns
URL as a String

Return type
str

`get_mark_as_canceled_html_id()`

EstimateModel Mark as Canceled HTML ID Tag.

Returns
HTML ID as a String

Return type
str

`get_mark_as_canceled_message()`

EstimateModel Mark as Canceled Message.

Returns
Message as a String

Return type
str

`get_mark_as_canceled_url()`

EstimateModel Mark as Canceled URL.

Returns
URL as a String

Return type
str

get_mark_as_completed_html_id()

EstimateModel Mark as Complete HTML ID Tag.

Returns
HTML ID as a String

Return type
str

get_mark_as_completed_message()

EstimateModel Mark as Completed Message.

Returns
Message as a String

Return type
str

get_mark_as_completed_url()

EstimateModel Mark as Completed URL.

Returns
URL as a String

Return type
str

get_mark_as_draft_html_id()

EstimateModel Mark as Draft HTML ID Tag.

Returns
HTML ID as a String

Return type
str

get_mark_as_draft_message()

EstimateModel Mark as Draft Message.

Returns
Message as a String

Return type
str

get_mark_as_draft_url()

EstimateModel Mark as Draft URL.

Returns
URL as a String

Return type
str

get_mark_as_review_html_id()

EstimateModel Mark as Review HTML ID Tag.

Returns
HTML ID as a String

Return type
str

`get_mark_as_review_message()`

EstimateModel Mark as In Review Message.

Returns
Message as a String

Return type
str

`get_mark_as_review_url()`

EstimateModel Mark as In Review URL.

Returns
URL as a String

Return type
str

`get_mark_as_void_html_id()`

EstimateModel Mark as Void HTML ID Tag.

Returns
HTML ID as a String

Return type
str

`get_mark_as_void_message()`

EstimateModel Mark as Void message.

Returns
Message as a String

Return type
str

`get_mark_as_void_url()`

EstimateModel Mark as Void URL.

Returns
URL as a String

Return type
str

`get_profit_estimate`(*as_float: bool = False*)

Calculates the total EstimateModel profit estimate.

Parameters
as_float (*bool*) – If True, will return value as float, else as Decimal.

Returns
The total EstimateModel profit.

Return type
Decimal or float

`get_revenue_estimate`(*as_float: bool = False*)

Calculates the total EstimateModel revenue.

Parameters
as_float (*bool*) – If True, will return value as float, else as Decimal.

Returns
The total EstimateModel revenue.

Return type
Decimal or float

`get_status_action_date()` → date

The date when the latest status took place.

Examples

If EstimateModel is Draft, will return date_draft. If EstimateModel is In Review, will return date_in_review. If EstimateModel is Approved, will return date_approved.

Returns
The latest action date.

Return type
date

`is_approved()` → bool

Determines if the EstimateModel is in Approved status.

Returns
True if EstimateModel is in Approved status, else False.

Return type
bool

`is_canceled()` → bool

Determines if the EstimateModel is in Canceled status.

Returns
True if EstimateModel is in Canceled status, else False.

Return type
bool

`is_completed()` → bool

Determines if the EstimateModel is in Completed status.

Returns
True if EstimateModel is in Completed status, else False.

Return type
bool

`is_configured()` → bool

Determines if the EstimateModel is configured.

Returns
True if EstimateModel is configured, else False.

Return type
bool

`is_contract()`

Determines if the EstimateModel is considered a Contract.

Returns
True if EstimateModel is a Contract, else False.

Return type
bool

`is_draft`() → bool

Determines if the EstimateModel is in Draft status.

Returns
True if EstimateModel is in Draft status, else False.

Return type
bool

`is_review`() → bool

Determines if the EstimateModel is In Review status.

Returns
True if EstimateModel is In Review status, else False.

Return type
bool

`is_void`()

Determines if the EstimateModel is in Void status.

Returns
True if EstimateModel is in Void status, else False.

Return type
bool

`mark_as_approved`(*commit=False*, *date_approved: Optional[date] = None*)

Marks the current EstimateModel instance as Approved.

Parameters

- **commit** (*bool*) – Commits transaction into current EstimateModel instance.
- **date_approved** (*date*) – Optional date when EstimateModel instance is Approved. Defaults to localdate().

`mark_as_canceled`(*commit: bool = False*, *date_canceled: Optional[date] = None*)

Marks the current EstimateModel instance as Canceled.

Parameters

- **commit** (*bool*) – Commits transaction into current EstimateModel instance.
- **date_canceled** (*date*) – Optional date when EstimateModel instance is canceled. Defaults to localdate().

`mark_as_completed`(*commit=False*, *date_completed: Optional[date] = None*)

Marks the current EstimateModel instance as Completed.

Parameters

- **commit** (*bool*) – Commits transaction into current EstimateModel instance.
- **date_completed** (*date*) – Optional date when EstimateModel instance is completed. Defaults to localdate().

`mark_as_draft`(*commit: bool = False*)

Marks the current EstimateModel instance as Draft.

Parameters
commit (*bool*) – Commits transaction into current EstimateModel instance.

mark_as_review(*itemtxs_qs: Optional[ItemTransactionModelQuerySet] = None*, *date_in_review: Optional[date] = None*, *commit: bool = True*)

Marks the current EstimateModel instance as In Review.

Parameters

- **commit** (*bool*) – Commits transaction into current EstimateModel instance.
- **itemtxs_qs** (*ItemTransactionModelQuerySet*) – Optional, pre-fetched ItemTransactionModelQuerySet. Avoids additional DB query. Validated if provided.
- **date_in_review** (*date*) – Optional date when EstimateModel instance is In Review. Defaults to localdate().

mark_as_void(*commit: bool = False*, *date_void: Optional[date] = None*)

Marks the current EstimateModel instance as Void.

Parameters

- **commit** (*bool*) – Commits transaction into current EstimateModel instance.
- **date_void** (*date*) – Optional date when EstimateModel instance is void. Defaults to localdate().

migrate_itemtxs(*itemtxs: Dict*, *operation: str*, *commit: bool = False*)

Migrates a predefined item transaction list.

Parameters

- **itemtxs** (*dict*) – A dictionary where keys are the document number (invoice/bill number, etc) and values are a dictionary:
- **operation** (*str*) – A choice of ITEMIZE_REPLACE, ITEMIZE_APPEND, ITEMIZE_UPDATE
- **commit** (*bool*) – If True, commits transaction into the DB. Default to False

Returns
A list of ItemTransactionModel appended or created.

Return type
list

save(***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

update_cost_estimate(*itemtxs_qs: Optional[ItemTransactionModelQuerySet] = None*, *commit: bool = False*)

Updates the cost estimate of the EstimateModel instance.

Parameters

- **itemtxs_qs** (*ItemTransactionModelQuerySet*) – Prefetched ItemTransactionModelQuerySet. A new ItemTransactionModelQuerySet will be fetched from DB if not provided. If provided will be validated.
- **commit** (*bool*) – If True, the new revenue estimate will be committed into the DB.

update_revenue_estimate(*itemtxs_qs: Optional[ItemTransactionModelQuerySet] = None*, *commit: bool = False*)

Updates the revenue estimate of the EstimateModel instance.

Parameters

- **itemtxs_qs** (ItemTransactionModelQuerySet) – Prefetched ItemTransactionModelQuerySet. A new ItemTransactionModelQuerySet will be fetched from DB if not provided. If provided will be validated.
- **commit** (*bool*) – If True, the new revenue estimate will be committed into the DB.

validate_bill_queryset(*bill_qs:* BillModelQuerySet) → *BillModelQuerySet*

Validates a prefetched BillModelQuerySet against the EstimateModel instance.

Parameters
bill_qs (BillModelQuerySet) – The BillModelQuerySet to validate

Returns
The validated BillModelQuerySet.

Return type
BillModelQuerySet

validate_invoice_queryset(*invoice_qs:* InvoiceModelQuerySet) → *InvoiceModelQuerySet*

Validates a prefetched InvoiceModelQuerySet against the EstimateModel instance.

Parameters
invoice_qs (InvoiceModelQuerySet) – The InvoiceModelQuerySet to validate

Returns
The validated InvoiceModelQuerySet.

Return type
InvoiceModelQuerySet

validate_item_transaction_qs(*itemtxs_qs: Union[ItemTransactionModelQuerySet, List[ItemTransactionModel]]*)

Validates that the entire ItemTransactionModelQuerySet is bound to the BillModel.

Parameters
itemtxs_qs (ItemTransactionModelQuerySet) – ItemTransactionModelQuerySet to validate.

validate_itemtxs_qs(*queryset: Union[ItemTransactionModelQuerySet, List[ItemTransactionModel]]*)

Validates that the entire ItemTransactionModelQuerySet is bound to the EstimateModel.

Parameters
queryset (ItemTransactionModelQuerySet *or list of ItemTransactionModel.*) – ItemTransactionModelQuerySet to validate.

validate_po_queryset(*po_qs:* PurchaseOrderModelQuerySet) → *PurchaseOrderModelQuerySet*

Validates a prefetched PurchaseOrderModelQuerySet against the EstimateModel instance.

Parameters
po_qs (PurchaseOrderModelQuerySet) – The PurchaseOrderModelQuerySet to validate

Returns
The validated PurchaseOrderModelQuerySet.

Return type
PurchaseOrderModelQuerySet

class django_ledger.models.estimate.**EstimateModelManager**(**args*, ***kwargs*)

A custom defined EstimateModelManager that that implements custom QuerySet methods related to the EstimateModel.

for_entity(*entity_slug: Union[*EntityModel*, str]*, *user_model*)

Fetches a QuerySet of EstimateModels associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> bill_model_qs = EstimateModel.objects.for_entity(user_model=request_user,␣
↪entity_slug=slug)
```

Returns
Returns a EstimateModelQuerySet with applied filters.

Return type
EstimateModelQuerySet

class django_ledger.models.estimate.**EstimateModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom-defined LedgerModelManager that implements custom QuerySet methods related to the EstimateModel.

approved()

Approved Estimates or Sales Orders are those that have been approved or completed.

Returns
A EstimateModelQuerySet with applied filters.

Return type
EstimateModelQuerySet

contracts()

A contract are Estimates or Sales Orders are those that have been approved or completed.

Returns
A EstimateModelQuerySet with applied filters. Equivalent to approve.

Return type
EstimateModelQuerySet

estimates()

Estimates or Sales Orders are those that have not been approved or completed.

Returns
A EstimateModelQuerySet with applied filters. Equivalent to not approved.

Return type
EstimateModelQuerySet

not_approved()

Not approved Estimates or Sales Orders are those that have not been approved or completed.

Returns
A EstimateModelQuerySet with applied filters.

Return type
EstimateModelQuerySet

exception django_ledger.models.estimate.**EstimateModelValidationError**(*message*, *code=None*, *params=None*)

14.15 Purchase Order Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <Ptulshyan77@gmail.com>

A purchase order is a commercial source document that is issued by a business purchasing department when placing an order with its vendors or suppliers. The document indicates the details on the items that are to be purchased, such as the types of goods, quantity, and price. In simple terms, it is the contract drafted by the buyer when purchasing goods from the seller.

The PurchaseOrderModel is designed to track the status of a Purchase Order and all its items. The PurchaseOrderModel starts in draft model by default and goes through different states including InReview, Approved, Fulfilled, Canceled and Void. The PurchaseOrderModel also keeps track of when these states take place.

class django_ledger.models.purchase_order.**PurchaseOrderModel**(**args*, ***kwargs*)

Purchase Order Base Model

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.purchase_order.**PurchaseOrderModelAbstract**(**args*, ***kwargs*)

The base implementation of the PurchaseOrderModel.

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

po_number

A unique human-readable and sequential PO Number identifier. Automatically generated.

Type
str

po_title

The PurchaseOrderModel instance title.

Type
str

po_status

One of PO_STATUS values representing the current status of the PurchaseOrderModel instance.

Type
str

po_amount

The total value of the PurchaseOrderModel instance.

Type
Decimal

po_amount_received

The PurchaseOrderModel instance total value received to date. Cannot be greater than PO amount.

Type
Decimal

entity

The EntityModel associated with the PurchaseOrderModel instance.

Type
EntityModel

date_draft

The draft date represents the date when the PurchaseOrderModel was first created. Defaults to `localdate`.

Type
date

date_in_review

The in review date represents the date when the PurchaseOrderModel was marked as In Review status. Will be null if PurchaseOrderModel is canceled during draft status. Defaults to `localdate`.

Type
date

date_approved

The approved date represents the date when the PurchaseOrderModel was approved. Will be null if PurchaseOrderModel is canceled. Defaults to `localdate`.

Type
date

date_fulfilled

The paid date represents the date when the PurchaseOrderModel was fulfilled and po_amount_received equals po_amount. Will be null if PurchaseOrderModel is canceled. Defaults to `localdate`.

Type
date

date_void

The void date represents the date when the PurchaseOrderModel was void, if applicable. Will be null unless PurchaseOrderModel is void. Defaults to `localdate`.

Type
date

`date_canceled`

The canceled date represents the date when the PurchaseOrderModel was canceled, if applicable. Will be null unless PurchaseOrderModel is canceled. Defaults to `localdate`.

Type
date

`po_items`

A foreign key reference to the list of ItemTransactionModel that make the PurchaseOrderModel amount.

`ce_model`

A foreign key reference to the EstimateModel associated with the PurchaseOrderModel, if any.

Type
EstimateModel

`PO_STATUS_CANCELED = 'canceled'`

The different valid PO Status and their representation in the Database

`action_bind_estimate`(*estimate_model*, *commit: bool = False*)

Binds a specific EstimateModel to the PurchaseOrderModel instance.

Parameters

- **estimate_model** (EstimateModel) – The EstimateModel to bind.
- **commit** (*bool*) – Commits the changes in the Database, if True. Defaults to False.

`can_approve`() → bool

Checks if the PurchaseOrderModel can be marked as Approved.

Returns
True if PurchaseOrderModel can be marked as Approved, else False.

Return type
bool

`can_bind_estimate`(*estimate_model*, *raise_exception: bool = False*) → bool

Checks if the PurchaseOrderModel ican be bound to an EstimateModel.

Returns
True if PurchaseOrderModel can be bound to an EstimateModel, else False.

Return type
bool

`can_cancel`() → bool

Checks if the PurchaseOrderModel can be marked as Canceled.

Returns
True if PurchaseOrderModel can be marked as Canceled, else False.

Return type
bool

`can_delete`() → bool

Checks if the PurchaseOrderModel can be deleted.

Returns
True if PurchaseOrderModel can be deleted, else False.

Return type
bool

can_draft() → bool

Checks if the PurchaseOrderModel can be marked as Draft.

Returns
True if PurchaseOrderModel can be marked as Draft, else False.

Return type
bool

can_edit_items() → bool

Checks if the PurchaseOrderModel items can be edited.

Returns
True if PurchaseOrderModel items can be edited, else False.

Return type
bool

can_fulfill() → bool

Checks if the PurchaseOrderModel can be marked as Fulfilled.

Returns
True if PurchaseOrderModel can be marked as Fulfilled, else False.

Return type
bool

can_generate_po_number()

Checks if PurchaseOrderModel can generate its Document Number.

Returns
True if PurchaseOrderModel can generate its po_number, else False.

Return type
bool

can_migrate_itemtxs() → bool

Checks if item transaction list can be migrated.

Return type
bool

can_review() → bool

Checks if the PurchaseOrderModel can be marked as In Review.

Returns
True if PurchaseOrderModel can be marked as In Review, else False.

Return type
bool

can_void() → bool

Checks if the PurchaseOrderModel can be marked as Void.

Returns
True if PurchaseOrderModel can be marked as Void, else False.

Return type
bool

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

configure(*entity_slug: Union[str, EntityModel]*, *po_title: Optional[str] = None*, *user_model: Optional[User] = None*, *draft_date: Optional[date] = None*, *estimate_model=None*, *commit: bool = False*)

A configuration hook which executes all initial PurchaseOrderModel setup on to the EntityModel and all initial values of the EntityModel. Can only call this method once in the lifetime of a PurchaseOrderModel.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel to associate the Bill with.
- **user_model** – The UserModel making the request to check for QuerySet permissions.
- **ledger_posted** – An option to mark the BillModel Ledger as posted at the time of configuration. Defaults to False.
- **bill_desc** (*str*) – An optional description appended to the LedgerModel name.
- **commit** (*bool*) – Saves the current BillModel after being configured.

Returns
The configured PurchaseOrderModel instance.

Return type
PurchaseOrderModel

generate_po_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next PurchaseOrder document number available.

Parameters
commit (*bool*) – Commits transaction into PurchaseOrderModel.

Returns
A String, representing the generated or current PurchaseOrderModel instance Document Number.

Return type
str

get_item_model_qs() → *ItemModelQuerySet*

Fetches the ItemModelQuerySet eligible to itemize.

Return type
ItemModelQuerySet

get_itemtxs_data(*queryset: Optional[Union[ItemTransactionModelQuerySet, List[ItemTransactionModel]]] = None*, *aggregate_on_db: bool = False*, *lazy_agg: bool = False*) → Tuple

Fetches the PurchaseOrderModel Items and aggregates the QuerySet.

Parameters

- **queryset** (*ItemTransactionModelQuerySet*) – Optional pre-fetched ItemModelQueryset to use. Avoids additional DB query if provided. Validated if provided.
- **aggregate_on_db** (*bool*) – If True, performs aggregation of ItemsTransactions in the DB resulting in one additional DB query.

Returns
A tuple

Return type
ItemTransactionModelQuerySet, dict

`get_mark_as_approved_html_id()`

PurchaseOrderModel Mark as Approved HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_approved_message()`

PurchaseOrderModel Mark as Approved Message.

Returns
Message as a String.

Return type
str

`get_mark_as_approved_url()`

PurchaseOrderModel Mark as Approved URL ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_canceled_html_id()`

PurchaseOrderModel Mark as Canceled HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_canceled_message()`

PurchaseOrderModel Mark as Canceled Message.

Returns
Message as a String.

Return type
str

`get_mark_as_canceled_url()`

PurchaseOrderModel Mark as Canceled URL ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_draft_html_id()`

PurchaseOrderModel Mark as Draft HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_draft_message()`

PurchaseOrderModel Mark as Draft Message.

Returns
Message as a String.

Return type
str

`get_mark_as_draft_url()`

PurchaseOrderModel Mark as Draft URL ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_fulfilled_html_id()`

PurchaseOrderModel Mark as Fulfilled HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_fulfilled_message()`

PurchaseOrderModel Mark as Fulfilled Message.

Returns
Message as a String.

Return type
str

`get_mark_as_fulfilled_url()`

PurchaseOrderModel Mark as Fulfilled URL ID Tag.

Returns
URL as a String.

Return type
str

`get_mark_as_review_html_id()`

PurchaseOrderModel Mark as In Review HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_review_message()

PurchaseOrderModel Mark as Review Message.

Returns
Message as a String.

Return type
str

get_mark_as_review_url()

PurchaseOrderModel Mark as In Review URL ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_void_html_id()

PurchaseOrderModel Mark as Void HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_void_message()

PurchaseOrderModel Mark as Void Message.

Returns
Message as a String.

Return type
str

get_mark_as_void_url()

PurchaseOrderModel Mark as Void URL ID Tag.

Returns
HTML ID as a String.

Return type
str

get_po_bill_queryset() → *BillModelQuerySet*

Fetches a BillModelQuerySet of all BillModels associated with the PurchaseOrderModel instance.

Return type
BillModelQuerySet

get_status_action_date()

Current status action date.

Returns
A date. i.e. If status is Approved, return date_approved. If In Review, return date_in_review.

Return type
date

is_approved() → bool

Checks if the PurchaseOrderModel is in Approved status.

Returns
True if PurchaseOrderModel is Approved, else False.

Return type
bool

is_canceled() → bool

Checks if the PurchaseOrderModel is in Canceled status.

Returns
True if PurchaseOrderModel is in Canceled, else False.

Return type
bool

is_contract_bound()

Checks if the PurchaseOrderModel is bound to an EstimateModel.

Returns
True if PurchaseOrderModel is bound to an EstimateModel, else False.

Return type
bool

is_draft() → bool

Checks if the PurchaseOrderModel is in Draft status.

Returns
True if PurchaseOrderModel is Draft, else False.

Return type
bool

is_fulfilled() → bool

Checks if the PurchaseOrderModel is in Fulfilled status.

Returns
True if PurchaseOrderModel is in Fulfilled status, else False.

Return type
bool

is_review() → bool

Checks if the PurchaseOrderModel is in Review status.

Returns
True if PurchaseOrderModel is Review, else False.

Return type
bool

is_void() → bool

Checks if the PurchaseOrderModel is in Void status.

Returns
True if PurchaseOrderModel is Void, else False.

Return type
bool

mark_as_approved(*date_approved: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks PurchaseOrderModel as Approved.

Parameters

- **date_approved** (*date*) – Approved date. If None, defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_canceled(*date_canceled: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks PurchaseOrderModel as Canceled.

Parameters

- **date_canceled** (*date*) – Canceled date. If None, defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_draft(*date_draft: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks PurchaseOrderModel as Draft.

Parameters

- **date_draft** (*date*) – Draft date. If None, defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_fulfilled(*date_fulfilled: Optional[date] = None*, *po_items: Optional[Union[*ItemTransactionModelQuerySet*, List[*ItemTransactionModel*]]] = None*, *commit: bool = False*, ***kwargs*)

Marks PurchaseOrderModel as Fulfilled.

Parameters

- **date_fulfilled** (*date*) – Fulfilled date. If None, defaults to localdate().
- **po_items** (*ItemTransactionModelQuerySet or list of ItemTransactionModel.*) – Pre-fetched ItemTransactionModelQuerySet or list of ItemTransactionModel. Validated if provided.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_review(*date_in_review: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks PurchaseOrderModel as In Review.

Parameters

- **date_in_review** (*date*) – Draft date. If None, defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_void(*void_date: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks PurchaseOrderModel as Fulfilled.

Parameters

- **void_date** (*date*) – Void date. If None, defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

migrate_itemtxs(*itemtxs: Dict*, *operation: str*, *commit: bool = False*)

Migrates a predefined item transaction list.

Parameters

- **itemtxs** (*dict*) – A dictionary where keys are the document number (invoice/bill number, etc) and values are a dictionary:
- **operation** (*str*) – A choice of ITEMIZE_REPLACE, ITEMIZE_APPEND, ITEMIZE_UPDATE
- **commit** (*bool*) – If True, commits transaction into the DB. Default to False

Returns
A list of ItemTransactionModel appended or created.

Return type
list

update_state(*itemtxs_qs: Optional[Union[*ItemTransactionModelQuerySet*, List[*ItemTransactionModel*]]] = None*) → Tuple

Updates the state of the PurchaseOrderModel.

Parameters
itemtxs_qs (*ItemTransactionModelQuerySet or list of ItemTransactionModel*) –

Returns
A tuple of ItemTransactionModels and Aggregation

Return type
tuple

validate_item_transaction_qs(*queryset: Union[*ItemTransactionModelQuerySet*, List[*ItemTransactionModel*]]*)

Validates that the entire ItemTransactionModelQuerySet is bound to the PurchaseOrderModel.

Parameters
queryset (*ItemTransactionModelQuerySet or list of ItemTransactionModel.*) – ItemTransactionModelQuerySet to validate.

class django_ledger.models.purchase_order.**PurchaseOrderModelManager**(**args*, ***kwargs*)

A custom defined PurchaseOrderModel Manager.

for_entity(*entity_slug*, *user_model*) → *PurchaseOrderModelQuerySet*

Fetches a QuerySet of PurchaseOrderModel associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Returns
A PurchaseOrderModelQuerySet with applied filters.

Return type
PurchaseOrderModelQuerySet

class django_ledger.models.purchase_order.**PurchaseOrderModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined PurchaseOrderModel QuerySet.

active()

Filters the QuerySet to include Active PurchaseOrderModels only. Active PurchaseOrderModels are either approved or fulfilled, which are those that may contain associated transactions on the Ledger.

Returns
A PurchaseOrderModelQuerySet with applied filters.

Return type
PurchaseOrderModelQuerySet

approved()

Filters the QuerySet to include Approved PurchaseOrderModels only.

Returns
A PurchaseOrderModelQuerySet with applied filters.

Return type
PurchaseOrderModelQuerySet

fulfilled()

Filters the QuerySet to include Fulfilled PurchaseOrderModels only.

Returns
A PurchaseOrderModelQuerySet with applied filters.

Return type
PurchaseOrderModelQuerySet

exception django_ledger.models.purchase_order.**PurchaseOrderModelValidationError**(*message*, *code=None*, *params=None*)

14.16 Invoice Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

This module implements the InvoiceModel, which represents the Sales Invoice/ Sales Invoice/ Tax Invoice/ Proof of Sale which the *EntityModel* issues to its customers for the supply of goods or services. The model manages all the Sales Invoices which are issued by the *EntityModel*. In addition to tracking the invoice amount , it tracks the receipt and due amount.

Examples

```
>>> user_model = request.user  # django UserModel
>>> entity_slug = kwargs['entity_slug'] # may come from view kwargs
>>> invoice_model = InvoiceModel()
>>> ledger_model, invoice_model = invoice_model.configure(entity_slug=entity_slug, user_
↪model=user_model)
>>> invoice_model.save()
```

class django_ledger.models.invoice.**InvoiceModel**(**args*, ***kwargs*)

Base Invoice Model from Abstract.

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.invoice.**InvoiceModelAbstract**(**args*, ***kwargs*)

This is the main abstract class which the InvoiceModel database will inherit from. The InvoiceModel inherits functionality from the following MixIns:

1. `LedgerWrapperMixIn`
2. *PaymentTermsMixIn*
3. *MarkdownNotesMixIn*
4. *CreateUpdateMixIn*

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

invoice_number

Auto assigned number at creation by generate_invoice_number() function. Prefix be customized with DJANGO_LEDGER_INVOICE_NUMBER_PREFIX setting. Includes a reference to the Fiscal Year, Entity Unit and a sequence number. Max Length is 20.

Type
str

invoice_status

Current status of the InvoiceModel. Must be one of the choices as mentioned under "INVOICE_STATUS". By default, the status will be "Draft".

Type
str

customer

This is the foreign key reference to the CustomerModel from whom the purchase has been made.

Type
`CustomerModel`

additional_info

Any additional metadata about the InvoiceModel may be stored here as a dictionary object. The data is serialized and stored as a JSON document in the Database.

Type
dict

invoice_items

A foreign key reference to the list of ItemTransactionModel that make the invoice amount.

ce_model

A foreign key to the InvoiceModel associated EstimateModel for overall Job/Contract tracking.

Type
EstimateModel

date_draft

The draft date represents the date when the InvoiceModel was first created. Defaults to `localdate`.

Type
date

date_in_review

The in review date represents the date when the InvoiceModel was marked as In Review status. Will be null if InvoiceModel is canceled during draft status. Defaults to `localdate`.

Type
date

date_approved

The approved date represents the date when the InvoiceModel was approved. Will be null if InvoiceModel is canceled. Defaults to `localdate`.

Type
date

date_paid

The paid date represents the date when the InvoiceModel was paid and amount_due equals amount_paid. Will be null if InvoiceModel is canceled. Defaults to `localdate`.

Type
date

date_void

The void date represents the date when the InvoiceModel was void, if applicable. Will be null unless InvoiceModel is void. Defaults to `localdate`.

Type
date

date_canceled

The canceled date represents the date when the InvoiceModel was canceled, if applicable. Will be null unless InvoiceModel is canceled. Defaults to `localdate`.

Type
date

INVOICE_STATUS = [('draft', 'Draft'), ('in_review', 'In Review'), ('approved', 'Approved'), ('paid', 'Paid'), ('void', 'Void'), ('canceled', 'Canceled')]

The different invoice status options and their representation in the Database.

bind_estimate(*estimate_model*, *commit: bool = False*)

Binds InvoiceModel to a given EstimateModel. Raises ValueError if EstimateModel cannot be bound.

Parameters

- **estimate_model** (EstimateModel) – EstimateModel to bind.
- **raise_exception** (*bool*) – Raises InvoiceModelValidationError if unable to bind EstimateModel.
- **commit** (*bool*) – Commits transaction into current InvoiceModel.

can_approve()

Checks if the InvoiceModel can be marked as Approved.

Returns
True if InvoiceModel can be marked as approved, else False.

Return type
bool

can_bind_estimate(*estimate_model*, *raise_exception: bool = False*) → bool

Checks if the InvoiceModel can be bound to a given EstimateModel.

Parameters

- **estimate_model** (EstimateModel) – EstimateModel to check against.
- **raise_exception** (*bool*) – If True, raises InvoiceModelValidationError if unable to bind. Else, returns False.

Returns
True if can bind provided EstimateModel, else False.

Return type
bool

can_cancel()

Checks if the InvoiceModel can be marked as Canceled status.

Returns
True if InvoiceModel can be marked as canceled, else False.

Return type
bool

can_delete()

Checks if the InvoiceModel can be deleted.

Returns
True if InvoiceModel can be deleted, else False.

Return type
bool

can_draft()

Checks if the InvoiceModel can be marked as Draft.

Returns
True if InvoiceModel can be marked as draft, else False.

Return type
bool

can_edit_items()

Checks if the InvoiceModel item transactions can be edited.

Returns
True if InvoiceModel items can be edited, else False.

Return type
bool

can_generate_invoice_number()

Checks if InvoiceModel can generate its Document Number.

Returns
True if InvoiceModel can generate its invoice_number, else False.

Return type
bool

can_make_payment() → bool

Checks if the BillModel can accept a payment.

Returns
True if can bind provided PurchaseOderModel, else False.

Return type
bool

can_migrate()

Checks if the InvoiceModel can be migrated.

Returns
True if InvoiceModel can be migrated, else False.

Return type
bool

can_migrate_itemtxs() → bool

Checks if item transaction list can be migrated.

Return type
bool

can_pay()

Checks if the InvoiceModel can be marked as Paid.

Returns
True if InvoiceModel can be marked as paid, else False.

Return type
bool

can_review()

Checks if the InvoiceModel can be marked as In Review.

Returns
True if InvoiceModel can be marked as in review, else False.

Return type
bool

can_void()

Checks if the InvoiceModel can be marked as Void status.

Returns
True if InvoiceModel can be marked as void, else False.

Return type
bool

clean(*commit: bool = True*)

Clean method for InvoiceModel. Results in a DB query if invoice number has not been generated and the InvoiceModel is eligible to generate an invoice_number.

Parameters
commit (*bool*) – If True, commits into DB the generated InvoiceModel number if generated.

configure(*entity_slug: Union[EntityModel, str]*, *user_model: Optional[User] = None*, *date_draft: Optional[date] = None*, *ledger_posted: bool = False*, *ledger_name: Optional[str] = None*, *commit: bool = False*, *commit_ledger: bool = False*)

A configuration hook which executes all initial InvoiceModel setup on to the LedgerModel and all initial values of the InvoiceModel. Can only call this method once in the lifetime of a InvoiceModel.

Parameters

- **entity_slug** (*str or EntityModel*) – The entity slug or EntityModel to associate the Invoice with.
- **user_model** (*UserModel*) – The UserModel making the request to check for QuerySet permissions.
- **ledger_posted** (*bool*) – An option to mark the InvoiceModel Ledger as posted at the time of configuration. Defaults to False.
- **ledger_name** (*str*) – Optional additional InvoiceModel ledger name or description.
- **invoice_desc** (*str*) – An optional description appended to the LedgerModel name.
- **commit** (*bool*) – Saves the current InvoiceModel after being configured.
- **commit_ledger** (*bool*) – Saves the InvoiceModel's LedgerModel while being configured.

Return type
A tuple of *LedgerModel*, *InvoiceModel*

generate_invoice_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next InvoiceModel document number available. The operation will result in two additional queries if the InvoiceModel & LedgerModel is not cached in QuerySet via select_related('ledger').

Parameters
commit (*bool*) – Commits transaction into InvoiceModel.

Returns
A String, representing the generated InvoiceModel instance Document Number.

Return type
str

get_document_id()

Human-readable document number. Defaults to invoice_number.

Returns
Document Number as a String.

Return type
str

get_html_amount_due_id()

Unique amount due HTML ID.

Returns
HTML ID as a String.

Return type
str

get_html_amount_paid_id()

Unique amount paid HTML ID

Returns
HTML ID as a String.

Return type
str

`get_html_form_id()`

Unique InvoiceModel Form HTML ID.

Returns
HTML ID as a String.

Return type
str

`get_html_id()`

Unique InvoiceNumber HTML ID.

Returns
HTML ID as a String.

Return type
str

`get_item_model_qs()` → *ItemModelQuerySet*

Fetches the ItemModelQuerySet eligible to itemize.

Return type
ItemModelQuerySet

`get_itemtxs_data(queryset: Optional[ItemTransactionModelQuerySet] = None, aggregate_on_db: bool = False, lazy_agg: bool = False)` → Tuple[*ItemTransactionModelQuerySet*, Dict]

Fetches the InvoiceModel Items and aggregates the QuerySet.

Parameters
queryset – Optional pre-fetched ItemModelQueryset to use. Avoids additional DB query if provided.

Returns
A tuple

Return type
ItemTransactionModelQuerySet, dict

`get_mark_as_approved_html_id()`

InvoiceModel Mark as Approved HTML ID.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_approved_message()`

Internationalized confirmation message with Invoice Number.

Returns
Mark-as-Approved InvoiceModel confirmation message as a String.

Return type
str

`get_mark_as_approved_url()`

InvoiceModel Mark-as-Approved action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
InvoiceModel mark-as-approved action URL.

Return type
str

`get_mark_as_canceled_html_id()`

InvoiceModel Mark as Canceled HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_canceled_message()`

Internationalized confirmation message with Invoice Number.

Returns
Mark-as-Canceled InvoiceModel confirmation message as a String.

Return type
str

`get_mark_as_canceled_url()`

InvoiceModel Mark-as-Canceled action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
InvoiceModel mark-as-canceled action URL.

Return type
str

`get_mark_as_draft_html_id()`

InvoiceModel Mark as Draft HTML ID Tag.

Returns
HTML ID as a String

Return type
str

`get_mark_as_draft_message()`

Internationalized confirmation message with Invoice Number.

Returns
Mark-as-Draft InvoiceModel confirmation message as a String.

Return type
str

`get_mark_as_draft_url()`

InvoiceModel Mark-as-Draft action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
HTML ID as a String

Return type
str

get_mark_as_paid_html_id()

InvoiceModel Mark as Paid HTML ID Tag.

Returns
HTML ID as a String

Return type
str

get_mark_as_paid_message()

Internationalized confirmation message with Invoice Number.

Returns
Mark-as-Paid InvoiceModel confirmation message as a String.

Return type
str

get_mark_as_paid_url(*entity_slug: Optional[str] = None*)

InvoiceModel Mark-as-Paid action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
InvoiceModel mark-as-paid action URL.

Return type
str

get_mark_as_review_html_id()

InvoiceModel Mark as In Review HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

get_mark_as_review_message()

Internationalized confirmation message with Invoice Number.

Returns
Mark-as-Review InvoiceModel confirmation message as a String.

Return type
str

get_mark_as_review_url()

InvoiceModel Mark-as-Review action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Returns
InvoiceModel mark-as-review action URL.

Return type
str

`get_mark_as_void_html_id()`

InvoiceModel Mark as Void HTML ID Tag.

Returns
HTML ID as a String.

Return type
str

`get_mark_as_void_message()`

Internationalized confirmation message with Invoice Number.

Returns
Mark-as-Void InvoiceModel confirmation message as a String.

Return type
str

`get_mark_as_void_url()`

InvoiceModel Mark-as-Void action URL.

Parameters
entity_slug (*str*) – Entity Slug kwarg. If not provided, will result in addition DB query if select_related('ledger__entity') is not cached on QuerySet.

Return type
InvoiceModel mark-as-void action URL.

`get_migrate_state_desc()`

Description used when migrating transactions into the LedgerModel.

Returns
Description as a string.

Return type
str

`get_migration_data(queryset: Optional[ItemTransactionModelQuerySet] = None) → ItemTransactionModelQuerySet`

Fetches necessary item transaction data to perform a migration into the LedgerModel.

Parameters
queryset (ItemTransactionModelQuerySet) – Optional pre-fetched ItemModelTransactionQueryset to use. Avoids additional DB query if provided.

`get_status_action_date()`

Current status action date.

Returns
A date. i.e. If status is Approved, return date_approved. If Paid, return date_paid.

Return type
date

`get_terms_start_date()` → date

Date where InvoiceModel term start to apply.

Returns
A date which represents the start of InvoiceModel terms.

Return type
date

`is_active()`

Checks if the InvoiceModel has the potential to impact the books and produce financial statements status.

Returns
True if InvoiceModel is Active, else False.

Return type
bool

`is_approved()` → bool

Checks if the InvoiceModel is in Approved status.

Returns
True if InvoiceModel is Approved, else False.

Return type
bool

`is_canceled()` → bool

Checks if the InvoiceModel is in Canceled status.

Returns
True if InvoiceModel is Canceled, else False.

Return type
bool

`is_configured()` → bool

Determines if the accruable financial instrument is properly configured.

Returns
True if configured, else False.

Return type
bool

`is_draft()` → bool

Checks if the InvoiceModel is in Draft status.

Returns
True if InvoiceModel is Draft, else False.

Return type
bool

`is_paid()` → bool

Checks if the InvoiceModel is in Paid status.

Returns
True if InvoiceModel is Paid, else False.

Return type
bool

is_past_due() → bool

Checks if the InvoiceModel is past due.

Returns
True if InvoiceModel is past due, else False.

Return type
bool

is_review() → bool

Checks if the InvoiceModel is In Review status.

Returns
True if InvoiceModel is in Review, else False.

Return type
bool

is_void() → bool

Checks if the InvoiceModel is in Void status.

Returns
True if InvoiceModel is Void, else False.

Return type
bool

make_payment(*payment_amount: Union[Decimal, float, int]*, *payment_date: Optional[Union[datetime, date]] = None*, *commit: bool = False*, *raise_exception: bool = True*)

Makes a payment to the InvoiceModel.

Parameters

- **payment_amount** (*Decimal ot float*) – The payment amount to process.
- **payment_date** (*datetime or date.*) – Date or timestamp of the payment being applied.
- **commit** (*bool*) – If True, commits the transaction into the DB. Defaults to False.
- **raise_exception** (*bool*) – If True, raises InvoiceModelValidationError if payment exceeds amount due, else False.

Returns
True if can make payment, else False.

Return type
bool

mark_as_approved(*entity_slug*, *user_model*, *date_approved: Optional[date] = None*, *commit: bool = False*, *force_migrate: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Marks InvoiceModel as Approved.

Parameters

- **entity_slug** – Entity slug associated with the InvoiceModel. Avoids additional DB query if passed.
- **user_model** – UserModel associated with request.
- **date_approved** (*date*) – InvoiceModel approved date. Defaults to localdate().

- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.
- **force_migrate** (*bool*) – Forces migration. True if Accounting Method is Accrual.

mark_as_canceled(*date_canceled: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Mark InvoiceModel as Canceled.

Parameters

- **date_canceled** (*date*) – InvoiceModel canceled date. Defaults to localdate() if None.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_draft(*draft_date: Union[date, datetime]*, *commit: bool = False*, ***kwargs*)

Marks InvoiceModel as Draft.

Parameters

- **date_draft** (*date*) – Draft date. If None, defaults to localdate().
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_paid(*entity_slug: str*, *user_model*, *date_paid: Optional[date] = None*, *commit: bool = False*, ***kwargs*)

Marks InvoiceModel as Paid.

Parameters

- **entity_slug** (*str*) – Entity slug associated with the InvoiceModel. Avoids additional DB query if passed.
- **user_model** – UserModel associated with request.
- **date_paid** (*date*) – InvoiceModel paid date. Defaults to localdate() if None.
- **itemtxs_qs** (ItemTransactionModelQuerySet) – Pre-fetched ItemTransactionModelQuerySet. Avoids additional DB query. Validated if passed.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.

mark_as_review(*date_in_review: Optional[date] = None*, *itemtxs_qs=None*, *commit: bool = False*, ***kwargs*)

Marks InvoiceModel as In Review.

Parameters

- **date_in_review** (*date*) – InvoiceModel in review date. Defaults to localdate() if None.
- **itemtxs_qs** (ItemTransactionModelQuerySet) – Pre fetched ItemTransactionModelQuerySet to use. Avoids additional DB Query if previously fetched.
- **commit** (*bool*) – Commits transaction into the Database. Defaults to False.
- **raise_exception** (*bool*) – Raises InvoiceModelValidationError if InvoiceModel cannot be marked as in review. Defaults to True.

mark_as_void(*entity_slug: str*, *user_model*, *date_void: Optional[Union[datetime, date]] = None*, *commit: bool = False*, ***kwargs*)

Marks InvoiceModel as Void. When mark as void, all transactions associated with InvoiceModel are reversed as of the void date.

Parameters

- **entity_slug** (*str*) – Entity slug associated with the InvoiceModel. Avoids additional DB query if passed.

- **user_model** – UserModel associated with request.
- **date_void** (*date*) – InvoiceModel void date. Defaults to localdate() if None.
- **commit** (*bool*) – Commits transaction into DB. Defaults to False.

migrate_itemtxs(*itemtxs: Dict*, *operation: str*, *commit: bool = False*)

Migrates a predefined item transaction list.

Parameters

- **itemtxs** (*dict*) – A dictionary where keys are the document number (invoice/bill number, etc) and values are a dictionary:
- **operation** (*str*) – A choice of ITEMIZE_REPLACE, ITEMIZE_APPEND, ITEMIZE_UPDATE
- **commit** (*bool*) – If True, commits transaction into the DB. Default to False

Returns
A list of ItemTransactionModel appended or created.

Return type
list

save(***kwargs*)

Save method for InvoiceModel. Results in a DB query if invoice number has not been generated and the InvoiceModel is eligible to generate a invoice_number.

update_amount_due(*itemtxs_qs: Optional[*ItemTransactionModelQuerySet*] = None*) → *ItemTransactionModelQuerySet*

Updates the InvoiceModel amount due.

Parameters
itemtxs_qs (ItemTransactionModelQuerySet) – Optional pre-fetched ItemTransactionModelQuerySet. Avoids additional DB if provided. Queryset is validated if provided.

Returns
Newly fetched of previously fetched ItemTransactionModelQuerySet if provided.

Return type
ItemTransactionModelQuerySet

validate_itemtxs_qs(*queryset:* ItemTransactionModelQuerySet)

Validates that the entire ItemTransactionModelQuerySet is bound to the InvoiceModel.

Parameters
queryset (ItemTransactionModelQuerySet) – ItemTransactionModelQuerySet to validate.

class django_ledger.models.invoice.**InvoiceModelManager**(**args*, ***kwargs*)

A custom defined InvoiceModel Manager that will act as an interface to handling the DB queries to the InvoiceModel. The default "get_queryset" has been overridden to refer the custom defined "InvoiceModelQuerySet"

for_entity(*entity_slug*, *user_model*) → *InvoiceModelQuerySet*

Returns a QuerySet of InvoiceModels associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.

- **user_model** – The request UserModel to check for privileges.

Returns
A Filtered InvoiceModelQuerySet.

Return type
InvoiceModelQuerySet

`get_queryset()`

Return a new QuerySet object. Subclasses can override this method to customize the behavior of the Manager.

`class django_ledger.models.invoice.InvoiceModelQuerySet(model=None, query=None, using=None, hints=None)`

A custom defined QuerySet for the InvoiceModel. This implements multiple methods or queries that we need to run to get a status of Invoices raised by the entity. For example, We might want to have list of invoices which are paid, unpaid, due , overDue, approved or in draft stage. All these separate functions will assist in making such queries and building customized reports.

`active()`

Active invoices are those that are approved or paid, which have impacted or have the potential to impact the Entity's Ledgers.

Returns
Returns a QuerySet of active invoices only.

Return type
InvoiceModelQuerySet

`approved()`

Approved invoices are those that have been reviewed and are expected to be paid before the due date.

Returns
Returns a QuerySet of approved invoices only.

Return type
InvoiceModelQuerySet

`canceled()`

Canceled invoices are those that are discarded during the draft or in review status. These invoices never had an impact on the books.

Returns
Returns a QuerySet of canceled invoices only.

Return type
InvoiceModelQuerySet

`draft()`

Default status of any invoice that is created. Draft invoices do not impact the Ledger.

Returns
Returns a QuerySet of draft invoices only.

Return type
InvoiceModelQuerySet

`in_review()`

In review invoices are those that need additional review or approvals before being approved. Draft invoices do not impact the Ledger.

Returns
Returns a QuerySet of in review invoices only.

Return type
InvoiceModelQuerySet

`overdue()`

Overdue invoices are those which due date is in the past.

Returns
Returns a QuerySet of overdue invoices only.

Return type
InvoiceModelQuerySet

`paid()`

Paid invoices are those that have received 100% of the amount due.

Returns
Returns a QuerySet of paid invoices only.

Return type
InvoiceModelQuerySet

`unpaid()`

Unpaid invoices are those that are approved but have not received 100% of the amount due. Equivalent to approved().

Returns
Returns a QuerySet of paid invoices only.

Return type
InvoiceModelQuerySet

`void()`

Void invoices are those that where rolled back after being approved. Void invoices rollback all transactions by creating a new set of transactions posted on the date_void.

Returns
Returns a QuerySet of void invoices only.

Return type
InvoiceModelQuerySet

exception django_ledger.models.invoice.**InvoiceModelValidationError**(*message*, *code=None*, *params=None*)

14.17 Customer Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>
- Pranav P Tulshyan <ptulshyan77@gmail.com>

A Customer refers to the person or entity that buys product and services. When issuing Invoices, a Customer must be created before it can be assigned to the InvoiceModel. Only customers who are active can be assigned to new Invoices.

class django_ledger.models.customer.**CustomerModel**(**args*, ***kwargs*)

Base Customer Model Implementation

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.customer.**CustomerModelAbstract**(**args*, ***kwargs*)

This is the main abstract class which the CustomerModel database will inherit from. The CustomerModel inherits functionality from the following MixIns:

1. *ContactInfoMixIn*
2. *CreateUpdateMixIn*

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

entity

The EntityModel associated with this Customer.

Type
EntityModel

customer_name

A string representing the name the customer uses to do business with the EntityModel.

Type
str

customer_number

A unique, auto-generated human-readable number which identifies the customer within the EntityModel.

Type
str

description

A text field to capture the description about the customer.

Type
str

active

We can set any customer code to be active or inactive. Defaults to True.

Type
bool

hidden

Hidden CustomerModels don't show on the UI. Defaults to False.

Type
bool

additional_info

Any additional information about the customer, stored as a JSON object using a JSONField.

Type
dict

can_generate_customer_number() → bool

Determines if the CustomerModel can be issued a Customer Number. CustomerModels have a unique sequential number, which is unique for each EntityMode/CustomerModel.

Returns
True if customer model can be generated, else False.

Return type
bool

clean()

Custom defined clean method that fetches the next customer number if not yet fetched. Additional validation may be provided.

generate_customer_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next Customer Number available.

Parameters
commit (*bool*) – Commits transaction into CustomerModel. Defaults to False.

Returns
A String, representing the current CustomerModel instance Document Number.

Return type
str

save(***kwargs*)

Custom-defined save method that automatically fetches the customer number if not present.

Parameters
kwargs – Keywords passed to the super().save() method of the CustomerModel.

class django_ledger.models.customer.**CustomerModelManager**(**args*, ***kwargs*)

A custom defined CustomerModelManager that will act as an interface to handling the DB queries to the CustomerModel.

for_entity(*entity_slug*, *user_model*) → *CustomerModelQueryset*

Fetches a QuerySet of CustomerModel associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> customer_model_qs = CustomerModel.objects.for_entity(user_model=request_
↪user, entity_slug=slug)
```

Returns
A filtered CustomerModel QuerySet.

Return type
CustomerModelQueryset

for_user(*user_model*)

Fetches a QuerySet of BillModels that the UserModel as access to. May include BillModels from multiple Entities.

The user has access to bills if:

1. Is listed as Manager of Entity.
2. Is the Admin of the Entity.

Parameters
user_model – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> customer_model_qs = CustomerModel.objects.for_user(user_model=request_user)
```

class django_ledger.models.customer.**CustomerModelQueryset**(*model=None*, *query=None*, *using=None*, *hints=None*)

A custom defined QuerySet for the CustomerModel. This implements multiple methods or queries needed to get a filtered QuerySet based on the CustomerModel status. For example, we might want to have list of Customers that are active or hidden. All these separate functions will assist in making such queries and building customized reports.

active() → QuerySet

Active customers can be assigned to new Invoices and show on dropdown menus and views.

Returns
A QuerySet of active Customers.

Return type
CustomerModelQueryset

hidden() → QuerySet

Hidden customers do not show on dropdown menus, but may be used via APIs or any other method that does not involve the UI.

Returns
A QuerySet of hidden Customers.

Return type
CustomerModelQueryset

inactive() → QuerySet

Active customers can be assigned to new Invoices and show on dropdown menus and views. Marking CustomerModels as inactive can help reduce Database load to populate select inputs and also inactivate CustomerModels that are not relevant to the Entity anymore. Also, it makes de UI cleaner by not populating unnecessary choices.

Returns
A QuerySet of inactive Customers.

Return type
CustomerModelQueryset

visible() → QuerySet

Visible customers show on dropdown menus and views. Visible customers are active and not hidden.

Returns
A QuerySet of visible Customers.

Return type
CustomerModelQueryset

14.18 Vendor Model

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

A Vendor refers to the person or entity that provides products and services to the business for a fee. Vendors are an integral part of the billing process as they are the providers of goods and services for the business.

Vendors can be flagged as active/inactive or hidden. Vendors who no longer conduct business with the EntityModel, whether temporarily or indefinitely may be flagged as inactive (i.e. active is False). Hidden Vendors will not show up as an option in the UI, but can still be used programmatically (via API).

class django_ledger.models.vendor.**VendorModel**(**args*, ***kwargs*)

Base Vendor Model Implementation

exception DoesNotExist

exception MultipleObjectsReturned

class django_ledger.models.vendor.**VendorModelAbstract**(**args*, ***kwargs*)

This is the main abstract class which the VendorModel database will inherit from. The VendorModel inherits functionality from the following MixIns:

1. *ContactInfoMixIn*
2. *BankAccountInfoMixIn*
3. *TaxInfoMixIn*
4. *CreateUpdateMixIn*

uuid

This is a unique primary key generated for the table. The default value of this field is uuid4().

Type
UUID

`entity_model`

The EntityModel associated with this Vendor.

Type
EntityModel

`vendor_name`

A string representing the name the customer uses to do business with the EntityModel.

Type
str

`vendor_number`

A unique, auto-generated human-readable number which identifies the vendor within the EntityModel.

Type
str

`description`

A text field to capture the description about the vendor.

Type
str

`active`

We can set any vendor to be active or inactive. Defaults to True.

Type
bool

`hidden`

Hidden VendorModel don't show on the UI. Defaults to False.

Type
bool

`additional_info`

Any additional information about the vendor, stored as a JSON object using a JSONField.

Type
dict

`can_generate_vendor_number()` → bool

Determines if the VendorModel can be issued a Vendor Number. VendorModel have a unique sequential number, which is unique for each EntityModel/VendorModel.

Returns
True if vendor number can be generated, else False.

Return type
bool

`clean()`

Custom defined clean method that fetches the next vendor number if not yet fetched. Additional validation may be provided.

generate_vendor_number(*commit: bool = False*) → str

Atomic Transaction. Generates the next Vendor Number available.

Parameters
commit (*bool*) – Commits transaction into VendorModel. Defaults to False.

Returns
A String, representing the current VendorModel instance document number.

Return type
str

save(***kwargs*)

Custom-defined save method that automatically fetches the vendor number if not present.

Parameters
kwargs – Keywords passed to the super().save() method of the VendorModel.

class django_ledger.models.vendor.**VendorModelManager**(**args*, ***kwargs*)

Custom defined VendorModel Manager, which defines many methods for initial query of the Database.

for_entity(*entity_slug*, *user_model*) → *VendorModelQuerySet*

Fetches a QuerySet of VendorModel associated with a specific EntityModel & UserModel. May pass an instance of EntityModel or a String representing the EntityModel slug.

Parameters

- **entity_slug** (*str or* EntityModel) – The entity slug or EntityModel used for filtering the QuerySet.
- **user_model** – Logged in and authenticated django UserModel instance.

Examples

```
>>> request_user = request.user
>>> slug = kwargs['entity_slug'] # may come from request kwargs
>>> vendor_model_qs = VendorModel.objects.for_entity(user_model=request_user,␣
↪entity_slug=slug)
```

Returns
A filtered VendorModel QuerySet.

Return type
VendorModelQuerySet

class django_ledger.models.vendor.**VendorModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Custom defined VendorModel QuerySet.

active() → QuerySet

Active vendors can be assigned to new bills and show on dropdown menus and views.

Returns
A QuerySet of active Vendors.

Return type
VendorModelQuerySet

hidden() → QuerySet

Hidden vendors do not show on dropdown menus, but may be used via APIs or any other method that does not involve the UI.

Returns
A QuerySet of hidden Vendors.

Return type
VendorModelQuerySet

inactive() → QuerySet

Active vendors can be assigned to new bills and show on dropdown menus and views. Marking VendorModels as inactive can help reduce Database load to populate select inputs and also inactivate VendorModels that are not relevant to the Entity anymore. Also, it makes de UI cleaner by not populating unnecessary choices.

Returns
A QuerySet of inactive Vendors.

Return type
VendorModelQuerySet

visible() → QuerySet

Visible vendors show on dropdown menus and views. Visible vendors are active and not hidden.

Returns
A QuerySet of visible Vendors.

Return type
VendorModelQuerySet

exception django_ledger.models.vendor.**VendorModelValidationError**(*message*, *code=None*, *params=None*)

14.19 MixIns

Django Ledger created by Miguel Sanda <msanda@arrobalytics.com>. Copyright© EDMA Group Inc licensed under the GPLv3 Agreement.

Contributions to this module:

- Miguel Sanda <msanda@arrobalytics.com>

This module implements the different model MixIns used on different Django Ledger Models to implement common functionality.

class django_ledger.models.mixins.**AccrualMixIn**(**args*, ***kwargs*)

Implements functionality used to track accruable financial instruments to a base Django Model. Examples of this include bills and invoices expenses/income, that depending on the Entity's accrual method, may be recognized on the Income Statement differently.

amount_due

The total amount due of the financial instrument.

Type
Decimal

amount_paid

The total amount paid or settled.

Type
Decimal

amount_receivable

The total amount allocated to Accounts Receivable based on the progress.

Type
Decimal

amount_unearned

The total amount allocated to Accounts Payable based on the progress.

Type
Decimal

amount_earned

The total amount that is recognized on the earnings based on progress.

accrue

If True, the financial instrument will follow the Accrual Method of Accounting, otherwise it will follow the Cash Method of Accounting. Defaults to the EntityModel preferred method of accounting.

Type
bool

progress

A decimal number representing the amount of progress of the financial instrument. Value is between 0.00 and 1.00.

Type
Decimal

ledger

The LedgerModel associated with the Accruable financial instrument.

Type
LedgerModel

cash_account

The AccountModel used to track cash payments to the financial instrument. Must be of role ASSET_CA_CASH.

Type
AccountModel

prepaid_account

The AccountModel used to track receivables to the financial instrument. Must be of role ASSET_CA_PREPAID.

Type
AccountModel

unearned_account

The AccountModel used to track receivables to the financial instrument. Must be of role LIABILITY_CL_DEFERRED_REVENUE.

Type
AccountModel

can_migrate() → bool

Determines if the Accruable financial instrument can be migrated to the books. Results in additional Database query if 'ledger' field is not pre-fetch on QuerySet.

Returns
True if can migrate, else False.

Return type
bool

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

get_amount_cash() → Union[Decimal, float]

Determines the impact to the EntityModel cash balance based on the financial instrument debit or credit configuration. i.e, Invoices are debit financial instrument because payments to invoices increase cash.

Returns
Financial instrument progress as a percent.

Return type
float

get_amount_earned() → Union[Decimal, float]

Determines the impact to the EntityModel earnings based on financial instrument progress.

Returns
Financial instrument amount earned.

Return type
float or Decimal

get_amount_open() → Union[Decimal, float]

Determines the open amount left to be progressed.

Returns
Financial instrument amount open.

Return type
float or Decimal

get_amount_prepaid() → Union[Decimal, float]

Determines the impact to the EntityModel Accounts Receivable based on financial instrument progress.

Returns
Financial instrument amount prepaid.

Return type
float or Decimal

get_amount_unearned() → Union[Decimal, float]

Determines the impact to the EntityModel Accounts Payable based on financial instrument progress.

Returns
Financial instrument amount unearned.

Return type
float or Decimal

get_progress() → Union[Decimal, float]

Determines the progress amount based on amount due, amount paid and accrue field.

Returns
Financial instrument progress as a Decimal.

Return type
Decimal

get_progress_percent() → float

Determines the progress amount as percent based on amount due, amount paid and accrue field.

Returns
Financial instrument progress as a percent.

Return type
float

get_state(*commit: bool = False*)

Determines the new state of the financial instrument based on progress.

Parameters
commit (*bool*) – Commits the new financial instrument state into the model.

Returns
A dictionary with new amount_paid, amount_receivable, amount_unearned and amount_earned as keys.

Return type
dict

get_tx_type(*acc_bal_type: dict*, *adjustment_amount: Decimal*)

Determines the transaction type associated with an increase/decrease of an account balance of the financial instrument.

Parameters

- **acc_bal_type** – The balance type of the account to be adjusted.
- **adjustment_amount** (*Decimal*) – The adjustment, whether positive or negative.

Returns
The transaction type of the account adjustment.

Return type
str

is_configured() → bool

Determines if the accruable financial instrument is properly configured.

Returns
True if configured, else False.

Return type
bool

is_posted()

Determines if the accruable financial instrument is posted. Results in additional Database query if 'ledger' field is not pre-fetch on QuerySet.

Returns
True if posted, else False.

Return type
bool

lock_ledger(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Convenience method to lock the LedgerModel associated with the Accruable financial instrument.

Parameters

- **commit** (*bool*) – Commits the transaction in the database. Defaults to False.
- **raise_exception** (*bool*) – If True, raises ValidationError if LedgerModel already locked.

migrate_state(*user_model*, *entity_slug: str*, *itemtxs_qs: Optional[QuerySet] = None*, *force_migrate: bool = False*, *commit: bool = True*, *void: bool = False*, *je_timestamp: Optional[Union[date, datetime, str]] = None*, *raise_exception: bool = True*, ***kwargs*)

Migrates the current Accruable financial instrument into the books. The main objective of the migrate_state method is to determine the JournalEntry and TransactionModels necessary to accurately reflect the financial instrument state in the books.

Parameters

- **user_model** – The Django User Model.
- **entity_slug** (*str*) – The EntityModel slug.
- **itemtxs_qs** (ItemTransactionModelQuerySet) – The pre-fetched ItemTransactionModelQuerySet containing the item information associated with the financial element migration. If provided, will avoid additional database query.
- **force_migrate** (*bool*) – Forces migration of the financial instrument bypassing the can_migrate() check.
- **commit** (*bool*) – If True the migration will be committed in the database. Defaults to True.
- **void** (*bool*) – If True, the migration will perform a VOID actions of the financial instrument.
- **je_timestamp** (*date*) – The JournalEntryModel date to be used for this migration.
- **raise_exception** (*bool*) – Raises ValidationError if migration is not allowed. Defaults to True.

Returns
A tuple of the ItemTransactionModel and the Digest Result from IOMixIn.

Return type
tuple

post_ledger(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Convenience method to post the LedgerModel associated with the Accruable financial instrument.

Parameters

- **commit** (*bool*) – Commits the transaction in the database. Defaults to False.
- **raise_exception** (*bool*) – If True, raises ValidationError if LedgerModel already locked.

classmethod split_amount(*amount: Union[Decimal, float]*, *unit_split: Dict*, *account_uuid: UUID*, *account_balance_type: str*) → Dict

Splits an amount into different proportions representing the unit splits. Makes sure that 100% of the amount is numerically allocated taking into consideration decimal points.

Parameters

- **amount** (*Decimal or float*) – The amount to be split.
- **unit_split** (*dict*) – A dictionary with information related to each unit split and proportions.
- **account_uuid** (*UUID*) – The AccountModel UUID associated with the splits.
- **account_balance_type** (*str*) – The AccountModel balance type to determine whether to perform a credit or a debit.

Returns
A dictionary with the split information.

Return type
dict

unlock_ledger(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Convenience method to un-lock the LedgerModel associated with the Accruable financial instrument.

Parameters

- **commit** (*bool*) – Commits the transaction in the database. Defaults to False.
- **raise_exception** (*bool*) – If True, raises ValidationError if LedgerModel already locked.

unpost_ledger(*commit: bool = False*, *raise_exception: bool = True*, ***kwargs*)

Convenience method to un-lock the LedgerModel associated with the Accruable financial instrument.

Parameters

- **commit** (*bool*) – Commits the transaction in the database. Defaults to False.
- **raise_exception** (*bool*) – If True, raises ValidationError if LedgerModel already locked.

update_state(*state: Optional[Dict] = None*)

Updates the state on the financial instrument.

Parameters
state (*dict*) – Optional user provided state to use.

void_state(*commit: bool = False*) → Dict

Determines the VOID state of the financial instrument.

Parameters
commit (*bool*) – Commits the new financial instrument state into the model.

Returns
A dictionary with new amount_paid, amount_receivable, amount_unearned and amount_earned as keys.

Return type
dict

class django_ledger.models.mixins.**BankAccountInfoMixIn**(**args*, ***kwargs*)

Implements functionality used to add bank account details to base Django Models.

account_number

The Bank Account number. Only Digits are allowed. Max 30 digists.

Type
str

routing_number

Routing number for the concerned bank account. Also called as 'Routing Transit Number (RTN)'. Max 30 digists.

Type
str

aba_number

The American Bankers Association Number assigned to each bank.

Type
str

account_type

A choice of ACCOUNT_TYPES. Each account will have to select from the available choices Checking, Savings.

Type
str

swift_number

SWIFT electronic communications network number of the bank institution.

Type
str

class django_ledger.models.mixins.**ContactInfoMixIn**(**args*, ***kwargs*)

Implements a common set of fields used to document contact information.

address_1

A string used to document the first line of an address. Mandatory. Max length is 70.

Type
str

address_2

A string used to document the first line of an address. Optional.

Type
str

city

A string used to document the city. Optional.

Type
str

state

A string used to document the State of Province. Optional.

Type
str

zip_code

A string used to document the ZIP code. Optional

Type
str

country

A string used to document the country. Optional.

Type
str

email

A string used to document the contact email. Uses django's EmailField for validation.

Type
str

website

A string used to document the contact website. Uses django's URLField for validation.

Type
str

phone

A string used to document the contact phone.

Type
str

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

class django_ledger.models.mixins.**CreateUpdateMixIn**(**args*, ***kwargs*)

Implements a created and an updated field to a base Django Model.

created

A created timestamp. Defaults to now().

Type
datetime

updated

An updated timestamp used to identify when models are updated.

Type
str

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

exception django_ledger.models.mixins.**ItemizeError**(*message*, *code=None*, *params=None*)

class django_ledger.models.mixins.**ItemizeMixIn**(**args*, ***kwargs*)

can_migrate_itemtxs() → bool

Checks if item transaction list can be migrated.

Return type
bool

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

get_item_model_qs()

Fetches the ItemModelQuerySet eligible to itemize.

Return type
ItemModelQuerySet

get_itemtxs_data(*queryset=None*, *aggregate_on_db: bool = False*, *lazy_agg: bool = False*)

Fetches the ItemTransactionModelQuerySet associated with the model.

Parameters

- **queryset** (ItemTransactionModelQuerySet) – Pre-fetched ItemTransactionModelQuerySet. Validated if provided.
- **aggregate_on_db** (*bool*) – If True, performs aggregation at the DB layer. Defaults to False.
- **lazy_agg** (*bool*) – If True, performs queryset aggregation metrics. Defaults to False.

Returns
ItemModelQuerySet, dict

Return type
tuple

migrate_itemtxs(*itemtxs: Dict*, *operation: str*, *commit: bool = False*)

Migrates a predefined item transaction list.

Parameters

- **itemtxs** (*dict*) – A dictionary where keys are the document number (invoice/bill number, etc) and values are a dictionary:
- **operation** (*str*) – A choice of ITEMIZE_REPLACE, ITEMIZE_APPEND, ITEMIZE_UPDATE
- **commit** (*bool*) – If True, commits transaction into the DB. Default to False

Returns
A list of ItemTransactionModel appended or created.

Return type
list

validate_itemtxs(*itemtxs*)

Validates the provided item transaction list.

Parameters
itemtxs (*dict*) – Item transaction list to replace/aggregate.

validate_itemtxs_qs()

Validates that the provided item transaction list is valid.

class django_ledger.models.mixins.**LoggingMixIn**

Implements functionality used to add logging capabilities to any python class. Useful for production and or testing environments.

class django_ledger.models.mixins.**MarkdownNotesMixIn**(**args*, ***kwargs*)

Implements functionality used to add a Mark-Down notes to a base Django Model.

markdown_notes

A string of text representing the mark-down document.

Type
str

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

notes_html()

Compiles the markdown_notes field into html.

Returns
Compiled HTML document as a string.

Return type
str

class django_ledger.models.mixins.**PaymentTermsMixIn**(**args*, ***kwargs*)

Implements functionality used to track dates relate to various payment terms. Examples of this include tracking bills and invoices that are due on receipt, 30, 60 or 90 days after they are approved.

terms

A choice of TERM_CHOICES that determines the payment terms.

Type
str

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

due_in_days() → Optional[int]

Determines how many days until the due date.

Returns
Days as integer.

Return type
int

get_terms_net_90_plus() → int

Determines the number of days for 90+ days terms of payment.

Returns
The date when terms of payment starts.

Return type
date

get_terms_start_date() → date

Determines the start date for the terms of payment.

Returns
The date when terms of payment starts.

Return type
date

`get_terms_timedelta()` → timedelta

Calculates a timedelta relative to the terms start date.

Returns
Timedelta relative to terms start date.

Return type
timedelta

`get_terms_timedelta_days()` → int

Determines the number of days from the terms start date.

Returns
The number of days as integer.

Return type
int

`net_due_group()`

Determines the group where the financial instrument falls based on the number of days until the due date.

Returns
The terms group as a string.

Return type
str

`class django_ledger.models.mixins.SlugNameMixIn(*args, **kwargs)`

Implements a slug field and a name field to a base Django Model.

`slug`

A unique slug field to use as an index. Validates that the slug is at least 10 characters long.

Type
str

`name`

A human-readable name for display purposes. Maximum 150 characters.

Type
str

`clean()`

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`class django_ledger.models.mixins.TaxCollectionMixIn(*args, **kwargs)`

Implements functionality used to add tax collection rates and or withholding to a base Django Model. This field may be used to set a pre-defined withholding rate to a financial instrument, customer, vendor, etc.

`sales_tax_rate`

The tax rate as a float. A Number between 0.00 and 1.00.

Type
float

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

class django_ledger.models.mixins.**TaxInfoMixIn**(**args*, ***kwargs*)

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Z