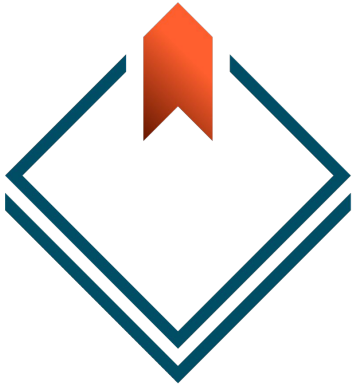


DJANGO LEDGER

< Object Oriented Accounting Engine />

Project Introduction & Roadmap

Miguel Sanda

Project Manager & Development Lead

arrobalytics.com

Intro

Miguel Sanda, M.Sc, MBA, PMP.

Project Manager & Lead Developer

Sr. Data Scientist. 15 Years experience in Project Management, Finance, Big Data & Full Stack Development. B.S. Civil Engineering, M. Sc. Project Management & MBA. Entrepreneur & Investor.

Collaborators & Sponsors:

Albert Salazar, Beechwood Financial.
https://www.linkedin.com/in/albert-salazar-8733427/

Michael Noel, CPA.
http://www.linkedin.com/in/michael-noel-cpa

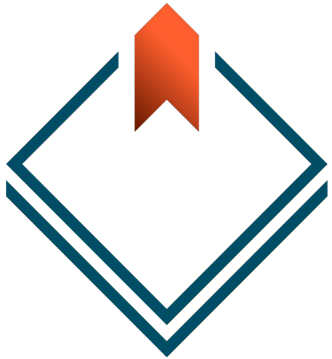
What is Django?

"Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source."

http://www.djangoproject.com

What is Django Ledger?

A plugable, extensible and scalable bookkeeping & financial analysis engine written in Python for the Django Framework.



DJANGO
LEDGER

< Object Oriented Accounting Engine />

Why Django Ledger?

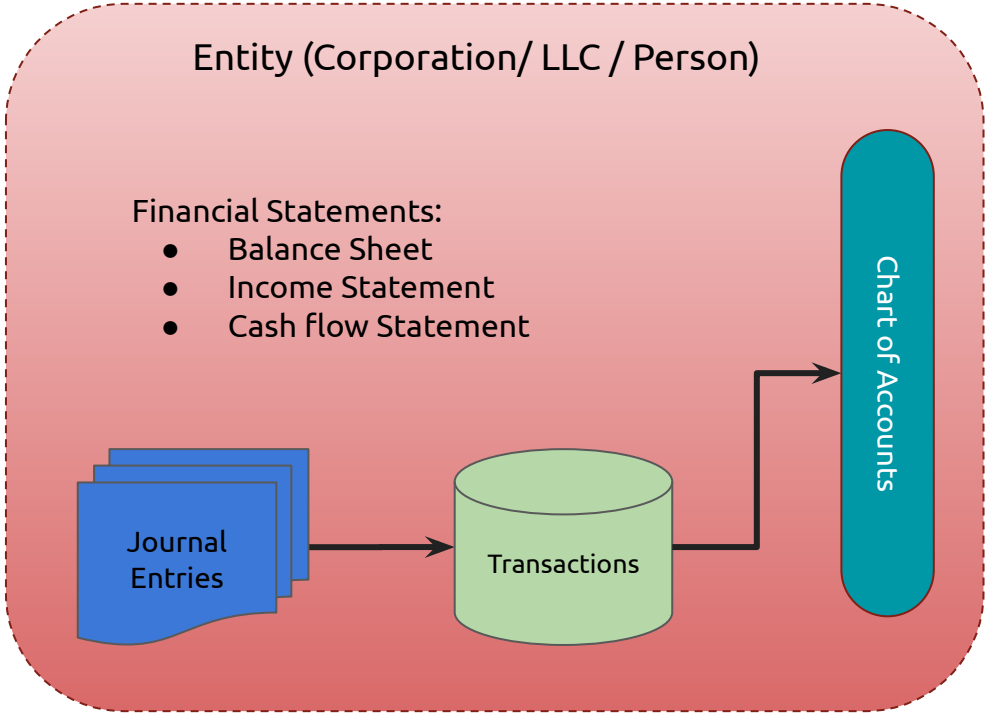
- Make accounting and money management open & transparent.
- Close the gap between developers and finance.
- Provide an accounting engine that is:
  - Out of the box, simple and ready to go for individuals and small business with limited resources or understanding of accounting.
  - Abstract enough to allow for corporate customization yet resilient enough to enforce the accounting controls needed at a corporate level.
- Provide out of the box, simple and extensible bookkeeping functionality to individuals and small businesses.
- Provide a solid, reliable, embeddable and scalable accounting engine to power financially-driven applications.

Why Python / Why Django?

- Simple and powerful programming language that attracts talent and domain experts.
- Python has the lowest time to market. Easy and fast to get started, easy to bind to other languages.

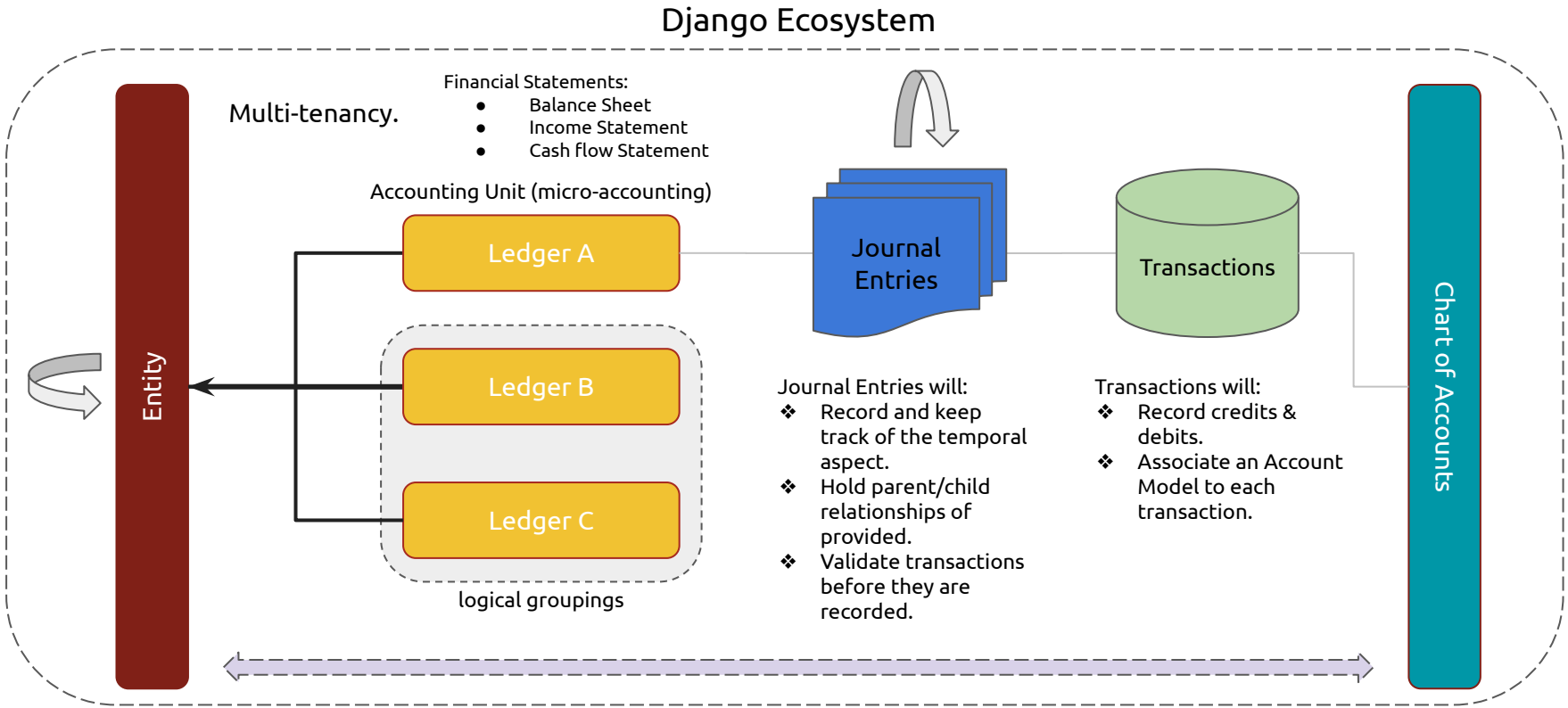
- Ability to focus more on the application and not on the framework.
- Fast, secure and scalable foundation to develop web applications.
- Proven winner, robust and battle tested.
- Easy to deploy and widely accepted.
- Out-of-the box Authorization and Authentication (accounting systems require separation of duties).
- Database agnostic (relational) and automatic migrations.
- Extensive documentation and use cases.
- Great community.

Traditional Accounting Systems



- All transactions and data is encapsulated within an Entity.
- Multi tenancy and collaborative functionality have limited support.
- Industry specific customizations are complicated.
- No standardized chart of accounts roles.
- High switching costs and typically require a subscription.

The Core Model



How Models Work in Django Ledger?

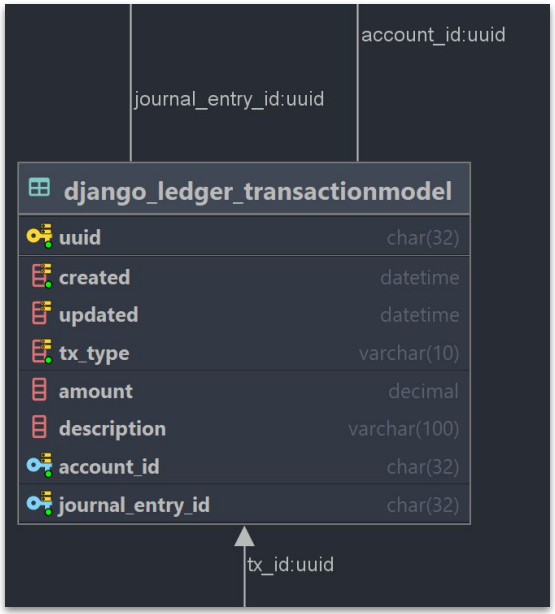
django_ledger/models/transactions.py

The Transaction Model implements the most basic Accounting system mutation: Changing the running balance of a single account with a credit or a debit.

```python
class TransactionModelAbstract(CreateUpdateMixIn):
    CREDIT = 'credit'
    DEBIT = 'debit'

    TX_TYPE = [
        (CREDIT, _('Credit')),
        (DEBIT, _('Debit'))
    ]

    uuid = models.UUIDField(default=uuid4, editable=False, primary_key=True)
    tx_type = models.CharField(max_length=10, choices=TX_TYPE, verbose_name=_('Tx Type'))
    journal_entry = models.ForeignKey('django_ledger.JournalEntryModel',
                                      related_name='txs',
                                      verbose_name=_('Journal Entry'),
                                      help_text=_('Journal Entry to be associated with this transaction.'),
                                      on_delete=models.CASCADE)
    account = models.ForeignKey('django_ledger.AccountModel',
                                related_name='txs',
                                verbose_name=_('Account'),
                                help_text=_('Account from Chart of Accounts to be associated with this transaction.'),
                                on_delete=models.CASCADE)
    amount = models.DecimalField(decimal_places=2,
                                 max_digits=20,
                                 null=True,
                                 blank=True,
                                 verbose_name=_('Amount'),
                                 help_text=_('Account of the transaction.'),
                                 validators=[MinValueValidator(0)])
    description = models.CharField(max_length=100, null=True, blank=True,
                                   verbose_name=_('Tx Description'),
                                   help_text=_('A description to be included with this individual transaction'))
    objects = TransactionModelAdmin()
```

DB Migration

django_ledger/ migrations/.py*



The Django ORM, Views and HTML Rendering.

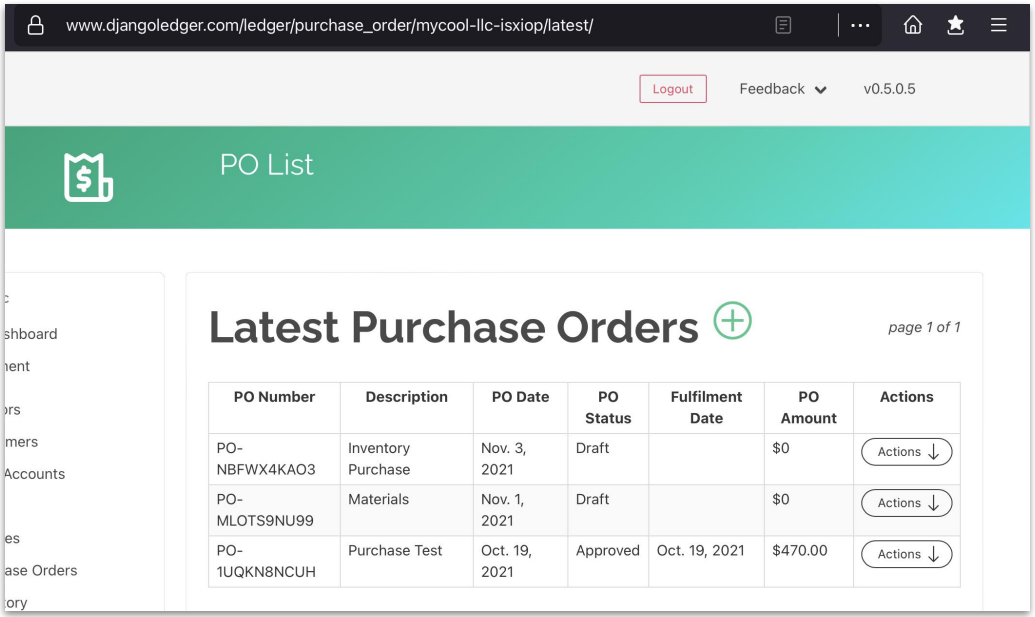
django_ledger/views/purchase_order.py

```python
class PurchaseOrderModelListView(LoginRequiredMixIn, ArchiveIndexView):
    template_name = 'django_ledger/po_list.html'
    context_object_name = 'po_list'
    PAGE_TITLE = _('PO List')
    date_field = 'po_date'
    paginate_by = 10
    paginate_orphans = 2
    allow_empty = True
    extra_context = {
        'page_title': PAGE_TITLE,
        'header_title': PAGE_TITLE,
        'header_subtitle_icon': 'uil:bill'
    }

    def get_queryset(self):
        return PurchaseOrderModel.objects.for_entity(
            entity_slug=self.kwargs['entity_slug'],
            user_model=self.request.user
        ).order_by('-po_date')
```

HTML template to render.

View context.

DB Query.



Form Validation & Rendering

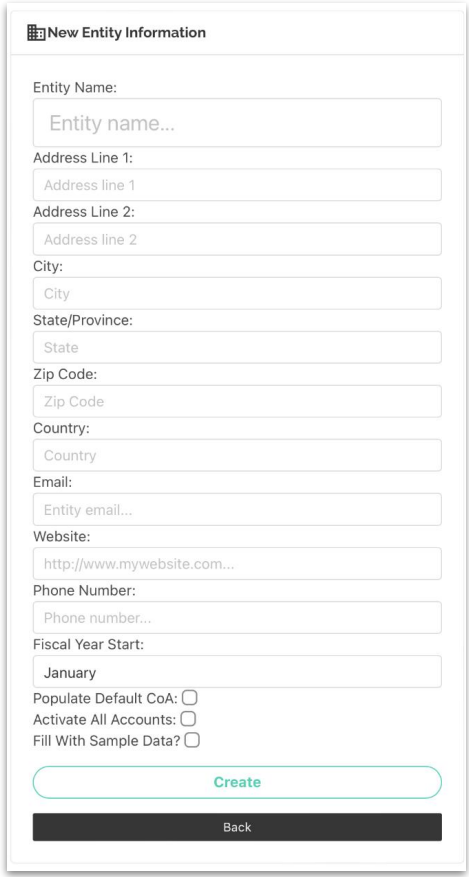
django_ledger/forms/entity.py

```python
class EntityModelCreateForm(ModelForm):
    default_coa = BooleanField(required=False, initial=False, label=_('Populate Default CoA'))
    activate_all_accounts = BooleanField(required=False, initial=False, label=_('Activate All Accounts'))
    generate_sample_data = BooleanField(required=False, initial=False, label=_('Fill With Sample Data?'))

    def clean_name(self):
        name = self.cleaned_data.get('name')
        if not name:
            raise ValidationError(_('Please provide a valid name for new Entity.'))
        if len(name) < 3:
            raise ValidationError(_('Looks like this entity name is too short...'))
        return name

    def clean(self):
        populate_coa = self.cleaned_data['default_coa']
        activate_all_accounts = self.cleaned_data['activate_all_accounts']
        sample_data = self.cleaned_data['generate_sample_data']

        if sample_data and not all([
            populate_coa,
            activate_all_accounts
        ]):
            raise ValidationError(f'Filling sample data requires using default CoA and activate all accounts
        validate_cszc(self.cleaned_data)

    class Meta:
        model = EntityModel
        fields = [
            'name',
            'address_1',
            'address_2',
            'city',
            'state',
            'zip_code',
            'country',
            'email',
```

Additional field validation...

Inheritance from EntityModel...

A fully functional HTML form represented by a simple Python class.



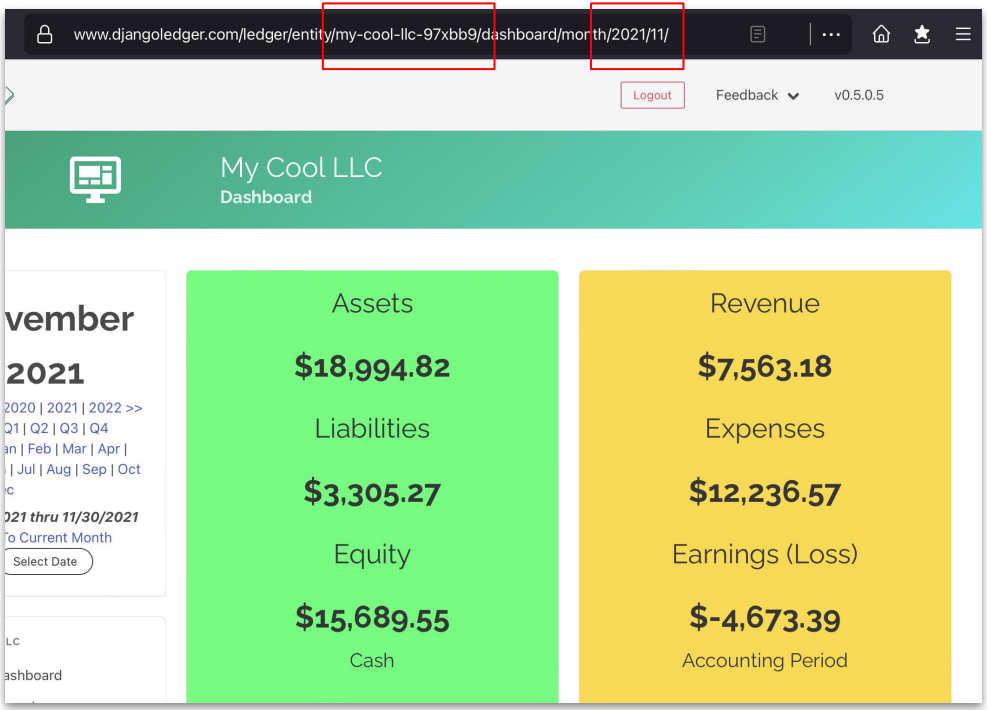
ARROBALYTICS

The URLs and Routing

django_ledger/urls/entity.py

```python
from django.urls import path

from django_ledger import views

urlpatterns = [
    # Entity Views ----
    path('list/', views.EntityModelListView.as_view(), name='entity-list'),
    path('create/', views.EntityModelCreateView.as_view(), name='entity-create'),

    # DASHBOARD Views...
    path('<slug:entity_slug>/dashboard/',
         views.EntityModelDetailView.as_view(),
         name='entity-dashboard'),
    path('<slug:entity_slug>/dashboard/year/<int:year>/',
         views.FiscalYearEntityModelDashboardView.as_view(),
         name='entity-dashboard-year'),
    path('<slug:entity_slug>/dashboard/quarter/<int:year>/<int:quarter>/',
         views.QuarterlyEntityDashboardView.as_view(),
         name='entity-dashboard-quarter'),
    path('<slug:entity_slug>/dashboard/month/<int:year>/<int:month>/',
         views.MonthlyEntityDashboardView.as_view(),
         name='entity-dashboard-month'),
    path('<slug:entity_slug>/dashboard/date/<int:year>/<int:month>/<int:day>/',
         views.DateEntityDashboardView.as_view(),
         name='entity-dashboard-date'),
```

entity_slug year/month



The IO MixIn

django_ledger/io/io_mixin.py

- The IO MixIn is the entry point responsible for requesting transaction data from the database and applying all accounting rules.
- Two digest phases:
  - Database Digest: Pushes the majority of calculations to the database layer and provides an aggregate of account balances with additional details if requested.
  - Python Digest: Only the necessary data is pulled into the Python Interpreter memory for further analysis data is grouped according to the nature of the request.
- Always returns a Python Dictionary with known schema, and the aggregated Transaction Model queryset.
- Can be abstracted and reimplemented using other distributed computing platforms like Spark and Dask, or other compiled modern programming languages like Rust.

```python
class IOMixIn:
    """..."""

    def database_digest(self,
                        user_model: UserModel,
                        queryset: QuerySet,
                        from_date: str or datetime = None,
                        to_date: str or datetime = None,
                        activity: str = None,
                        role: str = None,
                        entity_slug: str = None,
                        unit_slug: str = None,
                        accounts: str or List[str] or Set[str] = None,
                        posted: bool = True,
                        exclude_zero_bal: bool = True,
                        by_period: bool = False,
                        by_unit: bool = False):...

    def python_digest(self,
                      user_model: UserModel,
                      queryset: QuerySet,
                      to_date: str = None,
                      from_date: str = None,
                      equity_only: bool = False,
                      activity: str = None,
                      entity_slug: str = None,
                      unit_slug: str = None,
                      role: str = None,
                      accounts: set = None,
                      signs: bool = False,
                      by_unit: bool = False,
                      by_period: bool = False) -> list or tuple:...

    def digest(self,
               user_model: UserModel,
               accounts: set or list = None,
               activity: str = None,
               entity_slug: str = None,
               unit_slug: str = None,
               signs: bool = True,
               to_date: Union[str, datetime, date] = None,
               from_date: Union[str, datetime, date] = None,
               queryset: QuerySet = None,
               process_roles: bool = True,
               process_groups: bool = False,
               process_ratios: bool = False,
               equity_only: bool = False,
               by_period: bool = False,
               by_unit: bool = True,
               digest_name: str = None
               ) -> dict or tuple:...
```

How to get started?

Get familiar with Django.

How to install Django Ledger?

- Install Django Ledger

```
pip install git+https://github.com/arrobalytics/django-ledger.git
```

To install Django Virtual Environment `pip install pipenv`

- Or with pipenv:

```
pipenv install git+https://github.com/arrobalytics/django-ledger.git
```

Add Django Ledger to your project?

- Add django_ledger to INSTALLED_APPS

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

- Add URLs to your project:

```
from django.urls import include, path

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

Why you should contribute to Django Ledger?

- An opportunity to contribute to a financial project.
- Access to a network of like minded CPAs and accountants who want to learn Python.
- Django Ledger will teach you financial literacy in a language you can understand (Python).

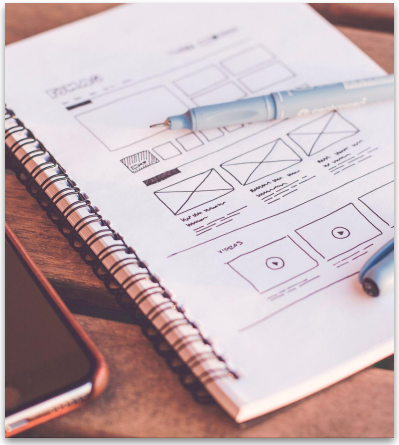
Opportunities to Contribute

Documentation

Easy

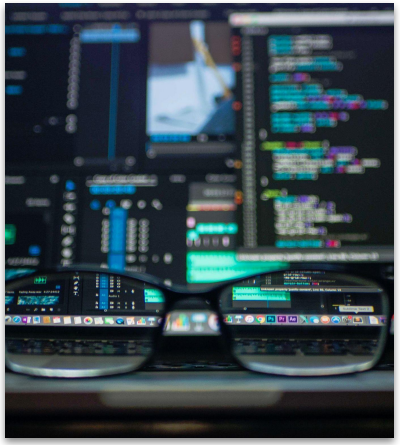
Bugs / Unit Tests

Easy



UI/UX

Medium



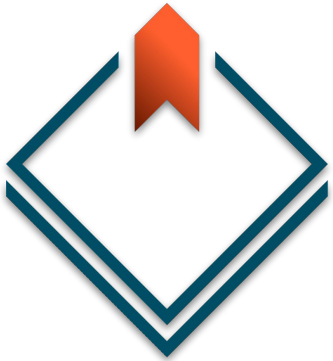
Data Models & Logic

Complex

The RoadMap

[django-ledger/ROADMAP.md at develop · arrobalytics/django-ledger]

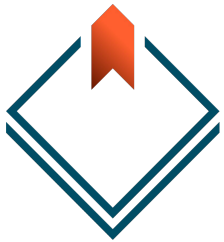
Start The Project On Github



DJANGO
LEDGER

< Object Oriented Accounting Engine />

Go To => http://www.djangoledger.com and click star!



DJANGO LEDGER

< Object Oriented Accounting Engine />

Demo