SWFramework

Overview

SWFramework is the AMNS Library, for development of Web Application, Automatically generates Masters REST APIs for Backend and UI screens for Frontend (i.e Listview & Formview), which effectively reduces the developemnt time of projects, as Masters screens are automatically generated.

SWFramework is divided in 2 modules :

- `SWBackend`
- `SWFrontend`

SWBackend Features :

- This package generates the dynamic REST API for the Django models.
- Handles drf Nested serialization dynamically for upto 10 levels.
- Handles nested one-to-many and nested many-to-one drf reverse nested serialization

SWBackend Features :

- This package generates the dynamic REST API for the Django models.
- Handles drf Nested serialization dynamically for upto 10 levels.
- Handles nested one-to-many and nested many-to-one drf reverse nested serialization dynamically.
- LDAP authentication for token generation
- API Authentication is handled with drf Token Authentication.
- Supports the POST, PUT, DELETE, operations.
- Each API is based on the permissions gievn RBAC handling.
- Excel Export for models Data.
- Maintain change history (i.e audit logs).
- LOGS for each API and Login calls.
- Configurable Generic Search and Filters for Masters
- Configurable Workflows

SWFrontend Features :

- Generates Masters screens automatically

- This package generates the dynamic REST API for the Django models.
- Handles drf Nested serialization dynamically for upto 10 levels.
- Handles nested one-to-many and nested many-to-one drf reverse nested serialization dynamically.
- LDAP authentication for token generation
- API Authentication is handled with drf Token Authentication.
- Supports the POST, PUT, DELETE, operations.
- Each API is based on the permissions gievn RBAC handling.
- Excel Export for models Data.
- Maintain change history (i.e audit logs).
- LOGS for each API and Login calls.
- Configurable Generic Search and Filters for Masters
- Configurable Workflows

SWFrontend Features :

- Generates Masters screens automatically
- Have Inbuilt sidebar & header with AMNS logo.

WHAT IS MDM ? →

What is MDM ?

Master Data Management

- Each project has its own master data like products, countries categories, states, customers etc, which are base Data for whole project, and reflects on transaction screens.
- Business Admin needs screens & UI to handle Master data and perform operations like add or remove data. Master Data Management implies simple CRUD operations with extension features like search and filters, as the Data size may be huge.
- Hence SWFramework provides Automation generation (p.s Requires Initial Configuration) of Frontend and Backend, which as a result reduce the development time of various projects. Developer can fully concentrate on Transactional APIs and UI Screen Develpment.

← OVERVIEW & FEATURES

WORKING BRIEF →

Working Brief

Working Brief (IN-Sync With Frontend)

- Every models, includes the `UI_Meta` class under the model class, which is the hard coded json, used for the UI screens generation. Initially, the OPTIONS call, generates all the models, of the Masters App, which are displayed on the Sidenavbar, on UI.
- On- click for any master screen, generates the Grid View with data for that respected model, with ADD/EDIT/DELETE buttons available w.r.t permissions available. Conclusively, **UI_Meta** class of Masters models is used for Masters Formview rendering and Listview is rendered based on the fields of the Masters models.

← WHAT IS MDM ?
BACKEND OVERVIEW →

Backend Overview

SWFBackend Contents

SWFBackend Contents

- The SWFBackend package, contains 3 apps, with functionality as follows :

```
- api :
        - dynamically generates REST Api for masters
        - handles masters CRUD operations
        - handles several utility functions
        - handles LDAP login

- rbac :
        - handles sidenavbar UI sections from backend i.e DB
        - handles users, roles, permissions

- workflow :
        - handles approval workflow fro master data
        - allows configurable workflows
```

Here is the project structure for development of backend of your project.

```
YOUR_PROJECT
    |- YOUR_APP             <-- Your custom app
        |- models
            |- __init__.py
            |- states.py
            |- cities.py
            ...
        |- constants.py
        |- entities.py      <-- Your proxy models lies here
        |- exceptions.py
        |- interface_tests.py
        |- managers.py      <-- Your model managers lies here
        |- responses.py
        |- models.py
        |- serializers.py
        |- service.py       <-- Your service lies here,
        |- tests.py
        |- urls.py
        |- validators.py
        |- value_objects.py
        |- views.py
    |- masters              <-- django app for master models CRUD
    |- users                <-- django app for users authentication
```

Folder	Description
constants.py	App / Module level constants.
entities.py	Proxy models which contains the Business Logic related to business model. Each functions should be atomic and single responsibility. The methods should be pure methods should not use anything other than instance variables and args.
exceptions	Globals for app/module level exceptions
interface_tests.py	test files relates to interface testing / Load Testing on API.
managers.py	Model manager which will do something analogous to a DAO.
models.py	Django Models
responses.py	Custom responses defined for the routes / api endpoints.
serializers.py	DRF Serializers.
service.py	Service Classes for a given module for each entities
tests.py	Unit Tests for given module
urls.py	Django URL Mapping
validators.py	Validation logic related to Domain Modules. These validators can

tests.py	Unit Tests for given module
urls.py	Django URL Mapping
validators.py	Validation logic related to Domain Modules. These validators can be injected in related serializers / forms / other orchestration logics.
views.py	API Views. Ideally views should call only the service classes for orchestration, workflow and wiring logic. Views should be free from logical detailing

Django Models Structure

Instead of using `models.py` . Define all your models into separate directory named `models` which contains all your models `e.g (states.py, cities.py)` . Here `states` and `cities` is two different models

```
YOUR_PROJECT
    |- YOUR_APP
        |- models
            |- __init__.py   <-- each model class, needs to be imported in here
            |- states.py
            |- cities.py
```

Django Models Structure

Instead of using `models.py` . Define all your models into separate directory named `models` which contains all your models `e.g (states.py, cities.py)` . Here `states` and `cities` is two different models

```
YOUR_PROJECT
    |- YOUR_APP
        |- models
            |- __init__.py   <-- each model class, needs to be imported in here
            |- states.py
            |- cities.py
        models.py
        urls.py
        ...
```

- Here, the models directory contains, the separate files for each and every model defined separately in their respected files
- The `__init__.py` , is mandatory which imports all the models from the different files, in order to detect the changes with Django migrations.
- Then, in order to activate models with migration, import each of the models in `__init__.py` , example :

```
models.py
urls.py
...
```

- Here, the models directory contains, the separate files for each and every model defined separately in their respected files
- The `__init__.py` , is mandatory which imports all the models from the different files, in order to detect the changes with Django migrations.
- Then, in order to activate models with migration, import each of the models in `__init__.py` , example :

```
from .states import states
from .cities import cities
```

← BACKEND OVERVIEW

INSTALLATION →

Configuration

Installed Apps Setup

- Register framework apps, `api` , `rbac` , `workflow` , in django `setting.py` in 'INSTALLED_APPS'.
- Before django's existing `INSTALLED_APPS` into `DEPENDENCY_APPS` and `PROJECT_APPS` into your `settings.py` as follows :

```
DEPENDENCY_APPS = [
    'corsheaders',
    'rest_framework',
    'rest_framework.authtoken',
    'rest_framework_swagger',
    'reversion',
    'multiselectfield',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

- Before django's existing `INSTALLED_APPS` into `DEPENDENCY_APPS` and `PROJECT_APPS` into your `settings.py` as follows :


```
DEPENDENCY_APPS = [
    'corsheaders',
    'rest_framework',
    'rest_framework.authtoken',
    'rest_framework_swagger',
    'reversion',
    'multiselectfield',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'whitenoise.runserver_nostatic',
    'django.contrib.staticfiles',
    'rbac',
    'workflow',
    'api',
]
PROJECT_APPS = []

INSTALLED_APPS = DEPENDENCY_APPS + PROJECT_APPS
```

Migrations

- Initially, migrate all the predefined inbuilt Django models, (i.e auth, user, permissions, etc), with the command :

```
python manage.py migrate
```

- Before starting, the migration for your project's app models, do migrate framework apps `api` , `workflow` models by :

```
python manage.py makemigrations workflow
python manage.py makemigrations api
python manage.py migrate
```

URLs Setup

- Include the following urls paths in your `masters` app's `urls.py` , to connect master Api calls with framework :

- `<your-prefix>/` will be same as we have defined in `REACT_APP_API_PREFIX` of our

```
python manage.py migrate
```


URLs Setup

- Include the following urls paths in your `masters` app's `urls.py` , to connect master Api calls with framework :
- `<your-prefix>/` will be same as we have defined in `REACT_APP_API_PREFIX` of our `.env` in frontend.

```
urlpatterns = [
    path('<your-prefix>/', include('api.urls'), name="API"),
    path('<your-prefix>/workflow/', include('workflow.urls'), name="WORKFLOW"),
    path('<your-prefix>/rbac/', include('rbac.urls'), name="RBAC")
]
```

Django & DRF Configuration

- Once we successfully configured django & DRF. Our `settings.py` file will look like as below :

Django & DRF Configuration

- Once we successfully configured django & DRF. Our `settings.py` file will look like as below :

```
"""
Django settings for mydjangoproject project.

Generated by 'django-admin startproject' using Django 3.1.6.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/
```

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '%(gn$)uyp--i+lf)r!jzyq)jj4_g^w+lah(!_(9ujy03jnoyj5'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'mydjangoproject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

```
WSGI_APPLICATION = 'mydjangoproject.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValid
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
```

```
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
```

Configure Settings.py

All the below mentioned constants are mandatory to define in `settings.py` , which are framework's configs.

MODEL_VALIDATORS

- This constant, is to apply mapping for, object level master models data validation.
- Add the key as the model name (lower case) and the value should be the object of the validation function to be applied on the model validation. (object level validation)

```
import LocationValidator
import PartnerValidator
...
...
MODEL_VALIDATORS = {
    'location' : LocationValidator.validate,     #<--- validate function object
    'partnermaster' : PartnerValidator.validate,
```

```
...
MODEL_VALIDATORS = {
    'location' : LocationValidator.validate,     #<--- validate function object
    'partnermaster' : PartnerValidator.validate,
}
```

DISPLAY_MODELS

- This constant, is to maintain the mapping for model : Display Names, to be displayed in Sidenavbar on UI.
- Add the key for the model class names (all lower case) and the value should be the name for which the model should be displayed as on the Side nav-bar screen (UI).

```
DISPLAY_MODELS = {
    'location': 'Location',
    'partnermaster' : 'Partner',
    ...
}
```


DISPLAY_MODEL_FIELDS

- This constant, is to configure the list of model fields, which are being displayed in Lisview grid in masters section.
- Here, key is the model class name (lower case) & values,the fields (field name) list, to be displayed in the grid-view on UI.

```
DISPLAY_MODEL_FIELDS = {
    'location': ['id', 'location_code', 'location_name', 'is_active'],      #<--- lis
    'partnermaster': ['code', 'partner_name', 'is_active', 'partner_type__type']
}
```

Id	Location code	Location name	Is active
6	BHGR	Bahadurghar	true
5	PRD	Paradeep	true
4	PUN	Pune	true
3	VZG	Vizag	true
2	MUM	Mumbai	true
1	HAZ	Hazira	true

Note :-

Note :-

- It is mandatory to include the , pimary key field for evey model. Also all the model names in lower case.
- Even if the Django's inbuilt primary key is used, then also it's mandatory to mentioned 'id' as theprimary key field for that model

EXTRA_USER_META

- This is important configuration constant, which handles the sections / subsections of sidenavbar of UI.
- The keys are the sections to be displayed, on the side navbar (UI-homepage), & the values i.e list of dictionary indicates the subsection.
- Here, `MASTERS` & `Reports` are the sections & under `Reports` , we have 2 subsections `SendBCReport` and `RequestBCReport` .

Note :

- Although we have mention all the sections into `EXTRA_USER_META` , None of the sections will be initially visible inside `UI Side NavBar` until proper `views`

- For more details, please visit Configure Side NavBar section.

```
EXTRA_USER_META = {
    'MASTERS': [
        {
            'app': 'masters',
            'title': 'MASTER'
        }
    ],

    'Reports': [
        {
            'screen': 'SendBCReport',              #<--- screen name reference, used f
            'type': 'subsection',                  #<--- mandatory keyword, type "subs
            'title': 'Confirmations Sent'          #<--- display name, for sidenavbar
        },
        {
            'screen': 'RequestBCReport',
            'type': 'subsection',
            'title': 'Confirmations Requested'
        }
    ],
}
```

SECTION_ICONS

- This constant, maintain the mapping for section icons, for all the sections maintained in `EXTRA_USER_META` .
- Provide the Section Icons, for each of the sections defined above, as shown:

```
SECTION_ICONS = {
    'MASTERS' : 'fa fa-wpforms',
    'Reports' : 'fa fa-comment',
}
```

Note :-

- The keys for `EXTRA_USER_META` and `SECTION_ICONS` , which indicates the user defined sections should remain same.
- For all the sections mentioned in `EXTRA_USER_META` , there should be compulsory key in `SECTION_ICONS` , (i.e. with same name)

AUTO_GENERATION_SECTIONS

- Configure the sections, which needs to be auto-generated, i.e completely handled by framework

AUTO_GENERATION_SECTIONS

- Configure the sections, which needs to be auto-generated, i.e completely handled by framework.
- The term `auto-generated` , means, all the subections i.e all the models of particular app, will be rendered with listview & formview as subsections, for the given section.
- Here, `MASTERS` , is the section, as mentioned in `EXTRA_USER_META` .

```
AUTO_GENERATION_SECTIONS = ['MASTERS']

EXTRA_USER_META = {
    'MASTERS': [
        {
            'app': 'masters',       #<--- Django app name, of which all models will b
            'title': 'MASTER'       #<--- Display name, of section in sidenavbar
        }
    ],
    ...
    ...

}
```

URL_METADATA_CLASS

- The OPTIONS calls for the `api` , `rbac` , and `workflow` app are handled, kindly paste as it is:

```
URL_METADATA_CLASS = {
    'api' : 'api.metadata.ApiMetadata',
    'User': 'rbac.metadata.RbacMetadata',
    'Suggest': 'rbac.metadata.RbacMetadata',
    'Group': 'rbac.metadata.RbacMetadata',
    'Permission': 'rbac.metadata.RbacMetadata',
}
```

Note :-

- Initially, The above settings are mandatory.
- Then, In case, if in your project You are extending the Django `SimpleMetadata` , for the OPTIONS call, then append the Dotted separated path for your Metadata class (which contains the determine_metadata overided definition) as shown above.

MODEL_FIELD_CHOICES

- In case of usage of Django-Multiselectfied, (i.e third party), mention the choices of the models, otherwise leave blank.

```
MODEL_FIELD_CHOICES = {
    ...
}
```

MASTERS_APP_RBAC

- Set the appname for Masters table for which the RBAC(frontend) needs to be applied.

```
MASTERS_APP_RBAC = 'master'      #<--- django app name, for which RBAC will be activa
```

WORKFLOW_MODEL_LIST

WORKFLOW_MODEL_LIST

- Provide the list of models (all lower case), for which the workflow needs to be applied, for add / edit operations.

```
WORKFLOW_MODEL_LIST = [
    ...
]
```

APPLY_SEARCH_MODEL_FIELDS

- It is used to provide dynamic `search` functionality which will be available from UI.
- Mention keys, as models class name (lower), & values as the list of model fields, on which the search value will be searched.
- As below, for value search on `country` model, will have look ups on fields, `country_id` , `country_code` & `country_name` .
- Models Foreign keys are also supported, the referenced model fields are to be mentioned `__` seperated.
- As below, model `partnermaster` , have `ForeignKey` field `partner_type` , referencing to model `PartnerType` , having field `type` , hence in list we mention

`country_id` , `country_code` & `country_name` .

- Models Foreign keys are also supported, the referenced model fields are to be mentioned `__` seperated.
- As below, model `partnermaster` , have `ForeignKey` field `partner_type` , referencing to model `PartnerType` , having field `type` , hence in list we mention `partner_type__type` , for which value will have look ups on `type` field of `PartnerType` .

```
APPLY_SEARCH_MODEL_FIELDS = {
    'country': ['country_id', 'country_code', 'country_name'],
    'partnermaster': ['code', 'partner_name', 'is_active', 'partner_type__type'],   #
    ...
}
```

FOREIGN_KEY_UI_NAME_MAP

- It is used, to configure generic `Filters` , functionality for `Foreignkey` fields of models, for Frameworks `Filters` , component i.e UI.
- Normal model fields are automatically configured & active for `Filters` , hence no need to mention here.

FOREIGN_KEY_UI_NAME_MAP

- It is used, to configure generic `Filters` , functionality for `Foreignkey` fields of models, for Frameworks `Filters` , component i.e UI.
- Normal model fields are automatically configured & active for `Filters` , hence no need to mention here.
- As in example below, model `balanceconfirmation` , has 2 `Foreignkey` fields, `partner` & `bc_status` .
    i. `partner` , is referencing to some model `PARTNER` , having fields, `partner_name` & `code` , hence to activate value look ups on both the fields, mention as list of dictionary as shown below in key `partner` .
    ii. `bc_status` , is referencing to some model `BC_STATUS` , having one of the field `status_name` , hence to activate value look up on the field, mention as list of dictionary as shown below in key `bc_status` .
- As of now, the only supported FK referenced field types are `text` & `integer` .
- Django reverse relations are also supported for look ups, i.e FK's `related_name` attribute.

```
FOREIGN_KEY_UI_NAME_MAP = {

    'balanceconfirmation': {
        'partner': [
            {
                'field': 'partner__partner_name',
                'type': 'text'                #<--- type of the referenced field, onl
            },
            {
                'field': 'partner__code',
                'type': 'text'
            },
        ],
        'bc_status':[
            {
                'field': 'bc_status__status_name',
                'type': 'text'
            },
        ],
        ....
        ....
    }
}
```

MASTERS_APP_NAME

- Mention the `masters` , django app name here.

```
MASTERS_APP_NAME = 'masters'   #<--- masters app name
```

DATE_INPUT_FORMAT

- Mention the Date Input formats for DRF serialization, for `DateField` , used in master models.

```
DATE_INPUT_FORMAT = []
```

DATE_TIME_INPUT_FORMAT

- Mention the Date Time Input formats for DRF serialization, for `DateTimeField` , used

DATE_TIME_INPUT_FORMAT

- Mention the Date Time Input formats for DRF serialization, for `DateTimeField`, used in master models.

```
DATE_TIME_INPUT_FORMAT = ['iso-8601']        #<--- example
```

TIME_INPUT_FORMAT

- Mention the Time Input formats for DRF serialization, for `TimeField`, used in master models.

```
TIME_INPUT_FORMAT = []
```

POST_SAVE_OBJ_RETURN_HOOKS

POST_SAVE_OBJ_RETURN_HOOKS

- Mention the hooks, for which the maste model object, will be return for custom operations, before saving to DB.

```
POST_SAVE_OBJ_RETURN_HOOKS = {

}
```

EXCEL_IMPORT_CONFIGS

- Configure the Excel Import varibales here.

```
EXCEL_IMPORT_CONFIGS = {}
```

Following constants are related to Moss DMS File uploads, which is under development, hence paste as it is:

```
MASTER_FILES_MOSS_UPLOAD = False
```

Following constants are related to Moss DMS File uploads, which is under development, hence paste as it is:

```
MASTER_FILES_MOSS_UPLOAD = False
MOSS_BASE_URL = ''
MOSS_USERNAME = ''
MOSS_PASSWORD = ''
RELATIVE_FOLDER_TO_UPLOAD_URL = ''
```

Following configurations are related to, `Users` management

CHECK_USER_REGISTRATION

- Bool `True` or `False`, to allow users able to login in project.
- If `False`, all the incoming AD Users are able to login, via LDAP authentication module.
- If `True`, Framework checks the user id existence in `userprofile`, model, then allow the login, hence, only registered users able to login.

```
CHECK_USER_REGISTRATION = False
```

CHECK_USER_REGISTRATION

- Bool `True` or `False`, to allow users able to login in project.
- If `False`, all the incoming AD Users are able to login, via LDAP authentication module.
- If `True`, Framework checks the user id existence in `userprofile`, model, then allow the login, hence, only registered users able to login.

```
CHECK_USER_REGISTRATION = False
```

USER_PROFILE_MODEL_DETAILS

- If `CHECK_USER_REGISTRATION = True`, mandatory to give below details, else leave empty values:
- Example as shown below:

```
USER_PROFILE_MODEL_DETAILS = {
    'app': 'users',                     #<--- 'users', django app name
    'model': 'userprofile',             #<--- model name, for user registration
    'search_on_field': 'outlook_id'     #<--- field name, for existence look ups
}
```

USER_PROFILE_SERIALIZER

- If `CHECK_USER_REGISTRATION = True` , mandatory to give below details, else leave empty values:
- Purpose, is to return User data on Login api call, via serializer mentioned as below.
- Example as shown below, give dot seperated path to serilizers class.

```
USER_PROFILE_SERIALIZER = 'users.serializers.UserProfileSerializer'   #<--- example p
```

Following constants are related to `workflow` , module of framework, if you're considering `workflow` , in your project configure each of as per requirments, else paste as shown below.

WORKFLOW_EMAIL_POLICY

- Configures the workflow email policy, kindly paste as,it is:

```
WORKFLOW_EMAIL_POLICY = {
    'default': {
        'Init': {
            'request': 'workflow.email_policy.notify_next_level',
```

as shown below.

WORKFLOW_EMAIL_POLICY

- Configures the workflow email policy, kindly paste as it is:

```
WORKFLOW_EMAIL_POLICY = {
    'default': {
        'Init': {
            'request': 'workflow.email_policy.notify_next_level',
        },
        'Approve': {
            'request': 'workflow.email_policy.notify_next_level',
            'notification': 'workflow.email_policy.get_request_initiator'
        },
        'Reject': {
            'notification': 'workflow.email_policy.notify_the_hierarchy'
        },
    }
}
```

RABBITMQ_CONF

RABBITMQ_CONF

- Configures Async mail component of framework.
- If not in usage, kindly paste as it is:

```
RABBITMQ_CONF = {
    "RABBITMQ_CON_URL": "",
    "RABBITMQ_QUEUE": "",
    "SENDER_MAIL": "",
    "MAIL_CREDENTIALS": {"sender_email": "", "mailserver": ""}
}
```

WORKFLOW_MAIL_MAP

- If not in usage, kindly paste as it is:

```
WORKFLOW_MAIL_MAP = {

}
```

MODELS_WITH_MAIL_APPROVAL

- If not in usage, kindly paste as it is:

```
MODELS_WITH_MAIL_APPROVAL = []
```

LANDING_PAGE_FMT

- If not in usage, kindly paste as it is:

```
LANDING_PAGE_FMT = "http://localhost:3000/app/ApprovalForm/{}"
```

APPROVAL_API_URLS

- If not in usage, kindly paste as it is:

```
APPROVAL_API_URLS = {
}
```

APPROVAL_GROUPS

- If not in usage, kindly paste as it is:

```
APPROVAL_GROUPS = []
```

TRAN_FILTER_QS_SERIALIZER

- If not in usage, kindly paste as it is:
- To learn more visit Tran QS Filter

```
TRAN_FILTER_QS_SERIALIZER = []
```

← CONFIGURATION

MANAGE.PY RUNSERVER →

Installation Complete

Runserver

- At this stage, SWFBackend is installed successfully, you can check by command :

```
python manage.py runserver
```

- Further, follow configurations of masters, rbac, workflow, as per developer requirements.

← CONFIGURE SETTINGS.PY

SETTINGS.PY REFERENCE →

Settings.py Reference

Sample Settings.py File (Only for reference purpose)

```
"""
Django settings for rpa_master project.

Generated by 'django-admin startproject' using Django 2.2.7.

For more information on this file, see
https://docs.djangoproject.com/en/2.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.2/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

Sample Settings.py File (Only for reference purpose)

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '9t25jai&9k2k_6-%ueo+4iv_hwblwqw(s4_5$^a3%iv67__8af'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
CSRF_TRUSTED_ORIGINS = [".myamns.in", ".amns.in"]

#YOUR PREFIX will be same as we have defined `REACT_APP_TOKEN_PREFIX` in `.env` file
CSRF_COOKIE_NAME = 'YOURPREFIX-csrftoken'
SESSION_COOKIE_NAME = 'YOURPREFIX-sessionid'

SESSION_COOKIE_HTTPONLY = False
SESSION_COOKIE_AGE = 14400

# Application definition
```

Sample Settings.py File (Only for reference purpose)

```
# Application definition

DEPENDENCY_APPS = [
    'corsheaders',
    'rest_framework',
    'rest_framework.authtoken',
    'rest_framework_swagger',
    'reversion',
    'multiselectfield',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'whitenoise.runserver_nostatic',
    'django.contrib.staticfiles',
    'rbac',
    'workflow',
    'api',
]
PROJECT_APPS = [
    # 'masters',
    'finance',
]

INSTALLED_APPS = DEPENDENCY_APPS + PROJECT_APPS
```

Sample Settings.py File (Only for reference purpose)

```
INSTALLED_APPS = DEPENDENCY_APPS + PROJECT_APPS

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

CORS_ORIGIN_ALLOW_ALL = True
CORS_ALLOW_CREDENTIALS = True
CORS_ALLOW_METHODS = [
    'DELETE',
    'GET',
    'OPTIONS',
    'PATCH',
    'POST',
    'PUT',
]
CORS_ALLOW_HEADERS = [
    'accept',
```

Sample Settings.py File (Only for reference purpose)

```
]
CORS_ALLOW_HEADERS = [
    'accept',
    'accept-encoding',
    'approval-authorization',
    'authorization',
    'content-type',
    'dnt',
    'origin',
    'user-agent',
    'x-csrftoken',
    'x-requested-with',
    'req',
    'source'
]

ROOT_URLCONF = 'rpa_master.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'build')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
```

Sample Settings.py File (Only for reference purpose)

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'build')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'rpa_master.wsgi.application'

# Database
# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'RPA_FIN_MASTER',
        'USER': 'postgres',
        'PASSWORD': 'admin',
```

Sample Settings.py File (Only for reference purpose)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'RPA_FIN_MASTER',
        'USER': 'postgres',
        'PASSWORD': 'admin',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': [
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
    ],

    'DEFAULT_PARSER_CLASSES': [
        'rest_framework.parsers.JSONParser',
        'rest_framework.parsers.MultiPartParser'
    ],

    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend'
```

Sample Settings.py File (Only for reference purpose)

```
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend'
    ],
    'DEFAULT_METADATA_CLASS': 'api.metadata.MinimalMetadata',

    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_SCHEMA_CLASS': 'rest_framework.schemas.coreapi.AutoSchema'
}
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
    }
}

# Password validation
# https://docs.djangoproject.com/en/2.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValid
    },
```

Sample Settings.py File (Only for reference purpose)

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValid
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/2.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Asia/Kolkata'

USE_I18N = True

USE_L10N = True
```

Sample Settings.py File (Only for reference purpose)

```
USE_L10N = True

USE_TZ = False

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.2/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATICFILES_DIRS = []
STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'

MODEL_VALIDATORS = {}

DISPLAY_MODELS = {
    # Models that will be displayed on Screen
    'grncreationuttam': 'GRN_CREATION_UTTAM',
    'grncreationsto': 'GRN_CREATION_STO',
    'porequestuttam': 'PO_REQUEST_UTTAM',
    'uttampocommercialchart': 'UTTAM_PO_COMMERCIAL_CHART',
    'uttamproductmapping': 'UTTAM_PRODUCT_MAPPING',
}

DISPLAY_MODEL_FIELDS = {
    'grncreationuttam': ['id', 'gst_invoice', 'invoice_date', 'batch', 'received_date
                         'vehicle_no', 'lr_no', 'gate_no', 'rj_number'],
    'grncreationsto': ['id', 'gst_invoice', 'invoice_date', 'batch', 'transporter_nam
```

Sample Settings.py File (Only for reference purpose)

```
DISPLAY_MODEL_FIELDS = {
    'grncreationuttam': ['id', 'gst_invoice', 'invoice_date', 'batch', 'received_date
                         'vehicle_no', 'lr_no', 'gate_no', 'rj_number'],
    'grncreationsto': ['id', 'gst_invoice', 'invoice_date', 'batch', 'transporter_nam
                       'gate_no'],
    'porequestuttam': ['id', 'coil_id', 'reference_number', 'monogram', 'ink_jet_prin
                       'product', 'material_code', 'hr_thickness', 'hr_width', 'finis
                       'finish_width', 'zinc_coating', 'coil_weight', 'spangle_type',
                       'back_coat', 'fg_equivalent_specs', 'qty'],
    'uttampocommercialchart': ['id', 'product', 'finish_thickness', 'rate'],
    'uttamproductmapping': ['id', 'product', 'commercial_product']
}

AUTO_GENERATION_SECTIONS = ['MASTERS', 'FINANCE']

EXTRA_USER_META = {
    'MASTERS': [
        {
            'app': 'masters',
            'title': 'MASTER'
        }
    ],
    'FINANCE': [
        {
            'app': 'finance',
            'title': 'FINANCE'
        }
```

Sample Settings.py File (Only for reference purpose)

```
    ],
    'FINANCE': [
        {
            'app': 'finance',
            'title': 'FINANCE'
        }
    ],
}

SECTION_ICONS = {
    'MASTERS': 'fa fa-th-list',
    'FINANCE': 'fa fa-th-list'
}

URL_METADATA_CLASS = {
    'api': 'api.metadata.ApiMetadata',
    'User': 'rbac.metadata.RbacMetadata',
    'Suggest': 'rbac.metadata.RbacMetadata',
    'Group': 'rbac.metadata.RbacMetadata',
    'Permission': 'rbac.metadata.RbacMetadata',
}

MODEL_FIELD_CHOICES = {
}

MASTERS_APP_RBAC = 'masters'
```

Sample Settings.py File (Only for reference purpose)

```
APPLY_SEARCH_MODEL_FIELDS = {
    'grncreationuttam': ['gst_invoice', 'invoice_date','batch', 'received_date', 'tra
                         'vehicle_no', 'lr_no', 'gate_no', 'rj_number'],
    'grncreationsto': ['gst_invoice', 'invoice_date', 'batch', 'transporter_name', 'v
                       'gate_no'],
    'porequestuttam': ['coil_id', 'reference_number','product', 'material_code',
                       'hr_thickness', 'hr_width', 'finish_thickness','finish_width']
    'uttampocommercialchart': ['product', 'finish_thickness'],
    'uttamproductmapping': ['product', 'commercial_product']
}


FOREIGN_KEY_UI_NAME_MAP = {
     'balanceconfirmation': {
         'location': [
             {
                 'field': 'location__location_name',
                 'type': 'text'
             }
         ],
     }
}


MEDIA_ROOT = BASE_DIR + '/media/'

MEDIA_URL = '/media/'
```

Sample Settings.py File (Only for reference purpose)

```
MEDIA_URL = '/media/'
X_FRAME_OPTIONS = 'SAMEORIGIN'

WORKFLOW_EMAIL_POLICY = {
    'default': {
        'Init': {
            'request': 'workflow.email_policy.notify_next_level',
        },
        'Approve': {
            'request': 'workflow.email_policy.notify_next_level',
            'notification': 'workflow.email_policy.get_request_initiator'
        },
        'Reject': {
            'notification': 'workflow.email_policy.notify_the_hierarchy'
        },
    }
}

RABBITMQ_CONF = {
    "RABBITMQ_CON_URL": "",
    "RABBITMQ_QUEUE": "",
    "SENDER_MAIL": "",
    "MAIL_CREDENTIALS": {"sender_email": "", "mailserver": ""}
}

WORKFLOW_MAIL_MAP = {
```

Sample Settings.py File (Only for reference purpose)

```
WORKFLOW_MAIL_MAP = {

}

MODELS_WITH_MAIL_APPROVAL = []

LANDING_PAGE_FMT = "http://localhost:3000/app/ApprovalForm/{}"

APPROVAL_API_URLS = {
}

# APPROVAL_GROUPS = []

MASTERS_APP_NAME = 'masters'

DATE_INPUT_FORMAT = []

DATE_TIME_INPUT_FORMAT = ['iso-8601']

TIME_INPUT_FORMAT = []



POST_SAVE_OBJ_RETURN_HOOKS = {
}
```

Sample Settings.py File (Only for reference purpose)

```
CHECK_USER_REGISTRATION = False

USER_PROFILE_MODEL_DETAILS = {
    'app': '',
    'model': '',
    'search_on_field': ''
}

USER_PROFILE_SERIALIZER = 'users.serializers.UserProfileSerializer'

DATA_RESTRICT_PERMISSIONS = {}

MASTER_FILES_MOSS_UPLOAD = False
MOSS_BASE_URL = ''
MOSS_USERNAME = ''
MOSS_PASSWORD = ''
RELATIVE_FOLDER_TO_UPLOAD_URL = ''

TRANS_FILTER_QS_SERIALIZER = {
        # 'careers.jobseeker': ('careers.trans_filter_support.return_jobseeker_qs', '
}


EXCEL_IMPORT_CONFIGS = {
    'grncreationuttam': {
        'gst_invoice': 0,
```

Sample Settings.py File (Only for reference purpose)

```
EXCEL_IMPORT_CONFIGS = {
    'grncreationuttam': {
        'gst_invoice': 0,
        'invoice_date': 1,
        'batch_no': 2
    },
    'grncreationsto': {
        'gst_invoice': 0,
        'invoice_date': 1,
        'batch_no': 2
    },
    'porequestuttam': {
    },
    'uttampocommercialchart': {
        'product': 0,
        'finish_thickness': 1
    },
}


WORKFLOW_MODEL_LIST = [
    'porequestuttam'
]

WORKFLOW_EMAIL_CONFIGS = {
}
```

```
            'finish_thickness': 1
        },
    }


    WORKFLOW_MODEL_LIST = [
        'porequestuttam'
    ]

    WORKFLOW_EMAIL_CONFIGS = {
    }


    APPROVAL_GROUPS = [
    ]

    GROUP_WISE_STATUS_MAPPING = {
        # 'HR': ['Pending For Approval', 'Initiated', 'Approved'],
    }
```

Sample Settings.py File (Only for reference purpose)

← MANAGE.PY RUNSERVER

WRITING MASTER MODELS →

Writing Master Models

While creating master model, following rules/configuration are mandatory :

- All the Master models should be inherited from the api's `BaseModel` .

```
from api.models import BaseModel

class Profile(BaseModel):
    ...
    ...
```

- All the fields defined for the models, should have `db_column` attribute, which defines the column names for the tables created in the database.

```
auditor_name = models.CharField(max_length=300, db_column='AUDITOR_NAME')
```

- All the fields defined for the models, should have `db_column` attribute, which defines the column names for the tables created in the database.

```
auditor_name = models.CharField(max_length=300, db_column='AUDITOR_NAME')
```

- For every models, for every Foreign key fields used, should have the `related_name` attribute, mandatorily:

```
address = models.ForeignKey(Address, on_delete=models.CASCADE, db_column="address", r
```

- All the models, should have the api's AliasField fields mandatorily, with the exact field names as mentioned below :-

```
from api.alias import AliasField

...
name = AliasField(db_column='< any referenced DB column>', blank=True, null=True)
```

- All the master models should be registered with `reversion` , in order to maintain

- All the master models should be registered with `reversion` , in order to maintain change history, as :

```
from reversion import revisions as reversion

...
...
reversion.register(Auditor)
```

- All the master models, should have `class Meta` , to define table name & associated app name as :

```
class Meta:
    db_table = 'MST_AUDITOR'
    app_label = 'masters'
```

- All the master models, should have `class UI_Meta` , which defines meta json to render UI forms, as :

```
class UI_Meta:
```

- All the master models, should have `class UI_Meta` , which defines meta json to render UI forms, as :

```
class UI_Meta:
    ui_specs = {
        "listview": [
            "this value"
        ],
        "formview": [
            {
                "sectionlabel": "Auditor Master",
                "cols": 2,
                "colComponent": [
                    {
                        "label": "Auditor Name",
                        "decorator": "auditor_name",
                        "type": "textbox",
                        "required": "true",
                        "message": "Enter Auditor Name",
                        "id": "auditor_name",
                        # "casetype":"uppercase",
                        "placeholder": "Please enter Auditor Name",
                        "disabled": False,
                        "maxlength": 300
                    },
                    {
                ]
```

- Once you have completed the above steps your model(s) will be look like as below :

```
...
from api.models import BaseModel
from api.alias import Aliasfield
from reversion import revisions as reversion

class Auditor(BaseModel):                        <--- Inheriting from BaseModel
    auditor_id = models.AutoField(
        primary_key=True, db_column="PARTNER_TYPE_ID")
    auditor_name = models.CharField(max_length=300, db_column='AUDITOR_NAME')
    auditor_email = models.EmailField(
        max_length=300, db_column='AUDITOR_EMAIL')
    auditor_address = models.TextField(db_column='AUDITOR_ADDRESS')
    is_current = models.BooleanField(db_column='IS_CURRENT', default=False)
    name = AliasField(db_column='AUDITOR_NAME', blank=True, null=True)        <--- de

    class UI_Meta:        <--- defining ui_meta
        ui_specs = {
            "listview": [
                "this value"
            ],
            "formview": [
                {
                    "sectionlabel": "Auditor Master",
                    "cols": 2
```

```
{
    "sectionlabel": "Auditor Master",
    "cols": 2,
    "colComponent": [
        {
            "label": "Auditor Name",
            "decorator": "auditor_name",
            "type": "textbox",
            "required": "true",
            "message": "Enter Auditor Name",
            "id": "auditor_name",
            # "casetype":"uppercase",
            "placeholder": "Please ente Auditor Name",
            "disabled": False,
            "maxlength": 300
        },
        {
            "label": "Auditor Email",
            "decorator": "auditor_email",
            "type": "textbox",
            "required": "true",
            "message": "Enter Auditor Email",
            "id": "auditor_email",
            # "casetype":"uppercase",
            "placeholder": "Please enter Auditor Email",
            "disabled": False,
            "maxlength": 300
```

```
},
{
    "label": "Auditor Address",
    "decorator": "auditor_address",
    "type": "textbox",
    "required": "true",
    "message": "Enter Auditor Address",
    "id": "auditor_address",
    # "casetype":"uppercase",
    "placeholder": "Please enter Auditor Address",
    "disabled": False,
    "maxlength": 300
},

{
    "label": "Is Current",
    "decorator": "is_current",
    "type": "radio",
    "radioType": "group",
    "id": "isActive",
    "required": "true",
    "message": "Please select the status",
    "listed": "yes",
    "list_data": [
        {
            "True": "True"
        },
```

```
            {
                "True": "True"
            },
            {
                "False": "False"
            },
        ],
        "link_api": "isactive",
        "disabled": False,
    },
    {
        "label": "Created By",
        "decorator": "created_by",
        "type": "textbox",
        "required": "true",
        "message": "Created By !",
        "id": "created_by",
        "placeholder": "This is Created By :",
        "disabled": True
    },
    {
        "label": "Created Date ",
        "decorator": "created_date",
        "required": "true",
        "message": "Created Date ",
        "placeholder": "Select Date",
        "type": "date",
```

```
        "type": "date",
        "id": "created_date",
        "dateFormatList": "YYYY-MM-DDTHH:mm:ss",
        "disabled": True
    },
    {
        "label": "Updated By ",
        "decorator": "last_updated_by",
        "type": "textbox",
        "required": "true",
        "message": "Last Updated By !",
        "id": "last_updated_by",
        "placeholder": "This was last Updated By:",
        "disabled": True
    },
    {
        "label": "Last Updated : ",
        "decorator": "last_updated_date",
        "required": "true",
        "message": "Last Updated Date",
        "placeholder": "Select Date",
        "type": "date",
        "id": "last_updated_date",
        "dateFormatList": "YYYY-MM-DDTHH:mm:ss",
        "readonly": "true",
        "disabled": True
    },
```

```
                        "label": "Last Updated : ",
                        "decorator": "last_updated_date",
                        "required": "true",
                        "message": "Last Updated Date",
                        "placeholder": "Select Date",
                        "type": "date",
                        "id": "last_updated_date",
                        "dateFormatList": "YYYY-MM-DDTHH:mm:ss",
                        "readonly": "true",
                        "disabled": True
                    },
                ]
            }
        ]
    }

    class Meta:
        db_table = 'MST_AUDITOR'
        app_label = 'masters'


reversion.register(Auditor)     <--- registering model with reversion
```

Writting UI_Meta

What is UI_Meta ?

- As describe earlier the framework itself generates dynamic UI for frontend CRUD operations. But how do framework know what kind of UI you want to generate ? That's where UI_Meta comes into frame.
- UI_Meta is kind of blueprint for your frontend UI. Suppose you need a textbox in frontend. Then you will write the following structure in `UI_Meta` class.

```
{
    "label": "State Name",
    "decorator": "sname",
    "type": "textbox",
    "required": "true",
    "message": "Enter State Name",
    "id": "sname",
    "placeholder": "Enter State Name",
    "disabled": False
```

- In the same way you can generate many controls like `textbox, combobox, radio ...` and so on.

Where to write UI_Meta ?

- Your each model contains it's own different UI design to display. So we will write `UI_Meta` inside our models for each models file as shown below.

```
"""
This file is used to display class for state
"""
import reversion
from api.alias import AliasField
from api.models import BaseModel
from django.db import models


class State(BaseModel):
    """
    State class
    """
    sid = models.AutoField(primary_key=True)
    sname = models.CharField(max_length=50, db_column="sname")
```

```
description = models.CharField(max_length=50, db_column="description")
status = models.CharField(db_column="STATUS", max_length=30, default="Draft")
name = AliasField(db_column="sname", blank=True, null=True)

class UI_Meta:                                  <--- UI_Meta for each model(s) you cr
    """
    ui meta class for state model
    """
    ui_specs = {
        "listview": [
            "this value"
        ],
        "formview": [
            {
                "sectionlabel": "State",
                "cols": 2,
                "colComponent": [
                    {
                        "label": "State Name",
                        "decorator": "sname",
                        "type": "textbox",
                        "required": "true",
                        "message": "Enter State Name",
                        "id": "sname",
                        "placeholder": "Enter State Name",
                        "disabled": False
                    },
```

```
},
{
    "label": "Description",
    "decorator": "description",
    "type": "textbox",
    "required": "true",
    "message": "Enter Description",
    "id": "description",
    "placeholder": "Enter Description",
    "disabled": False
},
{
    "label": "Created By",
    "decorator": "created_by",
    "type": "textbox",
    "required": "true",
    "message": "Created By !",
    "id": "created_by",
    "placeholder": "This is Created By :",
    "disabled": True
},
{
    "label": "Created Date ",
    "decorator": "created_date",
    "required": "true",
    "message": "Created Date ",
    "placeholder": "Select Date",
```

```
    "required": "true",
    "message": "Created Date ",
    "placeholder": "Select Date",
    "type": "date",
    "id": "created_date",
    "dateFormatList": "YYYY-MM-DDTHH:mm:ss",
    "disabled": True
},
{
    "label": "Updated By ",
    "decorator": "last_updated_by",
    "type": "textbox",
    "required": "true",
    "message": "Last Updated By !",
    "id": "last_updated_by",
    "placeholder": "This was last Updated By:",
    "disabled": True
},
{
    "label": "Last Updated : ",
    "decorator": "last_updated_date",
    "required": "true",
    "message": "Last Updated Date",
    "placeholder": "Select Date",
    "type": "date",
    "id": "last_updated_date",
    "dateFormatList": "YYYY-MM-DDTHH:mm:ss",
    "disabled": True
```

```
                "disabled": True
            },
            {
                "label": "Last Updated : ",
                "decorator": "last_updated_date",
                "required": "true",
                "message": "Last Updated Date",
                "placeholder": "Select Date",
                "type": "date",
                "id": "last_updated_date",
                "dateFormatList": "YYYY-MM-DDTHH:mm:ss",
                "disabled": True
            },
        ]
    }
]
}

reversion.register(State)
```

- You also need to add addtional `Created By` `Created Date` `Updated By` `Last Updated` into `UI_Meta` to maintain the transaction information for every model(s).

What about foreign key fields ?

- The framework itself handles foreign key UI generation. Here is the UI_Meta when your model(s) contains foreign key field.
- The example shown here is based on `State` and `City` model. Where `City` is having foreign key of `State`

```
"""
THis file is used to represent city master table
"""
import reversion
from api.alias import AliasField
from api.models import BaseModel
from django.db import models

from masters.models.State import State


class City(BaseModel):
    """
    City model class
    """
    cid = models.AutoField(primary_key=True, db_column="cid")
    cname = models.CharField(max_length=50, db_column="cname")
```

```
class City(BaseModel):
    """
    City model class
    """
    cid = models.AutoField(primary_key=True, db_column="cid")
    cname = models.CharField(max_length=50, db_column="cname")
    sid = models.ForeignKey(State, on_delete=models.CASCADE, db_column="sid", related
    status = models.CharField(db_column="STATUS", max_length=30, default="Draft")
    name = AliasField(db_column="cname", blank=True, null=True)

    class UI_Meta:
        """
        UI meta class for city
        """
        ui_specs = {
            "listview": [
                "this value"
            ],
            "formview": [
                {
                    "sectionlabel": "City",
                    "cols": 2,
                    "colComponent": [
                        {
                            "label": "City Name",
                            "decorator": "cname",
                            "type": "textbox",
```

```
{
    "label": "City Name",
    "decorator": "cname",
    "type": "textbox",
    "required": "true",
    "message": "Enter City Name",
    "id": "cname",
    "placeholder": "Enter City Name",
    "disabled": False
},
{
    "label": "State ID",
    "decorator": "sid",
    "type": "select",
    "required": "true",
    "message": "Select State ID",
    "validator": None,
    "validateStatus": None,
    "id": "sid",
    "placeholder": "Please Select State ID",
    "option": "sid",
    "linked": "yes",
    "link_api": "State",                    <--- FOREIGN KEY
    "primary_key_field": "sid",
    "foreign_table_attribute_name": "name",
    "disabled": False,
},
```

```
                    "foreign_table_attribute_name": "name",
                    "disabled": False,
                },
                ....
                ....
            ]
        }
    ]
}

reversion.register(City)
```

What about dependent foreign key fields ?

- Scenario may happen where you are having foreign key field which is depended onto some another foreign key field. For example state combobox is populated based on country selection.
- Here is the sample UI_Meta for dependent combobox

```
{
    "label": "Country",
    "decorator": "country_id",
    "type": "select",
```

- Here is the sample UI_Meta for dependent combobox

```
{
    "label": "Country",
    "decorator": "country_id",
    "type": "select",
    "required": "true",
    "message": "Select Country",
    "validator": None,
    "validateStatus": None,
    "id": "country_id",
    "placeholder": "Please Select Country",
    "option": "country_id",
    "linked": "yes",
    "link_api": "Country",
    "primary_key_field": "country_id",
    "foreign_table_attribute_name": "name",
    "disabled": False,
},
{
    "label": "State",
    "decorator": "sid",
    "type": "select",
    "required": "true",
    "message": "Select State",
    "validator": None,
```

```
        "foreign_table_attribute_name": "name",
        "disabled": False,
    },
    {
        "label": "State",
        "decorator": "sid",
        "type": "select",
        "required": "true",
        "message": "Select State",
        "validator": None,
        "validateStatus": None,
        "id": "sid",
        "placeholder": "Please Select State",
        "option": "country_id",
        "linked": "yes",
        "dropdown_dependent_field": [
            {
                "apiCall": "Country",
                "decorator": "country_id"
            },
        ],
        "link_api": "state",
        "primary_key_field": "sid",
        "foreign_table_attribute_name": "name",
        "disabled": False,
    },
```

Understanding UI_Meta

UI_Meta Understanding

- So now you know how framework generates UI based on UI_Meta. But there are few things you need to clarify to understand the meaning of each key value pair used in UI_Meta.

```
ui_specs = {
    ...
}

The Dictionary which returns the metadata for dynamic grid and form generation

"listview": [
 "this value"
],

The metadata information for the Gridview generation for the model data
(mostly remains same for every model)
```

```
"formview": [
    "sectionlabel" : "State",
    "cols": 2,
    "colComponent":[
        ...
    ]
 ],
```

The metadata information for the form generation for modals (i.e for Edit and Add), which includes the individual fields information (i.e types , formats, messages

```
"sectionlabel" : "State"
```

Section Label for the individual master screens for the UI

```
"cols": 2
```

Defined the columns (i.e sections) into which form will be generated.

```
"colComponent":[
    ...
]
```

The list of metadata, each with the respected model fields.

```
"colComponent":[
    ...
]
```

The list of metadata, each with the respected model fields.

Note :- Only this section is changed respected to the models defined, rest of the Ui_Meta structure remains same in most of the cases.

UI_Meta for textbox

- Now let's understand the UI_Meta for a single textbox generation. Here is the sample code for textbox generation using UI_Meta. Further we will understand it in brief.

```
{
    "label": "State Name",
    "decorator": "sname",
    "type": "textbox",
    "required": "true",
    "message": "Enter State Name",
    "id": "sname",
    "placeholder": "Enter State Name",
    "disabled": False
```

```
    "disabled": False
},

"label": "State Name",

The label describes the field label on the UI form.

"decorator": "sname",

For the Data mapping in React side (Frontend),

Note :- Should be Same as the field name (case - sensitive)

"type": "textbox",

The type of the field to be used according to the django fields for the model

Note :- Suggested Mapping Attached at the end of the document

"required": "true",

Normal UI function for the fields

"message": "Enter State Name!",
```

```
"message": "Enter State Name!",
```

Message to be displayed on the fields when left blank on submit

```
"id": "sname",
```

The unique identity for the field to be identify on frontend

Note :- Should be same as the decorator (Case- sensitive)

```
"placeholder": "Please enter State Name!",
```

The placeholder for the fields

```
"disabled": False,
```

Html field attribute, (mostly “False”)

UI_Meta for combobox

- Now let's understand the UI_Meta for a single combobox generation. Here is the
  oobox generation using UI_Meta. Further we will understand it in

UI_Meta for combobox

- Now let's understand the UI_Meta for a single combobox generation. Here is the sample code for combobox generation using UI_Meta. Further we will understand it in brief.


```
{
    "label": "State ID",
    "decorator": "sid",
    "type": "select",
    "required": "true",
    "message": "Select State ID",
    "validator": None,
    "validateStatus": None,
    "id": "sid",
    "placeholder": "Please Select State ID",
    "option": "sid",
    "linked": "yes",
    "link_api": "State",
    "primary_key_field": "sid",
    "foreign_table_attribute_name": "name",
    "disabled": False,
},


"option": "client_id",
```

```
"option": "client_id",

Should be same as the decorator

"linked": "yes",

Refers to the ForeignKey field of the model

"link_api": "State",

Refers to the model, on which the ForeignKey is referenced

Note :- The value should be same as mentioned in the Django ForeignKey field (case -

"primary_key_field": "sid",

The PrimaryKey field name for the ForeignKey referenced model

Note :- (Case - Sensitive)

"foreign_table_attribute_name": "name",

Key-value remains same for every ForeignKey field
```

UI_Meta for dependent combobox

- Now let's understand the UI_Meta for a dependent combobox generation. Here is the sample code for dependent combobox generation using UI_Meta. Further we will understand it in brief.

```
{
    "label": "State",
    "decorator": "sid",
    "type": "select",
    "required": "true",
    "message": "Select State",
    "validator": None,
    "validateStatus": None,
    "id": "sid",
    "placeholder": "Please Select State",
    "option": "country_id",
    "linked": "yes",
    "dependent": "yes",
    "dropdown_dependent_field": [
        {
            "apiCall": "Country",
            "decorator": "country_id"
        },
    ],
    "link_api": "state",
```

```
            },
        ],
        "link_api": "state",
        "primary_key_field": "sid",
        "foreign_table_attribute_name": "name",
        "disabled": False,
    }


    "dropdown_dependent_field": [
        {
            "apiCall": "Country",
            "decorator": "country_id"
        },
    ],

    Add this key when there is the requirement of Second Dropdown data
    from the Value of first dropdown (when there is dependency dropdowns)
```

What about UI_Meta for name and status fields in model(s) ?

- There is no need to write the json for `name` and `status` fields for all the model(s), as we don't need these fields in the form.

```
"dropdown_dependent_field": [
    {
        "apiCall": "Country",
        "decorator": "country_id"
    },
],
```

Add this key when there is the requirement of Second Dropdown data from the Value of first dropdown (when there is dependency dropdowns)

What about UI_Meta for name and status fields in model(s) ?

- There is no need to write the json for `name` and `status` fields for all the model(s), as we don't need these fields in the form.

← WRITTING UI_META

API CALLS →

API Calls (i.e For reference purpose only)

GET All records

```
http://localhost:8000/projectname/master/<str:app>.<str:model>/list
```

```
http://localhost:8000/projectname/master/appname.modelname/list?page=1
```

Example

```
http://localhost:8000/recogate/master/masters.country/list?page=1
```

Description

To get all the records from the particular model.

GET Single records

```
http://localhost:8000/projectname/master/<str:app>.<str:model>/<int:id>
```

GET Single records

```
http://localhost:8000/projectname/master/<str:app>.<str:model>/<int:id>
```

```
http://localhost:8000/projectname/master/appname.modelname/1
```

Example

```
http://localhost:8000/recogate/master/masters.country/1
```

Description

To get single record from the particular model.

POST Insert record

```
http://localhost:8000/projectname/master/<str:app>.<str:model>
```

```
http://localhost:8000/projectname/master/appname.modelname
```

Example

```
http://localhost:8000/recogate/master/masters.country
```

Example

```
http://localhost:8000/recogate/master/masters.country
```

Description

To create a new record.

PUT Update record

```
http://localhost:8000/projectname/master/<str:app>.<str:model>/<int:id>
```

```
http://localhost:8000/projectname/master/appname.modelname/1
```

Example

```
http://localhost:8000/recogate/master/masters.country/1
```

Description

To update existing record.

DELETE record

```
http://localhost:8000/projectname/master/<str:app>.<str:model>/<int:id>
```

```
http://localhost:8000/projectname/master/appname.modelname/1
```

Example

```
http://localhost:8000/recogate/master/masters.country/1
```

Description

To delete existing record.

OPTIONS Metadata for listview

```
http://localhost:8000/projectname/master/<str:app>.<str:model>?set=listview
```

```
http://localhost:8000/projectname/master/appname.modelname?set=listview
```

Example

```
http://localhost:8000/recogate/master/masters.country?set=listview
```

Example

```
http://localhost:8000/recogate/master/masters.country?set=listview
```

Description

To get listview information for a model.

OPTIONS Metadata for formview

```
http://localhost:8000/projectname/master/<str:app>.<str:model>?set=formview
```

```
http://localhost:8000/projectname/master/appname.modelname?set=formview
```

Example

```
http://localhost:8000/recogate/master/masters.country?set=formview
```

Description

To get formview information for a model.