GET Version history

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

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

Example

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

Description

To get transaction information for a model.

POST Excel export

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

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

Example

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

Example

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

Description

To export models data into excel file.

GET Search

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

```
http://localhost:8000/projectname/master/appname.modelname/list?page=1&search=123
```

Example

```
http://localhost:8000/recogate/master/masters.country/list?page=1&search=123
```

Description

To achieve search functionality for a model.

GET Filter

Description

To filter data based on different types of field including `Numeric` , `Text` , `Date` or in combind also.

Numeric

```
http://localhost:8000/recogate/master/masters.country/list?page=1&country_code=123&filter=1
```

Text

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

Date

```
http://localhost:8000/recogate/master/masters.country/list?page=1&created_date=2020-12-02&filter=1
```

Combind

```
http://localhost:8000/recogate/master/masters.country/list?page=1&country_id=1&country_name=icontains_IND&filter=1
```

```
http://localhost:8000/recogate/master/masters.country/list?page=1&country_code=123&filter=1
```

Text

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

Date

```
http://localhost:8000/recogate/master/masters.country/list?page=1&created_date=2020-12-02&filter=1
```

Combind

```
http://localhost:8000/recogate/master/masters.country/list?page=1&country_id=1&country_name=icontains_IND&filter=1
```

Note :-

Replace `appname` with the app name in which you have declared models.

RBAC Introduction

What is RBAC ?

- RBAC is known as `Role Based Access Control` . Django itself provides rich functionality to handle users and permission into django projects.

Why we need RBAC ?

- In your project the situation may happen where the functionality is depends onto the types of user. For e.g We can say that admin is having different permissions and access to feature compare to a regular user. So to maintain such scenarios we need RBAC module which will handles all the users and permissions for your project.

How RBAC Works ?

- The `RBAC` works based onto the `group` level permissions. In which you will create several permissions and based on those permissions the framework will act according to that.

Configure Side NavBar

Initial Side NavBar

- Initially the `sidebar` is empty. Because the framework only displays sidebar sections if user's group is having appropriate permission for a section.

What Next ?

- Before we move ahead. Let us assume that currently our `EXTRA USER META` inside

What Next ?

- Before we move ahead. Let us assume that currently our `EXTRA_USER_META` inside `settings.py` will look like as follow.

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

- Now suppose we want to add `reports` section inside our side navbar. So first of all we will add `reports` section into our `EXTRA_USER_META` . Once you add `reports` section our `EXTRA_USER_META` will look like as follow :

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

```
EXTRA_USER_META = {
    'MASTERS': [
        {
            'app': 'masters',
            'title': 'MASTER'
        }
    ],
    'Reports': [                    <--- Section which we want to add into side n
        {
            'screen': 'SendBCReport',   <--- screen 1
            'type': 'subsection',
            'title': 'Confirmations Sent'
        },
        {
            'screen': 'RequestBCReport', <--- screen 2
            'type': 'subsection',
            'title': 'Confirmations Requested'
        }
    ],
}
```

- Although the navbar still looks empty. Because we haven't assigned any permission related to `reports` section until now. As soon as we assign permissions our section will be displayed.

Create Section Permission

1. Open your django admin panel. Navigate to `Permissions` inside `AUTHENTICATION AND AUTHORIZATION` .

2. Click on `+ Add` to create new permission. Which can be later on assigned to group.

3. For `Content type` click on `+` sign.

Add content type

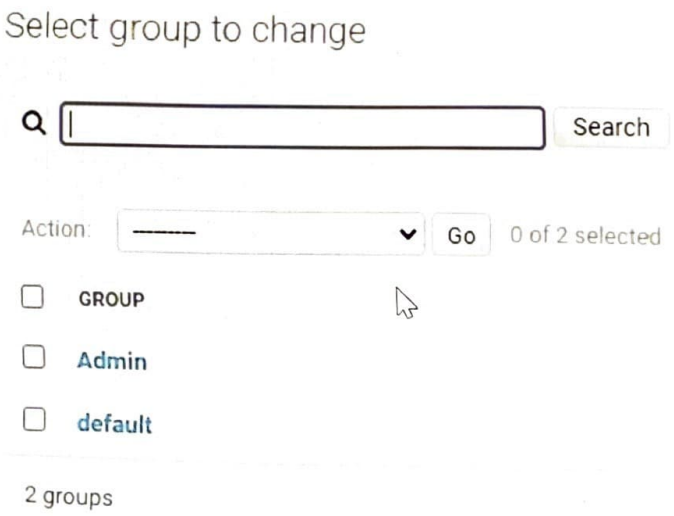
4. Click on `Save` to create your `Content type` and `Add Permission` .

Assign Permission To Group(s)

- After creation of permission we need to assign permission to group so that user related to that particular group can see the section into side navbar.

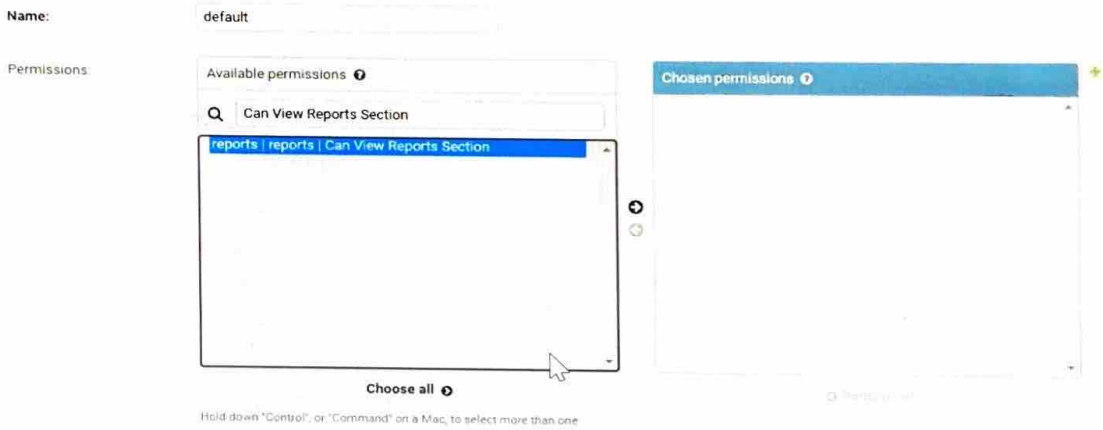
Assign Permission To Group(s)

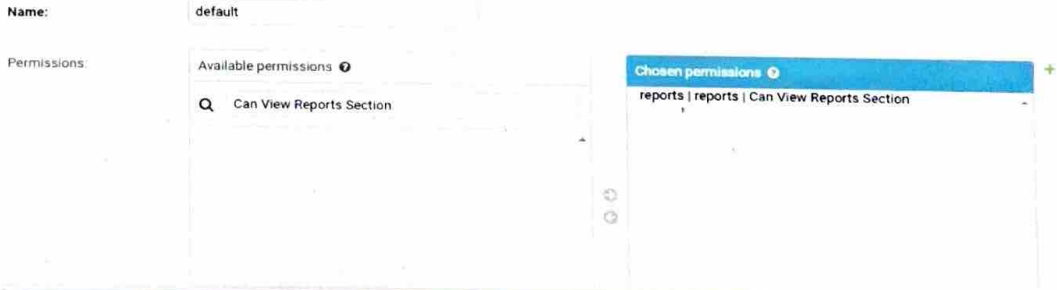
- After creation of permission we need to assign permission to group so that user related to that particular group can see the section into side navbar.

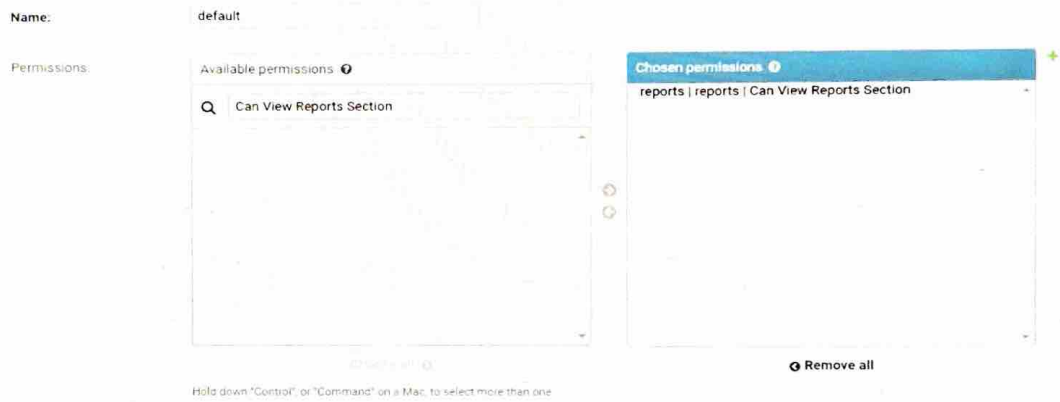


- Choose your group in which you want to assign permission. Here in the example we have taken `default` group.

- Choose your group in which you want to assign permission. Here in the example we have taken `default` group.







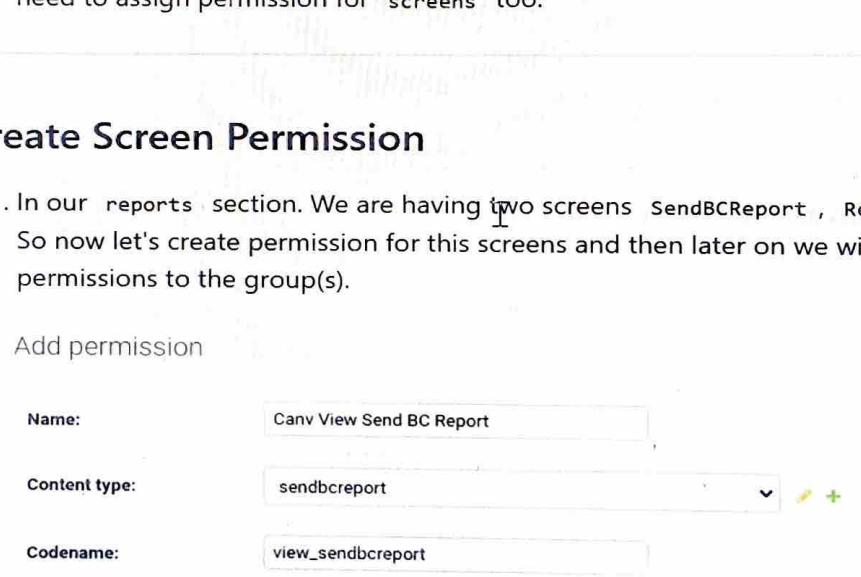
- Once we have completed the above steps and come back to `UI` part. After refreshing the screen you can see the `reports` section. Which will look like as follow :

Why Section Is Empty ?

- Although we have assigned a section permission to our group(s). Why is section still looks empty on expand ?
- The reason is we have only assigned the permission for `section` . In the same way we need to assign permission for `screens` too.

Create Screen Permission

1. In our `reports` section. We are having two screens `SendBCReport` , `RequestBCReport` . So now let's create permission for this screens and then later on we will assign this permissions to the group(s).



Add permission

Name: Canv View Send BC Report

Content type: sendbcreport

Codename: view_sendbcreport

Delete

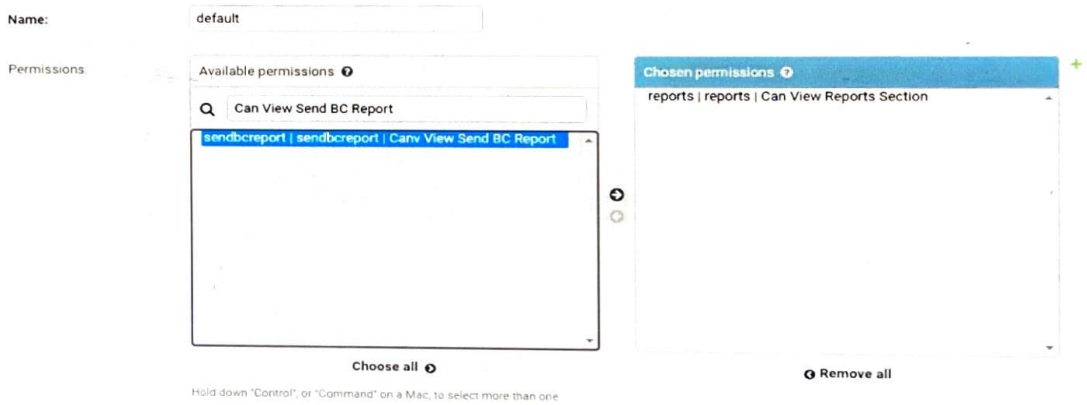
Add content type

App label: sendbcreport

Python model class name: sendbcreport

2. Click on `Save` to create your `Content type` and `Add Permission` .

3. Let's assign screen to our `default` group.



4. Once we have completed the above steps. We are able to see `RBAC` section inside our navbar.

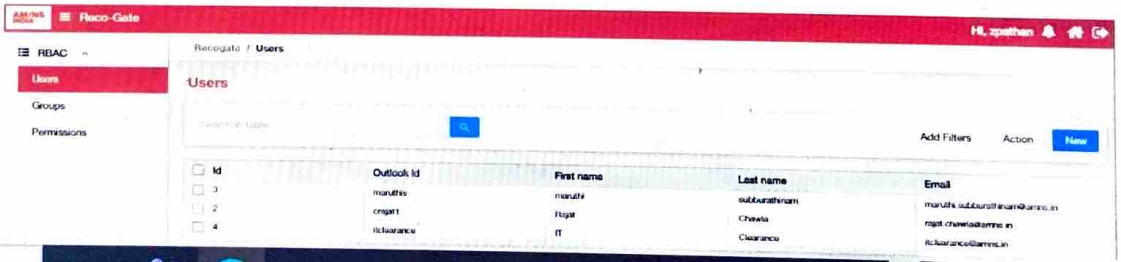
Conclusion

- Follow same procedure for every sections and subsections (screens) added into `EXTRA_USER_META` in `settings.py` of your project.

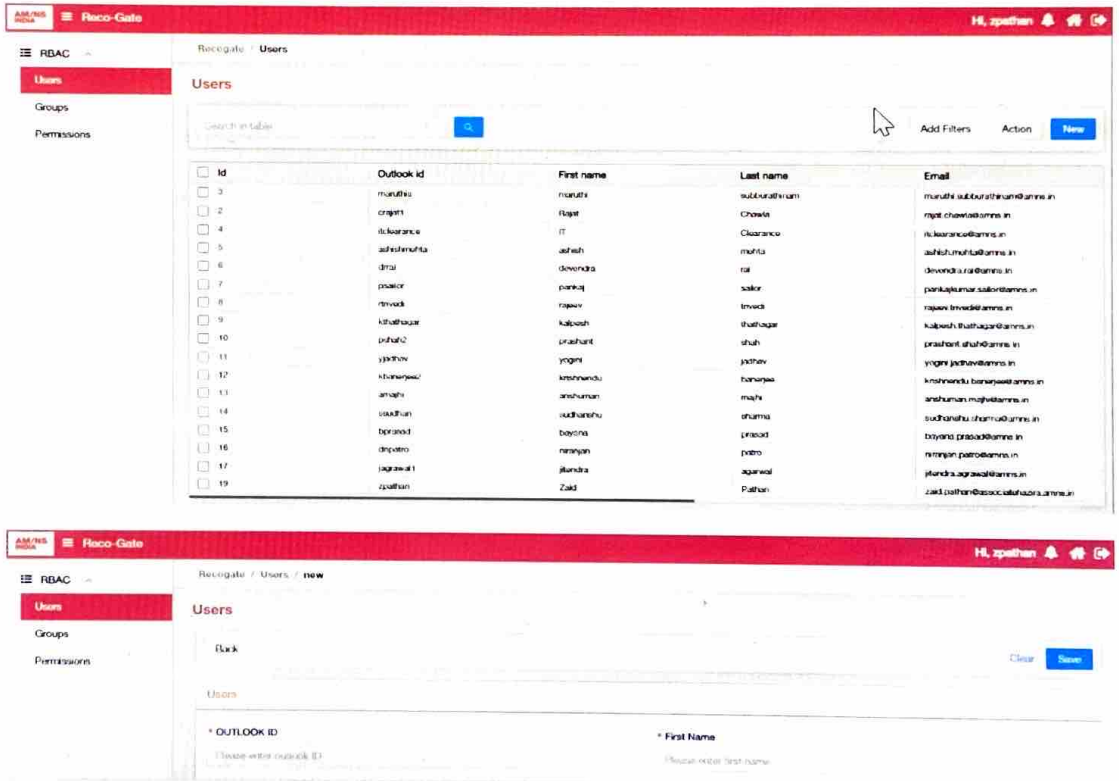
RBAC Panel

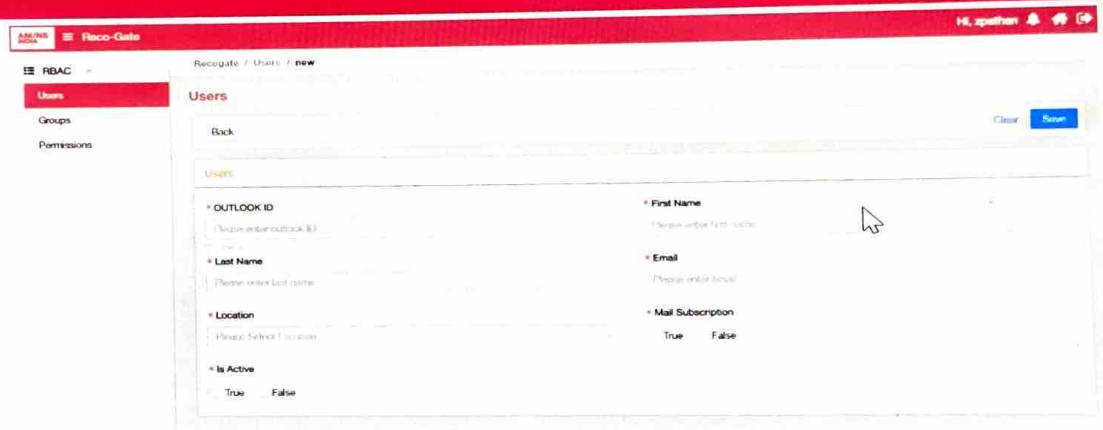
What is RBAC Panel ?

- Basically `RBAC` panel acts as an admin panel which handles `user(s)` registration, update user(s), assign roles to the user(s) and permission to the roles (groups). In simple words it handles `authorization matrix` of your project.
- Since in the installation steps we have already added `RBAC` app into our `settings.py` file. The `RBAC` contains functionality to manage your `users` , `groups` , `permissions` .
- Now we need to add some permissions into our django project so that `RBAC` module can be seen into side bar panel.



- Now we need to add some permissions into our django project so that `RBAC` module can be seen into side bar panel.





Features Of RBAC

- Side NavBar Sections
- Users ListView
- Users Creation, Modification, Deletion, Updation
- Groups ListView
- Groups Creation, Modification, Deletion, Updation
- Assign user(s) to group(s)
- Assign permission(s) to group(s)

Configuration

1. Open `settings.py` of your django project and append the following lines into `EXTRA_USER_META` section.

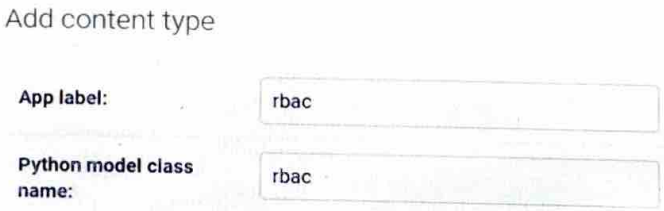
```
EXTRA_USER_META = {
    'MASTERS': [
        {
            'app': 'masters',
            'title': 'MASTER'
        }
    ],
    'RBAC':[
        {
            'screen':'Users',
            'type':'subsection',
            'title':'Users'
        },
        {
            'screen':'Groups',
            'type':'subsection',
            'title':'Groups'
        },
        {
            'screen':'Permissions',
            'type':'subsection',
```

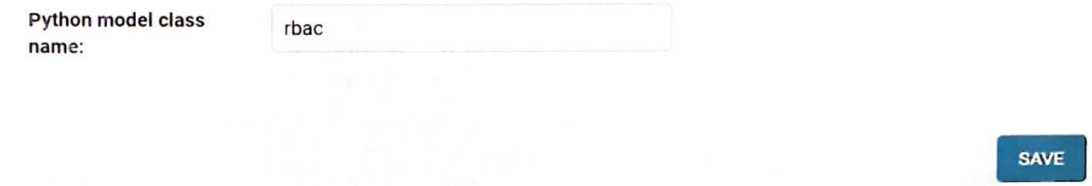
```
            },
            {
                'screen':'Permissions',
                'type':'subsection',
                'title':'Permissions'
            }
        ],
        ...
}
```

2. Open admin panel of your django project. And navigate to the `permissions` section of `AUTHENTICATION AND AUTHORIZATION` .
3. Click on `+ Add` sign. To add new permission.
4. Here you can see `Name` , `Content type` , `Codename` . Now we are going to add permissions as follow :
    i. For RBAC Section
    ii. For Users Screen
    iii. For Groups Screen
    iv. For Permissions Screen

Creating permission for RBAC Section

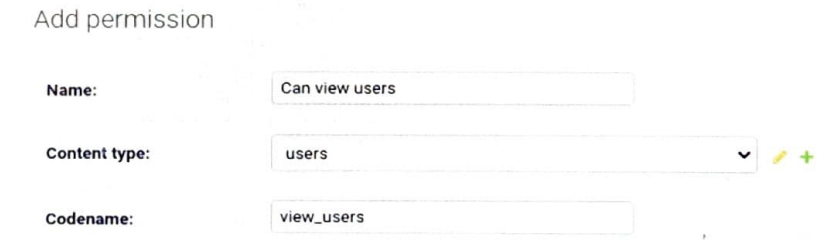
- Click on + sign for content type.





- Click on `Save` for content type and then click on `Save and add another` .

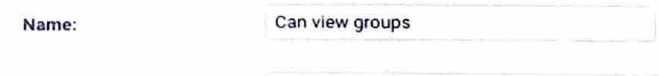
Creating permission for Users Screen



- Click on + sign for content type.

- Click on `Save` for content type and then click on `Save and add another` .

Creating permission for Groups Screen



Creating permission for Groups Screen

Add permission

Name: Can view groups

Content type: groups

Codename: view_groups

- Click on + sign for content type.

Change content type

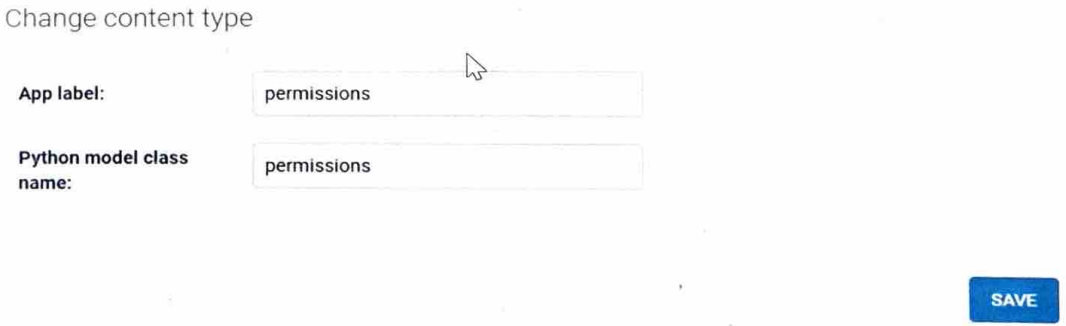
App label: groups

Python model class name: groups

- Click on `Save` for content type and then click on `Save and add another` .

Creating permission for Permissions Screen

- Click on + sign for content type.



- Click on `Save` for content type and then click on `Save and add another` .

Release Notes

Release Notes : v0.1.2 :

- Login with LDAP & added custom reverse username/passwd login
- Added filters & search for models, with specifying fields in "settings.py"
- appmeta OPTIONS call, handles the whole section/subsection view in UI sidebar with
- Contains first version of rbac & workflow

Release Notes : v1.0.0 :

- Added support for sections with space "i.e EXTRA_USER_META in settings.py".
- Login with LDAP & added custom reverse username/passwd login
- Moved commons and workflow constants in "settings.py", (i.e Addition of new constan
- api.BaseModel contains one added fields - "is_approved"
- api.BaseModel contains itself "status" field, i.e no need to mention in every maste
- Fixed the invalid login credentials status codes
- Contains latest version of rbac, workflow & commmons

Release Notes : v1.0.1 :

- Added master data listview pagination support
- Now all the Listview calls i.e search, filter & all data are handled by one api vie
- Now all the results for i.e search, filter are paginated.

Release Notes : v1.0.2 :

- Fixed get_paginated_queryset import error.
- Added support for columnfilter, through listview metadata call. (i.e queryparam : "
- Now whole subsection view, can be configured with "view" permissions from DB.
- Restricted Nested serilization for Transactional table data, via framwork url call.
- Restricted serialization of all fields for one-to-many relation, (i.e nested serial

Release Notes : v1.0.4 :

- Added support for extra Datetime, Date, Timefield formats , irrespective of BaseMod
- Added 3 constants for the same i.e DATE_INPUT_FORMAT, TIME_INPUT_FORMAT, DATE_TIME_
- Improvements and bug fixes.

Release Notes : v1.0.5 :

```
- Added support for POST object return hook for user defined function, which returns
- Added one constant i.e POST_SAVE_OBJ_RETURN_HOOKS, for the same, user can specify t

    follows :-
        - 'state' : 'complaints.views.save_location'
        i.e key :- model name (str), for the hook to be called.
            value :- the dot seperated string for the path, where user function is lc

        Note :-
            - Mandatory to call, (serial_data.save()), as user will get serialized da
            ex :-
            def save_location(serial_data):
                created_obj = serial_data.save()
                created_obj.location = "location name from views"
                created_obj.save()
                return created_obj

- Improvements and bug fixes.
```

Release Notes : v1.0.6 :

Release Notes : v1.0.6 :

- Added Restriction hook for AD authentication, for already registered users, need to

```
CHECK_USER_REGISTRATION = True
```

- If True, the successfull login, returns the data from user defined serializer, in

```
USER_PROFILE_MODEL_DETAILS = {
    'app' : 'users',
    'model' : 'userprofile',
    'search_on_field' : 'outlook_id'
}

USER_PROFILE_SERIALIZER = 'users.serializers.UserProfileSerializer' (i.e give loc
```

Note:-
- if flag = True, Mandatory to give details for two subsequent constants.

- Added generic date filters, with following use cases :
  - before
  - after
  - after - before
  - exact

- example urls format for date filters with prefix :-

```
- example urls format for date filters with prefix :-
    - http://localhost:8000/master/masters.lineitemcategory/list?page=1&created_d
    - http://localhost:8000/master/masters.lineitemcategory/list?page=1&created_d
    - http://localhost:8000/master/masters.lineitemcategory/list?page=1&line_item
    i.e here line_item_type is the FK.

- Note :-
    - date format should be in str & yyyy-mm-dd

- Now, to support foreign key filters, you can add following in FOREIGN_KEY_UI_NA

for type date as :
    'lineitemcategory' :{
    'field' : 'line_item_type__created_date',
    'type' : 'date'
    }
```

Release Notes : beta v2.0.0 :

1. Now Supports multiple app auto generation other than only masters :
    - Added one constant in settings.py file as :

```
AUTO_GENERATION_SECTIONS = ['MASTERS', 'BC']
```

Release Notes : beta v2.0.0 :

1. Now Supports multiple app auto generation other than only masters :
   - Added one constant in settings.py file as :

     AUTO_GENERATION_SECTIONS = ['MASTERS', 'BC']

     i.e Write all the sections here which are to be auto generated

   - Example :
     suppose you want two sections to be auto generated write EXTRA_USER_META as :

```
EXTRA_USER_META = {
    'MASTERS': [
        {
            'app': 'masters',
            'title': 'MASTER'
        }
    ],
    'BC': [
        {
            'app': 'balance_confirmations',
            'title': 'Balance Confirmation'
        }
    ],}
```

```
        AUTO_GENERATION_SECTIONS = ['MASTERS', 'BC']

    - Inbuilt django config "INSTALLED_APPS" should be divided as :

        DEPENDENCY_APPS = []
        PROJECT_APPS = []

        And finally,  INSTALLED_APPS = DEPENDENCY_APPS + PROJECT_APPS

    - Note :-
        - The CRUD operations of masters will be supported.

2. Added Master Files Upload to local server as well as MOSS :

    Configs to configure in settings.py file :

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

    - For a Filefield to use in Master model in compatibility to framework use field

        is_filefield = models.BooleanField(db_column='IS_FILES', default=False) (i.e
    - Added one model MasterFiles in api app, to handle file data
```

3. Added Excel Import Feature :-

- url :- 'master/<app>.<model>/import'
- Config to be done in settings.py file i.e example :-

```
EXCEL_IMPORT_CONFIGS = {
'mstdutydrawbackbenefit': {
'year': 0,
'month': 1,
'cap_value': 2,
'type': 3,
'product': 4
}
```


Release Notes : v2.0.2 :

- Fixed MasterFiles bug, i.e is_file_field flag added
- Added Support for all Foreignkey referenced fields with operators of type 'integer'
- Added combination filter i.e Searrch & apply filter'on search results.
- Added pre-hooks, for search & filters, for comaptibility with trans models & serial
- Following are the usage example & configurations :
- Added one constant in settings.py, where to mention the paths i.e dotted string for

```
- TRANS_FILTER_QS_SERIALIZER = {
    'careers.jobseeker': ('careers.trans_filter_support.return_jobseeker_qs', 'career
}
```

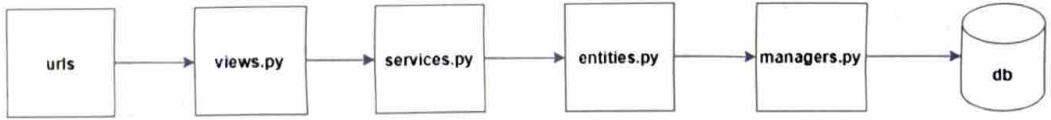
```
i.e key : app.model (lower case)
        value : tuple with 0th index as path to queryset function & 1th index as path

queryset funtion :

maintain seperate file, to have functions, which returns queryset for search & filter
Following is the mandatory syntax i.e structure example
```

```
def return_jobseeker_qs(request):
    return JobSeeker.objects.filter(first_name__icontains='esh')

- manadatory to have one argument i.e request, which is rest_framework Http request o
- No rules on funtion naming convention :)

trans serializer :
Trans model serializer, through whcih filtered queryset will be returned

- Improvements & Bug Fixes.
```

Transaction Apps

What is transaction apps ?

- Apps which are not generated based on `UI_Meta` is considered as transaction apps.
- To work with transaction apps first of all we need to understand 3 tier architecture.



URLS

- `urls.py` *is responsible for handling all the entrypoints for our api calls.*
- *Sample* `urls.py` *is shown below.*

```
from django.urls import path
```

- Sample `urls.py` is shown below.

```
from django.urls import path
from .views import BooksView

urlpatterns = [
    path('books', BooksView.as_view(), name="books_get_post"),
    path('books/<int:pk>', BooksView.as_view(), name="books_patch_delete"),
]
```


VIEWS

- Our `views.py` will contains all the `APIViews` to handle our rest-api calls like GET, POST, PATCH, DELETE.
- Sample `views.py` is shown below.

```
from django.shortcuts import render
from rest_framework.views import APIView
from masters.logger_directory import api_logs
from .services import BooksService
from rest_framework.response import Response
from rest_framework.status import HTTP_500_INTERNAL_SERVER_ERROR
from rest_framework.authentication import SessionAuthentication, TokenAuthentication
```

```python
from django.shortcuts import render
from rest_framework.views import APIView
from masters.logger_directory import api_logs
from .services import BooksService
from rest_framework.response import Response
from rest_framework.status import HTTP_500_INTERNAL_SERVER_ERROR
from rest_framework.authentication import SessionAuthentication, TokenAuthentication
from rest_framework.permissions import IsAuthenticated

logs = api_logs

class BooksView(APIView):
    authentication_classes = [SessionAuthentication, TokenAuthentication]
    permission_classes = [IsAuthenticated]

    """
    To handle API request fo books
    """

    def get(self, request, **kwrgs):
        """
        To get all books
        """
        try:
            logs.debug(f"-- GET REQUEST BY -- {request.user.username} -- VIEWS.PY --

            if kwrgs.get('pk') is not None:
```

```
def get(self, request, **kwrgs):
    """
    To get all books
    """
    try:
        logs.debug(f"-- GET REQUEST BY -- {request.user.username} -- VIEWS.PY --

        if kwrgs.get('pk') is not None:
            return BooksService().get_books_by_pk(kwrgs.get('pk'))

        elif kwrgs.get('pk') is None:
            return BooksService().get_all_books()
    except Exception as ex:
        logs.debug(f"--- EXCEPTION OCCURED -- {ex}")
        return Response(ex, status=HTTP_500_INTERNAL_SERVER_ERROR)


def post(self, request):
    """
    To add new books
    """
    try:
        logs.debug(f"-- POST REQUEST BY -- {request.user.username} -- VIEWS.PY --
        return BooksService().add_new_book(request)
    except Exception as ex:
        logs.debug(f"--- EXCEPTION OCCURED -- {ex}")
        return Response(ex, status=HTTP_500_INTERNAL_SERVER_ERROR)
```

```
def patch(self, request, *args, **kwargs):
    """
    To update books
    """
    try:
        logs.debug(f"-- PATCH REQUEST BY -- {request.user.username} -- VIEWS.PY -
        return BooksService().update_book(request, kwargs.get('pk'))
    except Exception as ex:
        logs.debug(f"--- EXCEPTION OCCURED -- {ex}")
        return Response(ex, status=HTTP_500_INTERNAL_SERVER_ERROR)


def delete(self, request, *args, **kwargs):
    """
    To soft delete books
    """
    try:
        logs.debug(f"-- DELETE REQUEST BY -- {request.user.username} -- VIEWS.PY
        return BooksService().delete_book(kwargs.get('pk'))
    except Exception as ex:
        logs.debug(f"--- EXCEPTION OCCURED -- {ex}")
        return Response(ex, status=HTTP_500_INTERNAL_SERVER_ERROR)
```

SERVICES

- Our `services.py` will help us to communicate with the `entity` and returns `rest_framework` response based on api request.
- Sample `services.py` is shown below.

```
import json
from rest_framework import status
from masters.logger_directory import api_logs
from .serializers import BooksSerializer, BooksDetailSerializer
from rest_framework.response import Response
from .models import Books
from rest_framework.status import HTTP_200_OK, HTTP_201_CREATED, HTTP_400_BAD_REQUEST
    HTTP_404_NOT_FOUND, HTTP_500_INTERNAL_SERVER_ERROR
from django.db import transaction
from reversion import revisions as reversion
from datetime import datetime
from .entities import BookProxy

logs = api_logs

class BooksService:
    """
    This class is responsible for handling GET-POST-PUT-DELETE db operations
    """
```

```python
class BooksService:
    """
    This class is responsible for handling GET-POST-PUT-DELETE db operations
    """

    def get_books_by_pk(self, pk):
        """
        To get book by pk
        """
        try:
            logs.debug(f"--- GET BOOK BY PK --")
            books = BookProxy.objects.get_book_by_pk(pk)
            res = BooksDetailSerializer(books).data
            return Response(res, status=HTTP_200_OK)
        except Exception as ex:
            logs.debug(f"-- EXCEPTION -- {ex}")
            return Response({'error':'internal server error'}, status=HTTP_500_INTERN


    def filter_books(self, query):
        """
        To filter books
        """
        try:
            logs.debug(f"--- FILTER BOOKS -- SERVICES.PY --")
            books = BookProxy.objects.filter_books(query)
            res = BooksDetailSerializer(books, many=True).data
```

```
        To filter books
        """
        try:
            logs.debug(f"--- FILTER BOOKS -- SERVICES.PY --")
            books = BookProxy.objects.filter_books(query)
            res = BooksDetailSerializer(books, many=True).data
            return Response(res, status=HTTP_200_OK)
        except Exception as ex:
            logs.debug(f"-- EXCEPTION -- {ex}")
            return Response({'error':'internal server error'}, status=HTTP_500_INTERN


    def get_all_books(self):
        """
        To get all books
        """
        try:
            logs.debug(f"--- GET ALL BOOKS --")
            books = BookProxy.objects.get_all_books()
            res = BooksDetailSerializer(books, many=True).data
            return Response(res, status=HTTP_200_OK)
        except Exception as ex:
            logs.debug(f"-- EXCEPTION -- {ex}")
            return Response({'error':'internal server error'}, status=HTTP_500_INTERN


    def add_new_book(self, request):
```

```
def add_new_book(self, request):
    """
    To create new books
    """
    try:
        logs.debug(f"--- CREATE NEW BOOK --")
        with reversion.create_revision():
            data = {}
            data = json.loads(request.data)
            data.update({'created_by':request.user.username})
            data.update({'created_date':datetime.now()})

            seri = BooksSerializer(data = data)
            if seri.is_valid():
                seri.save()
                return Response(seri.data, status=HTTP_201_CREATED)
            return Response(seri.errors, status=HTTP_500_INTERNAL_SERVER_ERROR)
    except Exception as ex:
        logs.debug(f"-- EXCEPTION -- {ex}")
        return Response({'error':'internal server error'}, status=HTTP_500_INTERN

def update_book(self, request, id):
    """
    To update book
    """
    try:
        logs.debug(f"--- GET BOOK FOR pk -- {id} --")
```

```
def update_book(self, request, id):
    """
    To update book
    """
    try:
        logs.debug(f"--- GET BOOK FOR pk -- {id} --")
        if Books.objects.filter(pk = id).exists():
            logs.debug(f"--- UPDATE book --")
            with reversion.create_revision():
                book = Books.objects.get(pk = id)
                data = {}
                data = json.loads(request.data)
                data.update({'last_updated_by':request.user.username})
                seri = BooksSerializer(book, data = data)
                if seri.is_valid():
                    seri.save()
                    return Response(seri.data, status=HTTP_201_CREATED)
                return Response(seri.errors, status=HTTP_500_INTERNAL_SERVER_ERRC
        else:
            return Response({'error':'Object does not exists'}, status=HTTP_404_N

    except Exception as ex:
        logs.debug(f"-- EXCEPTION -- {ex}")
        return Response({'error':'internal server error'}, status=HTTP_500_INTERN
```

```
def delete_book(self, id):
    """
    To soft delete books
    """
    try:
        logs.debug(f"--- GET BOOK FOR pk -- {id} --")
        if Books.objects.filter(pk = id).exists():
            logs.debug(f"--- SOFT DELETE BOOK --")
            with reversion.create_revision():
                books = Books.objects.get(pk = id)
                books.is_deleted = True
                books.is_active = False
                books.save()
                return Response(status=HTTP_200_OK)
        else:
            return Response({'error':'Object does not exists'}, status=HTTP_404_N

    except Exception as ex:
        logs.debug(f"-- EXCEPTION -- {ex}")
        return Response({'error':'internal server error'}, status=HTTP_500_INTERN
```


ENTITIES

ENTITIES

- Our `entities.py` or so called `proxy` is helpful to handle business logic. In simple words our `proxy` will contains all the bussiness logic required to provide needful output.
- Sample `entities.py` is shown below.

```
from .models.Books import Books
from .managers import BookManager

class BookProxy(Books):
    """
    To handle operations for book model
    """
    objects = BookManager()

    class Meta:
        proxy = True
```

MANAGERS

- Our `managers.py` will help us to communicate with db and can return queryset

MANAGERS

- Our `managers.py` will help us to communicate with db and can return `queryset` based on requirement.
- Sample `managers.py` is shown below.

```
from django.db import models
from django.db.models import Q

class BookManager(models.Manager):
    """
    Manager to handle db operations for book
    """

    def get_all_books(self):
        """
        To get all active records
        """
        return self.filter(is_deleted=False, is_active=True)


    def get_book_by_pk(self, pk):
        """
        To get single book by pk
        """
```

```
def get_book_by_pk(self, pk):
    """
    To get single book by pk
    """
    return self.get(pk=pk)


def filter_books(self, query):
    """
    To filter books by author and name
    """
    return self.filter(Q(bookname__contains=query) | Q(author__author__contains=q
```


Conclusion

- All the above code are for sample purpose. Actual file can be vary based on project requirement and specification.
- Above code can only used for understanding that how 3 tier architecture works for transaction apps.
- For more detailed info please refer to `AMNS CODING STANDARDS` section.