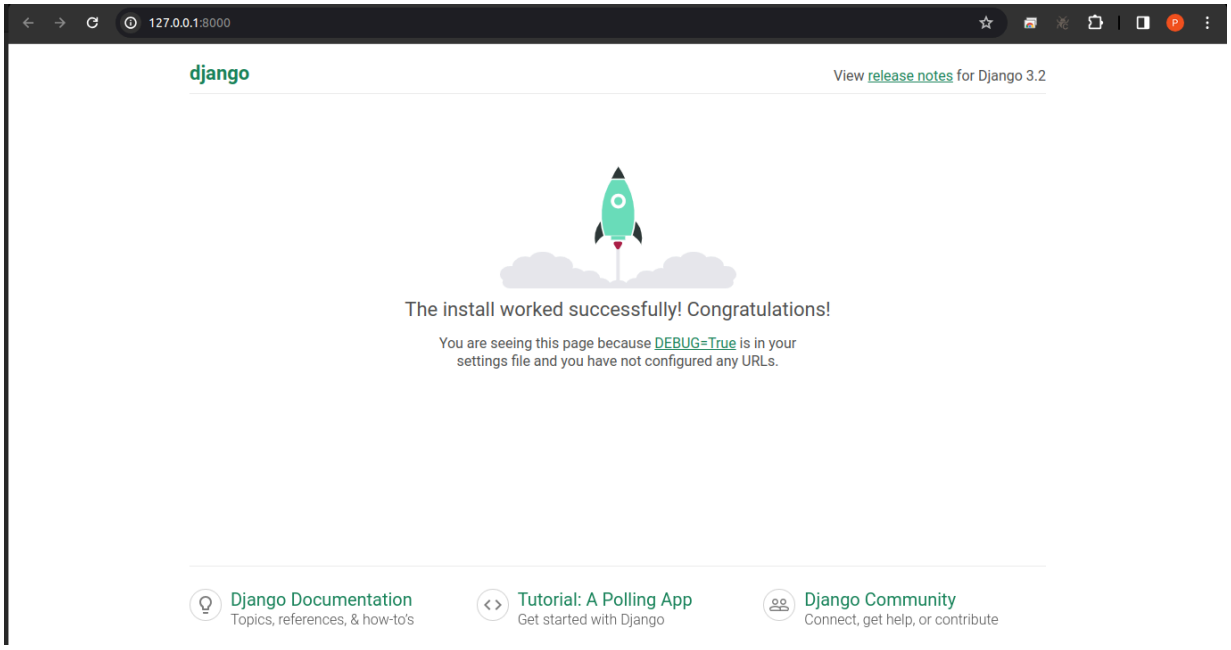
Week 1:

(Internship start date- 18 Dec 2023)

- installation of ubuntu operating system (DualBoot)
- Ubuntu setup
- Laptop Setup (https://gist.github.com/raunak-r/15fb871765e56ebe7a05bb6f42fc24d2)
- IDE's
  VSCode(https://www.youtube.com/watch?v=VqCgcpAypFQ&ab_channel=Academind)

  Pycharm (https://www.youtube.com/watch?v=56bPIGf4us0&ab_channel=TechWithTim)
- ubuntu-firstinstall.sh (installed .sh file includes. VSCode, Pycharm, Postgres etc.)
- Django installation (https://docs.djangoproject.com/en/3.2/intro/install/)
- Django Project



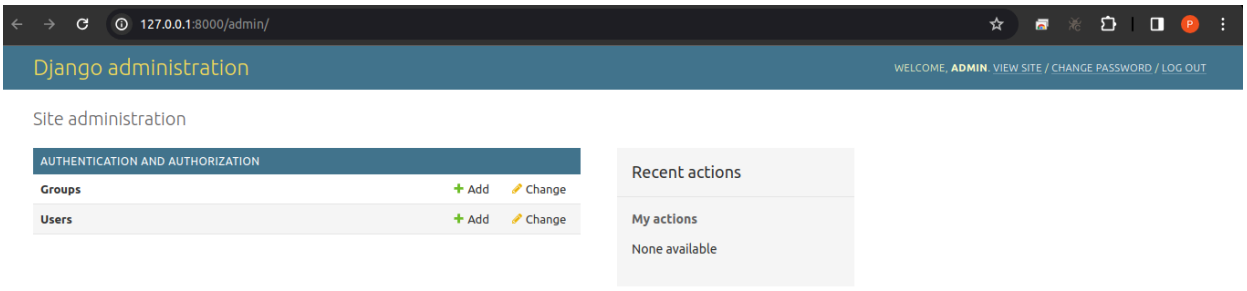
- First Django application (simple text printing appp, https://docs.djangoproject.com/en/3.2/intro/tutorial01/)

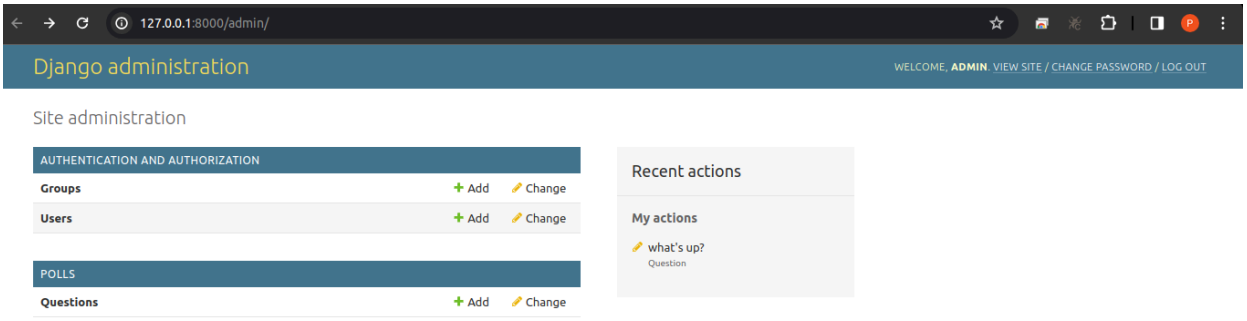
Week 2:

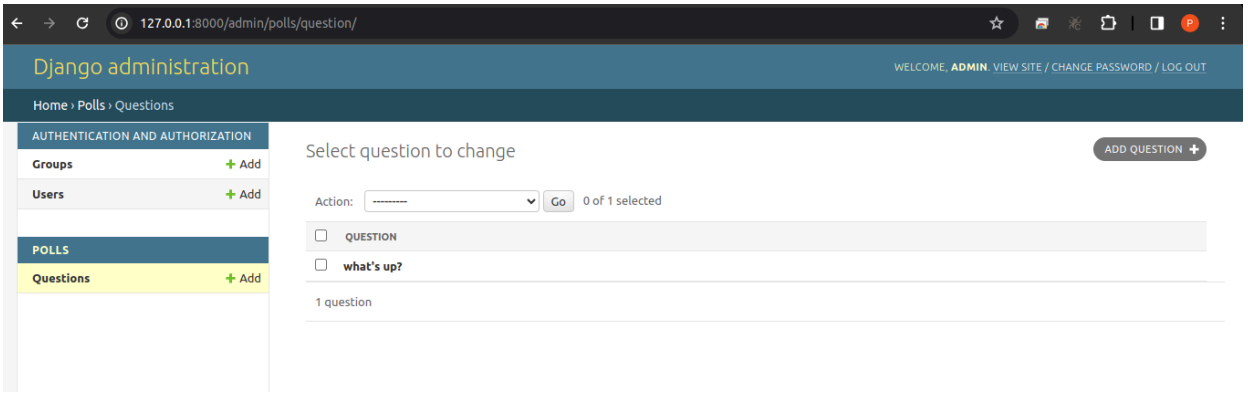
- DataBase setup (PostgresQL) (https://docs.djangoproject.com/en/3.2/intro/tutorial02/)

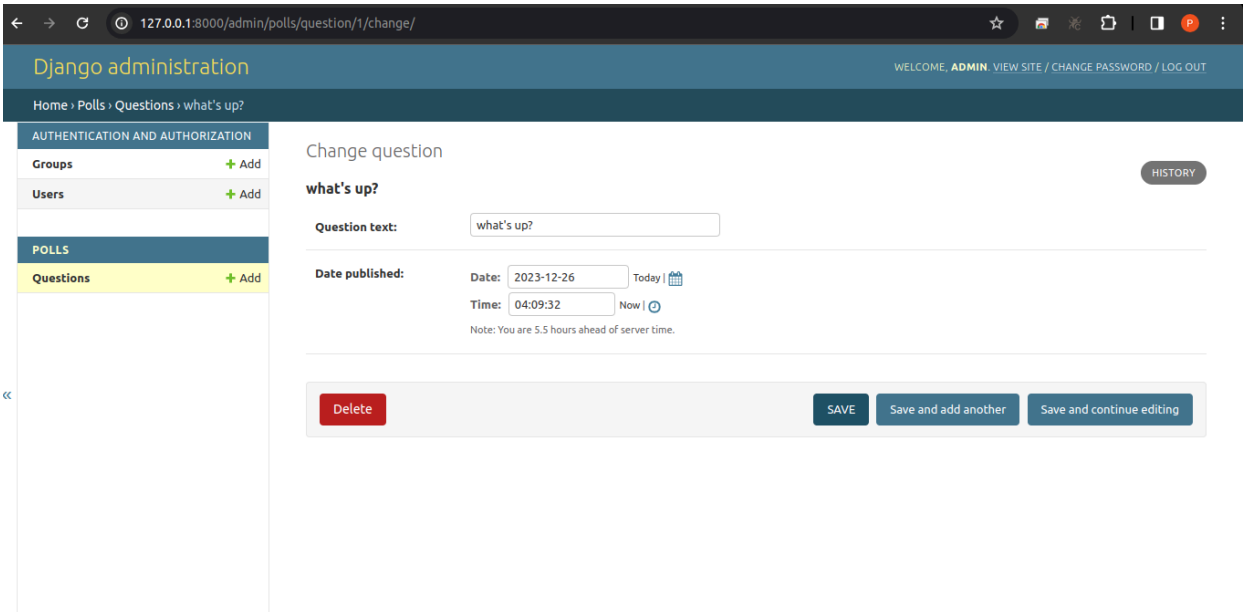
  Tables creation in database for Django application

- Django poll application(https://docs.djangoproject.com/en/3.2/intro/tutorial02/)

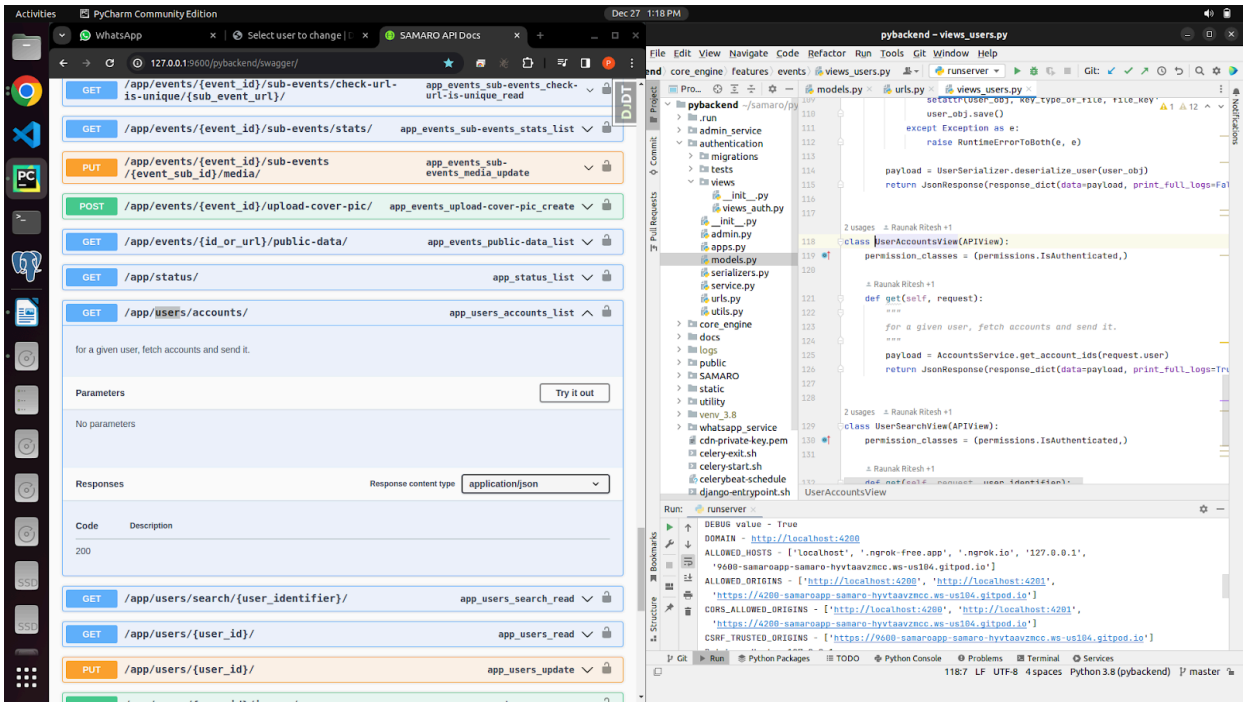






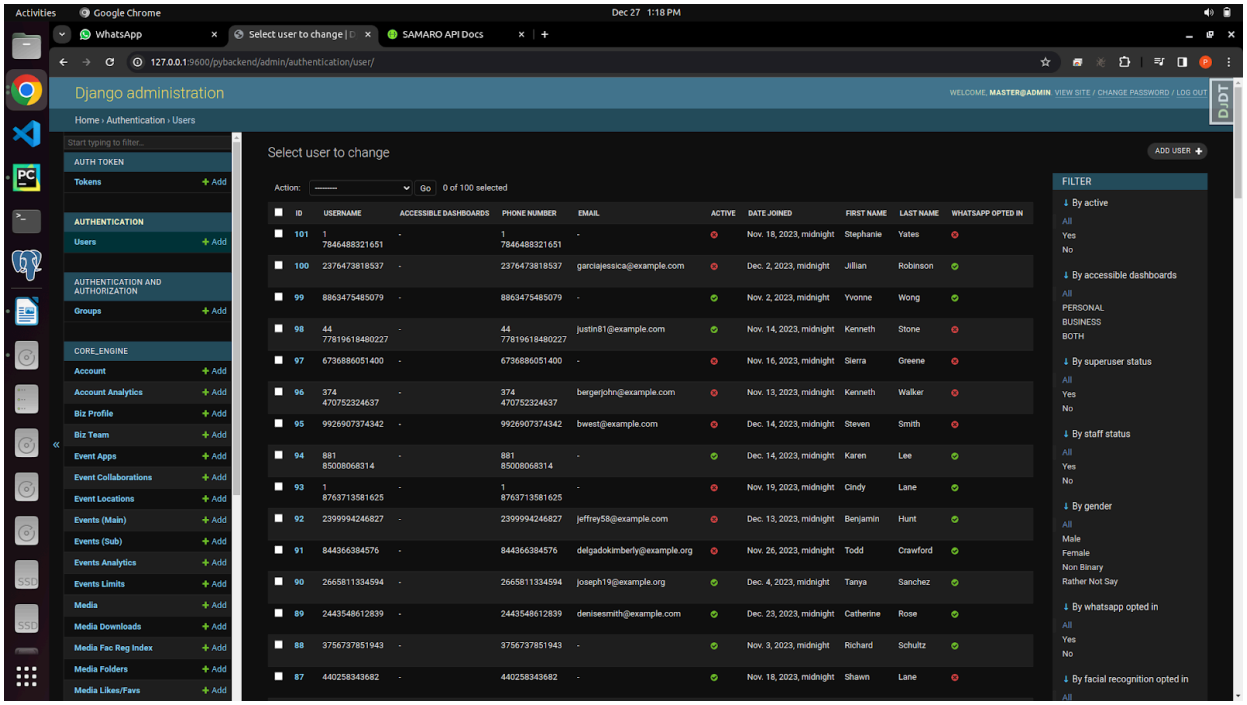


- Reading code / DB tables of SAMARO Project.

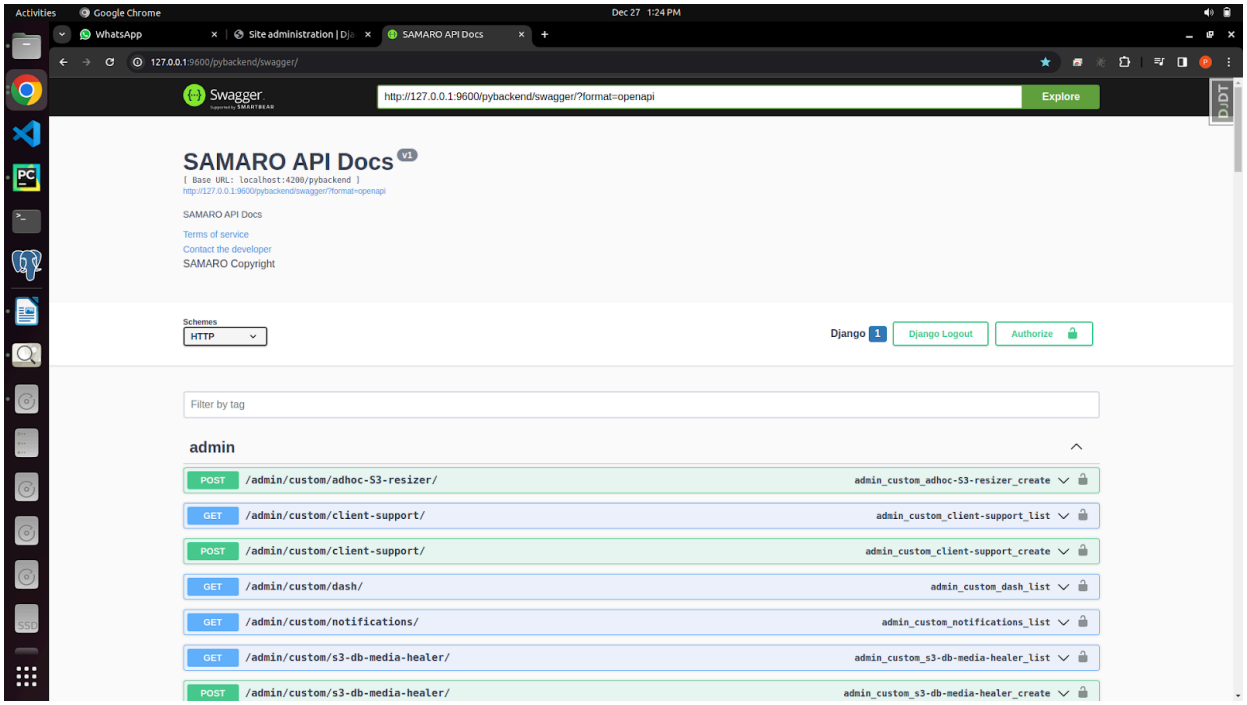


- Django administration (http://127.0.0.1:9600/pybackend/admin/)

(To open pybackend server>> pycharm>> samaro>>pybackend.py>>runserver.)



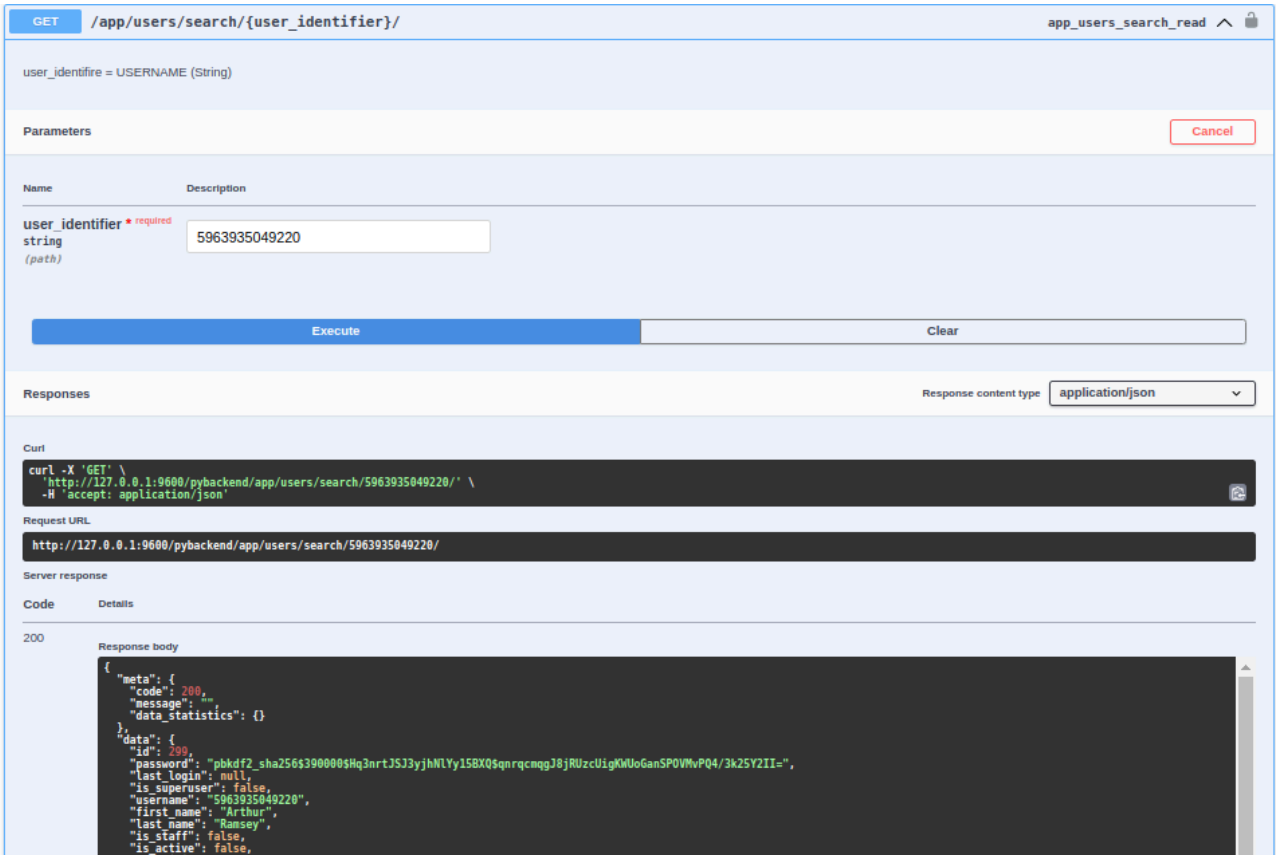
SAMARO API docs (Swagger)(http://127.0.0.1:9600/pybackend/swagger/)



Reading APIs

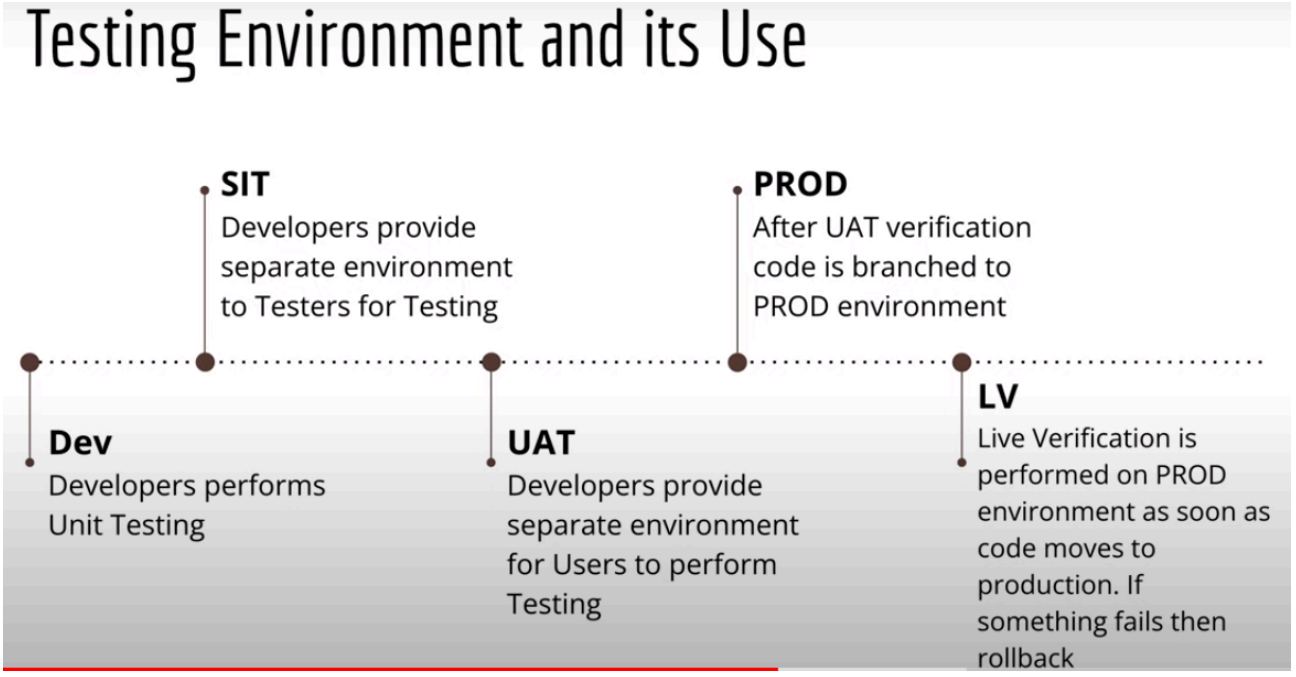
Parameter editing of APIs

```
2 usages  Raunak Ritesh +1 *
class UserSearchView(APIView):
    permission_classes = (permissions.IsAuthenticated,)

    Raunak Ritesh +1 *
    def get(self, request, user_identifier):
        """
            user_identifire = USERNAME (String)
        """
        try:
            user_obj = UserService.get_user(user_identifier, throw_error=False)
            return JsonResponse(response_dict(data=model_to_dict(user_obj)))
        except:
            raise RuntimeError("User not found")
```



Week 3:

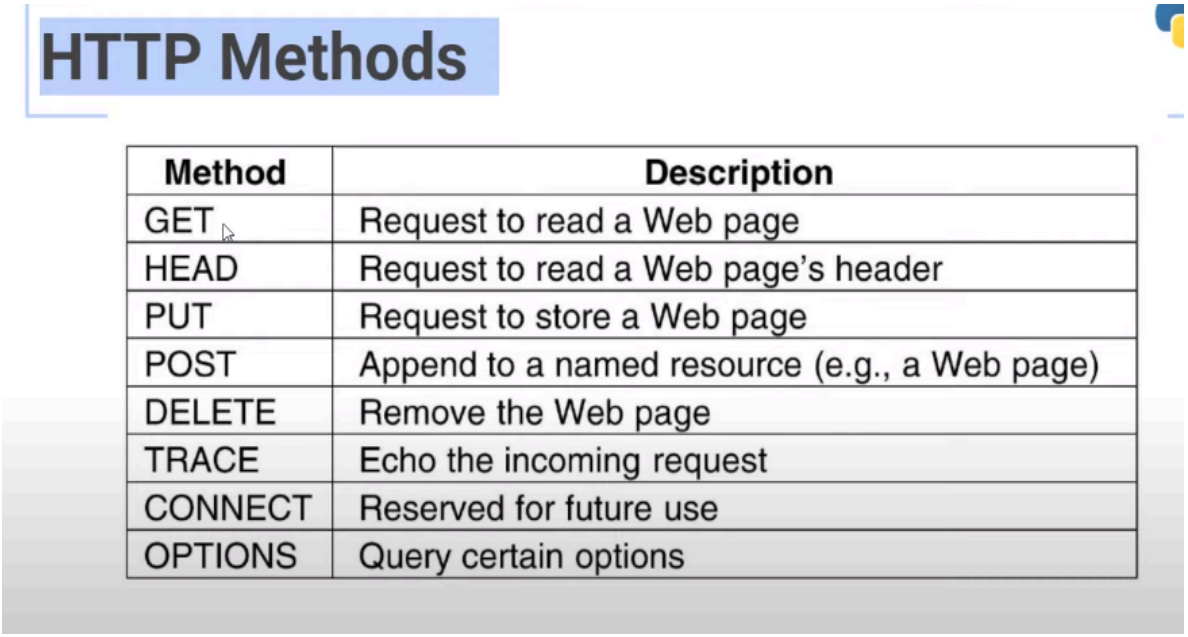
- Testing Environments



- commit code in https://github.com/samaroapp/samaro

```
(venv_3.8) prathmesh@prathmesh-HP-Laptop-15-da0xxx:~/samaro/pybackend$ git add .
(venv_3.8) prathmesh@prathmesh-HP-Laptop-15-da0xxx:~/samaro/pybackend$ git status
On branch be_swagger
Your branch is up to date with 'origin/be_swagger'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   authentication/views/views_auth.py
        modified:   core_engine/features/events/views_events.py
        modified:   core_engine/features/events/views_users.py
        modified:   public/views/open_apis.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../samaro/

(venv_3.8) prathmesh@prathmesh-HP-Laptop-15-da0xxx:~/samaro/pybackend$ git commit -m "add comment for GET EventURLView"
[be_swagger ec6adf21] add comment for GET EventURLView
 4 files changed, 12 insertions(+), 3 deletions(-)
(venv_3.8) prathmesh@prathmesh-HP-Laptop-15-da0xxx:~/samaro/pybackend$ git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 1.20 KiB | 1.20 MiB/s, done.
Total 14 (delta 11), reused 0 (delta 0), pack-reused 0
```

- GET /POST / PUT / DELETE Request



GET:

- Purpose: Retrieve data from the server.

- Usage: Requests data from a specified resource. Should only retrieve data and should have no other effect on the data.

POST:

- Purpose: Submit data to be processed to a specified resource.
- Usage: Used to send data to the server to create a new resource. Often used when submitting a form or when performing an action that results in a change to the server's state.

PUT:

- Purpose: Update data on the server.
- Usage: Replaces all current representations of the target resource with the request payload. It's idempotent, meaning multiple identical requests should have the same effect as a single request.

DELETE:

- Purpose: Remove data from the server.
- Usage: Deletes the specified resource. After a successful deletion, a 204 (No Content) response is returned.

- Spike Dashboard (daily basis monitoring for past 30 days)(https://app.clickup.com/t/86cu8z6cu)

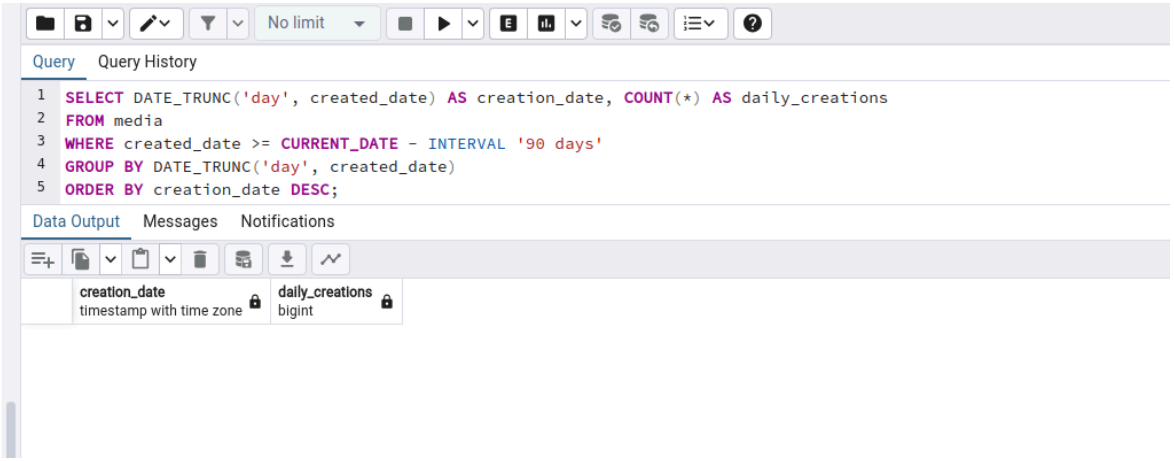
1. Daily user signups Query (By using data:- http://127.0.0.1:9600/pybackend/admin/authentication/user/?o=7)

```
SELECT DATE_TRUNC('day', date_joined) AS signup_date, COUNT(*) AS daily_signups
FROM authentication_user
WHERE date_joined >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY DATE_TRUNC('day', date_joined)
ORDER BY signup_date DESC;
```

(Refer : https://chat.openai.com/share/0337d646-65ec-4e72-a4fd-a7e9cb3fa2ae)

2. Daily media creations Query

```
SELECT DATE_TRUNC('day', created_date) AS creation_date, COUNT(*) AS daily_creations
FROM media
WHERE created_date >= CURRENT_DATE - INTERVAL '90 days'
GROUP BY DATE_TRUNC('day', created_date)
ORDER BY creation_date DESC;
```



3. Daily inactive signups.

SELECT DATE_TRUNC('day', date_joined) AS signup_date, COUNT(*) AS daily_inactive_signups

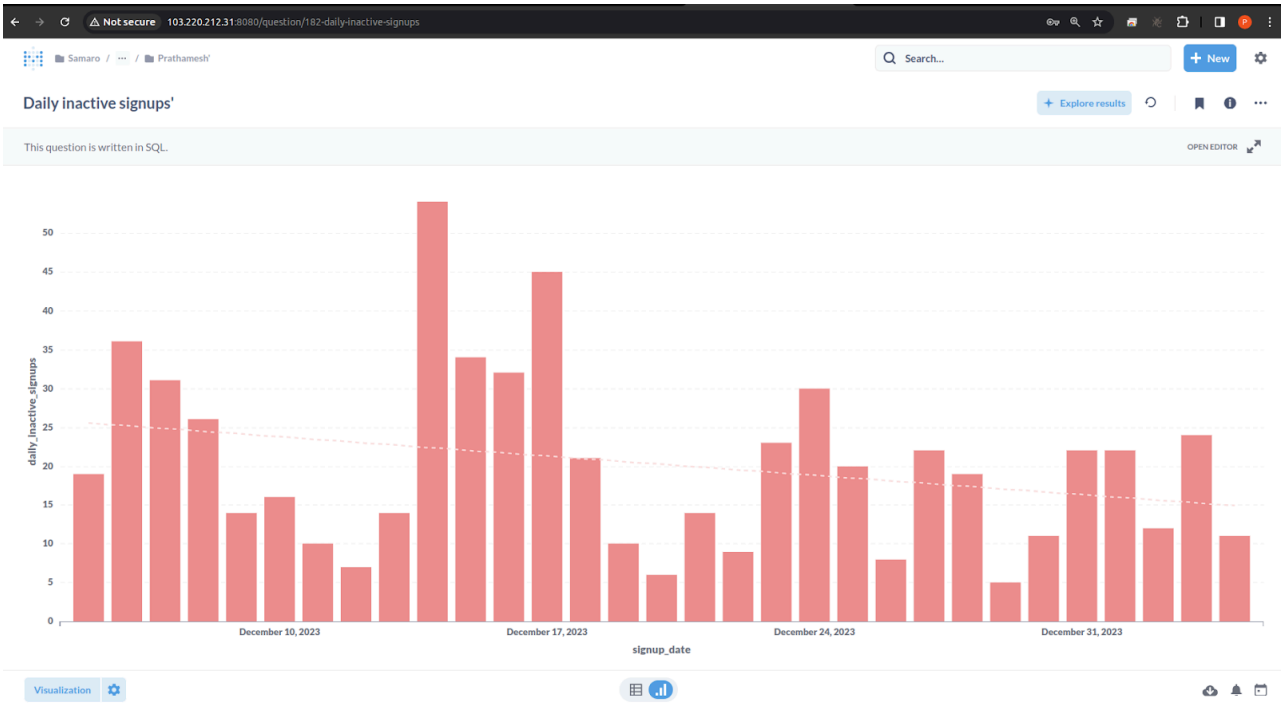
FROM authentication_user

WHERE date_joined >= CURRENT_DATE - INTERVAL '30 days'

AND is_active = FALSE

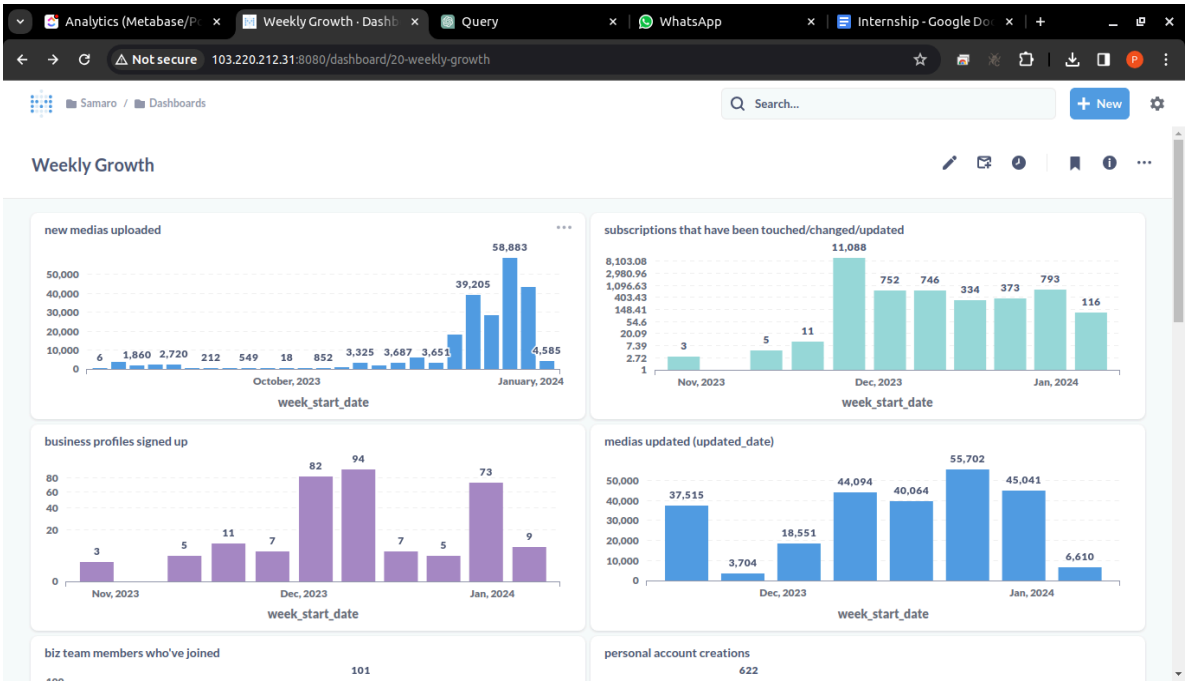
GROUP BY DATE_TRUNC('day', date_joined)

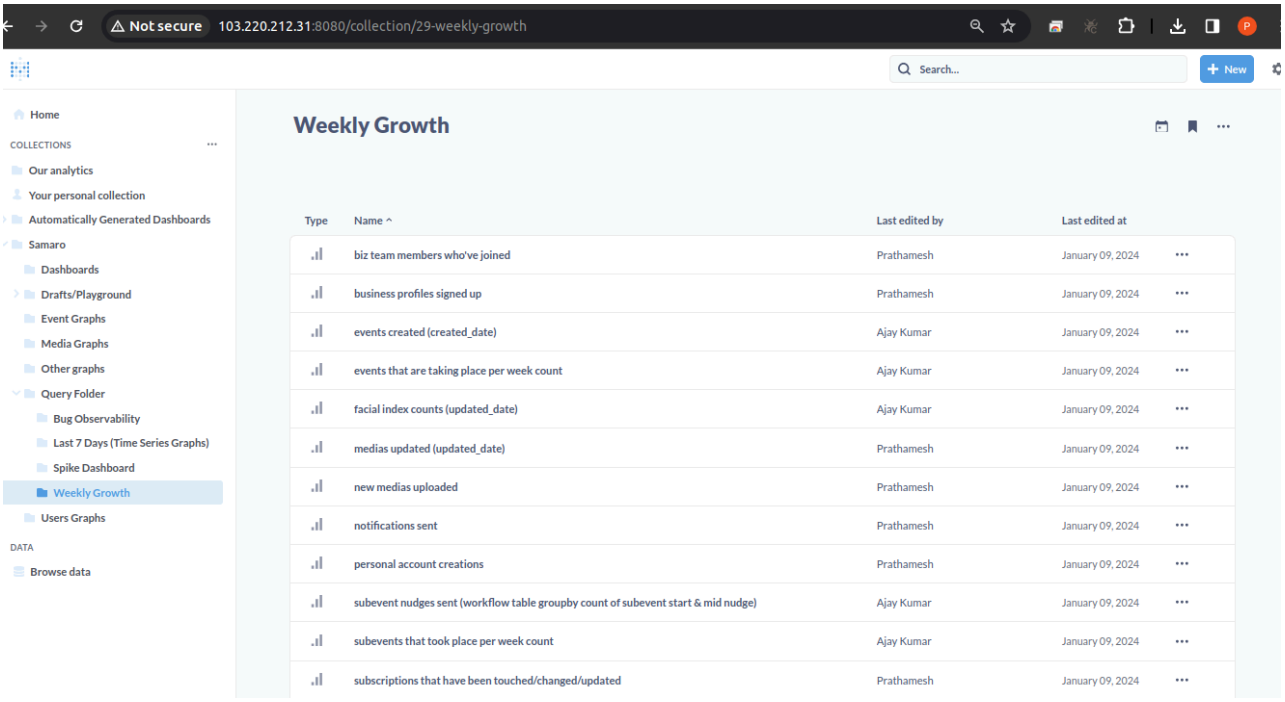
ORDER BY signup_date DESC;



Week 4:

- Weekly Growth Queries (Metabase)(http://103.220.212.31:8080/dashboard/20-weekly-growth)





- Frontend Setup

  nvm use 18

  npm install

  ng serve
- Practice ORM & select related & prefetch related.
  https://books.agiliq.com/projects/django-orm-cookbook/en/latest/

Week 5:

- 1. How to find the query associated with a queryset?
- 2. How to do OR queries in Django ORM?

```
def or_on_user(self):
    condition1 = Q(first_name__istartswith="R")
    condition2 = Q(last_name__istartswith="D")
    combined_condition = condition1 | condition2
    filtered_users = User.objects.filter(combined_condition)
    self.stdout.write(self.style.SUCCESS(f"OR Query Result: {filtered_users.values()}"))
```

- 3. How to do AND queries in Django ORM?

```
def and_on_users(self):
    condition1 = Q(first_name__istartswith='P')
    condition2 = Q(last_name__istartswith='D')
    filtered_users = User.objects.filter(condition1, condition2)
    self.stdout.write(self.style.SUCCESS(f"AND Query Result: {filtered_users.values()}"))
```

- 4. How to do a NOT query in Django queryset?

```
def not_on_users(self):
    excluded_value = "30"
    not_query_set = User.objects.filter(~Q(id=30))
    self.stdout.write(self.style.SUCCESS(f"NOT Query Result: {not_query_set.values()}"))
```

- 5. How to do union of two querysets from same or different models?

```
def union_users_event(self):
    queryset1 = User.objects.filter(Q(first_name__istartswith='A')).values('first_name', 'last_name')
    queryset2 = EventsMain.objects.filter(Q(event_name__istartswith='B')).values('event_name')
    resulting_queryset = list(queryset1) + list(queryset2)
    self.stdout.write(self.style.SUCCESS(f"Union Query Result: {resulting_queryset}"))
```

- 6. How to select some fields only in a queryset?

```
def select_only_name(self):
    filtered_users = User.objects.filter(first_name__istartswith='P').values('first_name', 'last_name')
    for user in filtered_users:
     print(f"First Name: {user['first_name']}, Last Name: {user['last_name']}")
```

- 7. How to do a subquery expression in Django?
- 8. How to filter a queryset with criteria based on comparing their field values

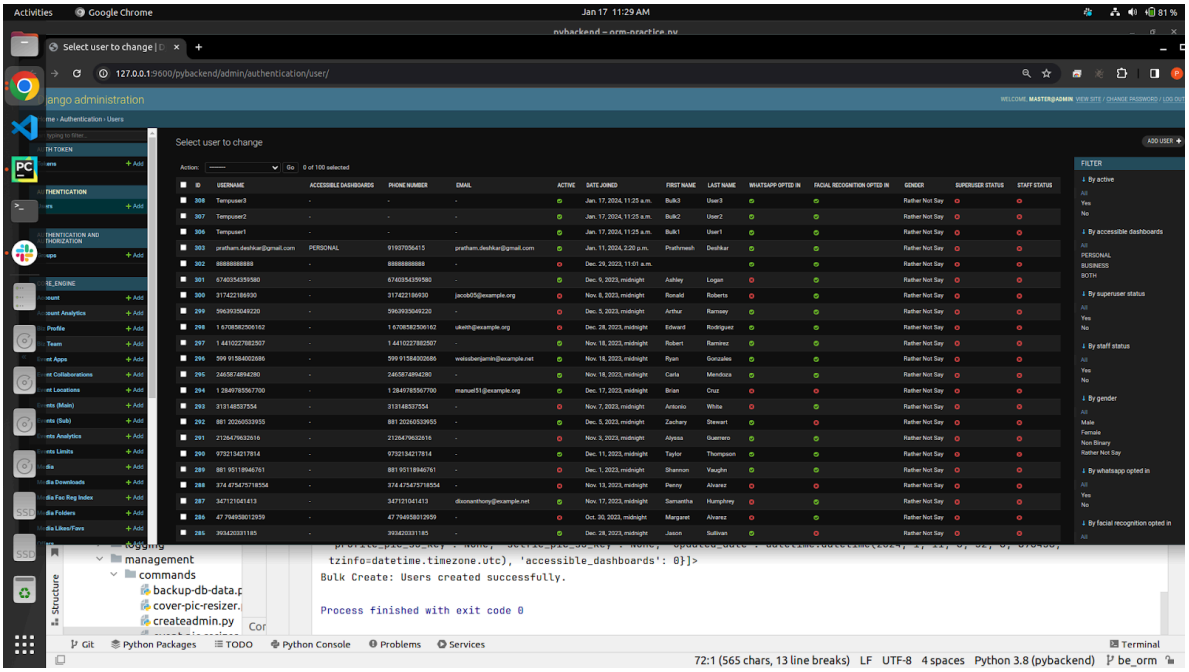
```
def compare_fields(self):
    filtered_users = User.objects.filter(first_name__icontains="P")
    self.stdout.write(self.style.SUCCESS(f"compare fields Query Result: {filtered_users.values()}"))
```

- 9. How to filter FileField without any file?
- 10. How to perform join operations in django ORM?

```
def create_users_in_bulk(self):
    users_to_create = [
        User(username='Tempuser1', first_name='Bulk1', last_name='User1'),
        User(username='Tempuser2', first_name='Bulk2', last_name='User2'),
        User(username='Tempuser3', first_name='Bulk3', last_name='User3'),
    ]
```

User.objects.bulk_create(users_to_create)

self.stdout.write(self.style.SUCCESS("Bulk Create: Users created successfully."))



- How to copy or clone an existing model object?

```
def create_users_in_bulk_with_clone(self):+

    users_to_create = [
        User(username='Tempuser1_clone', first_name='Bulk1', last_name='User1'),
        User(username='Tempuser2_clone', first_name='Bulk2', last_name='User2'),
        User(username='Tempuser3_clone', first_name='Bulk3', last_name='User3'),
    ]
    User.objects.bulk_create(users_to_create)
    self.stdout.write(self.style.SUCCESS("Bulk Create: Users created successfully."))

    # Clone an existing user
    original_user = User.objects.get(id=306)
    new_user = self.clone_user(original_user)

    self.stdout.write(self.style.SUCCESS(f"Clone: Original User - {original_user}, Cloned
User - {new_user}"))

# Method to clone a user
def clone_user(self, original_user):
    new_user = User(
        username=f"Clone_{original_user.username}",
        first_name=original_user.first_name,
        last_name=original_user.last_name,
    )
    new_user.save()
    return new_user
```

- How to ensure that only one object can be created?
- How to update denormalized fields in other models on save?
- How to perform truncate like operation using Django ORM

```
def truncate_users_table(self):
```

```
   User.objects.all().delete()
   self.stdout.write(self.style.SUCCESS("Truncate-like operation: Users table truncated."))
```

- ## 1. How to order a queryset in ascending or descending order?

```
def order_by_asc(self):
   ascending_users = User.objects.all().order_by('id')
   self.stdout.write(self.style.SUCCESS(f"Ascending Order Query Result:
{ascending_users.values()}"))

   # Descending order
   descending_users = User.objects.all().order_by('-id')
   self.stdout.write(self.style.SUCCESS(f"Descending Order Query Result:
{descending_users.values()}"))
```

- ## 2. How to order a queryset in case insensitive manner?

```
# Descending order with insensitive manner
         descending_users =
User.objects.all().order_by(Lower('first_name')).values_list('first_name', flat=True)
      self.stdout.write(self.style.SUCCESS(f"Descending Order Query Result:
{descending_users.values()}"))
```

- ## 3. How to order on two fields

```
queryset = User.objects.all().order_by('', '')     // for multiple fields
```

- ## 1. How to model one to one relationships?

class User(models.Model):

id= models.BigAutoField(primary_key=True)

username = models.CharField(max_length=255)

email = models.EmailField()

class EventsMain(models.Model):

id= models.BigAutoField(primary_key=True)

user = models.OneToOneField(User,on_delete=models.CASCADE)

event_name = models.CharField(max_length=255)

created_date = models.DateField()

def one_to_one_relationship(self):

user = User.objects.create(username='Prathmesh', email='example@example.com')

event = EventsMain.objects.create(user=user, event_name='Sample Event', created_date='2024-01-15')

- 2. How to model one to many relationships?

```
class User(models.Model):
    id= models.BigAutoField(primary_key=True)
    username = models.CharField(max_length=255)
    email = models.EmailField()
class EventsMain(models.Model):
    id= models.BigAutoField(primary_key=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    event_name = models.CharField(max_length=255)
    created_date = models.DateField()
def one_to_many_relationship(self):
    # Creating a user
    user = User.objects.create(username='Prathmesh', email='example@example.com')

    # Creating multiple events for the same user
 event1 = EventsMain.objects.create(user=user, event_name='Event 1', created_date='2024-01-15')
 event2 = EventsMain.objects.create(user=user, event_name='Event 2', created_date='2024-01-16')
  event3 = EventsMain.objects.create(user=user, event_name='Event 3', created_date='2024-01-17')
self.stdout.write(self.style.SUCCESS("One-to-Many Relationship: User and Events created successfully."))
```

- 3. How to model many to many relationships?

```
class User(models.Model):
    id = models.BigAutoField(primary_key=True)
    username = models.CharField(max_length=255)
    email = models.EmailField()
    events = models.ManyToManyField('EventsMain', related_name='users')  # Many-to-Many relationship

class EventsMain(models.Model):
    id = models.BigAutoField(primary_key=True)
    event_name = models.CharField(max_length=255)
    created_date = models.DateField()

def many_to_many_relationship(self):
        # Create users
        user1 = User.objects.create(username='User1', email='user1@example.com')
        user2 = User.objects.create(username='User2', email='user2@example.com')

        # Create events
        event1 = EventsMain.objects.create(event_name='Event1', created_date='2024-01-15')
        event2 = EventsMain.objects.create(event_name='Event2', created_date='2024-01-16')

        # Add events to users
        user1.events.add(event1, event2)
        user2.events.add(event1)
        # Query events for a user
        events_for_user1 = user1.events.all()
        events_for_user2 = user2.events.all()

    self.stdout.write(self.style.SUCCESS(f"Events for User1: {events_for_user1.values()}"))
        self.stdout.write(self.style.SUCCESS(f"Events for User2: {events_for_user2.values()}"))
```

- 4. How to include a self-referencing ForeignKey in a model

- 5. How to convert existing databases to Django models?

Week 6:

- GET, PUT, POST, DELETE API’s creation

(venv_3.8) prathmesh@prathmesh-HP-Laptop-15-da0xxx:~/samaro/pybackend/core_engine/features/subevents$ ***django-admin startapp folders***

creation of folder “folders” in pybackend>>core_engin>>subevents
1:39 PM

return JsonResponse(response_dict(data=pay

def delete(self, request, event_id):

"""

Delete Sub Event Associated with Media Files

"""

try:

Retrieve the subevent from the database

sub_event = EventSubs.objects.get(id=event_id, is_folder=True)

Filter the media items associated with the sub_event id

media_objects = Media.objects.filter(event_sub_id=event_id)

if not media_objects:

return JsonResponse(

{

shubh agarwal

load))

except Exception as e:

RuntimeErrorInternalOnly(e)

```
        return JsonResponse(response_dict(data=payload))
from SAMARO.constants import Collaborations, CustomExceptions
from utility.exception_handler import RuntimeErrorInternalOnly, RuntimeErrorToBoth
from utility.utils import response_dict
```

POST CODE using **`get_object_or_404`**

```
# def post(self, request, event_id):
#     """
#     Create a new sub-event based on event_id.
#     """
#     print("post")
#
#     try:
#         main_event = EventsMain.object.get(id=event_id)
#         new_sub_event = EventSubs.objects.create(
#             event_id=event_id,
#             sub_event_name=request.data.get("sub_event_name"),
#             description=request.data.get("description"),
#             is_folder=True,
#         )
#
#         return JsonResponse(
#             {"message": "POST request processed", "sub_event_id": id.id},
#             status=200,
#         )
#
#     except EventsMain.DoesNotExist:
#         return JsonResponse(
#             {"message": f"No matching EventMain found for id {event_id}"},
#             status=404,
```

```
      #    )
```

- created GET, PUT, POST and DELETE request for “Folders”

```
from django.http import HttpResponse, JsonResponse
from drf_yasg import openapi
from drf_yasg.utils import swagger_auto_schema
from rest_framework import status
from rest_framework.response import Response
from rest_framework.views import APIView

from core_engine.features.subevents.models import EventSubs
from core_engine.models import EventsMain, EventSubs, Media
from utility.utils import response_dict


class FolderView(APIView):
   def get(self, request, event_id):
       """
       GET records whose is_folder = True, Based on event_id
       """
       result_data = []
       try:
           sub_events_list = EventSubs.objects.filter(
               event_id_id=event_id, is_folder=True
           )

           if not sub_events_list:
               return JsonResponse(
                   {
                       "response": f"No matching FolderEventSubs found for event_id
{event_id}"
                   },
                   status=404,
               )

           for sub_event_item in sub_events_list:
               result_data.append(
                   {
                       "status": "GET request processed successfully",
                       "sub_event_name": sub_event_item.sub_event_name,
                       "description": sub_event_item.description,
                       "is_folder": sub_event_item.is_folder,
                   }
               )

           return JsonResponse(result_data, safe=False)

       except EventSubs.DoesNotExist:
           return JsonResponse(
               {
                   "response": f"No matching FolderEventSubs found for event_id
{event_id}"
               },
               status=404,
           )

   # PUT
   @swagger_auto_schema(
       request_body=openapi.Schema(
           type=openapi.TYPE_OBJECT,
           properties={
               "sub_event_name": openapi.Schema(type=openapi.TYPE_STRING),
               "description": openapi.Schema(type=openapi.TYPE_STRING),
           },
           required=["sub_event_name"],
       ),
   )
```

```
    def put(self, request, event_id):
        """
        event_id = ID (sub event id)
        @requestbody
        {
        "sub_event_name": "newSub Event name",
         "description": "New description"
        }
        """
        print("put")

        try:
            sub_event = EventSubs.objects.get(id=event_id)

            sub_event.sub_event_name = request.data.get(
                "sub_event_name", sub_event.sub_event_name
            )
            sub_event.description = request.data.get(
                "description", sub_event.description
            )

            sub_event.save()

            return JsonResponse(
                {
                    "message": "PUT request successfully updated subeventname,description
"
                },
                status=200,
            )

        except EventSubs.DoesNotExist:
            return JsonResponse(
                {
                    "message": f"No matching FolderEventSubs found for event_id
{event_id}"
                },
                status=404,
            )

    # POST
    @swagger_auto_schema(
        request_body=openapi.Schema(
            type=openapi.TYPE_OBJECT,
            properties={
                "sub_event_name": openapi.Schema(type=openapi.TYPE_STRING),
                "description": openapi.Schema(type=openapi.TYPE_STRING),
            },
            required=["sub_event_name"],
        ),
    )
    def post(self, request, event_id):
        """
        Create sub-event for main event (event_id).
        """

        folder_name = request.data.get("sub_event_name")
        if not folder_name:
            return JsonResponse(
                {"message": "sub_event_name not found in parameters"}, status=400
            )

        try:
            main_event = EventsMain.objects.get(id=event_id)

            if not main_event:
                return JsonResponse(
                    {"message": f"No matching Event found for event_id {event_id}"},
                    status=404,
                )
```

```
            new_sub_event = EventSubs.objects.create(
                sub_event_name=folder_name,
                event_id=main_event,
                description=request.data.get("description"),
                is_folder=True,
            )

            payload = {
                "sub_event": [
                    {
                        "Main Event": main_event.event_name,
                        "Sub Event": new_sub_event.sub_event_name,
                        "Description": new_sub_event.description,
                    }
                ]
            }

            return JsonResponse(response_dict(data=payload))

        except Exception as e:
            return JsonResponse({"message": f"Error: {str(e)}"}, status=500)

    def delete(self, request, event_id):
        """
        Delete a sub-event based on event_id.
        Also, set is_deleted=true for associated media.
        """
        try:
            sub_event = EventSubs.objects.get(EventSubs, id=event_id)
            media_list = Media.objects.filter(sub_event_id=event_id)
            for media in media_list:
                media.is_deleted = True
                media.save()

            sub_event.delete()

            return JsonResponse(
                {
                    "message": "DELETE request successfully deleted sub-event and updated
media"
                },
                status=200,
            )

        except EventSubs.DoesNotExist:
            return JsonResponse(
                {"message": f"No matching EventSubs found for id {event_id}"},
                status=404,
            )
```

Week 7:

(29 Jan - 2 Feb)

- Progress Internship Seminar 1st
- Integration of Backend API with frontend
- added new functionality in GET request

```
def get(self, request, event_id):
    """
    GET records whose is_folder = True, Based on event_id
    """
    result_data = []
    try:
        sub events list = EventSubs.objects.filter(
            event_id_id=event_id, is_folder=True
        )

        if not sub events list:
            return JsonResponse(
                {
                    "response": f"No matching FolderEventSubs found for event_id
{event id}"
                },
                status=404,
            )

        for sub event item in sub events list:
            associated media = Media.objects.filter(
                event_id=sub_event_item.event_id
            )

            sub event data = {
                "status": "GET request processed successfully",
                "sub event name": sub event item.sub event_name,
                "description": sub event item.description,
                "is_folder": sub event_item.is_folder,
                "associated_media": [
                    {
```

- Added test case for GET request

  >>>Python3 manage.py test core_engine.features.subevents.folders.test

- Added test case for PUT request
- Added test case for POST request
- Added test case for DELETE request
- Fix test cases for >>folders (https://uat.samaro.app/event/32/settings/folder)
- http://localhost:4200/event/211/subevents ↤(fe_server)

Week 8:

(5 Feb- 9 Feb)

- write GET api for download analytics component table

```
@swagger_auto_schema(
        manual_parameters=[
            openapi.Parameter(
                name="user_id",
                in_=openapi.IN_QUERY,
                type=openapi.TYPE_INTEGER,
                required=True,
                description="User ID for filtering media downloads",
            ),
        ],
        responses={200: "OK"},
    )
    def get(self, request, event_id):
        """
        Get media downloads associated with a user ID.
        """
        user_id = request.query_params.get("user_id")
        if not user_id:
            return JsonResponse({"error": "User ID is required"}, status=400)
        media_downloads = MediaDownloads.objects.filter(user_id=user_id)
        data = []
        for download in media_downloads:
            user_data = {
                "user_id": download.user_id,
                "user_first_name": download.user.first_name,
                "user_last_name": download.user.last_name,
                "media_id": download.id,
            }
            data.append(user_data)
        return JsonResponse(response_dict(data=data, print_full_logs=True))
```

https://github.com/samaroapp/samaro/pull/707/files

- added test cases for >>subevents>>tests.py

- subevents>> `cashing, query optimization, swagger.`

- Created Personal resume displaying Django project
  spring boot>>python>>html>>css

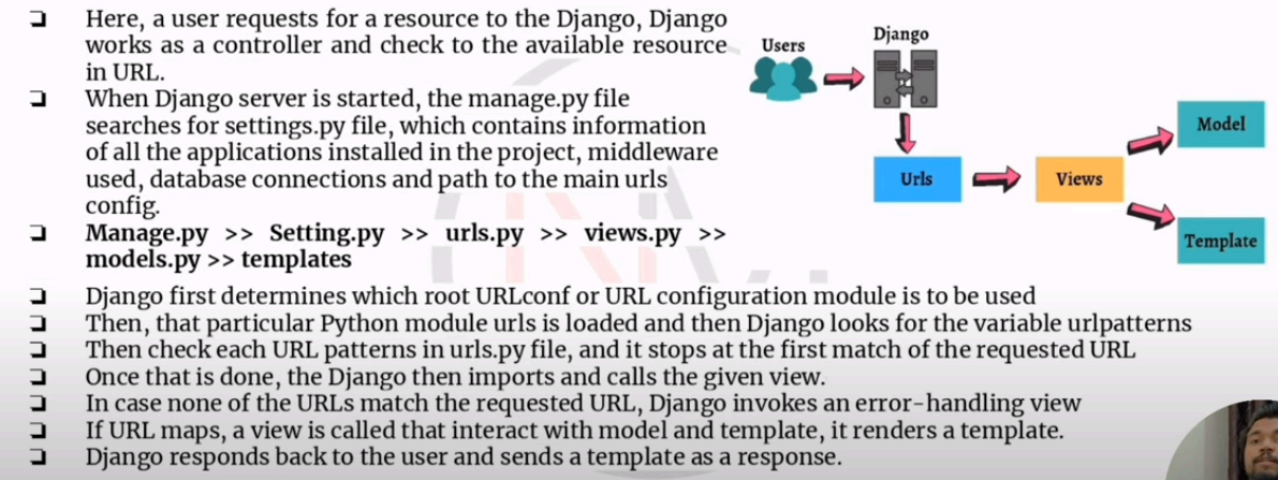
Week 9:

(12 Feb - 16 Feb)

code of subevents (test.py>> added test cases & views.py>> added swaggers for API’s)

```
            404: "SubEvent does not exist",
            500: "SubEventId is missing or Server Error",
        },
    )
    def get(self, request, event_id, sub_event_url):
        sub_event_id = request.GET.get(
            "subEventId", None
        )  # received only when it's an edit request.
        if sub_event_id is None:
            return JsonResponse(response_dict(data="SubEventId is missing"), status=500)
        try:
            obj: EventSubs = EventSubs.objects.get(
                event_id=event_id, sub_event_url=sub_event_url
            )
            if sub_event_id and obj.id == int(sub_event_id):
                return JsonResponse(response_dict(data="Success"))
        except EventSubs.DoesNotExist:
            return JsonResponse(
                response_dict(data="SubEvent does not exist"), status=404
            )
        except Exception as e:
            return JsonResponse(response_dict(data="Server Error"))

        raise RuntimeError("Url is taken")


class SubEventView(APIView):
    permission_classes = (permissions.IsAuthenticated,)

    @swagger_auto_schema(
        manual_parameters=[
            openapi.Parameter(
                name="getCondensed",
                in_=openapi.IN_QUERY,
                type=openapi.TYPE_STRING,
                required=False,
                description="Specify 'list', 'dict' / leave empty for the default view.",
            ),
```

- how a request is processed in Django



- Created employee management system Django project
  spring boot>>python>>html>>css

- razorpay & stripe understanding. to integrate one time payment on razorpay.
  - stripe docs
  - stripe web integration

- stripe checkout process
- stripe login
- youtube
- stripe.py
- txn view how it working
- localhost frontend test your stripe knowledge

Week 10:

(19 Feb - 23 Feb)

- Understanding stripe payment system in samaro
    - coupon code for stripes
    - one time payment in stripe
    - business payment stripe
    - where the coupon is applied and how we should apply coupon on payments in strip

- backend apis spec design for business public profile page

Week 11:

(16 feb - 1 mar)

- Following tasks on business profile page

BizReviews - biz_reviews
id, biz (FK to biz), user (FK to user), review (500 line), rating_star (1 - 5 ke bich ka float), created, updated

api - get - fetch all reviews for a given biz id
post - create new review for a given biz id
delete - delete a review from review id

create new folder biz_reviews in biz folder

- models_biz_reviews - model, admin decl,
- views_biz_reviews - get,post,del
  - get
    - - average_rating - average of all the reviews.rating_star.
    - total_reviews -
    - data - [... list of each review]
- service_biz_reviews - serializer class inside it only.

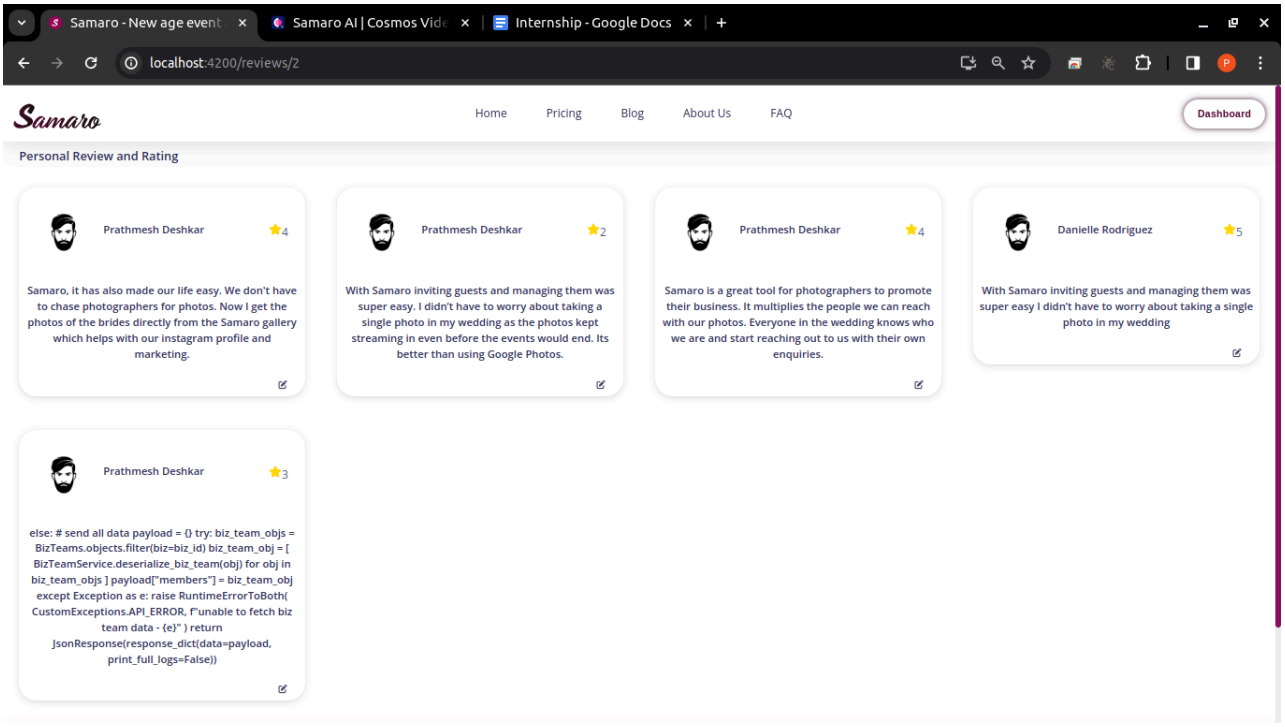
url - path(r"biz/<int:biz_id>/reviews/", BizReviewView.as_view(), name="BizReviewView"),

- Created GET PUT
- POST DELETE api for business profile

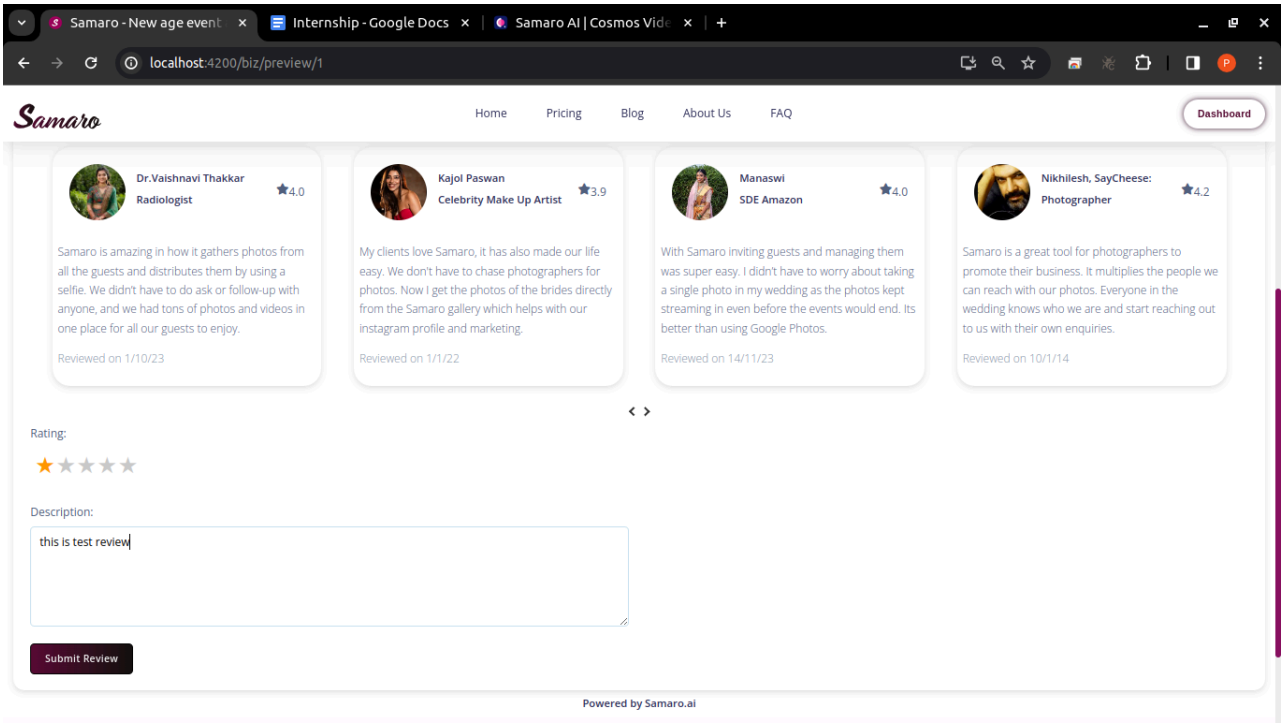
Week 12:

(4 mar - 8 mar)

- writing api for business profile
- reviews on business (by user)



- posting review (as user)



Week 13:

(11 mar - 15 mar)

- `select_related` :

used to retrieve related objects in a single query to the database.
It works by performing a SQL `JOIN` to retrieve the related objects along with the primary object in the same query.
It is useful when you know you'll need the related objects and want to minimize database hits.

One uses select_related when the object that you're going to be selecting is a single object, so OneToOneField or a ForeignKey

select_related obtains all data at one time through multi-table join Association query and improves performance by reducing the number of database queries.

`select_related` is used to fetch related objects in a single SQL query using SQL JOIN.

It works best for ForeignKey and OneToOne relationships.

if `BizReviews` had a ForeignKey relationship with `BizProfile`, using `select_related` would fetch `BizProfile` objects along with `BizReviews` objects in one query.

Using `select_related` reduces the number of database hits and can improve performance, especially when you need to access related fields frequently.

- `prefetch_related:`

used to retrieve related objects separately from the primary object's query.
It performs a separate database query for each relationship, but it does so in a more efficient way compared to individual queries for each object.
It is useful when you have a `ManyToMany` or reverse `ForeignKey` relationship, or when you're dealing with large sets of related objects.

use prefetch_related when you're going to get a "set" of things, so ManyToManyFields as you stated or reverse ForeignKeys.

`prefetch_related` is used to fetch related objects separately from the main query but in an optimized manner.

It's designed for ManyToMany and reverse ForeignKey relationships.

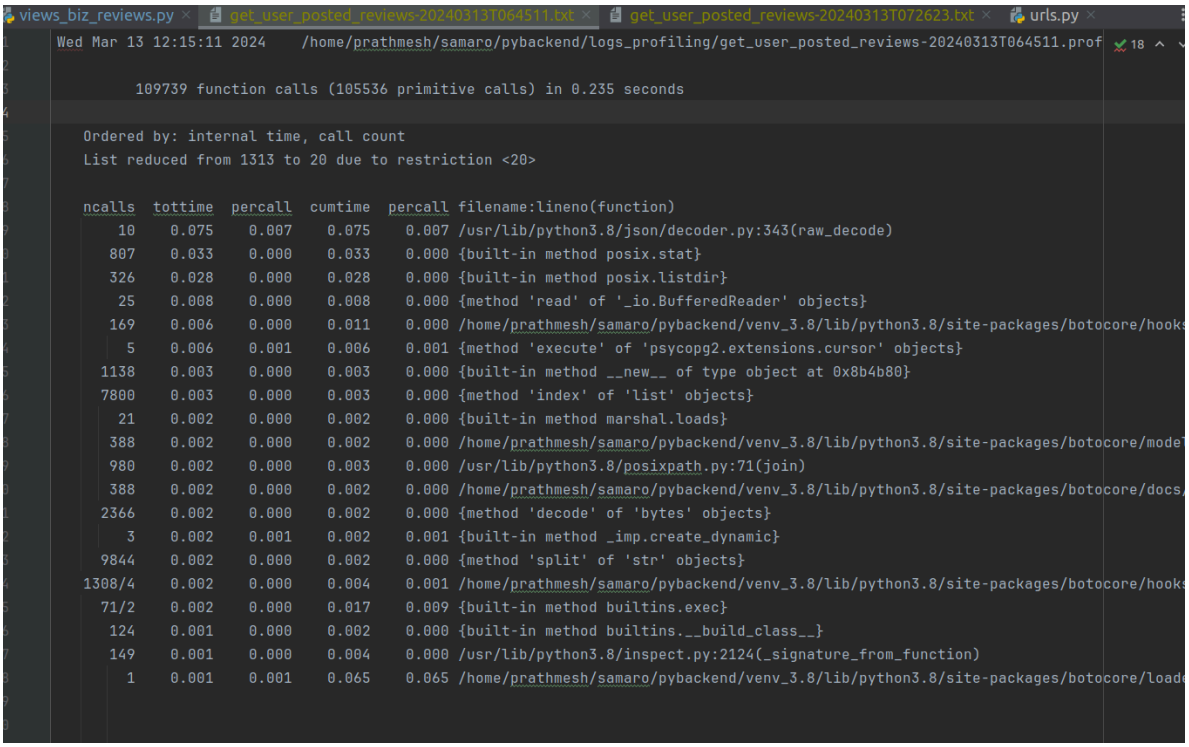
If `BizReviews` had a ManyToManyField or reverse ForeignKey relationship with another model, using `prefetch_related` would fetch related objects efficiently in separate queries.

`prefetch_related` is beneficial when dealing with ManyToMany relationships or reverse ForeignKey relationships, especially if you have large datasets or complex relationships.

gather sql code for this api, capture time, capture overhead, logs etc. Do profiling.
pybackend/utility/logging/profiling.py
now change the above code to use select_related or prefetch_related or optimize the query/api.
and profile it again. then show the comparison report.

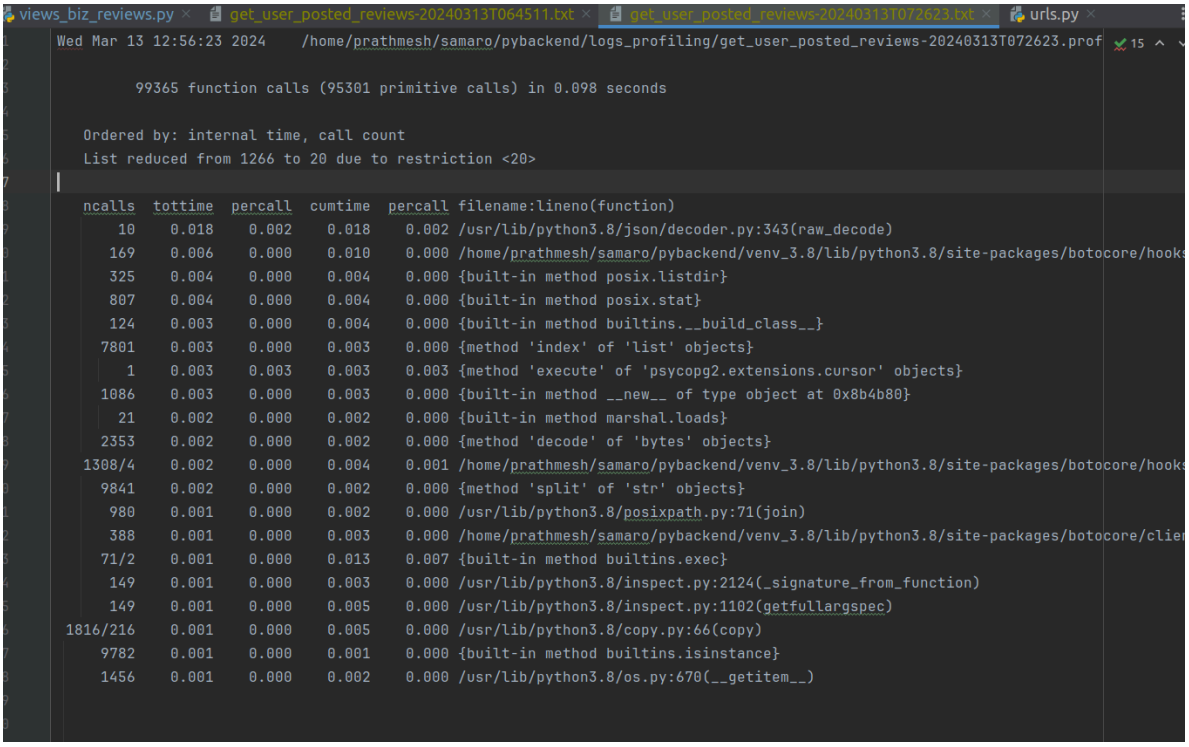
- Profiling Results of GET api (get_user_posted_reviews)

  Without using select_related



```
views_biz_reviews.py | get_user_posted_reviews-20240313T064511.txt | get_user_posted_reviews-20240313T072623.txt | urls.py

Wed Mar 13 12:15:11 2024    /home/prathmesh/samaro/pybackend/logs_profiling/get_user_posted_reviews-20240313T064511.prof

         109739 function calls (105536 primitive calls) in 0.235 seconds

   Ordered by: internal time, call count
   List reduced from 1313 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       10    0.075    0.007    0.075    0.007 /usr/lib/python3.8/json/decoder.py:343(raw_decode)
      807    0.033    0.000    0.033    0.000 {built-in method posix.stat}
      326    0.028    0.000    0.028    0.000 {built-in method posix.listdir}
       25    0.008    0.000    0.008    0.000 {method 'read' of '_io.BufferedReader' objects}
      169    0.006    0.000    0.011    0.000 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/hooks
        5    0.006    0.001    0.006    0.001 {method 'execute' of 'psycopg2.extensions.cursor' objects}
     1138    0.003    0.000    0.003    0.000 {built-in method __new__ of type object at 0x8b4b80}
     7800    0.003    0.000    0.003    0.000 {method 'index' of 'list' objects}
       21    0.002    0.000    0.002    0.000 {built-in method marshal.loads}
      388    0.002    0.000    0.002    0.000 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/model
      980    0.002    0.000    0.003    0.000 /usr/lib/python3.8/posixpath.py:71(join)
      388    0.002    0.000    0.002    0.000 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/docs/
     2366    0.002    0.000    0.002    0.000 {method 'decode' of 'bytes' objects}
        3    0.002    0.001    0.002    0.001 {built-in method _imp.create_dynamic}
     9844    0.002    0.000    0.002    0.000 {method 'split' of 'str' objects}
   1308/4    0.002    0.000    0.004    0.001 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/hooks
     71/2    0.002    0.000    0.017    0.009 {built-in method builtins.exec}
      124    0.001    0.000    0.002    0.000 {built-in method builtins.__build_class__}
      149    0.001    0.000    0.004    0.000 /usr/lib/python3.8/inspect.py:2124(_signature_from_function)
        1    0.001    0.001    0.065    0.065 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/load
```

  With using select_related



```
views_biz_reviews.py | get_user_posted_reviews-20240313T064511.txt | get_user_posted_reviews-20240313T072623.txt | urls.py

Wed Mar 13 12:56:23 2024    /home/prathmesh/samaro/pybackend/logs_profiling/get_user_posted_reviews-20240313T072623.prof

         99365 function calls (95301 primitive calls) in 0.098 seconds

   Ordered by: internal time, call count
   List reduced from 1266 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       10    0.018    0.002    0.018    0.002 /usr/lib/python3.8/json/decoder.py:343(raw_decode)
      169    0.006    0.000    0.010    0.000 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/hooks
      325    0.004    0.000    0.004    0.000 {built-in method posix.listdir}
      807    0.004    0.000    0.004    0.000 {built-in method posix.stat}
      124    0.003    0.000    0.004    0.000 {built-in method builtins.__build_class__}
     7801    0.003    0.000    0.003    0.000 {method 'index' of 'list' objects}
        1    0.003    0.003    0.003    0.003 {method 'execute' of 'psycopg2.extensions.cursor' objects}
     1086    0.003    0.000    0.003    0.000 {built-in method __new__ of type object at 0x8b4b80}
       21    0.002    0.000    0.002    0.000 {built-in method marshal.loads}
     2353    0.002    0.000    0.002    0.000 {method 'decode' of 'bytes' objects}
   1308/4    0.002    0.000    0.004    0.001 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/hooks
     9841    0.002    0.000    0.002    0.000 {method 'split' of 'str' objects}
      980    0.001    0.000    0.002    0.000 /usr/lib/python3.8/posixpath.py:71(join)
      388    0.001    0.000    0.003    0.000 /home/prathmesh/samaro/pybackend/venv_3.8/lib/python3.8/site-packages/botocore/clie
     71/2    0.001    0.000    0.013    0.007 {built-in method builtins.exec}
      149    0.001    0.000    0.003    0.000 /usr/lib/python3.8/inspect.py:2124(_signature_from_function)
      149    0.001    0.000    0.005    0.000 /usr/lib/python3.8/inspect.py:1102(getfullargspec)
 1816/216    0.001    0.000    0.005    0.000 /usr/lib/python3.8/copy.py:66(copy)
     9782    0.001    0.000    0.001    0.000 {built-in method builtins.isinstance}
     1456    0.001    0.000    0.002    0.000 /usr/lib/python3.8/os.py:670(__getitem__)
```

- Retool (https://retool.com/)

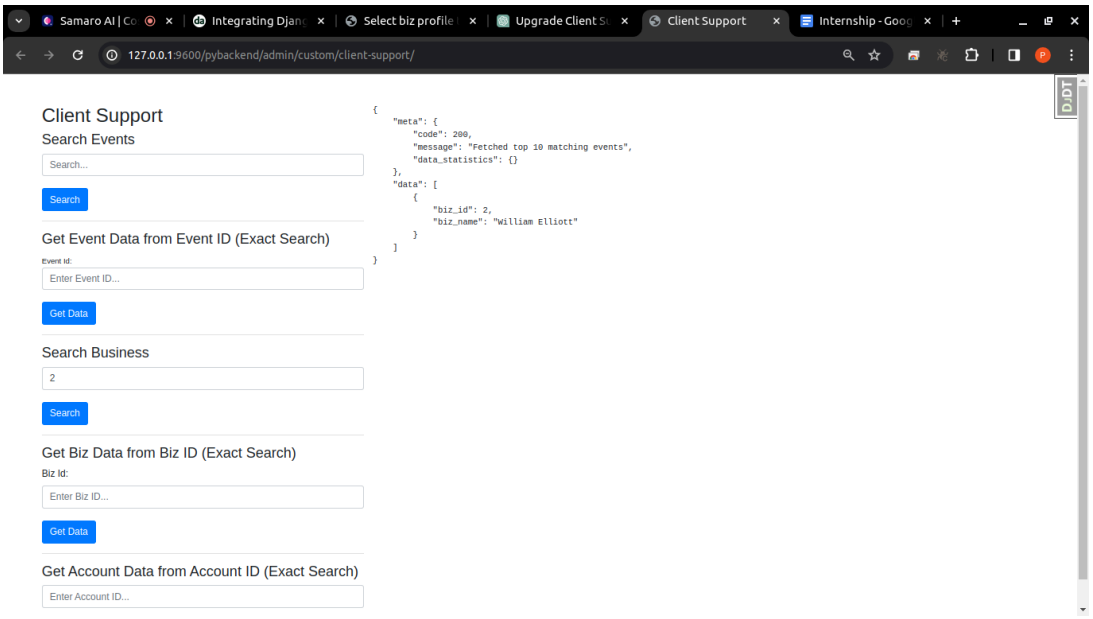
  bring table data from django admin to a retool dashboard
  And have some get, post apis integrated on retool dashboard

```
connection_string =
f"postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}"
```

Week 14:

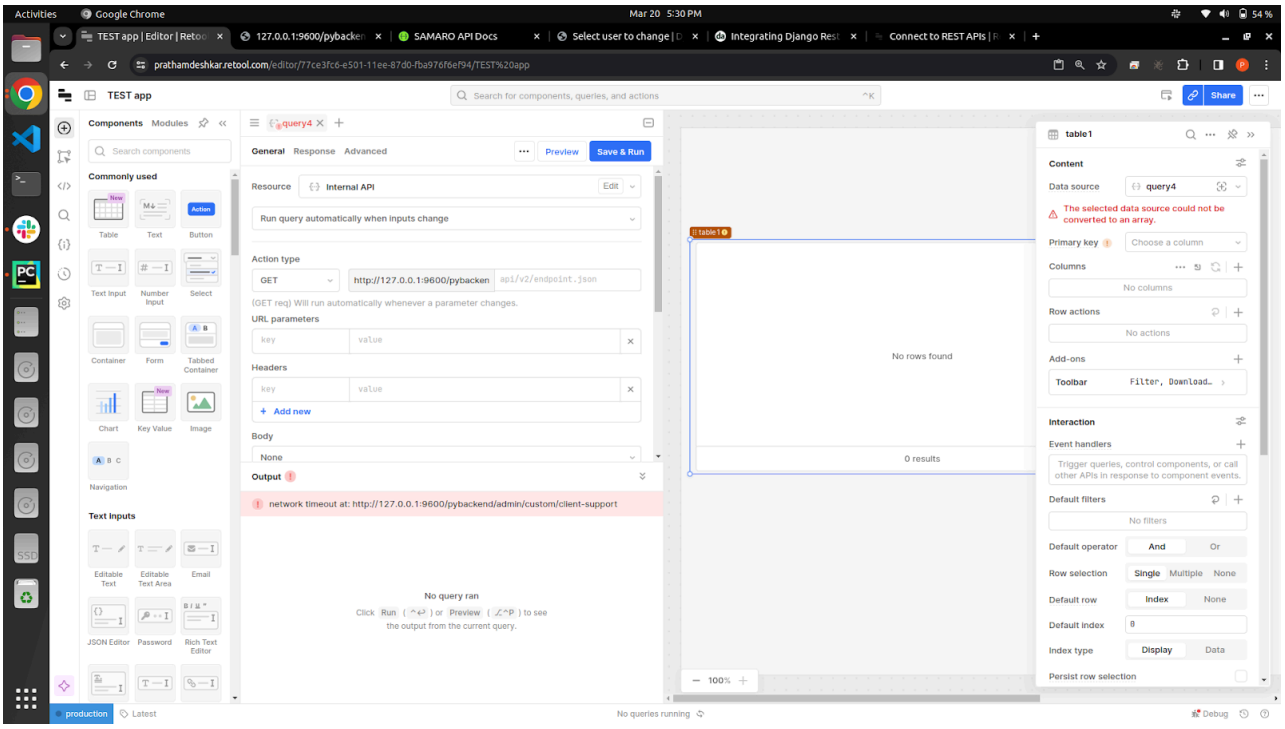
(18 mar - 22 mar)

- Upgrade client support page to a more beautiful UI like retool



  updated UI of client support page

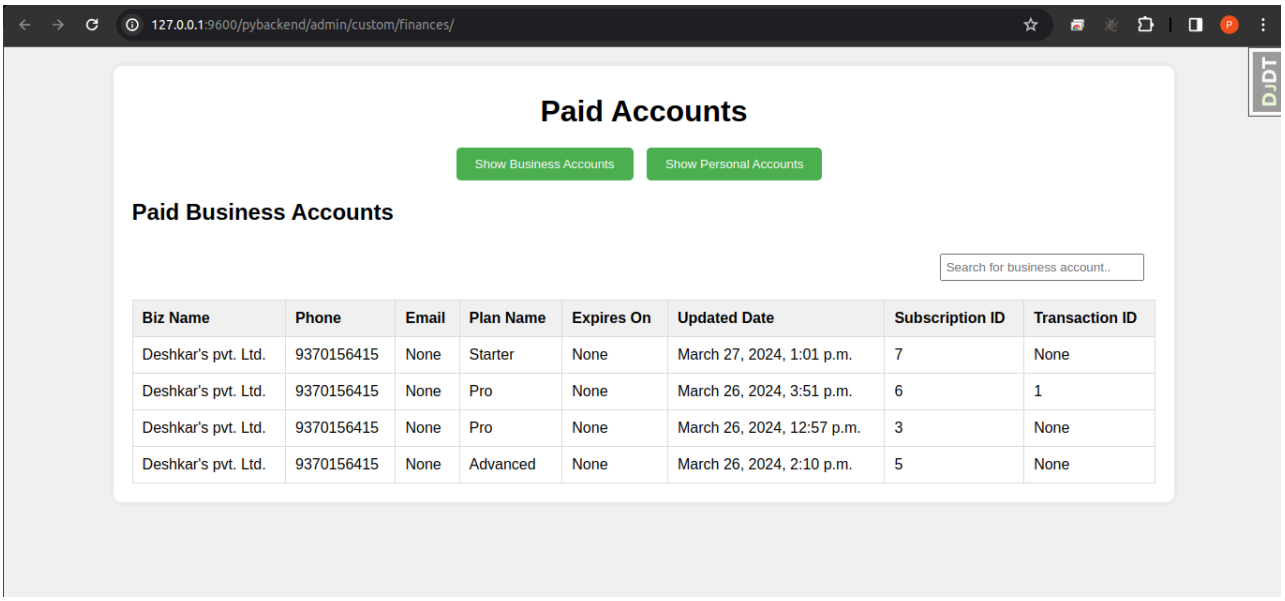
- created resources for api
- Create a ReTool Query:
- Design the UI (simple table for test)
- Bind Data to UI Components



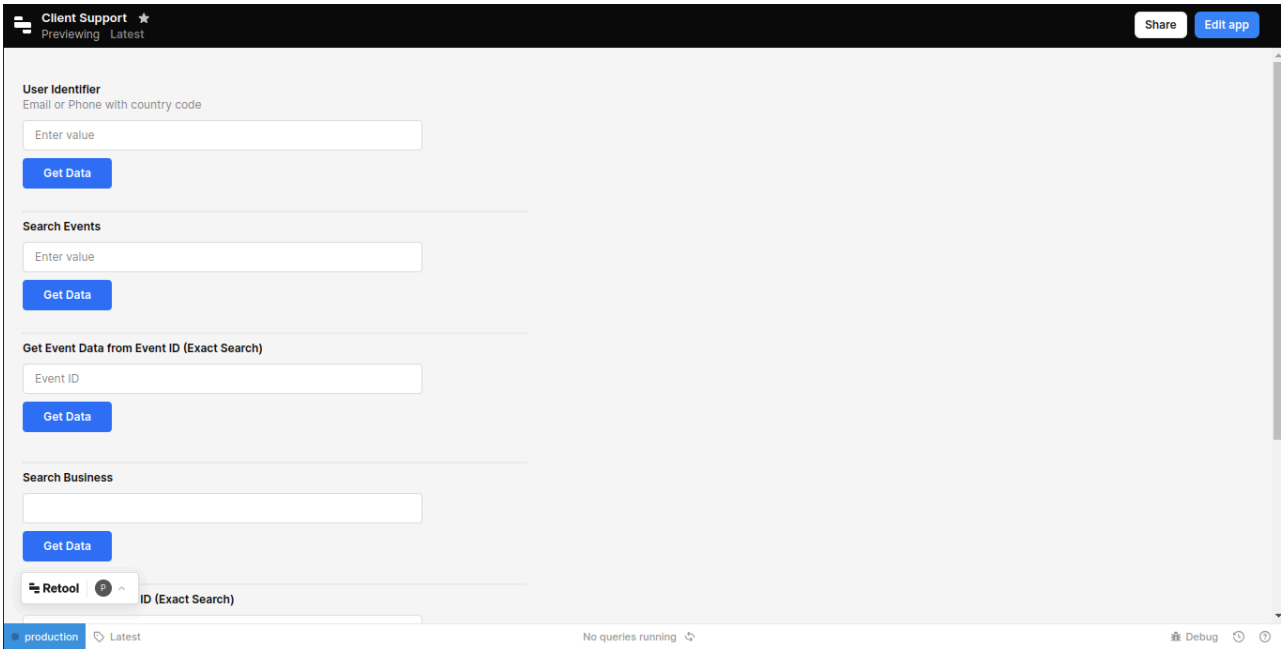
Week 15:

(25 mar - 29 mar)

- Custom Finance page (http://127.0.0.1:9600/pybackend/admin/custom/finances/)



- Retool client support page (https://samaro.retool.com/)



https://events.samaro.ai/pybackend/admin/custom/client-support/user-all-data/

Week 16:

(1 april - 6 april)

- retool client support page
- Progress Internship Seminar 2nd (6 apr)

- report of internship till now
  https://chat.openai.com/share/26647a02-2247-4ad4-879d-ce0ec73ef908

Week 17:

(8 april - 13 april)

- finances page

- For personal purchases:
  - If it's a non-Free Event Plan:
    - The status should be "INACTIVE" for both purchased and redeemed events.
    - The status should be "ACTIVE" for events that are purchased but not yet redeemed.

Week 18:

(15 april - 20 april)

- finance page query fix for Paid Personal Accounts

For personal accounts, it first checks if the plan is not a "Free Event" plan.
If it's not a Free Event plan, it checks if there's a transaction associated with the subscription and if the transaction is not redeemed.
If there is a transaction and it's not redeemed, it sets the status of the subscription to "ACTIVE".
If there's no transaction or it's redeemed, it sets the status to "INACTIVE".
Then, it constructs a dictionary containing information about the subscription along with its status and adds it to the list of paid personal accounts.

Week 19:

(22 april - 27 april)

- common QR for business

  Model - commonQR
  columns for commonQR
  1. id (pk)
  2. qr name
  3. account (fk to Account)
  4. event (fk to eventsMain)

User provides a QR name.
User selects an event to link with the QR.

api endpoint: path(r"biz/<int:biz_id>/common-qr/"

- GET api
    1. retrieve all QR codes associated with a specific Account.
    2. Fetch all common QR objects from the database linked to the specified account id ID.

- POST api
    1. Create a new common QR code associated with a specific business.
    2. Receive the data for the new common QR code from the request body.

- PUT api
    1. for editing qr,
    2. update an existing common QR code associated with a specific business.

Week 20:

(29 april - 4 may)

- Database cleanup script

    - in the new records -

```
{
  "tp": 1,
  "meta": {
      "isImage": "true", "folderId": null, "accountId": "10695", "filterTab": "OFFICIAL",
"subeventId": 1281
  },
  "parts": {"p_0": 1}
}
```

parts is a dictionary

    - for all the old records -

```
{"p_0": 1, "p_6": 1}
```

parts are directly written

so, move the parts data to a "parts" key such that the format of old and new db rows are same.
root level 3 keys - tp, meta: {}, parts: {move old data here}