Grabify

Aim : Is to connect all the shops in an area through an application to the Digital market

Technologies :
1. Flutter (Frontend)
2. Django (Python based framework for API)
3. Sqlite3 (Database Management)

Project Link: https://github.com/DPRIYATHAM/Grabify/tree/Grabify1.1

This app is here to revolutionize the way we used to order things in our college,we are trying to bring the whole buying experience of the small shops we have in our near vicinity (i.e. Neethi Medical Store, Milma Booth, Mandi Shop, AJ Bakers , etc.).
As, we know the major crowd we have here is Students the lines at these shops increases day by day and it is a very tiring process, So we are trying to minimize the waiting time by taking the order and then prompting the shop owner to prepare the order or pack the order and send a notification that the order is prepared so that the Person ordering can go and Collect the Order!

The Idea is , we will maintain User accounts using their roll numbers, name and Phone Number to authenticate the user similarly we will have shops to register themselves to the app and update their its stock everyday
Then there is the dashboard in which we will suggest the item from the specific selected shop,
We will also maintain liked item list so that the person can order items fastly.
We have finally the cart page which will store all the items and create a order and then send it to the shop
All of the backend stuff will be managed by Django for query writing in Python and the database for all items and the user accounts will be stored using SQLite3.
We will also implement SUggestion to product based on weather using open weather api and also analyzing the data to recommend food items with more click rates and buying probability.

We will also implement prescription upload feature to place medicinal orders at neethi medical store
As of now there is no transaction implemented for the project.

Backend Documentation

Create a folder `GRABIFY1.0` in your local machine.
Open that folder in VS Code, and start working.

Install Django

```
E:\BTech\Grabify1.0> pip install django
```

Install DjangoRestFramework

```
E:\BTech\Grabify1.0> pip install djangorestframework
```

Start a Project in Django

```
E:\BTech\Grabify1.0> django-admin startproject GrabifyBE
```

Navigate to `GrabifyBE` folder to work further.

```
E:\BTech\Grabify1.0> cd GrabifyBE
```

Start a App in Django

```
E:\BTech\Grabify1.0\GrabifyBE> django-admin startapp api
```

Adding the Installed apps to Django

Add the App `api` that we created and the rest_framework to installed apps in `settings.py` file.

```
INSTALLED_APPS = [
        ...
    'rest_framework',
    'api'
]
```

Setting up Database

We are using Sqlite3 Database so we should use the engine for Sqlite3, update the values of

`DATABASES` in `Settings.py` file.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'GrabifyDB.sqlite3',
    }
}
```

Creating a Model

Model is as similar to a table in MySQL, we store them as model in SQLite3.

```
from django.db import models

#Model Item with defined atributes.
class item(models.Model):
    item_id = models.IntegerField(primary_key=True) #PrimaryKey
defined
    item_name = models.CharField(max_length=1000)
    image_link = models.CharField(max_length=1000)
    item_price = models.CharField(max_length=1000)
    item_desc = models.CharField(max_length=1000)
```

Registering for Admin Access

Django provides a feature of Access page to your backend where you can actually go through the content, tables, and do CRUD operations without writing queries.
So let's register our Model in the Admin page.

```
from django.contrib import admi
from .models import item #should always write .models which says it
is a file
# Register your models here.
@admin.register(item)
class ItemAdmin(admin.ModelAdmin):
    list_display = ['item_id', 'item_name', 'image_link',
'item_price', 'item_desc']
```

Creating Serialiers

After creating the database, we actually need to Send it to frontend, as JSON (Javascript Object Notation), which can be used as input in the frontend, as it is easier to access a list.
Create a file `serializers.py` in `api`

```
from rest_framework import serializers
from .models import item

class ItemSerializer(serializers.ModelSerializer):
    class Meta:
        model = item #creating a serialized format of the model
item.
        fields = ['item_id', 'item_name', 'image_link',
'item_price', 'item_desc']
```

Creating Views

As we created serialixed format of data now we need to show it up somewhere so that we can access it to the frontend from the view,

```
from .models import item
from .serializers import ItemSerializer
from rest_framework.generics import ListAPIView

# Create your views here.
class ItemList(ListAPIView):
    queryset = item.objects.all() #We are writing a query to get
all elements
    serializer_class = ItemSerializer
```

Creating URLS

Now we created a view where we have all the JSON format text, but how do we send it to frontend?
We need to create a endpoint, where frontend can access all the JSON text,
So we are doing that now, we are adding this view to a url which we an access.
Create a file `urls.py` in `api`

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

```
urlpatterns = [
    path('items/', views.ItemList.as_view()),
]
```

Including URLS

Oh! Wow now we are done with creating URLS, but wait, this url we created above is inside *api*
Like http://127.0.0.1:8000/api/items
But we have actually missed to add the main app `api.urls` in `urls.py` of `GRABIFYBE`
So let's add the access to `/api` in the `GrabifyBE.urls` file.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls'))
    #including api.urls file urlpatterns here under /api
]
```

Flutter URL

As we had discussed about accessing urls, paths we should also take care that Android emulator can't directly access `127.0.0.1 localhost` We need `10.0.0.2` as host there,
In `GrabifyBE` go to `settings.py` and update Allowed hosts,

```
ALLOWED_HOSTS = [
    '10.0.2.2',
    '127.0.0.1'
]
```

Makingmigrations

All set, let's make all migrtions

```
E:\BTech\Grabify1.0\GrabifyBE> python manage.py makemigrations
```

Migrate

Now let's migrate

```
E:\BTech\Grabify1.0\GrabifyBE> python manage.py migrate
```

Creating Super User

`item` model is registered in the `Admin.py` file, so it says there is something as Admin, so now we need to create a `Superuser` aka `admin` who can access all the data and perfrom CRUD operations.

```
E:\BTech\Grabify1.0\GrabifyBE> python manage.py createsuperuser
#Email, Password is not neccessary.Click Enter to skip that
feild.
```

RunServer

```
E:\BTech\Grabify1.0\GrabifyBE> python manage.py runserver
```

Now we created the super user, we are free to perform CRUD operations from the
Django Site: http://127.0.0.1:8000/admin/

The `Endpoint` where we get all the data is http://127.0.0.1:8000/api/items

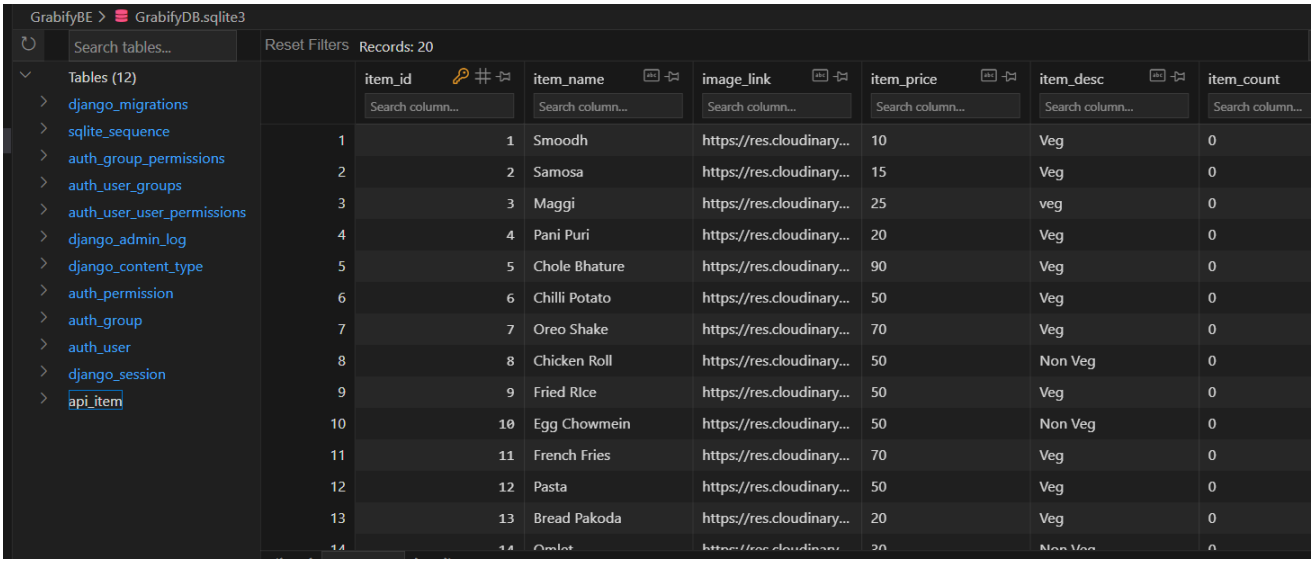
Disable Browser API

But did you notice that, the url returns some extra things apart from Json file? This might make a issue, so Just Disable that feature of getting those extra things.

```
REST_FRAMEWORK = {
        'DEFAULT_RENDERER_CLASSES': (
            'rest_framework.renderers.JSONRenderer',
        )
    }
```

Hurray, You are all set now, you will get the pure JSON formatted file of the requested queries.

DataBase:



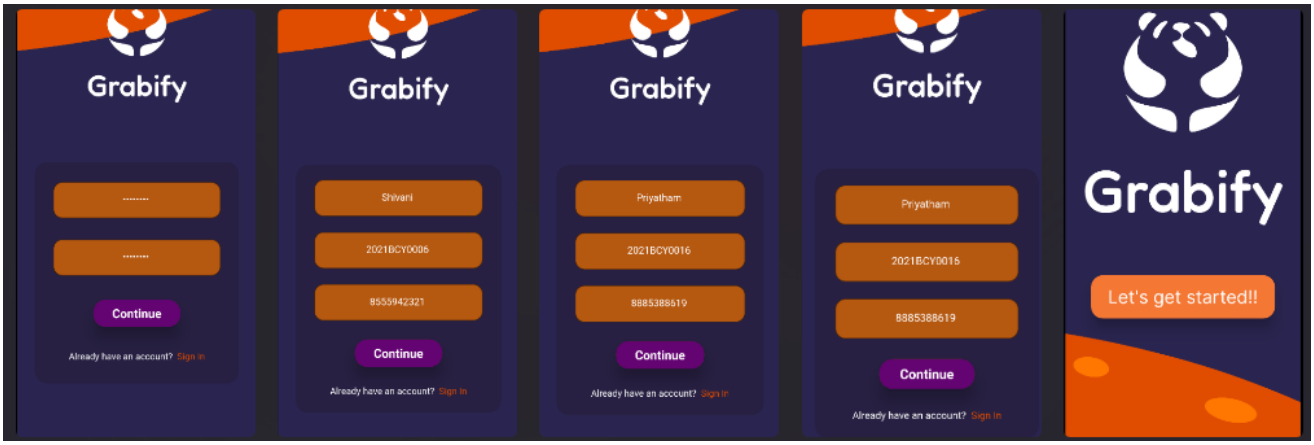
Frontend Documentation.

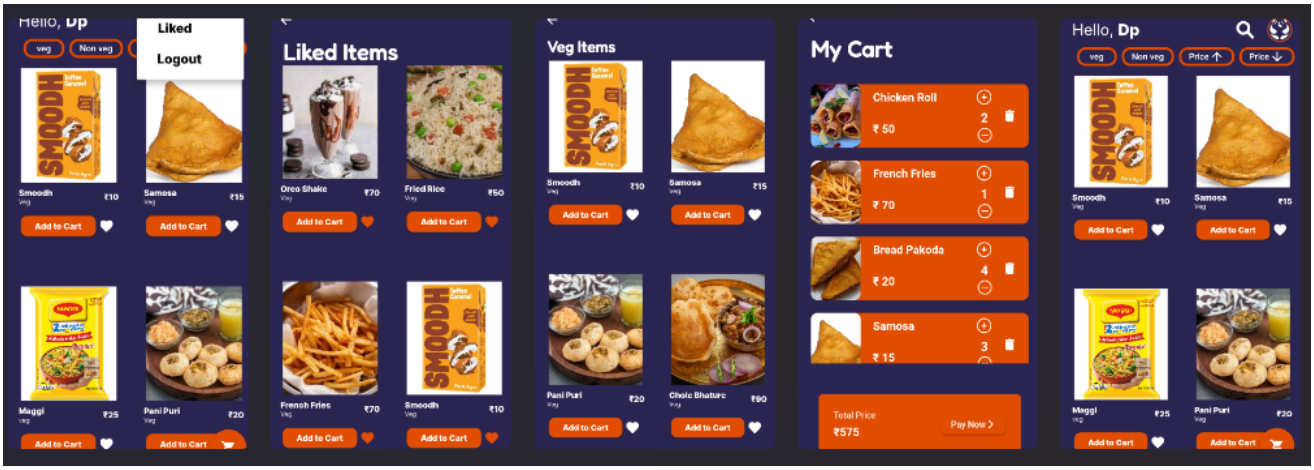
Building the User Interface

The frontend of the Gabify application will be built using Flutter, a popular open-source framework for building cross-platform applications.
The user interface of the application will consist of multiple screens, including a welcome screen, a login and signup screen, a home screen, a product list screen, a liked item screen, and a cart screen.

The home screen will display featured products and different filters. The product list screen will display a list of products. The liked Item screen shows the items which you frequently buy and like the most from the already present Items. The cart screen will display the items in the user's cart and allow them to place an order.

To build the user interface, we will use Flutter's rich collection of widgets, such as Container, GridView Builder, ListView Builder, Text, Image,FlatButton and ElevatedButton. We will also make use of Flutter's routing and navigation system to navigate between the screens.





You can find more `Images` in the Images folder.

Interacting with the Backend

The frontend of the E-commerce application will interact with the backend using APIs. The APIs provided by the backend will allow the frontend to retrieve data, such as products and categories, and send data, such as orders, to the backend.

To interact with the APIs, we will use the http package in Flutter, which provides a simple API for making HTTP requests.

Some Part of Code for Integrating frontend and Backend, calling the APIs...

```
class Cartmodel extends ChangeNotifier {
  String allurl = "http://10.0.2.2:8000/api/items/";
  Future<List> getAllItems() async {
    try {
      var response = await http.get(Uri.parse(allurl));
      if (response.statusCode == 200) {
        homeShopItems = (json.decode(response.body) as List);
        return homeShopItems;
      } else {
        return Future.error("Server Error");
```

```
      }
    } catch (e) {
      return Future.error(e);
    }
  }
```

Thank you for going through Our Documentation, New features and Updates on the way.