Implementation Document

for

Campus Craves

Version 0.1

Prepared by

Group 19 **Group Name: Supervised Learners**

Name	Roll no.	E-mail
Aayush Gautam	220020	aayushg22@iitk.ac.in
Abhishek Kumar	210040	abhikumar21@iitk.ac.in
Anand Chutani	220130	anandc22@iitk.ac.in
Anany Dev Choudhary	220135	ananydc22@iitk.ac.in
Chatla Sowmya Sri	200293	srisowmya20@iitk.ac.in
Lokesh Mehra	220591	lokeshm22@iitk.ac.in
Pranshu Mani Tripathi	220800	pranshumt22@iitk.ac.in
Siddharth Pathak	211034	siddharthp21@iitk.ac.in
Sidharth A S	221056	sidharthas22@iitk.ac.in
Sparsh Gupta	221084	gsparsh22@iitk.ac.in
Ujjwal Kumar	211123	ujjwalk21@iitk.ac.in

Course: CS253

Mentor TA: Mr. Ashish Singh

Date: 28 March, 2025

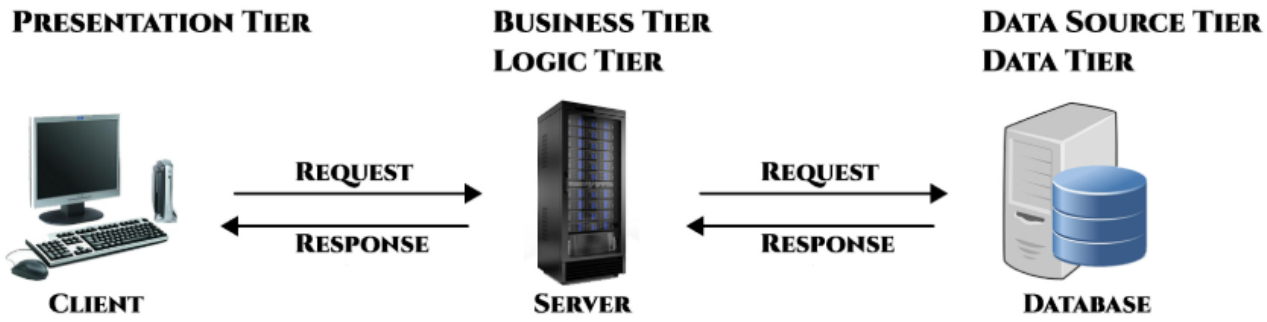
Revisions

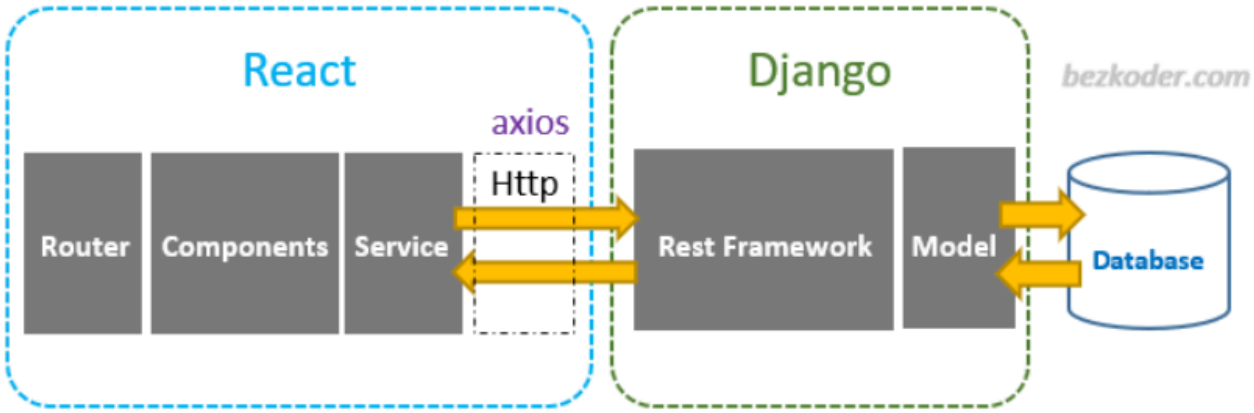
Version	Primary Author(s)	Description of Version	Date Completed
0.1	Chatla Sowmya Sri Siddharth Pathak	Initial Commit (First Draft)	28/03/2025

1 Implementation Details

Campus Craves implements a three-tier architecture, a popular pattern for user-facing applications. The tiers that comprise this architecture includes:

- **Presentation Tier (Frontend Development):** This represents the components users directly interact with. In our case, this is the React based web application that runs inside a browser.
- **Logic Tier (Backend Development):** This contains the code required to translate user actions to the functionality, written in python (utilizing django rest framework), that drives the application's behavior at the presentation tier.
- **Data Tier (Database Management):** This consists of databases that hold the data relevant to the application.





1.1. Presentation Layer

The presentation layer of our application is accessible through a web browser and consists of user interface components that enable interaction with the system. It is built using three primary technologies: HTML, CSS, and JavaScript. HTML defines the content of the website, while CSS controls its visual appearance. Tailwind CSS is chosen for its utility-first approach, allowing for rapid styling and customization without the need for extensive CSS knowledge.

On the backend, Python and Django frameworks enable the website to be interactive and responsive to user actions. React, a JavaScript framework was used to make the content on the page dynamic. ReactJS is preferred as compared to other frameworks due to its numerous benefits:

- **Speed:** React's modular structure allows developers to focus on individual components, significantly speeding up the development process.
- **Flexibility:** Its modular design makes the code easier to maintain and adapt to changing requirements.
- **Performance:** React's virtual DOM and server-side rendering capabilities ensure that complex applications run smoothly and efficiently.
- **Usability:** Deploying React applications is fairly easy.
- **Reusable Components:** It saves time as we don't have to write duplicate code for the same features.

In addition to React, we have utilized Tailwind CSS for styling. Tailwind offers a highly customizable and efficient way to design UI elements, allowing us to focus on creating a consistent and scalable user experience. The benefits of using Tailwind include:

- **Rapid Development:** With Tailwind's predefined classes, we can style our application quickly without requiring extensive CSS knowledge.
- **Consistency:** Tailwind helps us keep our design looking uniform across the entire application, which is crucial for a great user experience.
- **Scalability:** With Tailwind, we can easily scale our design by applying consistent styles across different components.

1.2. Logic Tier

The Logic Tier accepts user requests from the browser, processes them and determines the routes through which the data will be accessed. The workflows by which the data and requests travel through the backend are encoded in this layer. The logic tier is implemented in Python, one of the most widely used programming languages in the world. The advantages of using Python is as follows:

- **Simplicity:** Python has a clean and readable syntax, making development faster and reducing the chances of errors. Django follows the "Don't Repeat Yourself" (DRY) principle, ensuring maintainability and efficiency.

- **Cross-platform:** Python allows applications to run seamlessly on different operating systems, including Windows, macOS, and Linux. Django provides built-in support for easy deployment across various platforms.
- **Asynchronous Support:** Django supports asynchronous processing through Django Channels, enabling efficient real-time applications such as WebSockets for live notifications and background task execution.
- **Robust & Scalable:** Django includes built-in security features, automatic database migrations, and a powerful ORM (Object-Relational Mapping) system. It supports caching mechanisms and modular architecture, ensuring scalability for handling large-scale applications.
- **Rapid Development:** With Django's batteries-included approach, developers have access to pre-built components like authentication, session management, form handling, and an admin panel, allowing for quick and efficient development.

The framework used on top of Python is **Django**. Django is an open-source, high-level web framework that simplifies the development of secure and scalable web applications. Django offers the following advantages to its developers:

- Django follows a clean and readable syntax, making it beginner-friendly while enabling rapid development.
- It provides a robust Object-Relational Mapping (ORM) system that simplifies database interactions, supports multiple database backends, and allows easy migrations without manual SQL queries.
- Django REST Framework (DRF) enhances Django's capabilities by providing tools for building efficient and scalable RESTful APIs with minimal effort.
- Django comes with pre-built configurations for settings, middleware, security, and database integration, reducing the need for manual setup.
- Django uses decorators and class-based views to streamline functionality, improve reusability, and simplify development.
- Django manages dependencies efficiently through its package system and virtual environments, ensuring compatibility and stability.
- Django comes with a built-in lightweight web server for testing applications during development without requiring external configurations.

1.3. Data Tier

The Data Tier, sometimes called the Database Tier, is where the information processed by the application is stored and managed. We have used **SQLite (db.sqlite3)**, which is the inbuilt database provided by Django. SQLite is a lightweight, self-contained, and serverless database engine that requires minimal setup and is ideal for development and small to medium-scale applications.

Below are the main advantages/benefits of SQLite:

- SQLite comes pre-installed with Python and Django, eliminating the need for additional database setup or configuration.
- As a self-contained database engine, SQLite does not require a separate database server, making it highly efficient for development and local testing.
- No complex installation or database administration is required, making it easy to set up and use.
- SQLite follows ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable transactions and data integrity.
- SQLite databases can be easily shared across different operating systems without requiring modifications.
- It requires minimal management, making it an excellent choice for development, testing, and lightweight applications.

1.4. Development and version control environment

We have used **Git** as our version control system. It is the most commonly used version control system that is used for software development and other version control tasks. It tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. It also makes collaboration easier, allowing changes by multiple people to all to be merged into one source.

We used **GitHub**, a web-based Git repository hosting service to manage our repositories and collaborate amongst ourselves.

1.5. Authentication and Authorization

- To implement authentication we have stored the email and password in our database. The password uses cryptographic techniques of hashing and salting to prevent the data being exposed in case of data leaks.
- To implement authorization, we have made use of JWT (Json Web Token)

1.6. Rest Endpoints

POST **Signup OTP**

http://localhost:8000/users/signup-otp/

Headers

Content-Type	application/json

Body (json)

```
{
  “email”: “test1@example.com”
}
```

POST Buyer/Seller Signup

http://localhost:8000/users/signup/

Headers

Content-Type	application/json

Body (json)

```
{
  "username": "test1",
  "email": "test1@example.com",
  "password": "test_one",
  "role": "buyer"
}
```

POST Buyer/Seller Login

http://localhost:8000/users/login/

Headers

Content-Type	application/json

Body (json)

```
{
  “email”: “test1@example.com”,
  “password”: “test_one”
}
```

User Profile

http://localhost:8000/users/profile/

GET **Headers**

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYXQiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyNDRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4cM

POST **Creating Store**

http://localhost:8000/stores/create/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vlINTStm3SA1Gk7-KnhGzs

Body (json)

```
{
  "name": "RM Canteen",
  "description": "Tasty Food Available",
  "location": "RM, IITK",
  "status": "open"
}
```

GET **Stores**

http://localhost:8000/users/stores/

Headers

Content-Type	application/json

Authorization Bearer null

POST Adding Categories

http://localhost:8000/products/categories/

Headers

Authorization Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vllNTStm3SA1Gk7-KnhGzs

Body (json)

```
{
  "name": "snacks"
}
```

DELETE Deleting Categories

http://localhost:8000/products/categories/{id}/

Headers

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90e

Authorization

GET **Getting Products List**

http://localhost:8000/products/products/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vllNTStm3SA1Gk7-KnhGzs

GET **Getting a Specific Product Details**

http://localhost:8000/products/products/1/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vllNTStm3SA1Gk7-KnhGzs

DELETE **Deleting a Product**

http://localhost:8000/products/products/1/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vllNTStm3SA1Gk7-KnhGzs

POST **Adding a Product**

http://localhost:8000/products/products/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vllNTStm3SA1Gk7-KnhGzs

---

Body (json)

```
{
  "name": “samosa”,
  "price": 20,
  "category": 7
}
```

POST **Adding Cart**

http://localhost:8000/cart/add/

Headers

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90e

Authorization

XBIIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYX
QiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyN
DRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI
6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4
cM

GET **Cart**

http://localhost:8000/cart/{store_id}/

Headers

Authorization

Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90e
XBIIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYX
QiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyN
DRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI
6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4
cM

DELETE **Clear Cart**

http://localhost:8000/cart/clear/{store_id}/

Headers

Authorization

Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90e
XBIIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYX
QiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyN
DRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI
6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4
cM

POST **Checkout**

http://localhost:8000/orders/checkout/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYXQiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyNDRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4cM

GET **All Orders**

http://localhost:8000/orders/myorders/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYXQiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyNDRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4cM

GET **Store Orders**

http://localhost:8000/orders/sellerorders/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5ODc0LCJpYXQiOjE3NDMxNjgwNzQsImp0aSI6ImJjYmJhMTZlNzAzNjQyZGE5ZDY2Y2ZlZDZmY2JmNDJlIiwidXNlcl9pZCI6M30.y3aVocm6ZAvpghShRnMG-vlINTStm3SA1Gk7-KnhGzs

POST **Cancel Order**

http://localhost:8000/orders/cancel/{id}/

Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzQzMTY5Mzk2LCJpYXQiOjE3NDMxNjc1OTYsImp0aSI6IjgwZTg5OTA1MjQyNDRlYmNhM2MzYzJiMmRhOGM1YWVjIiwidXNlcl9pZCI6OH0.rzdNH7ciNQ6RZe82vaxw8Sy8_jcrmI2TsriKFiCt4cM

2 Codebase

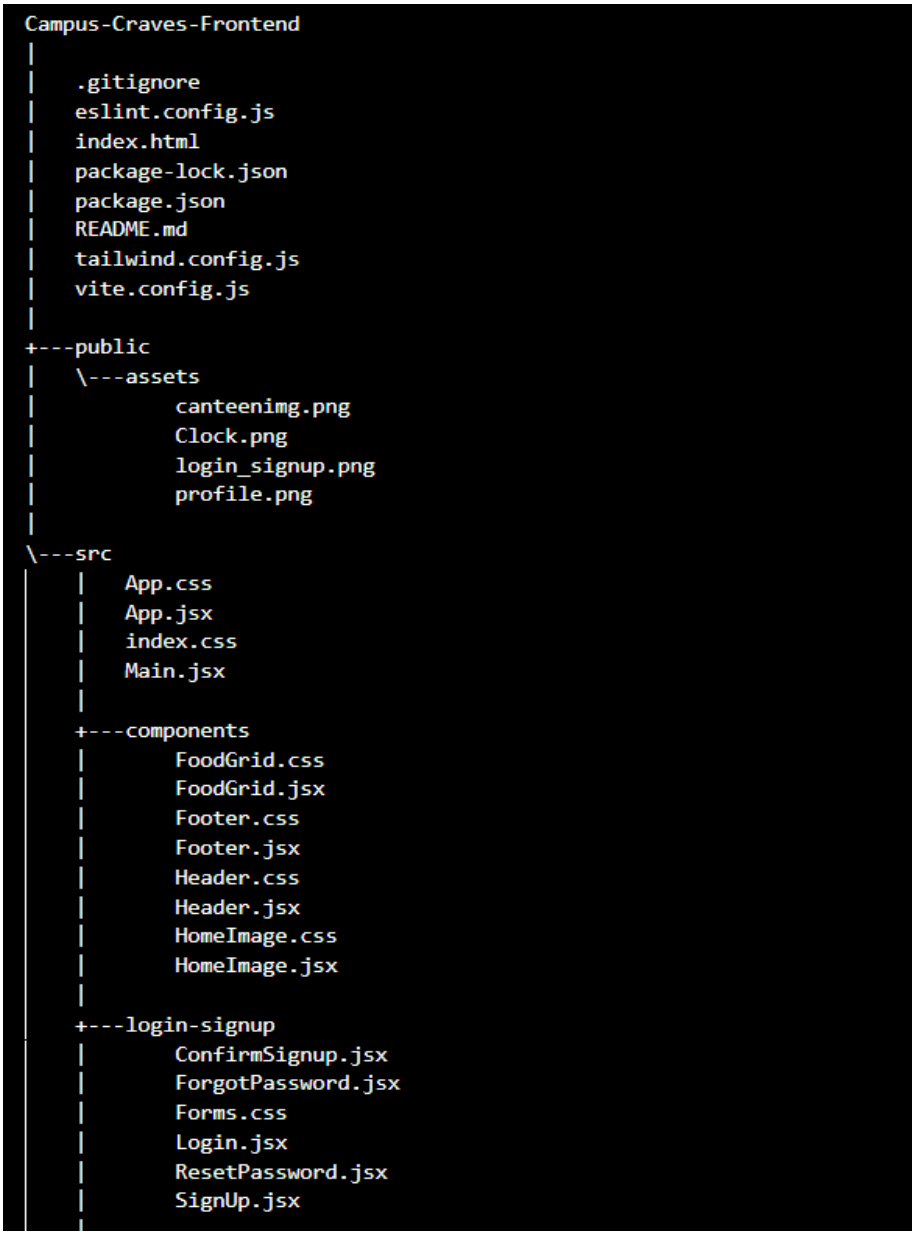
To streamline collaboration on Campus Craves, a GitHub organization has been created. This organization acts as the central hub where team members can work together on the project's source code. The project is hosted in the CS253-Course-Project repository.

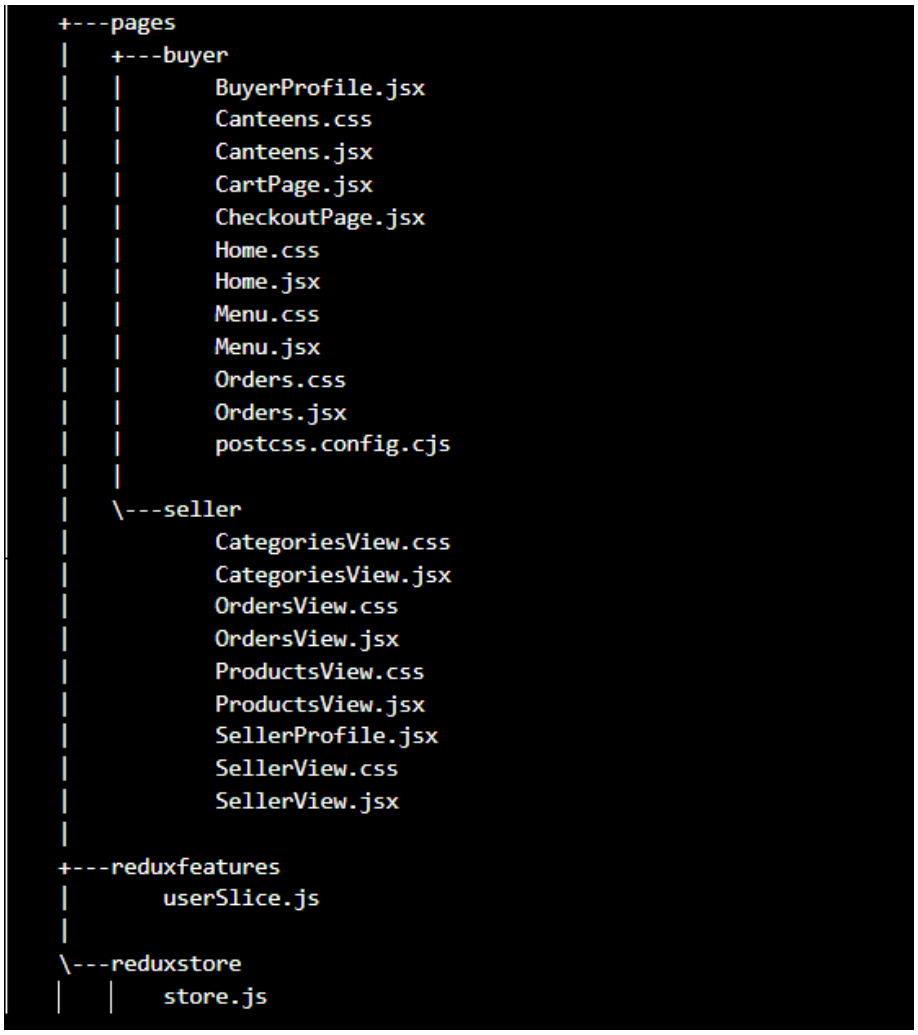
The repository is divided into two key parts:

1. **Campus-Craves-Frontend:** Handles the user interface and client-side logic.
2. **Campus-Craves-Backend:** Manages server-side operations and business logic, and database interactions through Django models.

2.1. Campus-Craves-Frontend

The directory contains the following files:





- **Top-level files:** These include essential configuration files like .gitignore, eslint.config.js, package.json, and README.md. The index.html serves as the entry point for the application, while tailwind.config.js and vite.config.js configure Tailwind CSS and Vite, respectively.
- **Public assets:** The public directory contains static assets such as images used throughout the application.
- **Source code:** The src directory holds the core frontend code:

  ➢ **Global styles and components:** Files like App.css, App.jsx, index.css, and Main.jsx provide the foundation for the application's layout and functionality.

  ➢ **Components:** Includes reusable UI components.

  ➢ **Login and signup functionality:** Contains components for user authentication.

  ➢ **Pages:** This section is divided into buyer and seller pages:

  ○ **Buyer Pages:** Includes components for the buyer's side

- **Seller Pages:** Features components for the seller's side

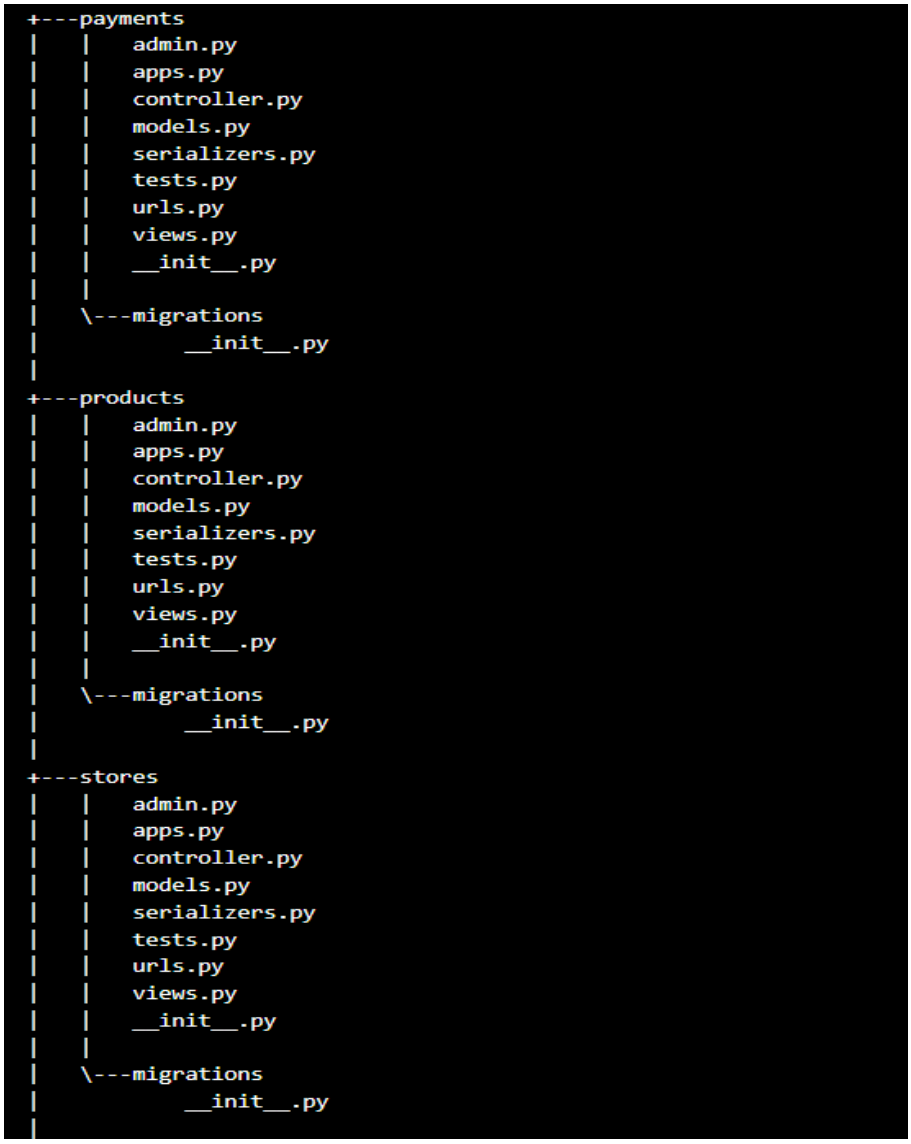
- **Redux features and store:** The reduxfeatures/userSlice.js file manages user state, and the reduxstore/store.js file sets up the Redux store for state management.

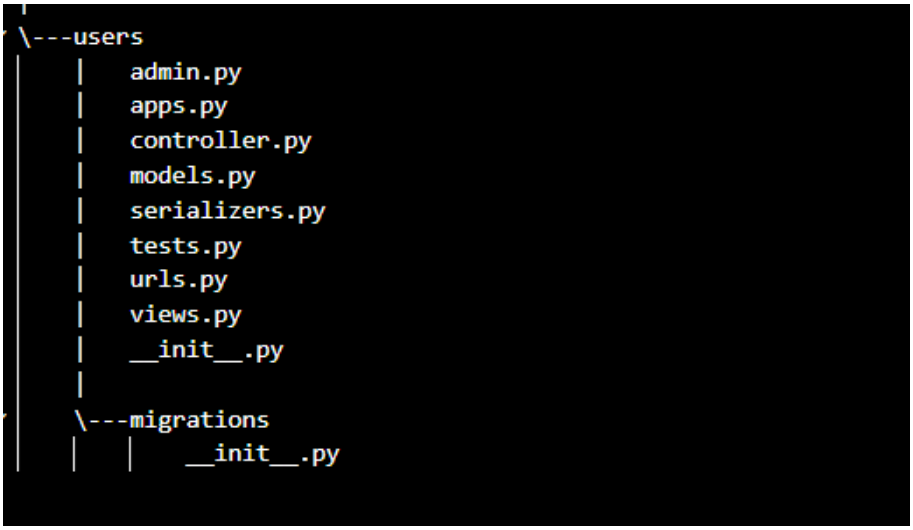
2.2. Campus-Craves-Backend

The directory contains the following files:

```
Campus-Craves-Backend

|   db.sqlite3
|   dump.rdb
|   insert_data.py
|   manage.py
|   requirements.txt
|
+---campus_craves_backend
|       asgi.py
|       celery.py
|       settings.py
|       urls.py
|       wsgi.py
|       __init__.py
|
+---cart
|   |   admin.py
|   |   apps.py
|   |   controller.py
|   |   models.py
|   |   serializers.py
|   |   tests.py
|   |   urls.py
|   |   views.py
|   |   __init__.py
|   |
|   \---migrations
|           __init__.py
|
+---orders
|   |   admin.py
|   |   apps.py
|   |   controller.py
|   |   models.py
|   |   serializers.py
|   |   tests.py
|   |   urls.py
|   |   views.py
|   |   __init__.py
|   |
|   \---migrations
|           __init__.py
|
```


+---payments
admin.py
apps.py
controller.py
models.py
serializers.py
tests.py
urls.py
views.py
__init__.py
\---migrations
__init__.py
+---products
admin.py
apps.py
controller.py
models.py
serializers.py
tests.py
urls.py
views.py
__init__.py
\---migrations
__init__.py
+---stores
admin.py
apps.py
controller.py
models.py
serializers.py
tests.py
urls.py
views.py
__init__.py
\---migrations
__init__.py


\---users
admin.py
apps.py
controller.py
models.py
serializers.py
tests.py
urls.py
views.py
__init__.py
\---migrations
__init__.py

- **Top-level files :** These include manage.py, which acts as a command-line interface to interact with the Django project, and requirements.txt, a list of all the necessary dependencies required to build and run the project.
- **campus_craves_backend :** This directory contains the main project settings and configurations. Key files here include settings.py, which configures the project's environment, and urls.py, which maps URLs to views.
- **Feature-specific modules**: These are organized into separate directories like cart, orders, payments, products, stores, and users. Each module contains:

  ➢ **Models :** It defines the structure of data entities, equivalent to schema design in databases.

  ➢ **Views:** It handles HTTP requests and returns HTTP responses, acting as the application’s endpoints.

  ➢ **Serializers:** Convert complex data into simple formats that can be easily rendered into JSON, XML, or other formats.

  ➢ **Controllers:** The controller.py files in each module manage the logic related to each feature.

  ➢ **Migrations:** It manages changes to the database schema over time.

3 Completeness

The “Section 3: Specific Requirements” of Software Requirements Specification Document lists all the desired product functionality.

- “Section 3.1: External Interface Requirements” is implemented in **Campus-Craves-Frontend**. Different user interfaces displayed in section 3.1 of the SRS document are implemented using React. The different views include - Home page, catalog, seller dashboard views (products, orders, categories), signup/login view, buyer’s view.
- All the features listed in “Section 3.2: Functional Requirements” except for 3.2.7, and 3.2.18 are implemented in **Campus-Craves-Backend**.

The application - “Campus Cravest” is aimed at digitizing various physical canteens and service providers at the IIT Kanpur campus. Currently, the implementation is done to provide the basic features for both the sellers and buyers, i.e., placing orders(for buyers), managing orders(for sellers), checking order history, browsing through store catalogs, etc. In addition to these features, we have a future development plan to improve the functionality of our product by adding more features which include:

- Adding an online payment option via credit/debit cards, net banking to increase the ease of transactions.
- Deploying a corresponding mobile app for enhancing the ease of use and accessibility. Most people prefer mobile apps over web browsers as it suits their convenience.
- Possibility of using other languages in the future for users belonging to different regional backgrounds.
- The user feedback system for each store to improvise and adapt according to the changing customer needs. This feedback will include a rating for the current transaction as well as subjective suggestions to improve the store services as well as the services provided by our application.
- Ratings for each store will be displayed along with their names so that the customer can prefer the stores with better services first. These ratings will be based upon the feedback and individual ratings provided by the user after the completion of each transaction. This will help the users to make a sound choice while selecting among different stores providing similar services.
- An order tracking mechanism might be added, which will enable the users to track their order in real-time as it reaches them. This might enhance the transparency in the whole mechanism as the user will be able to track the order and complain in case of any unnecessary delay made by the seller.

Appendix A - Group Log

The group activities during implementation were asynchronous, where we kept constant communication via our WhatsApp group. After the mid-semester recess, we also worked offline inside the KD Lab of the Dept. of Computer Science and Engineering.

Meeting Time	**Agenda/Discussions**
11 February 2025 (21:00 - 22:00 IST)	Discussed the software implementation plan, and work distribution.
15 February 2025 (21:00 - 22:00 IST)	Discussed the coding and implementation plan with TA, and made some changes in the plan.
4 March 2025 (16:30 - 17:00 IST)	Discussed the current process, and mid-semester recess tasks of the group members
16 March 2025 (14:30 - 15:30 IST)	Complete the backend and frontend code writing, review them with TA, and discuss the debugging and integration plan.
21 March 2025 (10:00 - 11:00 IST)	Debugging the backend together, tested the API calls by using Thunder Client.
25 March 2025 (20:00 - 3:00 IST)	Complete the backend frontend integration along with some minor testing
26 March 2025 (18:30 - 19:30 IST)	Final code review with TA