Implementation Document

for

Master Invoice

Version 1.0

Prepared by

Group 1: Group Name: while (1)

Cezan Vispi Damania	230310	cezanvd23@iitk.ac.in
Ch V Sai Koushik	230312	skoushik23@iitk.ac.in
Challa Kethan	230317	kethanc23@iitk.ac.in
Challa Umesh Varun	230318	cumesh23@iitk.ac.in
Chilamakuri Kundan Sai	230330	ckundans23@iitk.ac.in
Gudi Praneeth Sai	230425	gudips23@iitk.ac.in
Kishore Senthil Kumar	230566	kishores23@iitk.ac.in
Sai Saketh Mogillapalli	230895	saisaketh23@iitk.ac.in
Srijani Gadupudi	231033	srijanig23@iitk.ac.in
Vundela Obula Reddy	231178	voreddy23@iitk.ac.in

Course: CS253
Mentor TA: Hemang Mohanlal Khatri
Date: 28-03-2025

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Cezan Vispi Damania Ch V Sai Koushik Challa Kethan Challa Umesh Varun Chilamakuri Kundan Sai Gudi Praneeth Sai Kishore Senthil Kumar Sai Saketh Mogillapalli Srijani Gadupudi Vundela Obula Reddy	The first version of the implementation document	28/03/2025

1. Implementation Details

Our Web Application, **Master Invoice,** has three major components :

1.1 Backend

1. **Programming Language:** The backend of Master Invoice is built using Python. Python's simplicity, readability, and rich library support make it a highly flexible choice for development work.

2. **Framework:** We used the Django framework. We chose Django because we believe it to be robust yet simple, reliable, and scalable. Some other advantages of Django that make it a popular choice are:

    - **Simple Syntax:** Django is written in Python.

    - **Security:** Django offers robust user authentication functionality.

    - **Built-in Admin Interface:** Django includes an admin interface to quickly create, read, update, and delete data without writing additional code.

    - **Object-Relational Mapping:** Django's ORM simplifies database interactions by abstracting SQL queries into Python code, reducing efforts and increasing productivity.

    - **Rapid Development:** Django supports rapid development, like its "batteries-included" philosophy, and saves time and effort, making development faster and more efficient.

    - **Community Support:** Django has a large and active community of developers, providing extensive resources, support, and third-party packages.

    - **Automatic URL Routing:** Django automatically handles URL routing, simplifying the mapping of URLs to views and reducing manual configuration overhead.

    - **Modularity of Code:** Django's modularity allows developers to break down their code into smaller, reusable components, making it easier to manage and maintain.

3. **Libraries and Packages:**
    - **Django Admin LTE:** Django Admin LTE improves the appearance and functionality of Django admin pages. It also offers customization options for tweaking the design and includes appealing charts for better data presentation.

- **Django Crispy Forms:** A library that makes styling Django forms easier by integrating Bootstrap and other frameworks, allowing for clean and responsive form designs.
- **Django Crispy Bootstrap5**: An extension of Django Crispy Forms that specifically integrates Bootstrap 5, allowing for easy and visually appealing form styling with minimal effort.

4. **Authentication**:
    - User authentication is handled using Django's built-in authentication system with secure password storage and OTP verification.
        - **Password Handling :** Passwords are hashed using **PBKDF2** with SHA-256, as per Django's default security standards. Password verification uses constant-time comparison to mitigate timing attacks
        - **OTP Verification:** OTPs are generated using Python's `secrets` library for cryptographic security. Each OTP is hashed with **SHA-256** before storage and user input is hashed and compared securely during verification. **Time-based expiration** is used to prevent replay attacks additional to rate limiting.

1.2 Frontend

1. **Programming Languages:** Master Invoice's front end is built using **HTML, CSS, and JavaScript**. This combination allows for a clean, structured, and responsive user interface while ensuring compatibility across all modern web browsers.

2. **Framework:** We have not used any additional front-end frameworks or over-the-top libraries. Instead, we focused on **HTML, CSS, and JavaScript** to keep the application lightweight and maintainable.

3. **Libraries Used:** Used **JS Query** for dynamic content manipulation and **Chart.js** for interactive data visualization in my projects. These libraries help streamline development and enhance user experience.

1.3 Database

For the database, we have used **Django's built-in database** as our data storage solution. We chose this because it is lightweight, efficient and seamlessly integrates with Django, making development and deployment straightforward. Some advantages that make it a great choice include:

- **Lightweight and Easy to Set Up:** Django's built-in database requires no additional configuration, making it perfect for quick development and testing.
- **Seamless Django Integration:** Since it comes pre-configured with Django, it ensures smooth ORM (Object-Relational Mapping) functionality without extra setup.
- **Optimized for Development:** The built-in database is highly effective for small to medium-scale applications, enabling rapid prototyping and testing.
- **Zero Dependency Hassle:** Unlike external databases, Django's built-in database does not require separate installations or complex configurations, reducing setup time.
- **Cross-Platform Compatibility:** It works across various operating systems and environments, ensuring flexibility in development and deployment.

1.4 Building Systems:

As our project is a web application developed using **Django**, we utilize Django's built-in tools and commands for managing and deploying the application efficiently. Some key components include:

- **Django Management Commands:** The `manage.py` script provides various commands for tasks such as running the development server, applying database migrations, and handling static files.
- **Virtual Environments:** We use Python virtual environments (`venv`) to manage dependencies and ensure a consistent development environment.

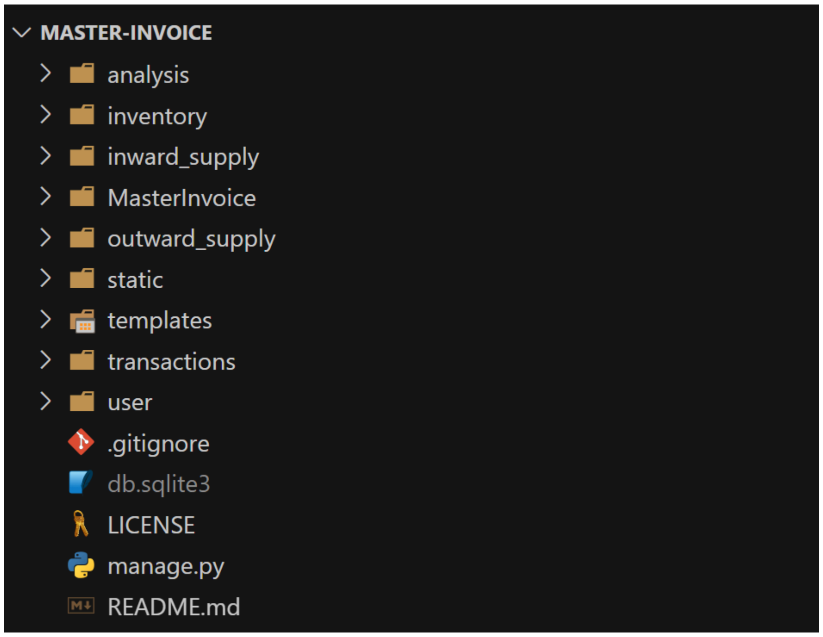
2. Codebase

2.1 Github Repository:

Link: https://github.com/me-kethanchalla/Master-Invoice

2.2 Codebase Structure:

Overview of Directories



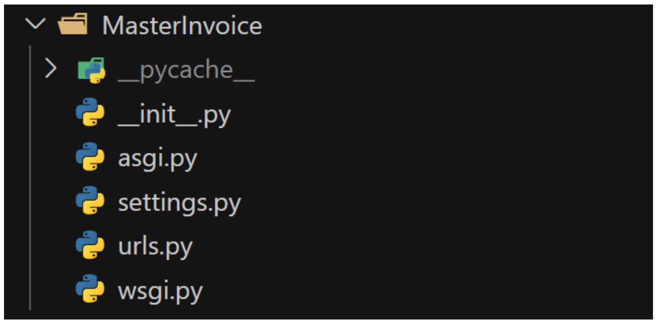
```
MASTER-INVOICE
  analysis
  inventory
  inward_supply
  MasterInvoice
  outward_supply
  static
  templates
  transactions
  user
  .gitignore
  db.sqlite3
  LICENSE
  manage.py
  README.md
```

The overall structure of the code, as displayed in the VSCode file explorer, consists of the following components:

- **Project Module (`MasterInvoice/`)**: Contains project SETTINGS , URLs, and WSGI/ASGI configurations.
- **Django Apps**: Includes `analysis`, `inventory`, `inward_supply`, `outward_supply`, `transactions`, and `user`.
- **Static Files (`static/`)**: Stores CSS stylesheets, images, and JavaScript files.
- **Templates (`templates/`)**: Holds HTML templates for web page structure and layout.
- **Supporting Files**:
  - `.gitignore` (Specifies files to be ignored by version control)
  - `manage.py` (Django's command-line utility)
  - `README.md` (Project documentation)
  - `LICENSE` (Legal terms for usage and distribution)
  - `db.sqlite3` (Development database)

Project Module

The overall structure of the inner Django project directory is given below:

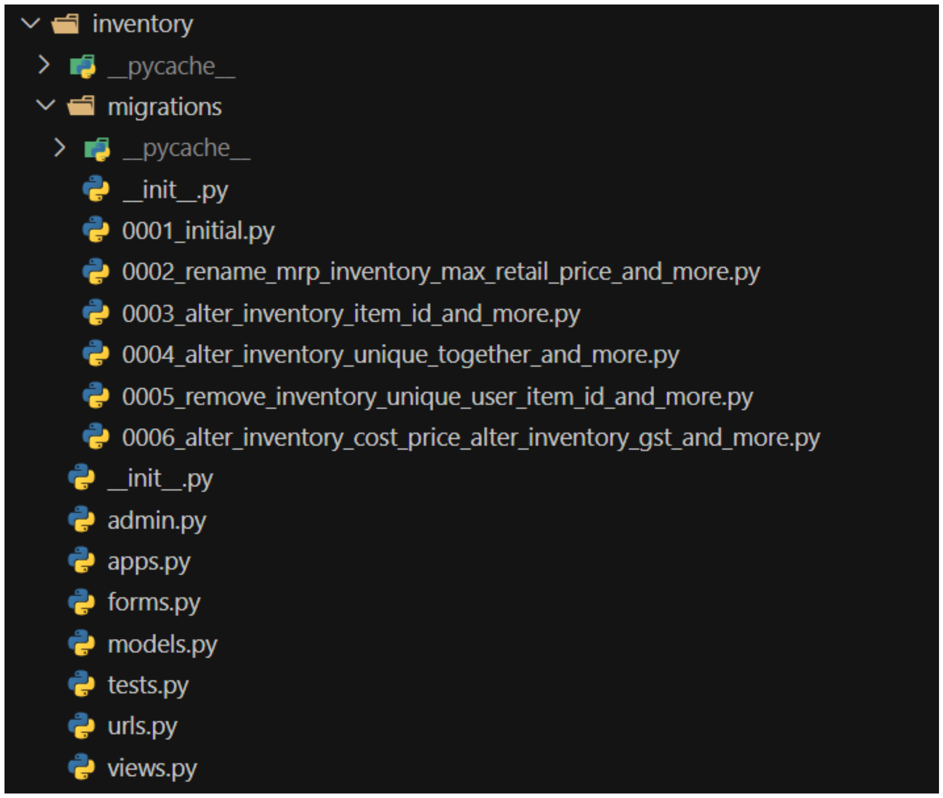


```
MasterInvoice
  __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
```

- `__init__.py`: Marks the directory as a Python package.
- `asgi.py` : Configures Django for ASGI.
- `settings.py`: Stores project settings (e.g., database and installed apps).
- `urls.py`: Defines global URL routing (maps URLs to apps/views).
- `wsgi.py` : Configures Django for WSGI.

Django Apps

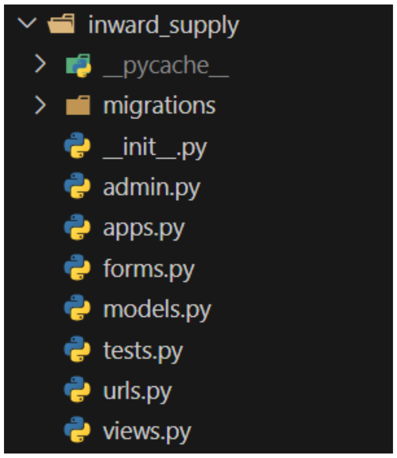
The overall structure of an app's directory is given below:



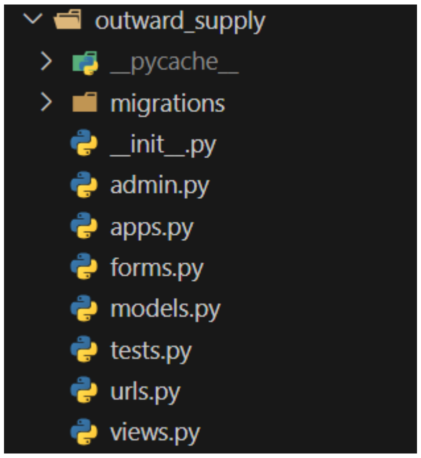
```
inventory
  __pycache__
  migrations
    __pycache__
    __init__.py
    0001_initial.py
    0002_rename_mrp_inventory_max_retail_price_and_more.py
    0003_alter_inventory_item_id_and_more.py
    0004_alter_inventory_unique_together_and_more.py
    0005_remove_inventory_unique_user_item_id_and_more.py
    0006_alter_inventory_cost_price_alter_inventory_gst_and_more.py
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```

It consists of components like:

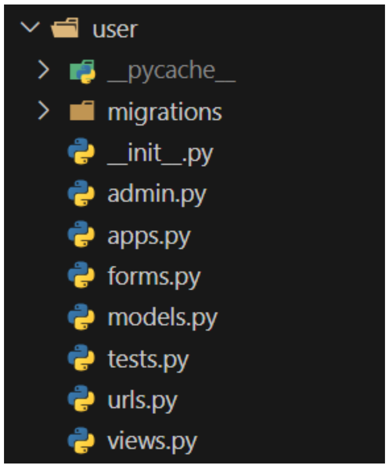
- `__init__.py`: Marks this directory as a Python package.
- `admin.py`: Registers models for Django's admin panel.
- `apps.py`: Stores app-specific configurations.
- `models.py`: Defines database models using Django ORM.
- `views.py`: Contains functions/classes to handle HTTP requests.
- `tests.py`: Defines test cases for automated testing.
- `urls.py`: Defines URL patterns for this app (manually created).
- `forms.py`: Defines Django forms to handle user input.
- `migrations/`: Stores migration files to track database schema changes.



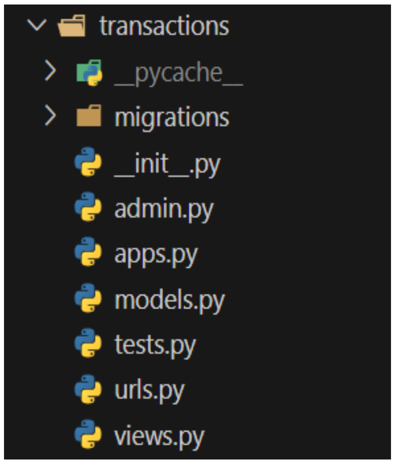
```
inward_supply
  __pycache__
  migrations
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```



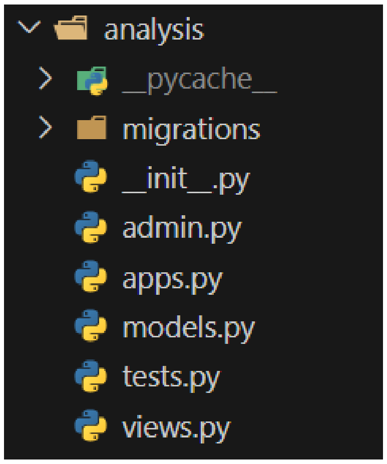
```
outward_supply
  __pycache__
  migrations
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```



```
user
  __pycache__
  migrations
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```



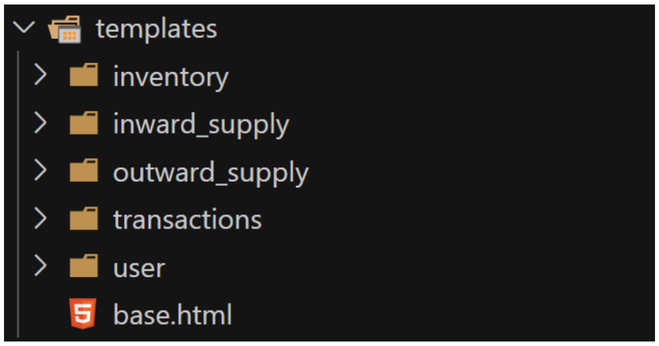
```
transactions
  __pycache__
  migrations
  __init__.py
  admin.py
  apps.py
  models.py
  tests.py
  urls.py
  views.py
```



```
analysis
  __pycache__
  migrations
  __init__.py
  admin.py
  apps.py
  models.py
  tests.py
  views.py
```

Templates Directory (for HTML files)

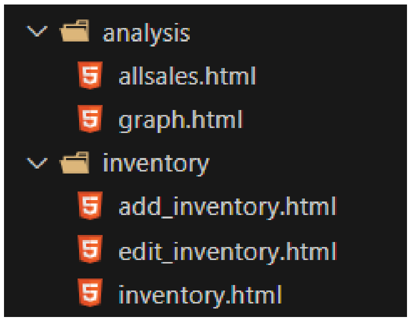
The overall structure of the templates directory is as below:



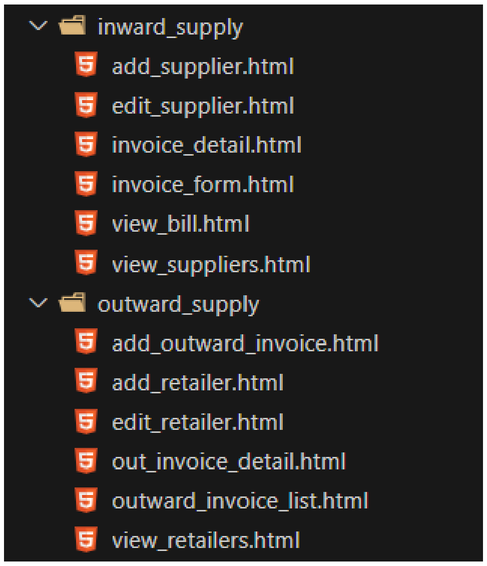
```
templates
  inventory
  inward_supply
  outward_supply
  transactions
  user
  base.html
```

The `templates/` folder contains HTML template files used for rendering web pages, structured as follows:

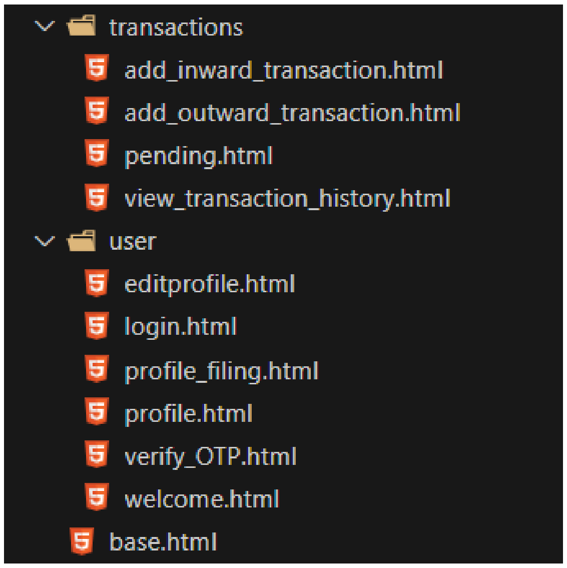
- **App-Specific Templates**: `inventory/`, `inward_supply/`, `outward_supply/`, `transactions/`, `user/`.
- **Base Template (`base.html`)**: A common layout template that other pages extend like navbar, footer, etc.



```
analysis
    allsales.html
    graph.html
inventory
    add_inventory.html
    edit_inventory.html
    inventory.html
```



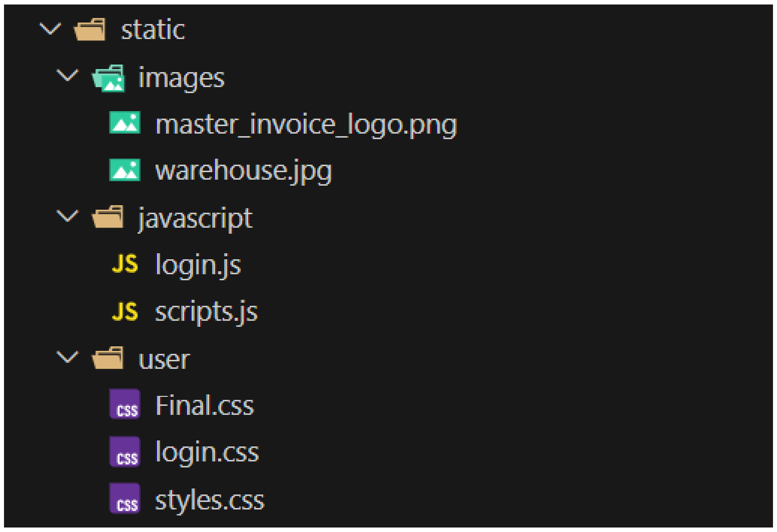
```
inward_supply
    add_supplier.html
    edit_supplier.html
    invoice_detail.html
    invoice_form.html
    view_bill.html
    view_suppliers.html
outward_supply
    add_outward_invoice.html
    add_retailer.html
    edit_retailer.html
    out_invoice_detail.html
    outward_invoice_list.html
    view_retailers.html
```



```
transactions
    add_inward_transaction.html
    add_outward_transaction.html
    pending.html
    view_transaction_history.html
user
    editprofile.html
    login.html
    profile_filing.html
    profile.html
    verify_OTP.html
    welcome.html
base.html
```

Static Directory (for images, css, js files)

The overall structure of the static directory is as below:



```
static
  images
    master_invoice_logo.png
    warehouse.jpg
  javascript
    login.js
    scripts.js
  user
    Final.css
    login.css
    styles.css
```

The `static/` folder contains static assets used in the project, categorized as follows:

- **Images (`images/`)**: Stores all the images needed in the application, (`master_invoice_logo.png`) and other visuals (`warehouse.jpg`).
- **JavaScript (`javascript/`)**: Contains JavaScript files for handling frontend interactions (`login.js`, `scripts.js`).
- **CSS (`user/`)**: Includes style sheets needed for multiple HTML files from templates.
  - `Final.css` : Contains the CSS file, which is used for the main landing page.
  - `login.css` : For the register or login page which is the first page visible for the user when the application is opened.
  - `styles.css :` We have used a single css file containing styling for the nav bar and the template box inside which the content in each page is visible individually. As of now, for the styling needed in any other locations of the code is specifically written inside the file as internal and inline css.

2.3 Instructions to Setup Development Environment:

1. **Fork the Repository:**
   Create a fork of this repository into your github account. To merge any changes made into the main repository, a pull request must be created.

2. **Clone into remote machine:**
   Clone the repository and open it in an IDE of your choice.

3. **Install Python:**
   Ensure that Python is installed on your system. You can verify the installation by running:

   ```
   python --version
   ```

4. **Setup a virtual python environment: (recommended) :**
   Create and activate a virtual environment before installing dependencies. Use the following commands:

   ```
   python -m venv venv
   venv\Scripts\activate  (windows)
   ```

5. **Install Django and Required Dependencies:**
   Install Django and other dependencies like bootstrap5, crispy-bootstrap5, and python date-time module using `pip`:

   ```
   pip install django bootstrap5 crispy-bootstrap5 python-dateutil
   ```

6. **Run Migrations:**
   Create migrations: (if you have made changes to models)

   ```
   python manage.py makemigrations
   ```

   Apply database migrations:

   ```
   python manage.py migrate
   ```

7. **Start the Development Server:**

   Run the Django development server:

   ```
   python manage.py runserver
   ```

   The application should now be accessible at `http://127.0.0.1:8000/`

3. Completeness

We were able to successfully implement nearly all of the features that we have mentioned in the SRS.

3.1 SignUp/Login:

- We have implemented the sign-up and Login features as written in the SRS.
- Allows users to register with Master Invoice by providing their full name, phone number, email address, and password.
- For a new user, we have given the SignUp feature in which the user can fill his details and register in the software, as we have mentioned in the SRS.
- We have successfully implemented OTP generation to verify the user during Sign-up.
- Users need to input credentials such as username and password to access their account.

3.2 Profile:

- This page is accessible only if the user is logged in.
- The profile page serves as a personalized space where users can view and manage their information like firm name, username, name, password, address, etc.
- This page provides a means for users to keep their profiles accurate and up-to-date..
- The edit profile page allows users to modify and update their personal information within the website. Users can typically change their firm name, username, name, password, address, etc.

3.3 Homepage:

- Upon successfully signing into their account, the users will be directed to the Dashboard/Homepage. The dashboard acts like a directory for all functional aspects. It can direct to functions like **Inventory**(used to manage current stock)**, Inward Supply**(used to manage the inflow of products), **Outward Supply**(used to manage the outflow of products)**, and Transactions**(used to manage monetary conditions), which hold their functionalities as mentioned in SRS.
- The navigation bar holds the above-mentioned functionalities as we move towards the right. There are more functionalities like **Analysis (**which can be used to understand trends)**, Contact Us (a** way to contact the technical team in case of any issue)**, Profile**(to view and edit user details)**, and Logout.**
- We also implemented this navigation bar on every page of the website, which allows the users to roam to any required page from the current page.
- This home page looks exactly like the image provided in the SRS document.

3.4 Inventory:

3.4.1 Add Product:

- We have implemented “ADD PRODUCT” using this user can add new product to the inventory by filling it’s details such as product name, item ID, quantity, cost price, sale price, maximum retail price, GST(%)

3.4.2 View Inventory:

- We have implemented “VIEW INVENTORY” by clicking on it. Users can view the list of all products in the inventory with details such as product name, item ID, quantity, cost price, sale price, maximum retail price, GST(in %), profit, and total sold.
- We have also implemented “SEARCH” by using this user can search an item by its name or product ID.
- The quantity of the product automatically changes when a new inward supply happens.
- We have implemented “EDIT,” using which users can edit the product’s information, such as product name, item ID, quantity, cost price, sale price, maximum retail price, and GST.
- We have implemented “DELETE” using which user can delete a particular product from the inventory list.

3.5 Supply Inflow:

3.5.1 Add Supplier:

- We have implemented “ADD SUPPLIER” using this user can add a new supplier to the supplier list by filling in details of the new supplier such as supplier name, firm name, phone number, email ID, address.

3.5.2 View Suppliers:

- We have implemented “VIEW SUPPLIERS”, using which user can view the list of all suppliers with details such as supplier name, firm name, phone number, email, address,
- We have also implemented “SEARCH” by using this user can search a supplier by his firm name or his name.
- The debit of the supplier automatically when a new inward transaction happens.
- We have implemented “EDIT,” using which users can edit the supplier’s information such as supplier name, firm name, phone number, email, and address.
- We have implemented “DELETE” using which user can delete a particular supplier from the suppliers list.

3.5.3 Enter Supplier's Invoice Bill:

- The user will be able to add an invoice bill generated by the supplier, using which the system updates the user's inventory and finances instantaneously.
  The user can manually enter the invoice bill with the below details.
  - Supplier
  - Bill Number (given by supplier)
  - Date
  - Details of products (product and quantity)

3.5.4 View Supplier's Invoice Bills:

- The user will be able to view all the invoice bills added from suppliers.
- We have also implemented "SEARCH" to search the bill with firm name or bill number or with the date.

3.6 Supply Outflow:

3.6.1 Add Retailer :

- We have implemented "ADD RETAILER" using this user can add a new retailer to the retailer list by filling in details of the new retailer such as retailer name, firm name, phone number, email ID, address.

3.6.2 View Retailers :

- We have implemented "VIEW RETAILERS", using which user can view the list of all retailers with details such as retailer name, firm name, phone number, email, address,
- We have also implemented "SEARCH" by using this user can search a retailer by his firm name or his name.
- The credit of the retailer automatically when a new outward transaction happens.
- We have implemented "EDIT" in which users can edit the retailer's information, such as the retailer's name, firm name, phone number, email, and address.

3.6.3 Billing to Retailer :

- The user will be able to add a retail invoice bill generated for a retailer, which updates the system's inventory and finances instantaneously.

The user can manually enter the retail invoice bill with the following details:

- Retailer
- Date
- Details of products (product, quantity, cost, etc.)

3.6.4 View Retailer’s Invoice Bills :

- The user will be able to view all the retail invoice bills generated for retailers.
- We have also implemented “SEARCH” to search the bill with firm name or bill number or with the date.

3.7 Transactions:

3.7.1 Add Inward Transactions :

- It deals with the payment inflow for the user.
- The user may add a transaction when it is completed. Each added transaction requires the user to fill in data for attributes like ‘Supplier/Retailer’ and ‘Amount’. The system also records the date on which the transaction has been added and adds it to the ‘Date’ attribute.
- Once a transaction is added, the pending transaction to that supplier/retailer is automatically updated in the Credit/Debit tables.

3.7.2 Add Outward Transactions :

- It deals with the payment outflow for the user.
- The user may add a transaction when it is completed. Each added transaction requires the user to fill in data for attributes like ‘Supplier/Retailer’ and ‘Amount’. The system also records the date on which the transaction has been added and adds it to the ‘Date’ attribute.
- Once a transaction is added, the pending transaction to that supplier/retailer is automatically updated in the Credit/Debit tables.

3.7.3 View Transaction History :

- The user can view all their previous transactions with recent transactions on top.

3.7.4 Pending Transactions :

- The transactions are divided into two tables: ‘Credit’ and ‘Debit’. The ‘Credit’ table displays all the transactions in which the user must pay the supplier, and the ‘Debit’ table displays all the debts owed by the retailers.

- The user can view a list of all the pending transactions to suppliers/retailers, search using different filters, and sort based on different parameters. Each pending transaction has two attributes: ‘Name of Supplier/Retailer’ and ‘Amount’.
- The pending transactions are updated automatically by the system whenever the user adds invoice data and are modified based on added transactions.

3.8 Analysis :

- The user can analyze all of his product exchanges and their transactions using various data representation methods(like graphs, etc.)
- Analysis of data makes the user understand trends about how certain products are performing and their role in business, and it may help him make new business choices more effectively based on his previous products' performance.

3.8.1 Graphical Representation :

- The user can analyze trends in business in a specific selected period over various useful variable choices.

3.8.2 Tabular Representation :

- The user can analyze the whole data corresponding to products and overall transactions happening.

3.8.3 Statistical Representation :

- The user can analyze data corresponding to a particular product supplier or retailer in a very simple yet effective way of representation.

3.9 Contact Us :

- In case of any issue with the application, the user can contact the technical team using the contact us option.

3.10 Future Implementations :

- The requirement mentioned in the SRS document regarding generating a way bill and displaying the bill in a specific format is not implemented. This feature will be implemented in the next version.
- As of now, we have not given the user the ability to change his email ID. This will be implemented in the next version using OTP verification.
- We are currently implementing a test version of OTP verification, which will be replaced with SendGrid API.
- There is a minor issue related to the Log-out option. This will be fixed in the next version.

- In the next version, the user will also be able to sort and filter the bills by price, date, etc.
- The CSS is not uniform throughout the website and is somewhat different from the images provided before. This will be fixed in the next version.
- We are currently using the django inbuilt database db.sqlite3, which will be replaced by PostGreSQL due to its more extensibility and scalability in the next version.

Appendix A – Group Log

The table only records the meetings among group members. Each member has been consistently working on their assigned tasks, and the actual time and effort invested by the team far exceed what is documented here.

SL. No.	Date	Timings	Venue	Description
1	10/02/2025	17:00 - 19:30	RM Building	Split the team into Frontend and Backend to plan implementation.
2	17/02/2025	17:00 - 18:30	RM Building	Discussed the progress made and merged all progress via pull requests and made development in basic outline of codes.
3	23/02/2025	16:00 - 18:30	KD Library	Further django app-specific work distribution and discussion of brief details regarding requirements.
4	03/03/2025	14:00 - 19:00	KD Library	Resolved issues with current progress and merged further changes.
5	12/03/2025	14:30 - 16:30	Google meet	Further distribution of work. Unit testing of different functionalities work distribution has been done by different groups
6	20/03/2025	14:00 - 17:00	KD Library	Work Distribution regarding Implementation Document, Deployment, Frontend and Backend modifications
7	23/03/2025	20:30 - 23:00	KD Library	Began work on the Implementation doc and merged progress.
8	27/03/2025	19:30 - 22:00	KD Library	Worked on the Implementation document. Merged further changes.