Implementation Document

for

Re_Store

Version 1.0

Prepared by

Group 4 **Group Name : Divide and Conquer**

Bharatula Anirudh Srivatsa	230290	bharatula23@iitk.ac.in
Voora Rakesh	231174	vrakesh23@iitk.ac.in
Soma Koushik	231018	koushiks23@iitk.ac.in
Sanjay Raghav Vangala	230916	sanjayra23@iitk.ac.in
Yashwanth Reddy Junutula	231194	yashwanth23@iitk.ac.in
Saatvik Gundapaneni	230428	gundapanen23@iitk.ac.in
Vempati Prem Santhosh	231137	psanthosh23@iitk.ac.in
Ayush Yadav	230272	ayushydv23@iitk.ac.in
Meghana Kadari	230512	kmeghana23@iitk.ac.in
Chapati Venkata Pranaya	230324	chapative23@iitk.ac.in

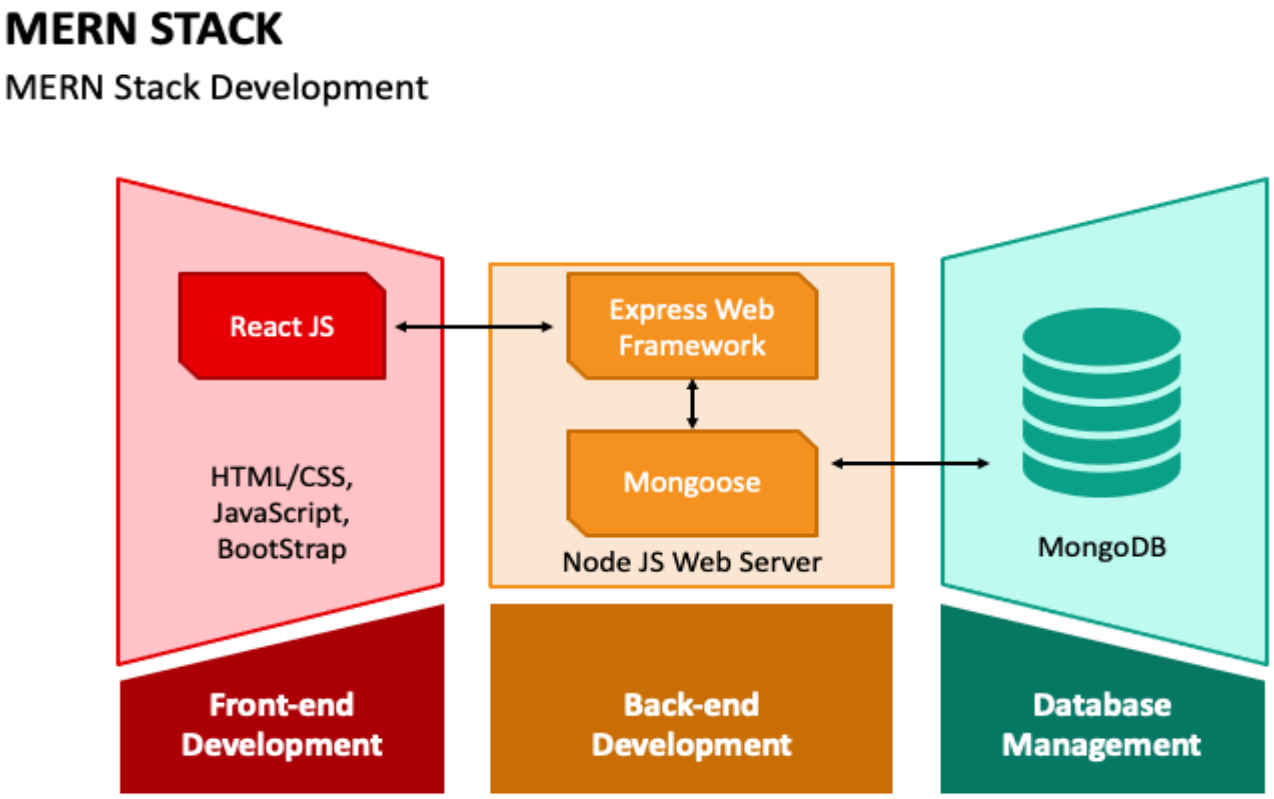
Course: CS253
Mentor TA: Jeswaanth Gogula
Date of Submission: 28th March 2025

Table Of Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.00	Bharatula Anirudh Srivatsa Voora Rakesh Soma Koushik Sanjay Raghav Vangala Yashwanth Reddy Junutula Saatvik Gundapaneni Vempati Prem Santhosh Ayush Yadav Meghana Kadari Chapati Venkata Pranaya	The First Draft of Re_Store Implementation.	26/03/25

1 Implementation Details



Re_Store is a comprehensive web-based e-commerce platform built using the MERN stack, which leverages MongoDB, Express.js, React.js, and Node.js. This technology stack was chosen for its ability to seamlessly handle the complex requirements of our platform.

MongoDB, a NoSQL database, provides a flexible and scalable data storage solution, ideal for managing diverse billing data efficiently.

Express.js, a lightweight and flexible web application framework for Node.js, simplifies backend development by offering robust routing and middleware capabilities.

React.js, a powerful JavaScript library, enables us to create dynamic and interactive user interfaces, ensuring a seamless user experience.

Node.js, with its event-driven architecture, allows for non-blocking I/O operations, ensuring high performance and scalability for our web application.

Programming Language, Framework, and Libraries

Programming Language:

Re_Store has been developed primarily using **Javascript** programming language. Re_Store follows the model-view-controller architectural pattern.

For the **Backend**, we have used:

1. 1. **Javascript**: The benefit of using **Javascript (Node.js)** for the backend is that, as compared to C++/Java, it offers much more flexibility by providing a wide range of frameworks enabling efficient asynchronous operations, code reusability with React.js, and scalability with its robust ecosystem of tools and resources.

For the **Frontend**, we have used:

1. **HTML**: HTML is used for structuring web pages. Every browser supports HTML, and it is very easy to use.
2. **CSS**: Cascading Style Sheets (CSS) are employed for styling HTML elements, providing a consistent and visually appealing layout across multiple web pages. It is also easy to maintain. Making a global change is simple: just update the style, and all elements across all web pages will be automatically updated.
3. **JS**(Javascript): JavaScript plays a dual role here. While Node.js handles the backend, JavaScript on the front end adds interactivity to the user interface and makes the application dynamic for the user. JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server. Also, it is easy to implement.

For the **DBMS** (Database Management System), We have used:

1. **MongoDB**: The reason behind this is that MongoDB is faster than MySQL due to its ability to handle large amounts of unstructured data when it comes to speed. This makes it perfect for storing the diverse product information managed by Re_Store.

Frameworks:

Build System: We have used **Node.js** and **NPM** due to the following benefits:

1. **NPM** (Node Package Manager): Node.js comes with a built-in package manager called npm, which is the largest ecosystem of open-source libraries in the world.
2. Npm allows developers to easily install, manage, and share reusable code packages, making it quick and convenient to add functionality to Node.js applications.
3. It provides numerous libraries and reusable templates.

Node.js: We have used **Node.js** as our backend framework, which has the following features:

1. Node.js is a powerful JavaScript runtime environment that allows developers to build scalable and high-performance applications.
2. Node.js is designed to handle asynchronous I/O operations efficiently. It uses an event-driven, non-blocking I/O model, which allows multiple operations to be performed concurrently without blocking the execution thread. This makes Node.js well-suited for handling high-throughput applications, such as web servers and real-time applications.
3. Node.js is cross-platform, meaning it runs on various operating systems such as Windows, macOS, and Linux. This allows developers to write applications once and deploy them across different platforms without modification, increasing development efficiency and flexibility.
4. Node.js provides support for streaming data, allowing developers to process large files or data streams incrementally without loading the entire dataset into memory. This is particularly useful for handling file I/O, network communication, and real-time data processing.
5. Node.js facilitates seamless integration with cloud platforms, enabling effortless deployment, scaling, and management of applications in distributed environments.

Libraries:

We have used the **Express** and **ReactJS** libraries for our project due to the following benefits:

EXPRESS

1. The Express library is a popular web application framework for Node.js.
2. It simplifies the process of building web applications and APIs in Node.js by providing a minimalistic and flexible set of features.
3. It allows developers to use middleware functions to perform tasks such as parsing request bodies, handling sessions, authentication, logging, etc. Middleware functions can be chained together to handle requests in a modular and reusable way.
4. It provides a simple and intuitive way to define routes for handling different HTTP requests (GET, POST, PUT, DELETE, etc.) on various URLs.
5. Express simplifies error handling by providing middleware that can catch and process errors, making it easier to manage and debug applications.

REACT.JS

1. React.js is a popular JavaScript library for building user interfaces, particularly for web applications. Developed by Facebook, React.js focuses on component-based architecture and provides a declarative syntax for creating interactive UIs.
2. React.js utilizes a virtual DOM to improve performance. Instead of directly manipulating the browser's DOM. React.js works with a lightweight representation of the DOM called the virtual virtual DOM. React then compares the virtual DOM with the actual DOM and only updates the parts that have changed, resulting in faster rendering.
3. React.js follows a unidirectional data flow pattern, where data flows down from parent components to child components via props. This helps in maintaining a clear and predictable data flow throughout the application.

AUTHENTICATION:

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. For this we have used the Bcrypt library. For verification of email we have used the nodemailer library to send emails. In conclusion, Re_Store leverages a well-orchestrated blend of technologies, each meticulously chosen to optimize performance and streamline development. JavaScript, acting as both foundation (Node.js) and interactive layer (frontend), provides flexibility and reusability. The tried-and-true trio of HTML, CSS, and JavaScript on the front end ensures a user-friendly and visually appealing interface. MongoDB's strength in handling large amounts of unstructured data makes it the perfect fit for storing billing information. Finally, the power of frameworks like Node.js with NPM and libraries like Express and ReactJS expedites development, promotes code modularity, and prioritizes a smooth user experience. This strategic selection of technologies empowers Re_Store to deliver a robust, scalable, and user-centric Software.

2 Codebase

Github Repository- Re_Store
Hosted Website- Re_Store

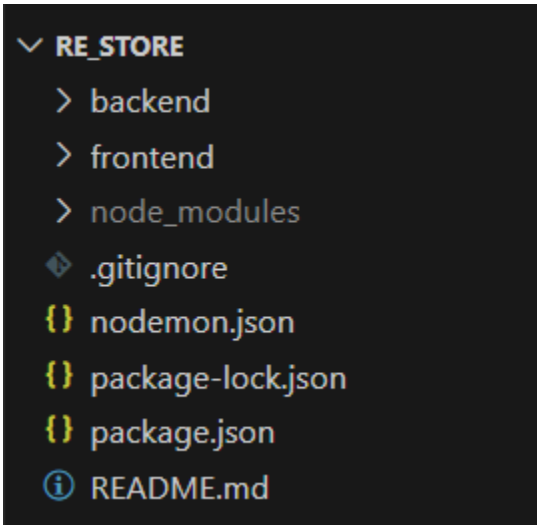
Code Structure:

The above Github link contains all the source code of our web application along with a README file, which contains the instructions for using the code.

The repository is divided into 2 main parts:

- Frontend
- Backend

The following images show the overall structure of the Re_Store repository:



Frontend Folder Structure Overview

The frontend folder contains all the necessary files for the React-based client-side application, managing UI components, routing, and API interactions. Below is a breakdown of its structure:

- **`index.html` –** Main HTML File
  The root HTML file serves as the entry point for the React app. It contains the `<div id="root"></div>` where React components are mounted and typically includes script links for loading the React application.

- **`src/` –** Main Source Code Directory
  This directory contains all the source code and logic for the React application.

  - **`assets/` –** Static Files
    Stores static assets such as images, icons, fonts, or other media files required for the UI. These files are typically imported into components when needed.

  - **`chat/` –** Chat Functionality
    Likely contains chat-related components and logic, including WebSocket integration or message-handling functions for real-time communication.

  - **`components/` –** Reusable UI Components
    Holds modular UI components like buttons, modals, navigation bars, and forms that are used across different pages. These components help maintain a consistent UI design.

  - **`context/` –** React Context for State Management
    Contains React Context API implementations that manage global state across the application. It may include files for handling user authentication, theme settings, or chat state.

  - **`pages/` –** Page Components for Routing
    Stores full-page components used in React Router for navigation. Examples include Home, Login, Signup, Dashboard, Profile, and other views that represent different sections of the application.

  - **`services/` –** API Calls & Utility Functions
    Likely includes functions for making HTTP requests using `fetch` or `axios`, handling authentication, and interacting with the backend. This separation improves code organization and reusability.
  - **`App.css` & `index.css` –** Global Stylesheets

    - `App.css` – Contains styles specific to the `App.jsx` component.

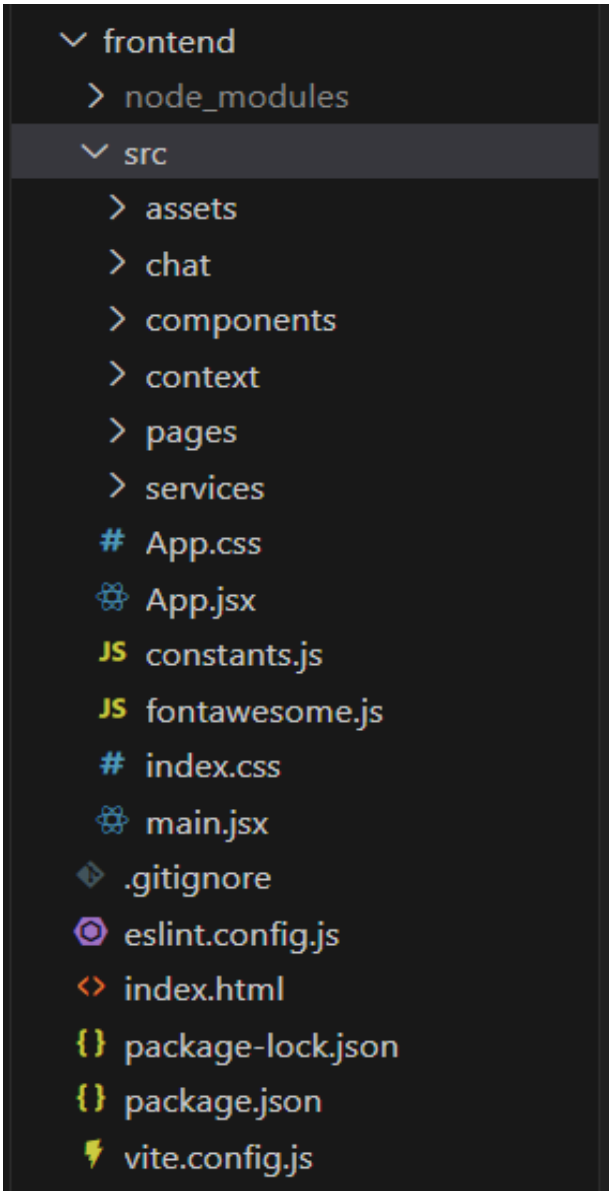
    - `index.css` – Defines global styles applied across the entire application.

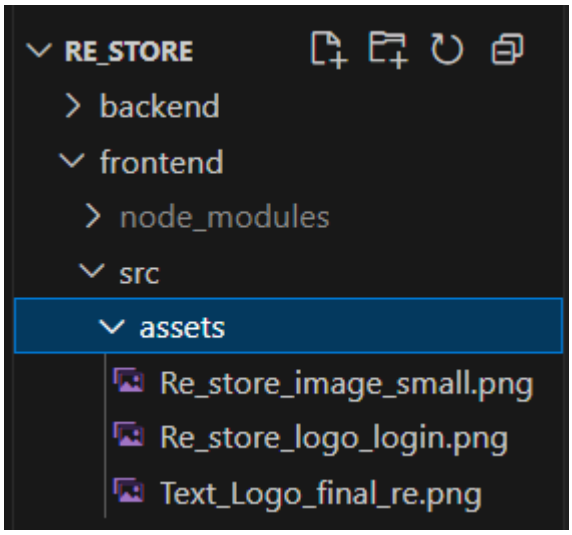
  - **`App.jsx` –** Main Application Component
    The top-level React component that renders the app's structure, including routing, layout, and context providers. It acts as the root component that encapsulates all other components.

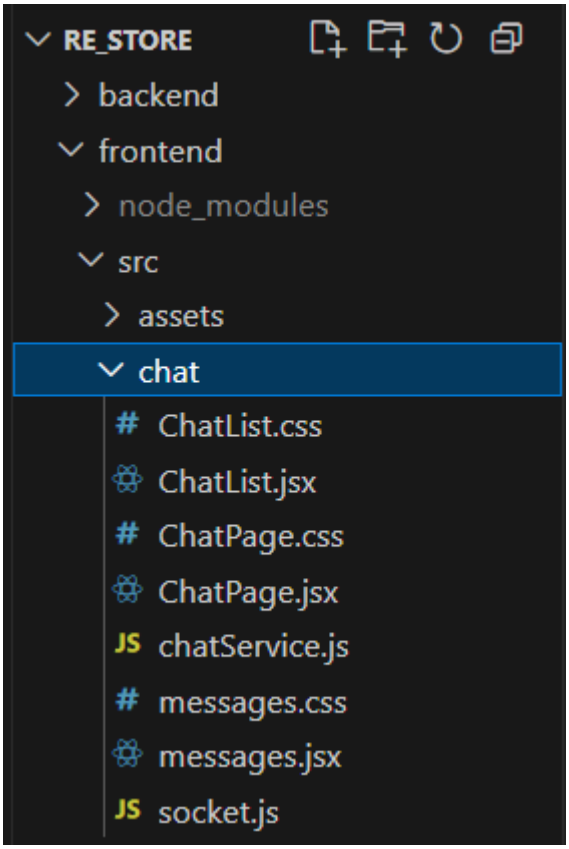
  - **`constants.js` –** Static Configuration Values
    Stores predefined constants such as API endpoints, theme settings, default values, or configuration data used throughout the app.

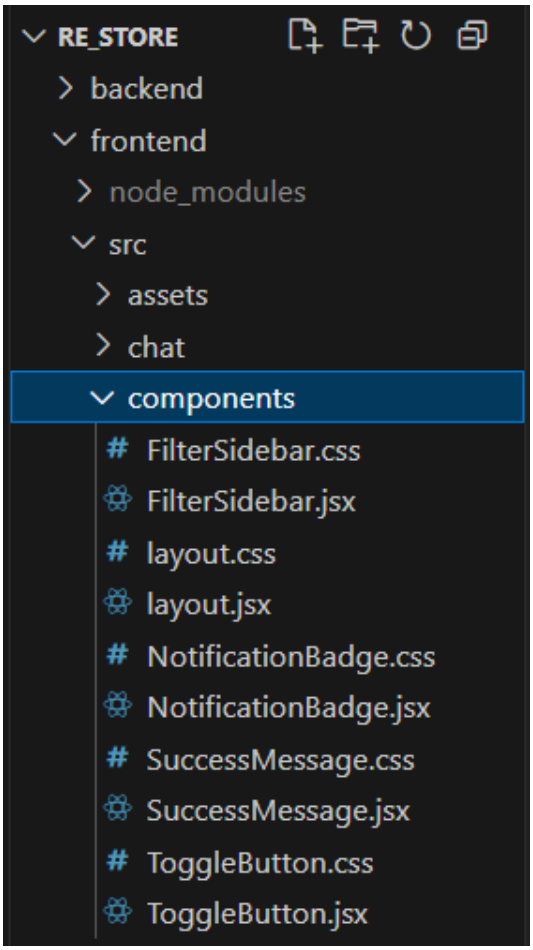
- **`fontawesome.js` –** FontAwesome Icon Imports
  Manages the import of FontAwesome icons, making them available for use across different components without needing to import them multiple times.

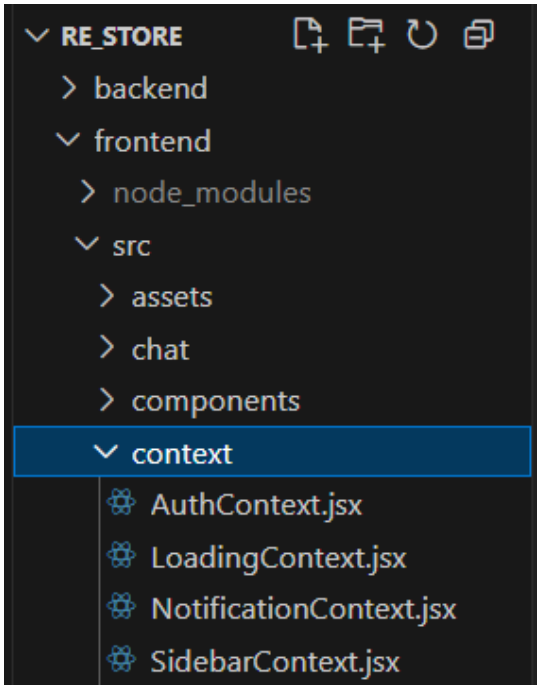
- **`main.jsx` –** React App Entry Point
  The main file responsible for rendering the React app inside `index.html`. It typically includes React's `createRoot` function and wraps the application with providers.

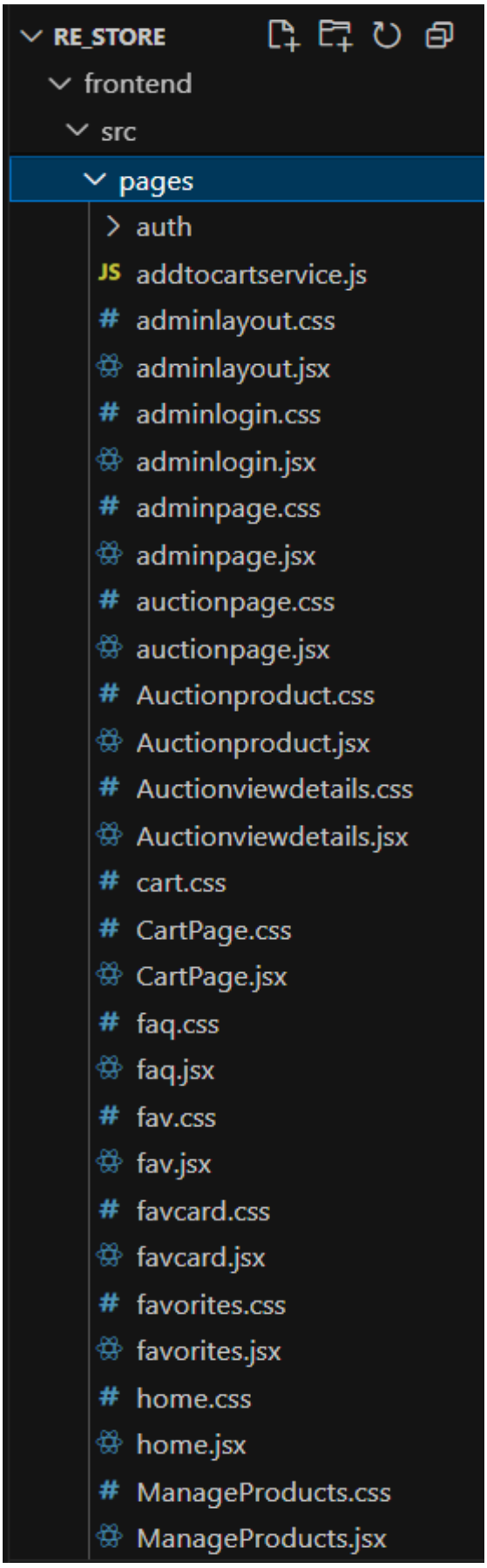


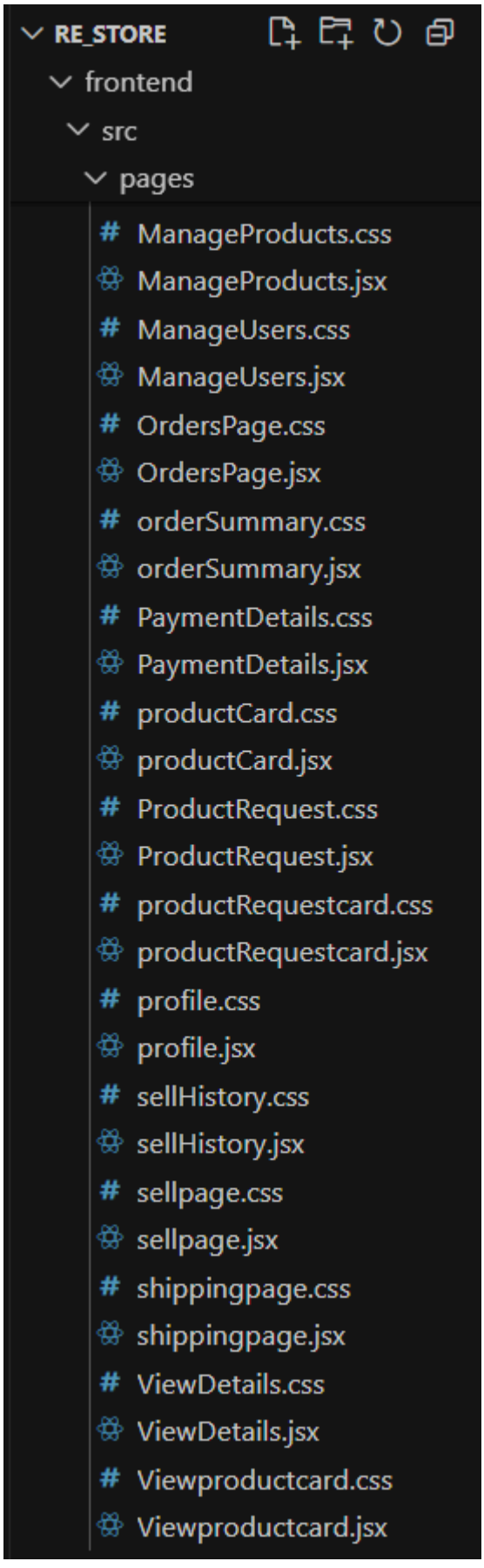

RE_STORE
backend
frontend
node_modules
src
assets
Re_store_image_small.png
Re_store_logo_login.png
Text_Logo_final_re.png

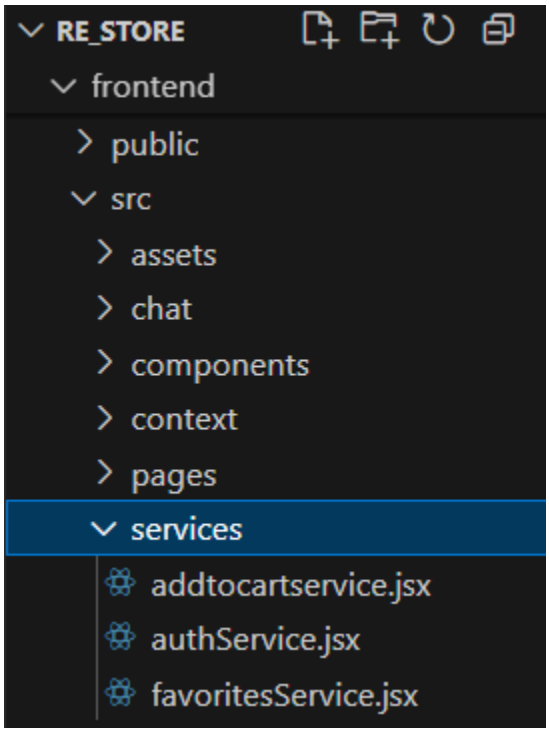

RE_STORE
backend
frontend
node_modules
src
assets
chat
ChatList.css
ChatList.jsx
ChatPage.css
ChatPage.jsx
chatService.js
messages.css
messages.jsx
socket.js


RE_STORE
backend
frontend
node_modules
src
assets
chat
components
FilterSidebar.css
FilterSidebar.jsx
layout.css
layout.jsx
NotificationBadge.css
NotificationBadge.jsx
SuccessMessage.css
SuccessMessage.jsx
ToggleButton.css
ToggleButton.jsx


RE_STORE
backend
frontend
node_modules
src
assets
chat
components
context
AuthContext.jsx
LoadingContext.jsx
NotificationContext.jsx
SidebarContext.jsx


RE_STORE
frontend
src
pages
auth
addtocartservice.js
adminlayout.css
adminlayout.jsx
adminlogin.css
adminlogin.jsx
adminpage.css
adminpage.jsx
auctionpage.css
auctionpage.jsx
Auctionproduct.css
Auctionproduct.jsx
Auctionviewdetails.css
Auctionviewdetails.jsx
cart.css
CartPage.css
CartPage.jsx
faq.css
faq.jsx
fav.css
fav.jsx
favcard.css
favcard.jsx
favorites.css
favorites.jsx
home.css
home.jsx
ManageProducts.css
ManageProducts.jsx


RE_STORE
frontend
src
pages
ManageProducts.css
ManageProducts.jsx
ManageUsers.css
ManageUsers.jsx
OrdersPage.css
OrdersPage.jsx
orderSummary.css
orderSummary.jsx
PaymentDetails.css
PaymentDetails.jsx
productCard.css
productCard.jsx
ProductRequest.css
ProductRequest.jsx
productRequestcard.css
productRequestcard.jsx
profile.css
profile.jsx
sellHistory.css
sellHistory.jsx
sellpage.css
sellpage.jsx
shippingpage.css
shippingpage.jsx
ViewDetails.css
ViewDetails.jsx
Viewproductcard.css
Viewproductcard.jsx



Backend Folder Structure Overview:

- **controllers/–** Business Logic Layer
  Handles the core logic for different features such as authentication, product management, order processing, and user operations. It interacts with models and sends appropriate responses to API requests.

- **models/–** Database Schemas
  Defines the structure of database collections using MongoDB. It ensures data consistency and manages relationships between entities like users, products, and orders.

- **public/img/–** Image Storage
  Stores uploaded images, categorized into sub-directories:

  - products/– Contains images related to products.

  - users/– Stores user profile pictures.

- **routes/ –** API Endpoints
  Defines the backend's API routes, linking HTTP requests to controllers. Each route file corresponds to a specific feature like authentication, products, users, or orders, ensuring modularity.

- **`scripts/` –** Utility Scripts
  Includes standalone scripts for administrative tasks such as seeding the database, creating an admin user, or performing batch operations, which are executed manually when needed.

- **`utils/` –** Helper Functions
  Contains reusable utility functions for common operations like email handling, token generation, API response formatting, and authentication checks to improve code maintainability.

- **`views/` –** Pug Templates
  Holds server-rendered templates used for email formatting, verification pages, and other dynamic HTML-based responses required for user notifications.

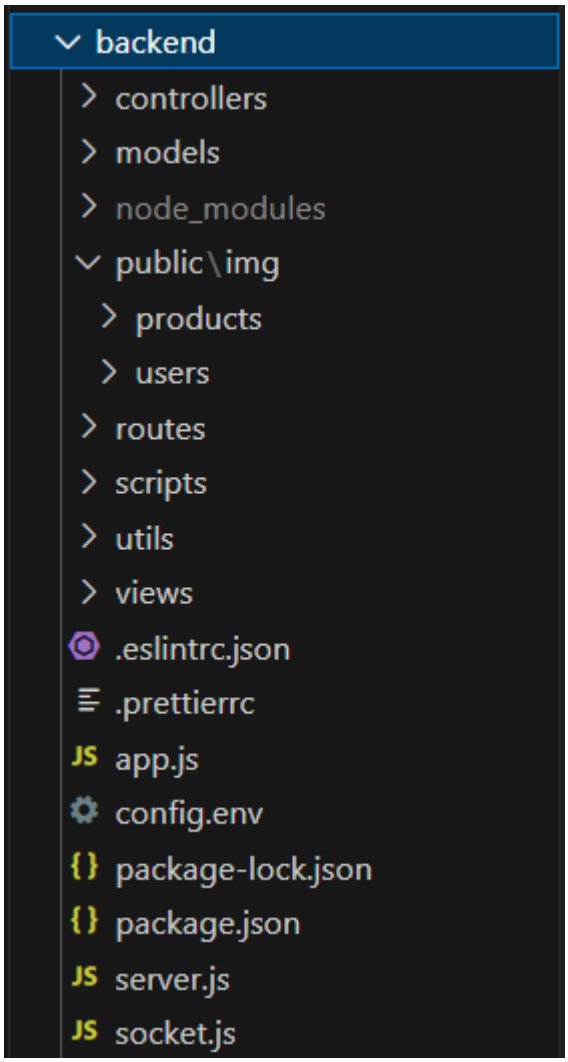
- **Core Backend Files**

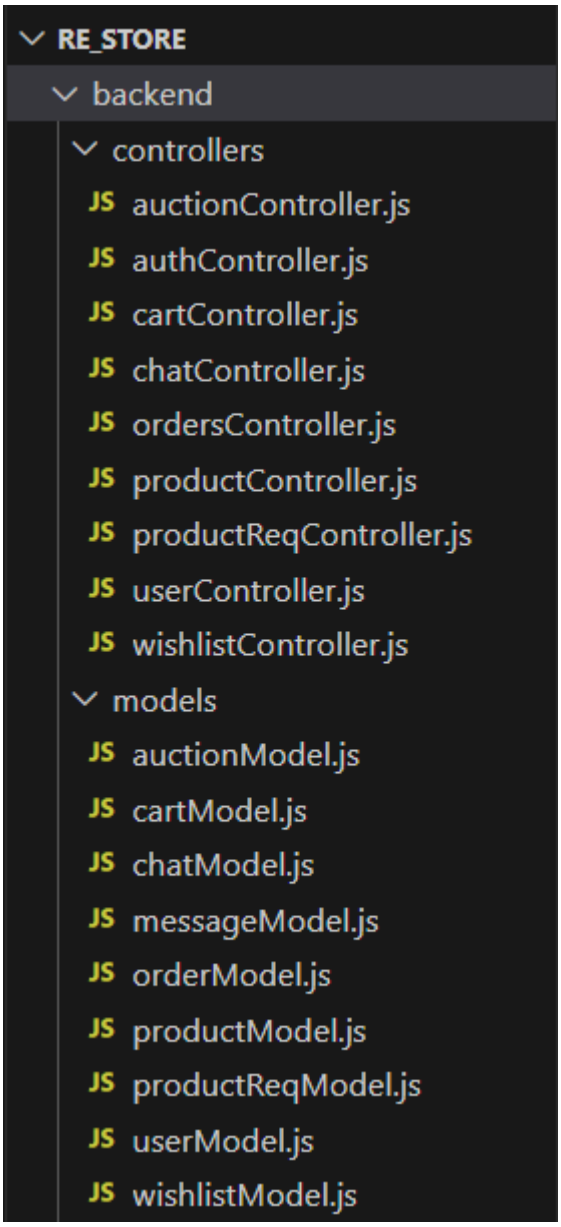
  - **`app.js`** – Initializes the Express application, sets up middleware, and connects to the database.

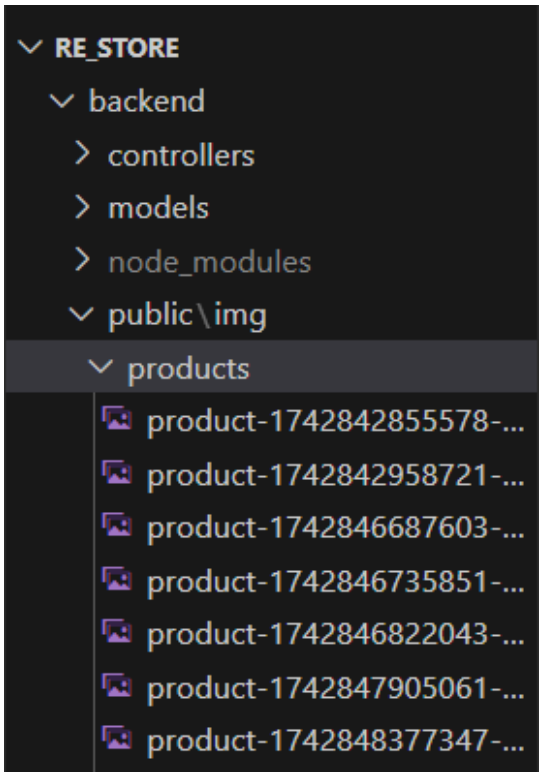
  - **`config.env`** – Stores environment variables like database credentials, API keys, and secret tokens.

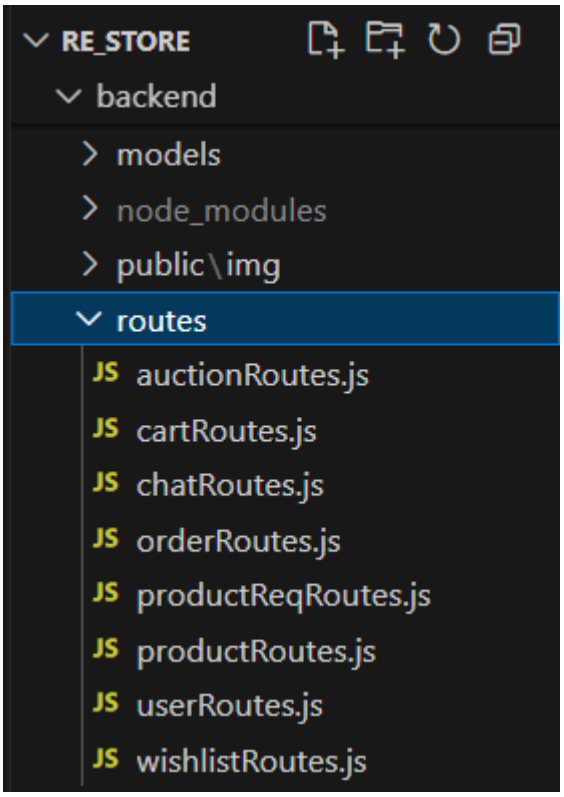
  - **`server.js`** – The main entry point that starts the Express server and listens for requests.

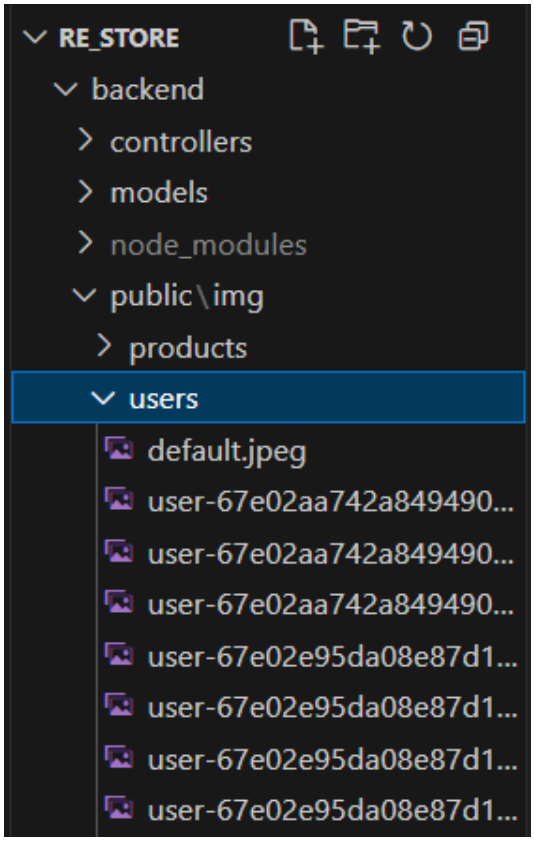
  - **`socket.js`** – Manages WebSocket-based real-time communication for features like live bidding, chat, or notifications.

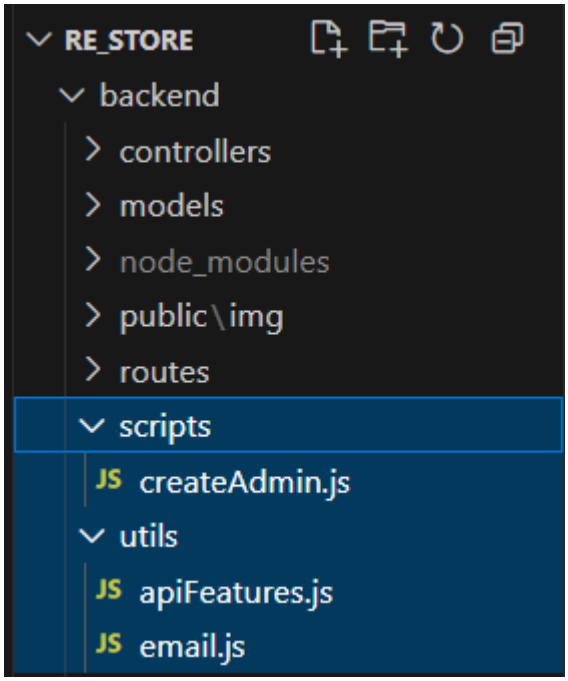

backend
controllers
models
node_modules
public\img
products
users
routes
scripts
utils
views
.eslintrc.json
.prettierrc
app.js
config.env
package-lock.json
package.json
server.js
socket.js

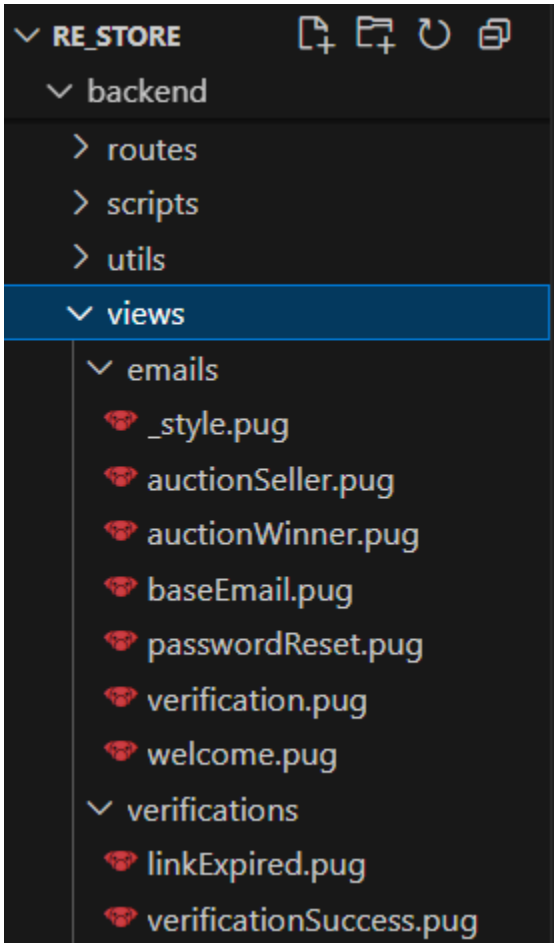

RE_STORE
backend
controllers
auctionController.js
authController.js
cartController.js
chatController.js
ordersController.js
productController.js
productReqController.js
userController.js
wishlistController.js
models
auctionModel.js
cartModel.js
chatModel.js
messageModel.js
orderModel.js
productModel.js
productReqModel.js
userModel.js
wishlistModel.js


RE_STORE
backend
controllers
models
node_modules
public\img
products
product-1742842855578-...
product-1742842958721-...
product-1742846687603-...
product-1742846735851-...
product-1742846822043-...
product-1742847905061-...
product-1742848377347-...


RE_STORE
backend
models
node_modules
public\img
routes
auctionRoutes.js
cartRoutes.js
chatRoutes.js
orderRoutes.js
productReqRoutes.js
productRoutes.js
userRoutes.js
wishlistRoutes.js


RE_STORE
backend
controllers
models
node_modules
public\img
products
users
default.jpeg
user-67e02aa742a849490...
user-67e02aa742a849490...
user-67e02aa742a849490...
user-67e02e95da08e87d1...
user-67e02e95da08e87d1...
user-67e02e95da08e87d1...
user-67e02e95da08e87d1...


RE_STORE
backend
controllers
models
node_modules
public\img
routes
scripts
createAdmin.js
utils
apiFeatures.js
email.js


RE_STORE
backend
routes
scripts
utils
views
emails
_style.pug
auctionSeller.pug
auctionWinner.pug
baseEmail.pug
passwordReset.pug
verification.pug
welcome.pug
verifications
linkExpired.pug
verificationSuccess.pug

3 Completeness

We were able to successfully implement nearly all of the features that we have mentioned in the SRS. We have listed them further in the sections below:

Login Page

The Login Page serves as a secure gateway for registered users to access the platform. Users can log in by entering their email and password.

"Forgot Password?": An option is provided for easy password recovery - if a user forgets their password, they can enter their email in the input box provided and an email will be sent to the user with a link which redirects to a reset password page where users can change their password.

New users who haven't registered yet can click on the "Sign Up" link to create an account, here users have to enter the details asked, when submitted will redirect to a verify email page, the user will receive an email for email verification which when clicked verifies the user. There is also an option to skip verification for now.

Additionally, an Admin Login option is available for administrators to access the system with special permissions.

The page is designed with a clean and intuitive interface, ensuring a smooth and user-friendly login experience.

Home Page

The Home Page of the platform provides users with a structured interface to explore, search, and interact with products listed for sale. It enables buyers to browse through available items, access product details, and manage their shopping preferences.

User can:

- Search Bar: A search bar at the top allows users to enter keywords related to product names. The system fetches relevant results based on user queries, making it easier to find specific items.

- Product Listings: The homepage displays multiple products with key details such as images, prices, and names. Each product has a "View Details" button that redirects users to a dedicated product page for more information.

● Buying Options: Users can choose between different purchasing modes, including "Buy it now," "Auctions," and "Request." This feature provides flexibility in acquiring products based on preference.

● Favourites: Each product listing includes a heart icon, allowing users to add items to their wishlist for future reference. Saved items can be accessed in the Favorites section.

● Sidebar Navigation: A sidebar menu is available on the left side, providing quick access to essential features such as:

1. Home (to browse products)
2. Messages (to chat with sellers)
3. Favorites (to view wishlist items)
4. My Orders (to track purchases)
5. Sell Items (to list products for sale)
6. Help (for frequently asked questions)
7. Logout (to sign out of the platform)

● User Account Options: In the top-right corner, users can navigate to Home, Cart, Sell History and Profile, allowing them to manage their shopping activities efficiently.

The Home Page is designed with an intuitive layout, ensuring a seamless user experience by integrating search, product discovery, and navigation features for efficient browsing and purchasing.

● Product Search & Requests: We have also successfully implemented the product search feature so that users can filter the data and if it is not available ,user can request for it by the Request button, on clicking it will be directed to the product request page.

● My Orders: We have also successfully implemented the my orders feature so that user can view their order history.

Chat Page

The Chat Page enables seamless communication between buyers and sellers, allowing them to negotiate prices, clarify doubts, and discuss product details before making a purchase. Additionally, it supports discussions related to the auction feature, where buyers can communicate with sellers about ongoing bids, ask questions, and finalize deals. This real-time interaction ensures transparency, helping users make informed decisions and enhancing the overall buying and selling experience on the platform.

Sell page

The Sell Page allows users to list items they want to sell within the platform. Users can enter the product name, upload images of the item, and choose a selling mode (Sell it now or Auctions).

They must specify the price and provide a brief description highlighting key details such as condition, specifications, and usage duration.

Additionally, the "Continue to add payment details" button guides the user through the selling process, ensuring they complete all necessary steps before listing the item. The interface is designed for a seamless experience, making it easy to upload and manage product listings.

Profile Page

The Profile Page serves as the user's personal dashboard, providing essential details and functionalities related to their account.

Features:

- View Profile: Displays user information, including name and official IITK email ID. The user’s profile picture is also shown, with an option to upload or update it.
- Edit Profile: Users can modify their profile details, ensuring their information remains up to date.

This page is designed to provide a seamless experience while allowing users to manage their personal details efficiently.

Auction Page

Our Auction Page is a real-time bidding platform. Each listed product is presented with transparent details, including the base price, current highest bid, seller information, and a live countdown timer, ensuring you're fully informed throughout the process. Actively participate by placing bids, tracking the auction's progress, and engaging with sellers via our integrated chat feature for any inquiries. This interactive experience transforms online shopping, empowering you to secure desired products at competitive prices through a fair and engaging bidding environment.

Help Page

The Help page is designed to assist users by addressing common queries and concerns about the platform. It features an FAQ section with expandable questions covering topics like registration eligibility, payment methods, price negotiations, handling defective items, and editing or deleting listings.

Future Improvements

- **Mobile Application Development:** To reach a broader audience, develop a mobile-dedicated application for Re_Store to provide mobile users an optimized user experience.We will also launch these apps with push notifications, real-time updates.

- **Sustainability & Community Growth:** To further promote sustainability, we plan to introduce a refurbished product certification, ensuring buyers get reliable, high-quality used products. Additionally, an eco-incentive program could reward sellers who frequently list used products, encouraging responsible reselling. These developments will make second-hand shopping more reliable, efficient, and environmentally friendly.

- **Expanding Beyond the Campus Community:** Currently, our platform is designed for the campus community, providing a localized environment for buying and selling used products. However, in the future, we plan to expand it to a larger audience beyond the campus, making the platform accessible in different cities and regions. This expansion will help connect more buyers and sellers, increase the variety of available products, and create a larger, more active marketplace.

- **User Trust & Safety Enhancements:** To improve security, we plan to collaborate with IITK's official management for user verification, reducing fake accounts and fraud. A user rating and review system will help buyers and sellers assess trustworthiness based on past transactions. A buyer-seller agreement system will also be introduced to ensure transparency and security in transactions.

- **AI-Powered Chatbot for User Assistance:** To enhance user support, we plan to integrate an AI-powered chatbot that will provide instant assistance for queries related to product listings, auctions, payment processes, and security concerns. This will replace the traditional FAQ section, offering real-time guidance and improving user experience.

Appendix A - Group Log

Date	Team	Summary of the Meeting	Duration	Location
10th Feb 2025	All	Divided the work among the team and decided on a timeline for the completion of implementation. Allocated frontend and backend work to each member of the group.	180 Minutes	RM Building
13th Feb 2025	All	Completed the Login and Signup frontend and backend	120 Minutes	RM Building
15th Feb 2025	All	Divided the frontend and backend files among each group member.	90 Minutes	RM Building
28th Feb 2025	All	Worked on the pages allocated to individual members.	240 Minutes	RM Building
02nd Mar 2025	All	Completed the pages allocated to individual members.	120 Minutes	RM Building
06th Mar 2025	All	Discussed the fixes in each other's pages.	120 Minutes	RM Building
08th Mar 2025	All	Decided folder structuring	75 Minutes	RM Building
18th Mar 2025	All	Updates regarding the work progress.	30 Minutes	Online
20th Mar 2025	All	Discussion regarding the changes in some functionalities of the feed development.	90 Minutes	RM Building
22nd Mar 2025	All	Clarification regarding some changes in the functionalities.	120 Minutes	RM Building
25th Mar 2025	All	Integration of frontend and backend and fixing a few bugs.	240 Minutes	RM Building
27th Mar 2025	All	Clearing bugs and fixing the problems in the system.	300 Minutes	RM Building
28th Mar 2025	All	Implemented the last adjustments to the documentation.	240 Minutes	RM Building