Implementation Document

for

Billing 360

Version 1.00

Prepared by

Group 7: **Group Name: TEAM ARJUNA**

Name	Roll No.	Email
Abhishek Khandelwal	220040	abhishekkh22@iitk.ac.in
Ansh Agarwal	220165	ansha22@iitk.ac.in
Nipun Nohria	220717	nipun22@iitk.ac.in
Pallav Goyal	220747	pallavg22@iitk.ac.in
Poojal Katiyar	220770	poojalk22@iitk.ac.in
Venkatesh Akula	220109	akulav22@iitk.ac.in
Dhruv Gupta	220361	dhruvgupta22@iitk.ac.in
Pragati Agrawal	220779	apragati22@iitk.ac.in
Saagar K V	220927	saagar22@iitk.ac.in
Kundan Kumar	220568	kundank22@iitk.ac.in

Course : CS253

Mentor TA : Somesh

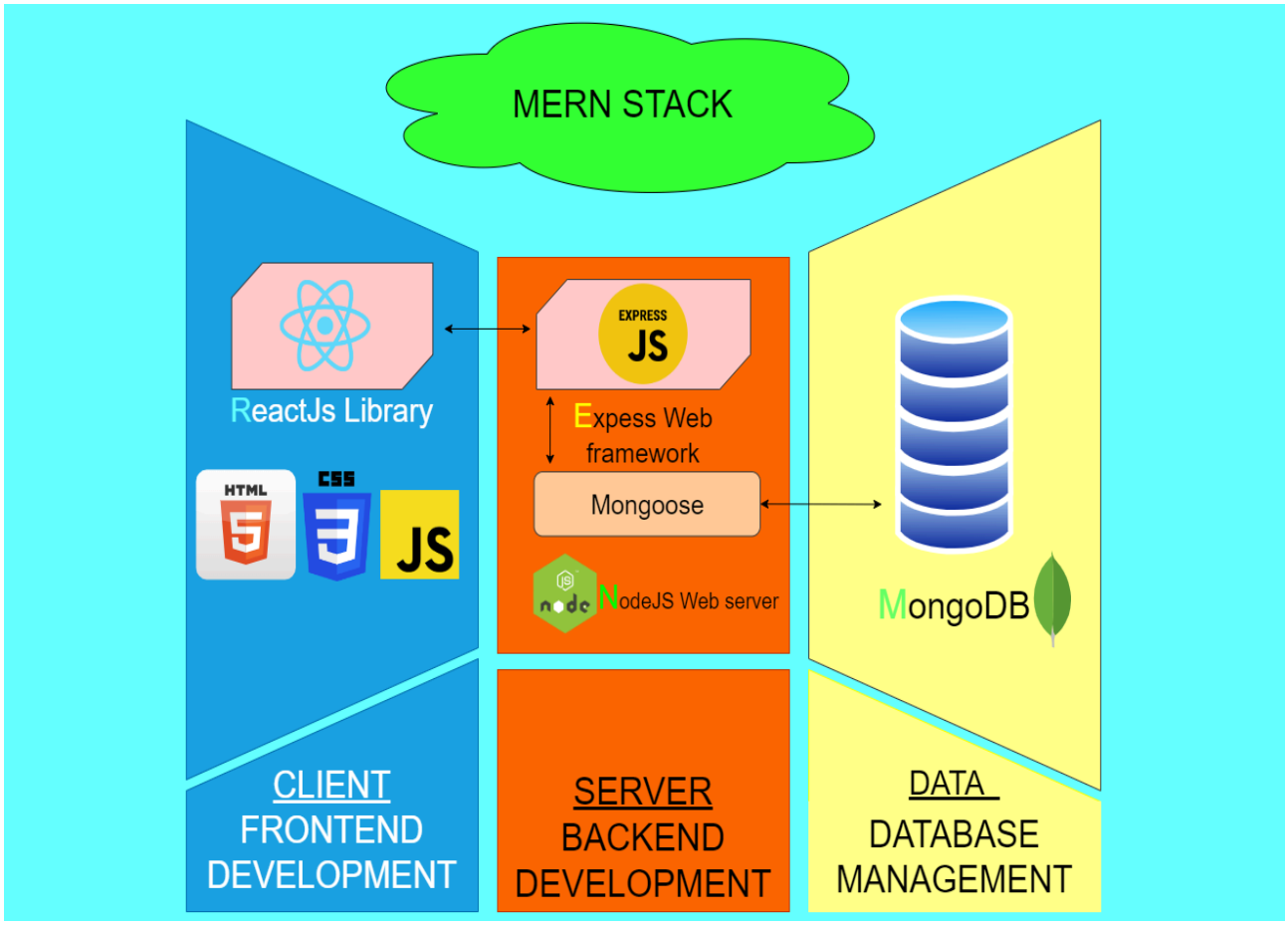
Date : 18/03/2024

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.00	Abhishek Khandelwal Dhruv Gupta Pragati Agrawal Poojal Katiyar Ansh Agarwal Nipun Nohria Venkatesh Akula Saagar K V Kundan Kumar Pallav Goyal	The First version of Implementation Document	18/03/2024

1 Implementation Details



Billing360 is a comprehensive web-based billing software 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 billing software.

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:

Billing 360 has been developed primarily using **Javascript** programming language. Billing 360 follows the **model-view-controller** architectural pattern.

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

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 billing information managed by Billing 360.

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 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 by OTP we have used the **nodemailer library** to send emails.

In conclusion, Billing 360 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 Billing 360 to deliver a robust, scalable, and user-centric Software.

2 Codebase

Github Repository- https://github.com/Billing-360/Billing-360-Dev

Hosted Website- https://billing-360-dev-1.onrender.com/

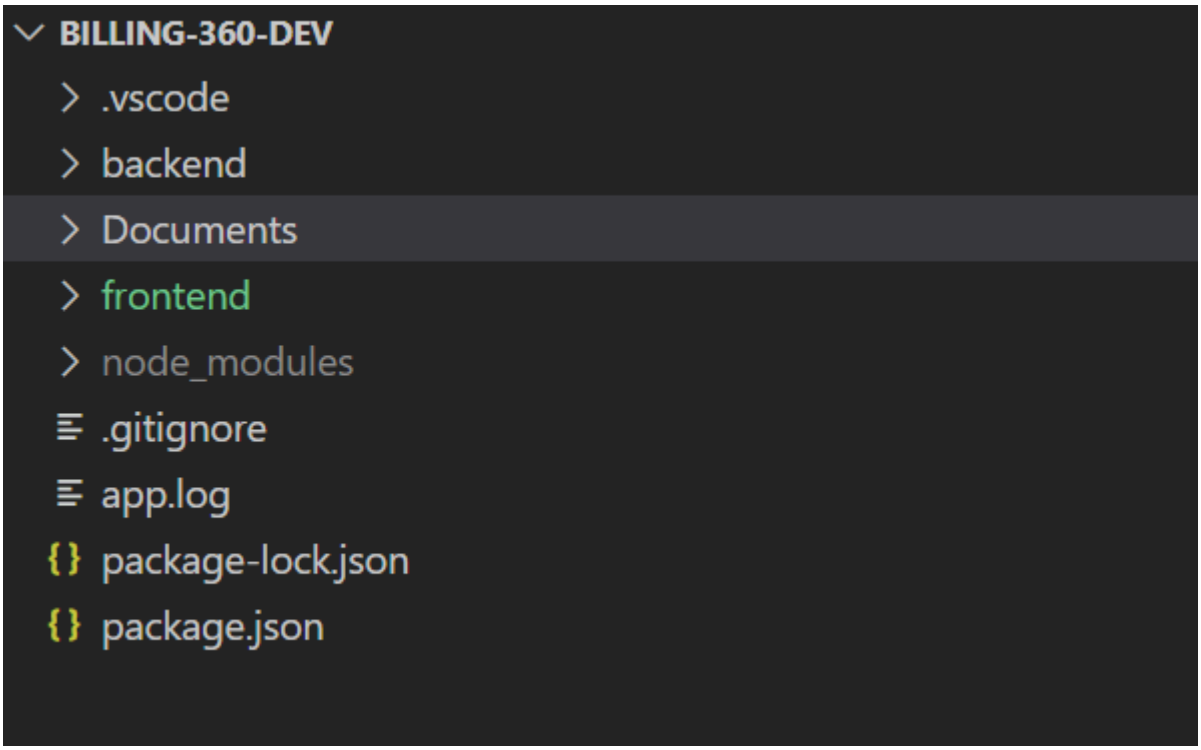
Code Structure:

The above link takes you to the GitHub repository of the Billing 360, which contains all the source code for our web application.

The project repository is mainly composed of 3 main parts:

- Frontend
- Backend
- Documents

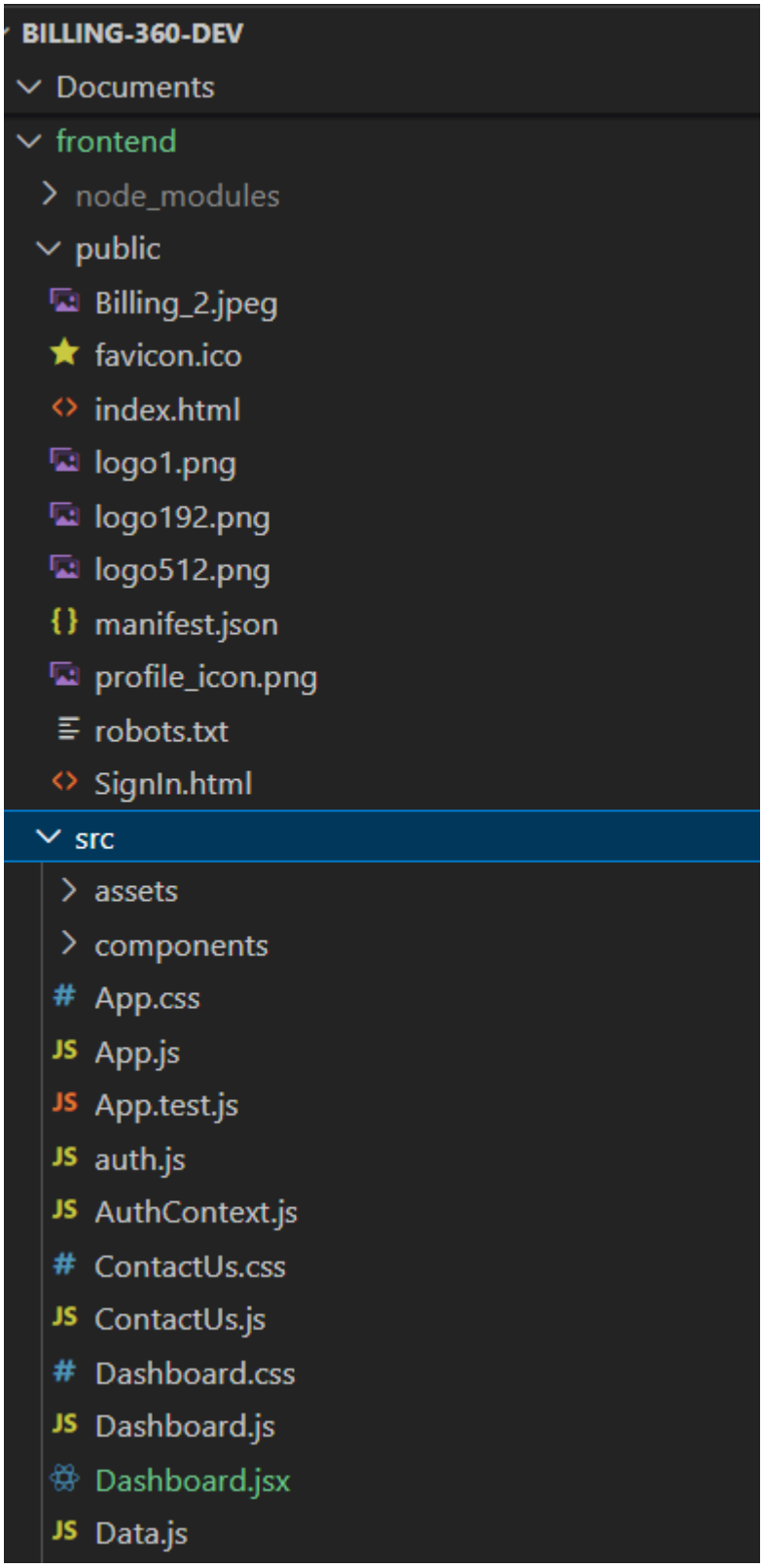
The overall image of the Billing 360 repository when cloned in VSCode is as shown:

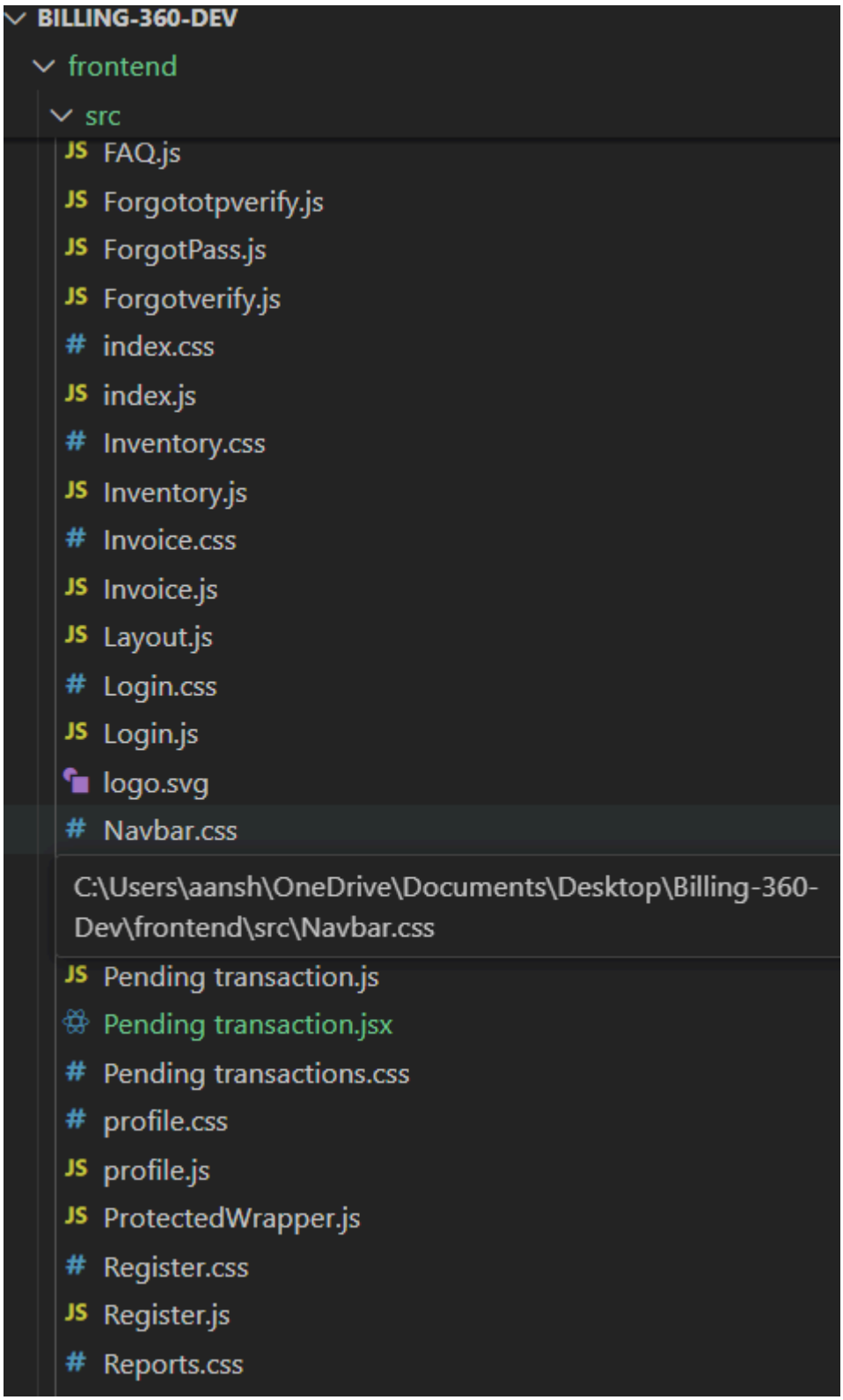


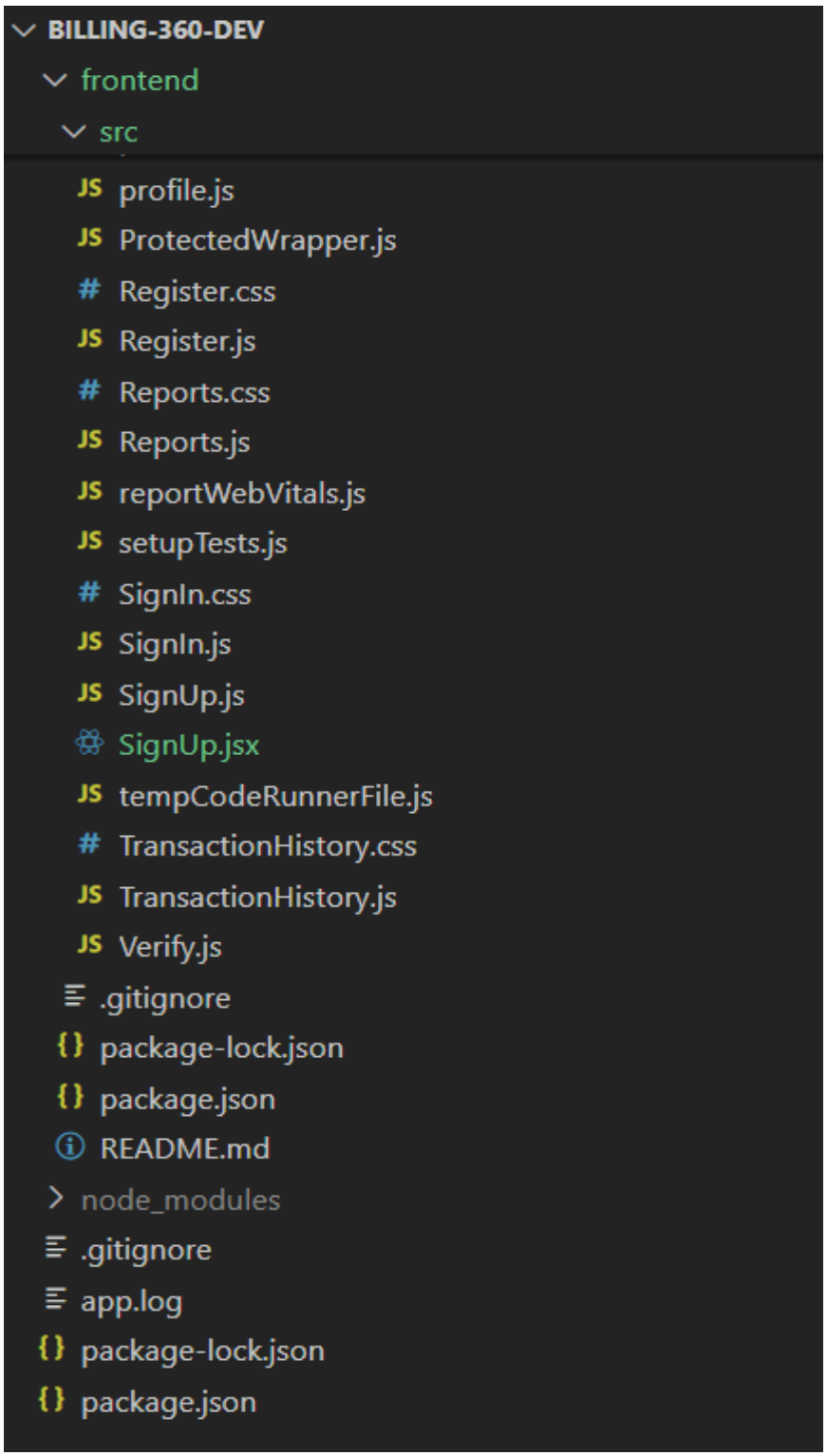
When you open the Frontend, the subsequent images are as shown: It has mainly the following parts:

- Public: It contains the images/logos that are used in the project and the index.HTML which is the basic HTML that is used in all the files of the project.

- Src: It contains the component section that has all the frontend React components(.jsx, .js, .css) used in the project.



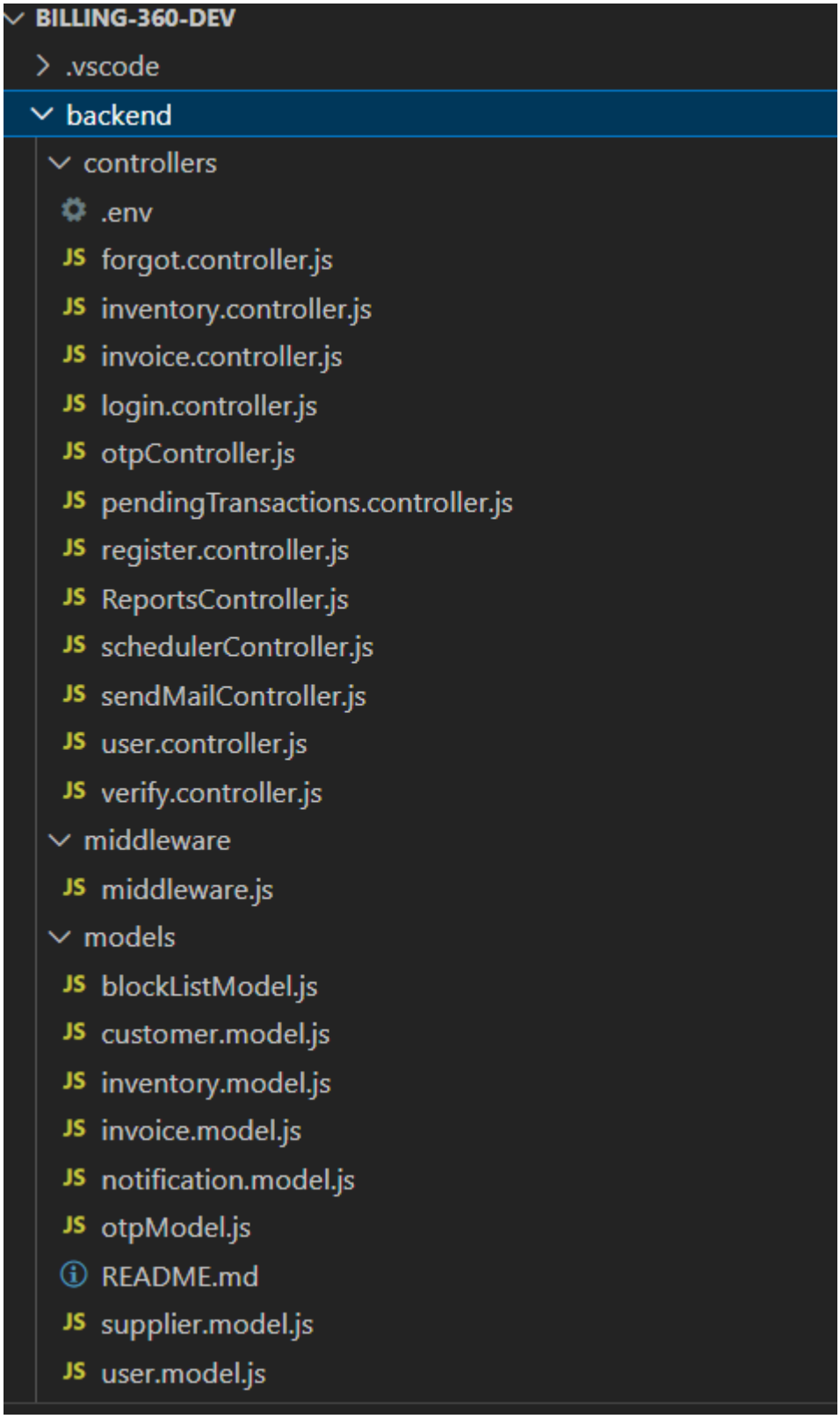

BILLING-360-DEV
frontend
src
FAQ.js
Forgototpverify.js
ForgotPass.js
Forgotverify.js
index.css
index.js
Inventory.css
Inventory.js
Invoice.css
Invoice.js
Layout.js
Login.css
Login.js
logo.svg
Navbar.css
C:\Users\aansh\OneDrive\Documents\Desktop\Billing-360-Dev\frontend\src\Navbar.css
Pending transaction.js
Pending transaction.jsx
Pending transactions.css
profile.css
profile.js
ProtectedWrapper.js
Register.css
Register.js
Reports.css

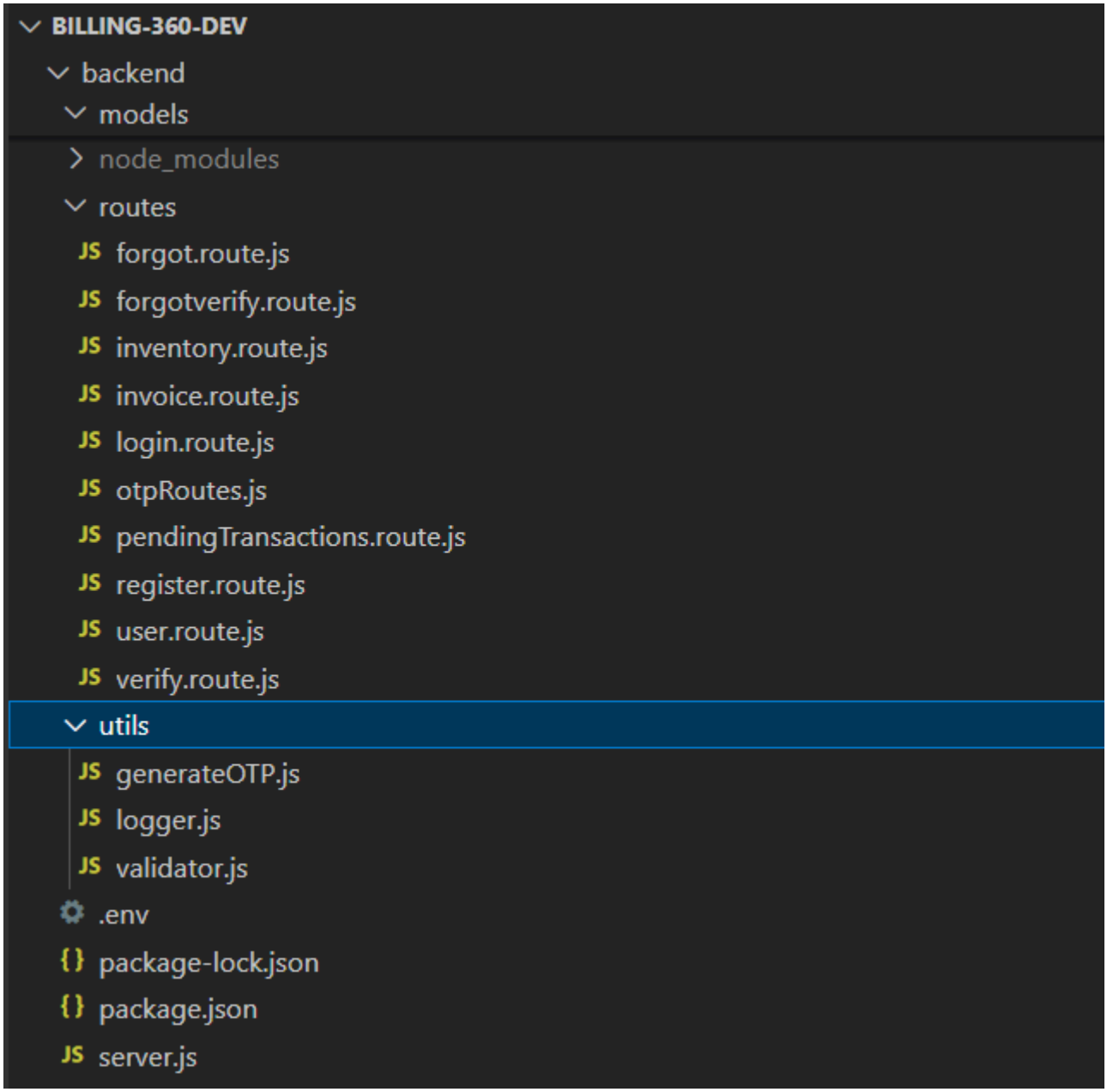


When you open the Backend, the subsequent images are as shown: It has mainly the following parts:

- Controller
- Routes

- Models


BILLING-360-DEV
.vscode
backend
controllers
.env
forgot.controller.js
inventory.controller.js
invoice.controller.js
login.controller.js
otpController.js
pendingTransactions.controller.js
register.controller.js
ReportsController.js
schedulerController.js
sendMailController.js
user.controller.js
verify.controller.js
middleware
middleware.js
models
blockListModel.js
customer.model.js
inventory.model.js
invoice.model.js
notification.model.js
otpModel.js
README.md
supplier.model.js
user.model.js



We have a used **Model-View-Controller architecture** for our software. This architecture is well suited for the software since there are multiple ways to interact with the same data presented in different views. Also, the data has to be changed independent of the view.

The code structure reflects the architectural pattern used. The three components constituting the software are:

1. **Model**: Consists of MongoDB schemas for objects of various classes. The classes have been described in more detail in the Design document. The code for this

component('models' folder) is located in the 'backend' directory. Each file within the folder contains the schema for a particular class.

2. **View:** Controls what the user sees and how he/she can interact with the software. The code for this component rests in the 'src' folder within the 'frontend' directory. Each Javascript file within the folder contains code that controls the view presented to the user. A corresponding CSS file, which controls the design of the webpage, has been included in each Javascript file. In addition, we have the components folder (which has files containing react components that handle user interaction) and the assets folder (which has other necessary files (like images)).
3. **Controller**: Consists of business logic that makes the software perform the desired tasks. The code for this component ('controllers' folder) is located in the 'backend' directory. Each file within the folder contains react components dealing with a specific functionality.

The code is organised in accordance with the functionalities, which in turn correspond to webpages rendered before the user:

- Authentication : This is the sign-in/sign-up page that allows a user to create a new account and login to access his data and use the software's functionalities.
- Dashboard : The home page of the user that is rendered when the user logs in.
- Invoice : On this page, the user can create a new invoice and generate a pdf of the invoice.
- Inventory : Here, the user can manage his inventory - adding, editing and deleting items or batches of an item
- Pending Transactions: The user can manage the record of customers who owe him money as well as of suppliers to who the user owes money.
- Reports: This page provides useful insights to the user by analyzing temporal data and rendering meaningful graphs.
- FAQ and Contact Us: These pages provide help to the user in using the software and in reaching out to us.

Other important files:

- index.js and App.js : They are located in the 'src' folder within the 'frontend' directory. 'index.js' is the main file that drives the View component. It renders the App component from 'App.js'. 'App.js' maintains the authenticated user and imports most other files that constitute the frontend. It is responsible for routing between various web pages.
- server.js : It is located in the 'backend' directory. It forms the heart of the software as it connects the frontend with the backend logic. It handles requests from the web by using a corresponding router which further directs to the controller logic. Controller then interacts with the database (using the models) and performs the desired task. The response is then sent back to the fetch point.
- routes : This is a folder in the 'backend' directory. 'server.js' directs to a specific file in this folder depending on the request. Further details about the request are used by this file to reach out to the appropriate controller.

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/Register:

- We have completed the login features as written in the SRS.
- We have also successfully implemented the forgot password feature in which the user can generate his new password by way of OTP generation in the email he has registered in case he forgets his previous password.
- For the new user, we have given the SignUp feature in which the user can fill in his details and register in the software, as we have mentioned in the SRS.

Dashboard:

- We have implemented the requirement stating that the user must be able to see 'Today's sale, Today's profit, Today's customer, and Yesterday's sale' which will be updated on a daily basis.
- We have implemented the user navigation feature in which users can navigate to any page like inventory, invoice, and pending transactions.
- The user is provided view and edit access to his profile. The user has other options like signing out and profile in the upper right corner. All these have been mentioned in the SRS. We were able to implement these requirements successfully.
- The user can see graphs that show how sales and profit have varied over the past 7 days.

Invoice:

- The main feature that our software was based on, was the billing or invoice section. We have successfully implemented the feature in which the user can enter the customer details like name, email and phone number. The invoice id is automatically updated using the total number of invoices in the database.
- The user can search and select an item from the inventory by entering the item name. There are options to edit and delete items. The details like rate, GST and amount will be calculated automatically from the inventory. There is also a box to enter discount after which the total amount of the bill would be shown. Additionally, some notes can be added to the invoice.
- We have also implemented a feature where the user can add the amount as credit if the customer wants to pay later or set it as paid if the customer wants to pay it on the spot. For credit customers, data is updated in the customer's profile in the database. If a new customer comes, a new profile is created in the database. These changes are reflected in the Pending Transactions page
- On clicking the Preview Bill, he can generate the pdf of the transaction and preview or download it. The generate bill button downloads the invoice with a unique name "invoice_id.pdf", updates the inventory, and the invoice is then added to the Transaction History page.
  NOTE: We have also implemented a feature to send the invoice to the customer email directly. It is pushed on github and tested on localhost. However, this feature is not yet deployed on our hosted website.

Inventory:

- We have implemented the feature for searching for an item in the inventory as well as adding a new item to the inventory. We have also made batches for each item where the items will be reduced from the batch with closest expiry when making the invoice.
- By clicking on titles like item Name,Item ID, etc users will be able to sort the items in any order.
- The quantity of the item is updated when the batch quantity is entered and total sum is hereby updated.
- The user can edit/add/delete/update an item/batch.
- We have also implemented an additional feature. The items shown in red indicate that the items are expired, items in yellow are those that are going to expire in 3 days and the items in green are the rest of the items which don't expire in the next 3 days.In this way we are making use of the expiry date to alert the user.

Pending Transactions:

- We have successfully implemented the credit feature in which the user can view a list of customers with pending payments, including their contact details. The user can also search for the customer's name in the credit section.
- Total credit is displayed at the top of the page when the credit tab is active. The user sees the total credit that his customers owe him.
- As mentioned in the SRS, we have also implemented the debit feature in which the user can maintain a record of suppliers along with their contact details and the money he owes them.
- Total debit is also displayed on top of the page when the debit tab is active.
- We have also implemented the feature of notifying customers via email which contains the pdf of invoice as attachment, whenever we update their credit dues but right now, we have done it on local host and we are soon going to deploy it.

Contact Us:

- We have implemented the "Contact Us" option using which the user will be able to contact the developers to resolve an issue or to give a suggestion that will probably make the software more user-friendly.

FAQs(Frequently Asked Questions):

- As mentioned in the SRS we have implemented the FAQs section in which the user will be able to view common problems that he might face while running the software, along with their solutions .

Reports:

- In the report section, the user can select the start date and the end date indicating the duration pertaining to which he wants the data to be analyzed.
- The user is shown the variation of sales, profit, and number of bills with days in the given duration using a line graph. We have also implemented a pie chart of the sales distribution which shows the paid sales vs credit sales.
- The user can view the top 5 customers with highest sales.
- Thus, the requirement pertaining to the rendering of insights to the user has been successfully implemented.

Future Improvements:

<u>GENERAL NOTE:</u> Our code is working perfectly on our local host. But on deployment, some features are lagging like generating invoice pdf which we are going to improve in the testing phase.

- The requirement mentioned in the SRS regarding the feature of scan using barcode has not been implemented due to lack of time and hardware resources. This feature will be implemented in the next version and the user will be able to scan the barcode of the items using a barcode scanner.
- The possibility of using other languages for users belonging to different regional backgrounds is expected to be implemented.
- At present, we have not implemented a regular feedback system in which the user or the retailer can give ratings or feedback to the app developers regularly, but it is expected to be implemented in future versions.
- We will also take in the logo of the firm of our user which can be printed on his invoice.
- We will also have customised reports according to the sales and inventory of our user.
- We will have an option of partial credit for a customer in the invoice.

Appendix A - Group Log

The frontend part (View) was first written, followed by the backend part (Model and Controller). In each stage, tasks were divided among team members, and pair programming was used to carry out tasks. Thus, all team members are familiar with all stages of the development process as well as all aspects of the software.

Prior to 1 March 2024, the frontend part was divided among 4 groups of 2-3 team members. All team members were supposed to complete their tasks by 1 March 2024.

DATE	TIME	DURATION	AGENDA
1 March 2024	9 pm - 1 am	4hrs	● Review the frontend work of all the members of the team and discuss the problems that they had faced in completing their work and resolve them. ● Installation of the MongoDB and linking it with VSCode for all the members.
2 March 2024	9:30 pm - 11:00 pm	1.5 hrs	● Divided segments of the software among the groups of 2-3 persons where each group was assigned a segment to gather information and explain how it can be implemented.
3 March 2024	9 am - 11 am	2hrs	● Discuss the previous work assigned and divide the further work among groups of 2-3-3-2 depending upon the intensity of the workload.
5 March 2024	9 am - 12 noon	3hrs	● Discuss the work assigned and decide how to do the rest of the work on GitHub so that everyone can work in a collaborative manner. ● At this time, we finally divided the team into groups of size 2-2-2-2-2 for the implementation part and these groups remained intact till the implementation was completed. ● Each pair was assigned the complete responsibility of a particular webpage.

6 March 2024 -10 March 2024	- - -	- -	● During these days, everyone did their work in pairs as assigned and if anyone faced any error, it was solved on the spot by the other members. ● We also met daily for 1-2 hours to report the work each person and each pair had done that day.
13 March 2024	10 pm - 12 am	2 hrs	● Major tasks on each webpage were done and reported. ● The task of writing this document was assigned to a pair and other pairs were asked to implement the optional features that had been mentioned in SRS. ● Also, there were a lot of errors we were facing while running the application by integrating everyone's work. Hence, each group was also asked to debug and to do the best they could to remove errors and make the software run smoothly.
18 March 2024	6 pm -11 pm	5 hrs	● Everything was now working fine. ● The software was deployed. ● This document was reviewed and finalized.

Here, only the meetings of the group members have been mentioned. Each pair has continuously worked on their assigned task. The actual time and effort put in by the team is much more than what is reflected in the above table.