

---

# Implementation Document

for

## DBMS

Version 1.2

Prepared by

### Group #: 5

Vishal Kumar	221201
Yash Pratap Singh	221223
Shriya Garg	221038
Aditya Gupta	220065
Abhishek Kumar	220041
Nandini Akolkar	220692
Rishikesh Sahil	220892
Udbhav Singh Sikarwar	221150
Anshu Yadav	220171
Kushagra Singh	220572

### Group Name: Dev Dynamos

[vishalkmr22@iitk.ac.in](mailto:vishalkmr22@iitk.ac.in)

[yashps22@iitk.ac.in](mailto:yashps22@iitk.ac.in)

[shriyag22@iitk.ac.in](mailto:shriyag22@iitk.ac.in)

[adityagu22@iitk.ac.in](mailto:adityagu22@iitk.ac.in)

[kumara22@iitk.ac.in](mailto:kumara22@iitk.ac.in)

[anandini2222@iitk.ac.in](mailto:anandini2222@iitk.ac.in)

[rishikeshs22@iitk.ac.in](mailto:rishikeshs22@iitk.ac.in)

[udbhavss22@iitk.ac.in](mailto:udbhavss22@iitk.ac.in)

[anshuyadav22@iitk.ac.in](mailto:anshuyadav22@iitk.ac.in)

[kushagras22@iitk.ac.in](mailto:kushagras22@iitk.ac.in)

Course: CS253

Mentor TA: Bharat

Date: 18.3.24

## INDEX

CONTENTS	ii
REVISIONS	ii
<b>1 IMPLEMENTATION DETAILS</b>	<b>1</b>
<b>2 CODEBASE</b>	<b>9</b>
<b>3 COMPLETENESS</b>	<b>11</b>
<b>APPENDIX A - GROUP LOG</b>	<b>13</b>

--

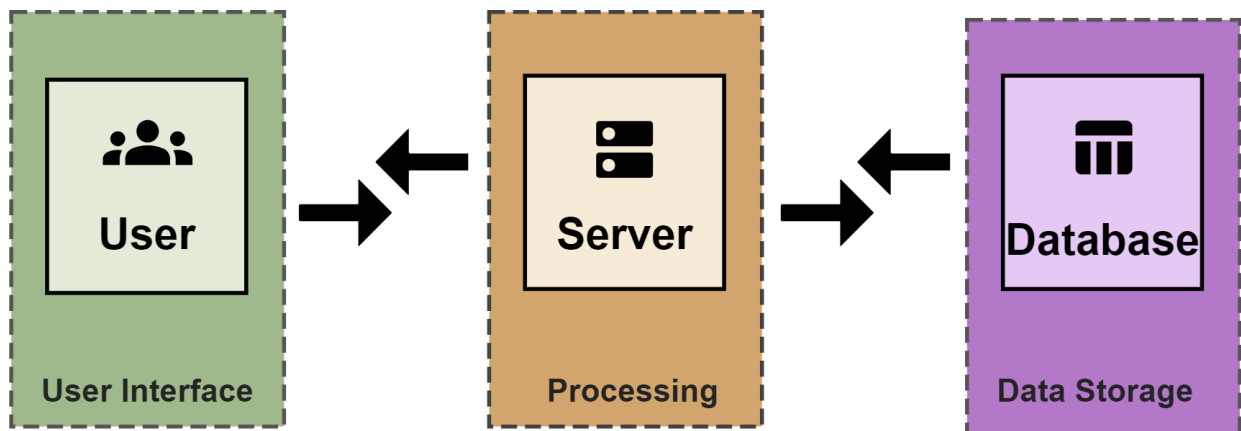
Version	Primary Author(s)	Description of Version	Date Completed
1.1	<b>Group 5 - DevDynamos</b>	-A basic version of the document was drafted. -Implementation and Completeness was done.	10/03/2024
1.2	<b>Group 5 - DevDynamos</b>	-Code base and Group log was completed and the document was finalized	18/03/2024

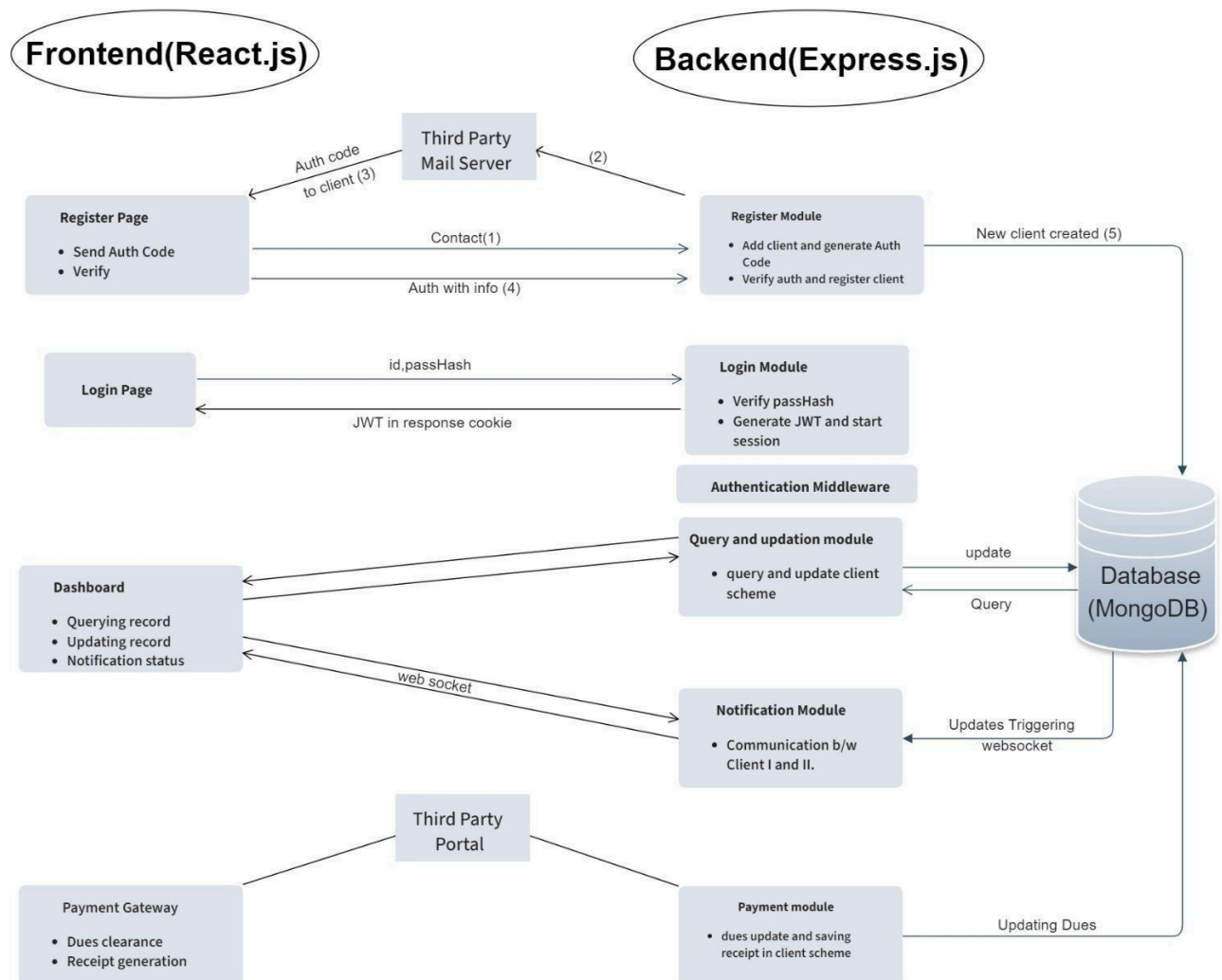
## 1 Implementation Details

“DBMS” implements a three-tier architecture, a popular pattern for user-facing applications.

The tiers that comprise this architecture includes:

- **User Interface component:** This encompasses the elements users directly engage with. In our context, this refers to the React-based web application operating within a browser.
- **Processing Component:** This segment encompasses the code responsible for interpreting user interactions and executing the application's functionalities, coded in JavaScript, which govern the behavior at the user interface tier.
- **Data Storage Component:** This comprises databases housing the pertinent data for the application.





## 1.1 User Interface Component:

The user interface layer is accessible to users through a web browser and comprises components facilitating interaction with the system. It's constructed using three fundamental technologies: **HTML, CSS, and JavaScript**.

HTML defines the website's content, while CSS dictates its appearance. JavaScript and its frameworks add interactivity and responsiveness to user actions.

**React**, a JavaScript framework, was chosen to enhance the dynamism of the page. ReactJS is favored over other frameworks due to the following reasons:

- 
- **Speed:** React empowers developers to employ specific parts of their application, thus accelerating the development process.
  - **Flexibility:** React code is easier to maintain and more flexible due to its modular structure compared to other frontend frameworks.
  - **Performance:** The framework's core features, such as a virtual DOM program and server-side rendering, enable complex applications to run exceptionally fast.
  - **Usability:** Deploying React is straightforward, enhancing the usability of the framework.
  - **Reusable Components:** It saves time by eliminating the need to rewrite code for similar features, thanks to its reusable component structure.

## **1.2 Processing Component:**

### **The advantages of Javascript programming language :**

1. Platform independence (Write once, run anywhere)
2. Object-oriented programming (OOPs) paradigm
3. Rich standard libraries (APIs)
4. Strong community support
5. Automatic memory management (Garbage collection)
6. Multi-threading support for concurrent execution
7. Built-in security features (Bytecode verification, sandboxing)
8. Improved performance with Just-In-Time compilation
9. Versatility for various application domains

---

## 1.3 Data Storage Component:

- In our software we used MongoDB for data storage due to following reasons:
  - **Flexibility:** MongoDB offers a flexible Schema design, whereas MySQL/PostgreSQL require a predefined schema making it easier to accommodate changing data models.
  - **Document-Oriented Data Model:** MongoDB stores data in JSON-like documents, which helps to represent complex relationships and nested data structures. Relational databases store data in rows and columns which may increase complexity.
  - **Speed:** MongoDB is faster than MySQL due to its better structural management.

## 1.4 Development and version control environment:

We employed **Git** as our chosen version control system, a widely adopted tool for managing software development and versioning tasks.

It enables tracking of file modifications, ensuring a detailed history of changes and the ability to revert to specific versions when necessary. Additionally, Git facilitates seamless collaboration, enabling multiple contributors to merge their changes into a unified source. To manage our repositories we utilized **GitHub**, a web-based platform for hosting Git repositories.

## 1.5 API Details:

The following are the links where different api rcalls are sent to the backend to fetch/update date.

API NAME (from backend)	WHAT IT DOES
<b>Session Routes</b>	
/session/admin/login	This authenticates admin users based on their credentials provided in the request body, generating a JWT token upon successful

	authentication, setting the token as a cookie, and providing appropriate responses for success and failure cases.
/session/student/login	This is for authentication of student, similar to admin
/session/washerman/login	This is for authentication of washerman, similar to admin
/session/logout	A route for logging out a user's session, typically implemented to clear any session data or authentication tokens associated with the user.
<b>Register Routes</b>	
/sendAuthCode	Utilizes a Mongoose model named AuthCode to generate and store authentication codes according to the specified schema, enabling functionalities such as two-factor authentication or email/phone verification within an application
/student/register	This is to register the student using the credential asked
/student/resetPassword	This is to reset the password of the student.
<b>Admin Routes</b>	
/admin/washerman/register	Through this admin can register the washerman
/admin/addHallData	This is to add halls and their wings in the hall database
<b>Student Routes</b>	
/student/fetchRecords	Through this students can view the data of a particular date events if any event is done.
/student/requestWash	This is to request the cloths washing confirmation from the washerman
/student/fetchDates	Through this student can get the data of the dates when events done like cloths given to washerman
/student/payment/fetchReceipt	This is to get the payment receipt after the payment done
/student/payment/fetchDates	This is to get the payment summary for a particular date



/student/payment/clearDue	This facilitates the clearance of due payments for students. Upon invocation, it processes payment verification, updates student payment records, and transfers due amounts to the assigned washerman, ensuring accurate payment reconciliation and providing feedback on the success or failure of the payment clearance process.
/student/fetchUpcoming	This updates the upcoming date of washerman for students.
<b>Washerman Routes</b>	
/washerman/wing/fetchRecord	Through this washer man can view the record of the halls and the wings assigned to the washerman
/washerman/upcomingDate	Through this washerman can change the tentative dates of coming ,
/washerman/wing/addEvents	This add events to students of that wing for that washerman.
/washerman/wing/collectCloths	fetches the cloths to be accepted for that particular wing.
/washerman/wing/fetchSummary	fetches the summary of cloths for the given wing.
/washerman/wing/accept	accepts the cloths for which request has been made.
<b>Payment verification from Razorpay</b>	
/verifyPayment	Through this student gets the verification notification from the payment gateway

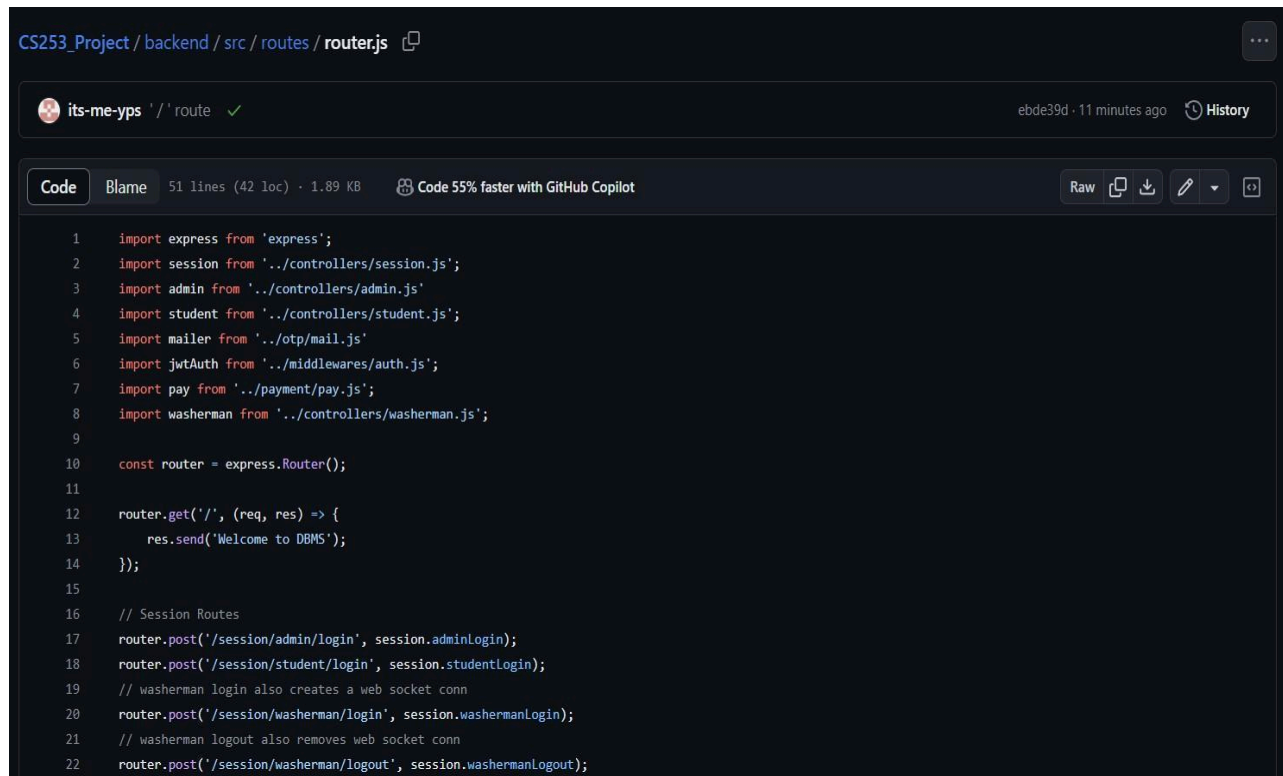
We used Authentication middleware to access other routes **jwtauth** :

1. **Importing JWT:** The code imports the JWT library/module. JWT is a compact, URL-safe means of representing claims to be transferred between two parties. The **JWT** library is being used here to verify and decode JWT tokens.
2. **jwtAuth Middleware Function:** This middleware function takes three parameters: **req** (request), **res** (response), and **next** (callback function to call when the middleware is finished). It's a common pattern in Node.js middleware.

3. **Token Verification:** Inside the middleware function, it checks if there are cookies in the request (**!req.cookies**) or if there is no token property in the cookies. If either of these conditions is true, it responds with a 401 Unauthorized status and a JSON object containing a message indicating that no token was provided.
4. **Extract Token:** If there is a token in the cookies, it extracts it from the requested cookies.
5. **Token Verification:** The **jwt.verify** function is then called with the token and a secret key (**process.env.JWT\_SECRET**). This function decodes and verifies the

token against the secret key. If the token is valid, it returns the decoded payload. If not, it throws an error.

6. **Valid Token:** If the token is valid, the decoded payload (usually containing user information or permissions) is attached to the request object (**req.user**) for subsequent middleware or route handlers to use.
7. **Invalid Token:** If the token is invalid (either expired, tampered with, or simply not matching the expected format), it catches the error thrown by **jwt.verify** and responds with a 403 Forbidden status and a JSON object containing a message indicating that the token is invalid.
8. **Exporting Middleware:** Finally, the middleware function is exported to be used in other parts of the application.



```
CS253_Project / backend / src / routes / router.js

its-me-yps '/' route ✓ ebde39d · 11 minutes ago History

Code Blame 51 lines (42 loc) · 1.89 KB Code 55% faster with GitHub Copilot Raw Copy Download Edit View Log

1  import express from 'express';
2  import session from '../controllers/session.js';
3  import admin from '../controllers/admin.js';
4  import student from '../controllers/student.js';
5  import mailer from '../otp/mail.js';
6  import jwtAuth from '../middlewares/auth.js';
7  import pay from '../payment/pay.js';
8  import washerman from '../controllers/washerman.js';
9
10 const router = express.Router();
11
12 router.get('/', (req, res) => {
13   res.send('Welcome to DBMS');
14 });
15
16 // Session Routes
17 router.post('/session/admin/login', session.adminLogin);
18 router.post('/session/student/login', session.studentLogin);
19 // washerman login also creates a web socket conn
20 router.post('/session/washerman/login', session.washermanLogin);
21 // washerman logout also removes web socket conn
22 router.post('/session/washerman/logout', session.washermanLogout);
```

```
23 router.get('/session/logout', session.logout);
24
25 // Register Routes
26 router.post('/sendAuthCode', mailer.sendOTP);
27 router.post('/student/register', student.register);
28
29 // Authentication Middleware to access other routes
30 router.use(jwtAuth);
31
32 // Admin Routes
33 router.post('/admin/washerman/register', admin.registerWasherman);
34 router.post('/admin/addHallData', admin.addHallData);
35
36 // Student Routes
37 // requestWash is done through web socket conn and response is received through event listener
38 router.post('/student/requestWash', student.requestWash);
39 router.post('/student/fetchRecord', student.fetchRecord);
40 router.get('/student/fetchDates', student.fetchDates);
41 router.post('/student/payment/fetchReceipt', student.fetchReceipt);
42 router.get('/student/payment/fetchDates', student.paymentDates);
43 router.get('/student/payment/clearDue', student.clearDue);
44 // For payment verification from RazorPay
45 router.post('/verifyPayment', pay.verifyPayment);
46
47 // Washerman routes
48 router.post('/washerman/wing/fetchRecord', washerman.wingRecord);
49 router.post('/washerman/upcomingDate', washerman.upcomingDate);
50
51 export default router;
```

Link to github repository of our team :

[https://github.com/its-me-yps/CS253\\_Project](https://github.com/its-me-yps/CS253_Project)

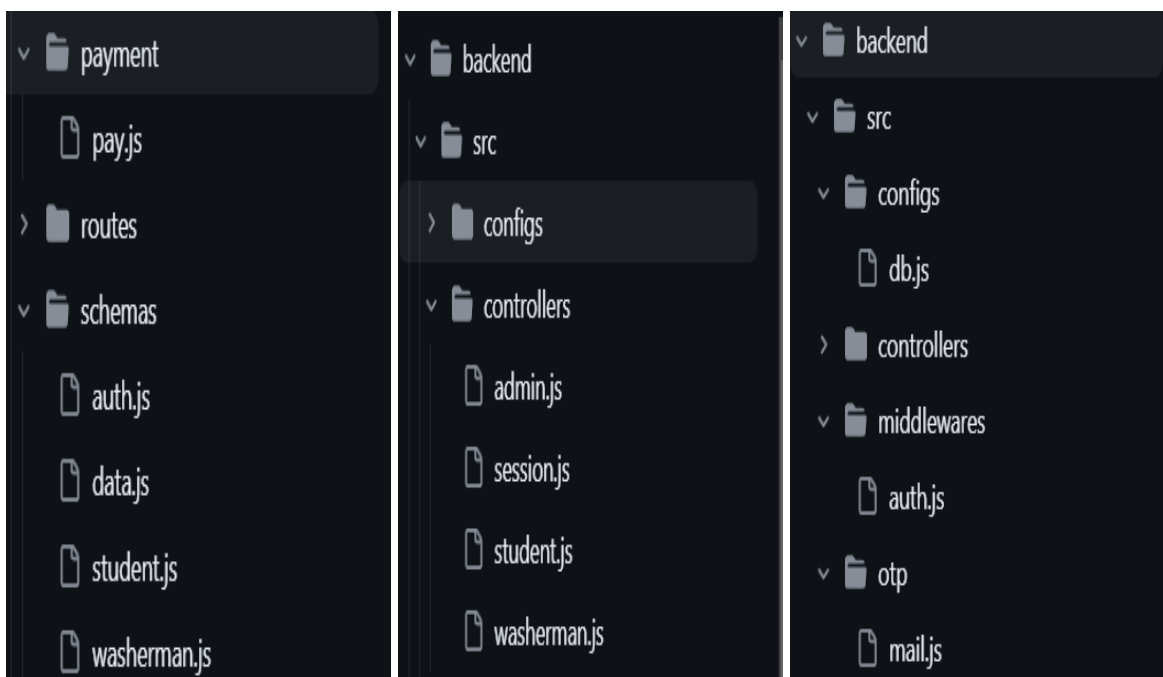
Frontend Repository : [https://github.com/its-me-yps/CS253\\_Project/tree/main/frontend](https://github.com/its-me-yps/CS253_Project/tree/main/frontend)

Backend Repository : [https://github.com/its-me-yps/CS253\\_Project/tree/main/backend](https://github.com/its-me-yps/CS253_Project/tree/main/backend)

**The link for the deployed website is provided in the readme file in the github repository.**

This organization has **two separate repositories** for **backend and frontend** components. The repositories are private as of now.

## **2.1 Backend Repository:**



---

### FIG : BACKEND REPOSITORY

Content of the **backend** Repository:

- (i) **Routes** folder contain the file **route.js**(contains all the endpoints)
- (ii) **Controllers** folder contains files
  - 1) **admin.js** - Containing functions for washerman registration,adding hall data, conversion of password into hash.
  - 2) **session.js**- Contains functions for admin login,student login,washerman login and generates a token while login,the token will expire upon logout.
  - 3) **student.js** -Contains functions for registration for student,clear due,fetching record.
- (iii) The folder **payment** contains the file **pay.js** containing the functions for Payment Order,verification of payment and transfer of payment.The transaction has involvement of a third party api Razorpay.
- (iv) The folder **Schemas** contains the files:-
  - 1) **student.js** :-schema for students. .
  - 2) **washerman.js** :-schema for washerman
  - 3) **data.js**:- containing the wing and hall schemas.
  - 4) **auth.js** :- contains the schemas for authentication,roll\_id.
- (v) The folder **OTP** contains the file:-
  - 1) **mail.js**:- containing functions for sending otp to the student's mail and for generating authentication code.

## 2.2 Frontend Repository:

✓ src	✓ src
> apiCalls	> apiCalls
> components	> components
> images	> images
✓ pages	> pages
🌀 LandingPage.jsx	✓ styles
🌀 Login.jsx	# LandingPage.css
🌀 NotFound.jsx	# Login.css
🌀 RegisterStudent.jsx	# RegisterStudent.css
🌀 ResetPassword.jsx	# resetPassword.css
🌀 StudentDashboard.jsx	# WashermanSelection.css
JS userdata.js	
🌀 WashClothes.jsx	
🌀 WashermanDashboard.jsx	
🌀 WashermanSelection.jsx	

WashermanDashboard(for

--

Index in SRS	SRS Requirement	Status
3.2.1	Account Creation for student	Completed
3.2.2	Student login.	Completed
3.2.3	Student homepage with calendar.	Completed
3.2.4	Colour Coded Calendar	Completed
3.2.5	Price Rate Chart	In progress
3.2.6	Payment Option choice	Completed
3.2.7	Login page for Washerman	Completed
3.2.8	Monthly summary generation	Completed
3.2.9	Accepting/Denying Student request	Completed
3.2.10	Notification upon receiving payment.	Completed
3.2.11	Translation feature	In progress

### 3.1 Updated Feature(s):

- ~~Registration for washerman:~~  
~~The student shall create an account using his email, wing, and room number.~~  
~~The student will create a password for the website.~~
- This feature has been removed, instead the washerman will be registered by an implicit admin who will provide them with their password. This is to make the experience more convenient for the washerman and avoid additional hassle for them.



---

## **3.2 Future Developments:**

- Price Chart: Adding a Price chart to the user dashboard will make the experience more informative for the user. This can be developed further by specifying prices for each type of clothing.
- Translation Feature: A translation feature which translates the contents of the website to Hindi (or possibly other regional languages) will be helpful to the user who are not very comfortable with English
- Details Updation: A feature for updating details of both students and washerman can be added to keep the database updated.
- Creating a corresponding mobile application might make the functioning more convenient for the users.
- A feedback system can be put into place which can take feedback from the user and improve upon the current functioning of the website.

SL no.	Date	Timings	Venue	Description
1	10 February 2024	2 pm -4pm	RM Building	Had a meeting to decide the course of our implementation.It was decided that we would do frontend first, followed by backend and then integration.
2	15 February 2024	9pm -11pm	RM building	Finalised our implementation details, framework and distributed the work amongst ourselves.
3	17 February 2024	4pm- 7pm	RM building	We initiated the development of project features, gathering input from each team member as they contributed their ideas..
4	18 February 2024	9pm -11:30pm	Google Meet	Started working on key aspects of the project and everyone gave their ideas.
5	20 February 2024	8pm-9:30 pm	RM Building	Setup the git repository and decided some rules before pushing in the main branch.
7	26 February 2024	12pm-3pm	RM building	We created the final frontend for our product. And started the backend work.
8	11 March 2024	9pm-11pm	RM building	Met to combine the work of different persons and cleared each other's doubts.
9	13 March 2024	9pm-10pm	Google Meet	Started making the implementation document and decided briefly on the template.
10	15 March 2024	5pm-7pm	RM building	Started integration of frontend and backend parts.
11	16 March 2024	8pm to 11pm	RM building	Continued the integration and completed the Implementation Details

				part of the document
<b>12</b>	17 March 2024	3pm to 5pm	RM Building	Completed the integration and completed the completeness part of the document.
<b>13</b>	18 March 2024	8pm to 12pm	RM building	Had a final meeting before submission, reviewed the entire repository, completed the Implementation document.