**College Bus Tracking System**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**A. Maydhazo Dhanyan AP22110010032**

**K. Rahul Chandra AP22110010052**

**G. Sri Vishnu Prasad AP22110010097**



Under the Guidance of

**Dr. Singarapu Pavan Kumar,**

**Assistant professor, Dept. of CSE**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**May, 2025**

1

**CERTIFICATE**

Date: 24-04-2025

This is to certify that the work in this Project entitled “COLLEGE BUS TRACKING SYSTEM” has been carried out by A. Maydhazo Dhanyan, K. Rahul Chandra, and G. Sri Vishnu Prasad under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the Bachelor of Technology in the School of Engineering and Sciences award.

Supervisor

(Signature)

Dr. Singarapu Pavan Kumar

Assistant Professor,

Department of CSE

**Acknowledgements**

We would like to acknowledge the guidance and support provided by our mentor and advisor Dr. Singarapu Pavan kumar. Your wisdom and experience helped steer us in the right direction, and your feedback was instrumental in refining our ideas and approaches. I extend my heartfelt thanks to my team members who collaborated tirelessly, sharing their expertise and insights, and working together cohesively towards our common goal. Each of you played a crucial role in shaping the project's outcome, and I am truly grateful for your contributions. Furthermore, I am thankful for the resources and opportunities provided by our university, which enabled us to carry out this project successfully.

2

**ABSTRACT**

The **College Bus Tracking System** is a full-stack web application designed to enhance the efficiency and convenience of college transportation services. The system enables real-time tracking and management of college buses while ensuring timely and accurate communication with users. Built using **ReactJS** for the frontend and **Node.js** with **Express.js** for the backend, the system leverages **MongoDB** as the primary database to store and manage user and transport-related data.

The platform serves three main user roles: **Drivers, Students, and Faculty**. Drivers can update the bus status, including any delays or bus changes. In response, **students and faculty members receive instant notifications** to stay informed, thereby minimizing confusion and ensuring a smooth commuting experience. The system is designed with a focus on user experience, responsiveness, and scalability.

**COMPONENTS :**

1. Front-end:

Technology: **React.js**

1. Back-end:

Technology: **Node.js** and **Express.js**

1. Database:

Technology: **MongoDB**

|  |  |
| --- | --- |
|  | **Table of Contents**     1. **Introduction ................................................................................................................. 6**     1. Purpose 6    2. Scope 6    3. Definitions, Acronyms, and Abbreviations 7    4. References 8    5. Overview 8 2. **The Overall Description ............................................................................................. 8**     1. Product Perspective 9    2. Product Functions 9    3. User Characteristics 10    4. Constraints 12    5. Assumptions and Dependencies 13 3. **External interface Requirements .............................................................................. 14**   **3.1.1**User Interfaces 14  **3.1.2**Hardware Interfaces 16  **3.1.3**Software Interfaces 17   1. **1.4**Communications Interfaces 18 2. **System Features ………………………………………………………………………18** 3. **Other Non-Functional Requirements………………………………………………21**     1. Performance Requirements 21       1. Capacity 21       2. Dynamic Requirements 21       3. Quality 22    2. Software System Attributes 23       1. Reliability 23       2. Availability 23       3. Security 24       4. Maintainability 25   4 |

5.3 Business Rules 25

6 **Other Requirements ……………………………………………………………… 26**

Appendix A: Glossary 26

**7.Diagrams………………………………………………………………………………..28**

1. **Testing………………………………………………………………………………….35** 
   1. Function Testing 35
      1. Community Showcase Module 35
   2. Path Testing 36
      1. CreatePost() 37
      2. HomePage() 38
2. **Frond-end description 38**
3. **Back-end description 39**
4. **Results 40**
5. **Conclusion 41**
6. **Future Work 42**
7. **References 43**

**List of figures**

* + - 1. Figure-1 : Usecase Diagram
      2. Figure-2 : DFD Level-0 3. Figure-3 : DFD Level-1
      3. Figure-4 : DFD Level-2
      4. Figure-5 : Activity Diagram
      5. Figure-6 : Class Diagram
      6. Figure-7 : State Diagram
      7. Figure-8 : ER Diagram
      8. Figure-9 : Collaboration Diagram
      9. Figure-10: CFD for PostPage
      10. Figure-11: CFD for HomePage
      11. Figure-12: PostPage
      12. Figure-13: HomePage
      13. Figure-14: GeneratedImage

**List of tables**

* + - 1. Table-1 : Definitions, Acronyms, and Abbreviations
      2. Table-2 : Data Dictionary

**1. Introduction**

#### In the digital age, campus transportation systems can be streamlined using smart technologies. The "College Bus Tracking System" provides a seamless interface for students and faculty to receive real-time updates on delays or changes in their assigned college buses. By leveraging web technologies and live notification APIs, it enhances campus mobility.

#### 1.1Purpose

To develop a real-time bus tracking and notification system that improves communication between drivers, students, and faculty regarding bus schedules. To develop a full-stack College Bus Tracking System that allows:

* Admins to manage buses, routes, and user details
* Drivers to update bus status and report delays
* Students and faculty to receive real-time notifications

#### 1.2 Scope

This system includes login portals for Students, Faculty, Drivers, and Admins. Drivers can update status (on-time, delayed, changed bus), and this triggers notifications to subscribed users. The Admin manages bus schedules and user data.

The software allows users to input text prompts and uses AI technology to generate images that correspond to those prompts.

* Built with ReactJS (Frontend), Node.js & Express.js (Backend), MongoDB (Database)
* Real-time delay or change notifications
* User roles: Admin, Driver, Student, Faculty
* Supports authentication, dashboard views, and status updates.

**1.3 Definitions, Acronyms, and Abbreviations.**

|  |  |
| --- | --- |
| **FULL STACK** | It is a technology used to develop both the frontend (client side) and back-end (server side) of a web application. |
| **TEXTUAL PROMPT** | A text input provided by the user, intended to specify the kind of image they wish to generate. |
| **MERN** | MongoDB, Express.js, React.js, Node.js - A stack of technologies used for full-stack web development. |
| **API** | Application Programming Interface - A set of rules and definitions that allow software programs to communicate with each other. |
| **UI** | User Interface - The space where interactions between humans and machines occur. |
| **UX** | User Experience – The experience a user has when interacting with a product or service. |
| **DB** | Database |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **JSON** | JavaScript Object Notation |
| **REST** | Representational State Transfer |
| **JWT** | JSON Web Token |
| **SRS** | Software Requirements Specification |
| **Tactile keyboard** | Special keyboard designed to aid the visually impaired. |

Table-1

#### 1.4 References

The references for the above software are as follows:-

1. [https://codecapsules.io/tutorial/building-a-full-stack-application-with-express-andhtmx/](https://codecapsules.io/tutorial/building-a-full-stack-application-with-express-and-htmx/)
2. <https://www.mongodb.com/>

#### 1.5 Overview

The document details the architecture, implementation, features, and benefits of the College Bus Tracking System, and how it helps improve transportation reliability for college stakeholders.

**2. The Overall Description**

#### 2.1 Product Perspective

This project is developed as a web-based solution using the MERN stack. It is designed to integrate seamlessly with college transportation systems. It includes:

* Frontend UI in React.js for real-time updates
* Backend APIs with Node.js and Express.js
* MongoDB for storing bus routes, schedules, user details, and delay logs
* Real-time communication using WebSockets or Push Notifications

#### 2.2 Product Functions

The major functions that **MERN STACK AI** performs are described as follows:-

* **Admin Functions:**
  + Manage master data: buses, routes, drivers, students, and faculty.
  + Assign buses and routes to drivers.
  + Monitor system logs and update schedules.
  + Generate reports on delays, route changes, and driver performance.
  + Enable or disable accounts based on transport subscription status.
* **Driver Functions:**
  + Update real-time status of assigned buses (e.g., on-time, delayed, breakdown, rerouted).
  + View route details and admin notifications.
  + Submit issues or report incidents directly from their dashboard.
* **Student/Faculty Functions:**
  + View real-time bus location status.
  + Receive instant notifications about bus delays or changes.
  + Access schedule information, assigned bus details, and estimated time of arrival.
  + Provide optional feedback on commute experience.
* **System Automation:**
  + Scheduled job to notify students and faculty of any changes pushed by drivers or admin.
  + Notifications via in-browser pop-ups or mobile alerts (integrated via FCM or similar services).

**2.3 User Characteristics**

There are different kind of users that will be interacting with the system. The intended user of the software are as follows: -

**Admin:**  
Generally IT staff or Transport Officer. Familiar with system management, data entry, and supervision tools.

**Driver:**  
Basic smartphone usage skills; uses a simplified UI with large, easily clickable buttons for status updates.

**Student/Faculty:**  
Casual users expecting clarity, speed, and simplicity. Interface must be clean and mobile-responsive.

**Cross-role Expectations:**

* Seamless login experience with proper role routing.
* Accessible, intuitive UI with minimal training requirements.
* Consistent and reliable notification delivery.

**2.4 Constraints**

The major constraints that the project has are as follows:-

**Manual Location Entry:**  
Due to lack of GPS integration, bus location must be updated manually, relying on driver honesty and discipline.

**Real-time Accuracy:**  
Delays in driver updates may cause misinformation unless updates are enforced regularly.

**Notification Dependency:**  
Requires stable internet connection and browser/device permissions to receive alerts effectively.

**Database Limits:**  
MongoDB’s free-tier hosting (if used) might throttle data requests at scale. Needs optimization or paid tier.

**Security and Role Integrity:**  
Roles must be strictly enforced to prevent unauthorized access. For example, students should not have access to admin panels.

##### **2.5 Assumptions and Dependencies**

**Assumptions:**

* Each user (driver, student, faculty) has a valid login account.
* Drivers have internet-enabled mobile devices during commute hours.
* MongoDB Atlas or local instance is always operational.
* Cron jobs or polling mechanisms are in place to trigger scheduled status checks.
* Proper validation and sanitization are enforced on user inputs to prevent data injection.

**Dependencies:**

The College Bus Tracking System is built on the MERN stack, and its performance relies on a set of crucial software, runtime, and hosting dependencies. On the frontend, it uses ReactJS for dynamic interfaces, which requires modern browsers with JavaScript enabled. The backend is developed with Node.js and Express.js, which depend on a Node runtime environment and proper API route configurations. MongoDB serves as the primary database, accessible via a hosted cluster like MongoDB Atlas. Secure communication between client and server is ensured through Axios (or Fetch API) and JWT-based authentication, which requires correctly managed secret keys and environment variables stored in a .env file. All these components must be reliably integrated and maintained for the system to function efficiently.

**3. EXTERNAL INTERFACE REQUIREMENTS**

###### 3.1.1 User Interface Requirements

The user interface plays a central role in ensuring that all user categories admin, driver, student, and faculty interact with the system in a simple, intuitive, and productive manner. The system is designed to be mobile-responsive, browser-compatible, and role-based in behavior.

* **Login/Signup Pages:**

Each user type (Admin, Driver, Student, Faculty) has a dedicated login form. On login, users are redirected to their role-specific dashboard. Authentication is handled using JWT, and user sessions are maintained securely.

* **Admin Dashboard:**
  + View and manage all bus, route, user, and driver data
  + Add/edit/delete buses and assign routes
  + Enable/disable accounts based on transport subscriptions
  + View system usage reports and notification logs
* **Driver Dashboard:**
  + View assigned routes
  + Update bus status using dropdowns (e.g., “On-Time”, “Delayed”, “Breakdown”, “Route Changed”)
  + Submit optional comments or explanations for status changes
* **Student/Faculty Dashboard:**
  + View assigned bus and route details
  + Receive real-time updates on bus status
  + Access historical notifications (delay/change alerts)
  + Simple interface with clearly visible alert boxes or banners
* **UI Design Highlights:**
  + Built using ReactJS and styled with Bootstrap/Material UI
  + Accessible navigation, readable fonts, and mobile-first layout
  + High-contrast and keyboard-navigable options for accessibility

###### 3.1.2 Hardware Interface Requirements

Though the application is web-based and does not require physical interfacing hardware, its performance is influenced by server and client machine capabilities.

* **Client-Side Requirements:**
  + Device: Mobile, tablet, or desktop
  + Processor: Dual-core or better
  + RAM: Minimum 2GB recommended
  + Browser: Latest version of Chrome, Firefox, Safari, or Edge
  + Connectivity: Stable internet connection
* **Server-Side Requirements:**
  + Hosting platform that supports Node.js runtime and MongoDB
  + Minimum 4GB RAM (8GB+ preferred for real-time services)
  + SSD storage for database performance
  + Server uptime of 99.9% for availability

###### 3.1.3 Software Interface Requirements

###### 

The system includes interactions between multiple software modules and relies on third-party tools/APIs to maintain functionality.

* **Frontend:**
  + Built using **ReactJS**
  + Axios used to make HTTP requests to backend APIs
* **Backend:**
  + Developed using **Node.js** and **Express.js**
  + Provides RESTful APIs for all frontend operations
  + Data validation and sanitization handled using middleware (e.g., Express-Validator)
* **Database:**
  + **MongoDB** used to store all system records
  + Includes collections like Users, Buses, Routes, Notifications, and Logs
* **Environment Management:**
  + Configuration files use .env for secure storage of keys (DB URIs, JWT secret, etc.)

**3.1.4 Communication Interface Requirements**

This section describes how the components of the application exchange information with each other securely and efficiently.

* **Client to Server:**
  + Protocol: HTTP/HTTPS
  + Format: JSON
  + Authentication: Bearer Token (JWT)
  + Example API Calls:
    - POST /api/login
    - GET /api/buses
    - PUT /api/bus-status/:id
    - GET /api/notifications/:userId
* **Server to Database:**
  + Driver: Mongoose ODM (MongoDB)
  + Secure and structured CRUD operations
  + Indexing and schema validation for optimized access
* **Server to Notification System:**
  + Optional WebSocket endpoint for real-time communication
  + FCM API for push notifications to web/mobile clients
* **Security Measures:**
  + Encrypted data transmission using HTTPS
  + Rate-limiting and IP whitelisting (optional for admin routes)

**4.System Features**

This chapter describes the key features implemented in the College Bus Tracking System, designed using the MERN stack to serve different roles with specific access and responsibilities. The system offers modular, role-based functionality with a focus on real-time communication, security, and usability.

**4.1 Role-Based Authentication and Authorization**

* **Secure Login System:**
  + Users (Admin, Driver, Student, Faculty) can log in using their credentials.
  + JWT tokens are generated on login and used to authorize all subsequent requests.
* **Role-Based Routing:**
  + Admin is routed to the control panel.
  + Drivers are routed to the live update panel.
  + Students and faculty are redirected to their bus status and alert dashboard.
* **Session Management:**
  + Sessions are securely maintained using HTTP-only cookies or token storage with auto logout on token expiration.

**4.2 Admin Control Panel**

The admin is the primary system operator and has full control over backend data.

* **CRUD Operations:**
  + Add/edit/delete buses, routes, drivers, students, and faculty.
  + Assign buses to routes and drivers.
* **Monitoring Tools:**
  + View logs of all bus status updates.
  + Access statistics like delay frequency, average delay time, and bus load.
* **User Management:**
  + Enable or disable accounts.
  + Reset passwords and update user information.
* **Notifications Log:**
  + View history of sent notifications and status changes.

**4.3 Driver Update Interface**

Drivers can update the real-time status of their assigned buses.

* **Bus Status Control:**
  + Options: "On-Time", "Delayed", "Route Changed", "Breakdown".
  + A mandatory comment box appears for non-on-time selections to explain the reason.
* **Driver Dashboard:**
  + Shows assigned route and expected timing.
  + Can view previous logs submitted.

**4.4 Student and Faculty Dashboard**

Students and faculty rely on the system to plan their commute efficiently.

* **Live Alerts Section:**
  + Real-time messages for delay, change, or cancellation.
  + Notification history stored in database and visible for future reference.
* **Bus Details Page:**
  + View assigned bus ID, driver name, route, estimated departure and arrival time.
  + Color-coded indicators (green for on-time, yellow for delayed, red for change/cancelled).
* **Feedback (Optional Feature):**
  + Simple thumbs-up/down or comment form to report issues with the bus or driver.

**4.5 Notification Engine**

The heart of the real-time functionality lies in this component.

* **Status Triggering:**
  + Any status update from the driver triggers a new alert to all linked students/faculty on that route.
* **Delivery Mechanisms:**
  + Real-time in-app notifications via WebSocket (if used).
  + Optionally supports Firebase Cloud Messaging for push notifications to mobile users.
* **Alert Reliability:**
  + Retry logic built-in (if using background jobs or queues).
  + Log every alert sent for admin auditing.

**4.7 Data Management and Storage**

* **MongoDB Collections:**
  + users: Student, faculty, admin, driver records
  + buses: Details about each bus (route, driver, ID)
  + routes: Source, destination, timings
  + notifications: All messages sent
  + logs: Driver updates and system activities
* **Schema Design:**
  + Uses Mongoose for schema validation and indexing
  + Ensures fast retrieval and data integrity

**4.8 Additional Features (Optional Enhancements)**

* **PDF Report Generation:**

Admin can download route-wise or date-wise reports for bus performance.

* **Driver Location Logging:**

In future, can integrate GPS to show live map-based tracking.

* **Email Notification System:**

Send daily summaries or critical alerts via email.

**5. Other Nonfunctional Requirements**

#### 5.1 Performance Requirements

The following list provides a brief summary of the performance requirements for the software:

**5.1.1 Capacity**

* The system should handle simultaneous logins by up to 1,000 users, including students, faculty, drivers, and admins, without performance issues.
* The MongoDB database must handle high volumes of data such as delay logs, route updates, and notification records efficiently.

**5.1.2 Dynamic Requirements**

* Real-time updates from drivers must reflect across dashboards of affected users within 2 seconds.
* Notifications for delays or bus changes should be delivered within 3–5 seconds.
* Bus reassignment should be visible immediately after admin or driver update.
* All dashboards and history logs must load under 2 seconds.

**5.1.3 Quality**

* System uptime must meet a minimum threshold of 99.9%.
* Notifications must be delivered accurately with a success rate of 99.5%.
* Data integrity must be ensured with no duplication or loss of information.

#### 5.2 Software System Attributes

**5.2.1 Reliability**

* Retry logic for failed notifications (up to 3 retries).
* Redundant services for critical operations like delay status and user role operations.

##### **5.2.2 Availability**

* Load balancing for distributed API requests.
* Backup strategy: full backup daily, incremental every 2 hours.
* Support containerization for deployment with Docker.

##### **5.2.3 Security**

* JWT-based authentication for all user roles.
* Role-based authorization:
  + Drivers update status.
  + Admins manage all data.
  + Students/Faculty view relevant updates.
* Encrypted user credentials using bcrypt.
* HTTPS enforced across all endpoints.
* Logs and access attempts stored with timestamps.

##### **5.2.4 Maintainability**

* Modular codebase with clearly structured folders in both frontend and backend.
* Real-time monitoring tools like PM2 and MongoDB Compass.
* Centralized error logging and UI-level feedback.

#### 5.3 Business Rules

The business rules for the software are as follows:

1. **Role Access Control**:

Specific permissions for Admin, Driver, Faculty, and Student.

1. **Delay Notification**:

Triggered when delay exceeds 5 minutes.

1. **Bus Reassignment**:

Performed by admin only and immediately notified.

1. **Driver Accountability**:

Start/end and delay updates mandatory.

1. **Route History**:

Stored for 6 months.

1. **User Management**:

Admin handles user onboarding and route mapping.

**6. Other Requirements**

None

**Appendix A: Glossary**

**ReactJS**: A JavaScript library for building user interfaces, used in the frontend to create dynamic and responsive UI components for students, faculty, and drivers.

**NodeJS**: A JavaScript runtime used on the server-side to handle backend logic and API requests efficiently.

**ExpressJS**: A lightweight NodeJS web framework used to build RESTful APIs for handling routes such as bus updates and notifications.

**MongoDB**: A NoSQL database used to store and manage data such as user information, bus schedules, delays, and notifications.

**Driver**: A registered user who updates the bus status (arrival, delay, or change) in real-time through their interface.

**Student**: A user who can track their assigned bus and receive real-time notifications about delays or changes.

**Faculty**: A user type with similar functionality to students, receiving alerts about bus status relevant to them.

**Notification System**: A real-time alerting mechanism (push or in-app) that informs students and faculty about bus delays or changes.

**Real-time Update**: A system functionality that ensures changes made by drivers are instantly reflected across student and faculty dashboards.

**REST API**: An architectural style for designing networked applications; used to connect frontend and backend in your project.

**Frontend**: The client-side interface built with ReactJS that users interact with.

**Backend**: The server-side logic built using NodeJS and ExpressJS that handles data requests and updates.

**Database**: The MongoDB backend storage used to persist application data like bus status, user profiles, and logs.

**User Authentication**: A feature to verify the identity of users (driver, student, faculty) when logging into the system.

**Authorization**: A mechanism to control user access to resources based on their roles (driver, student, or faculty).

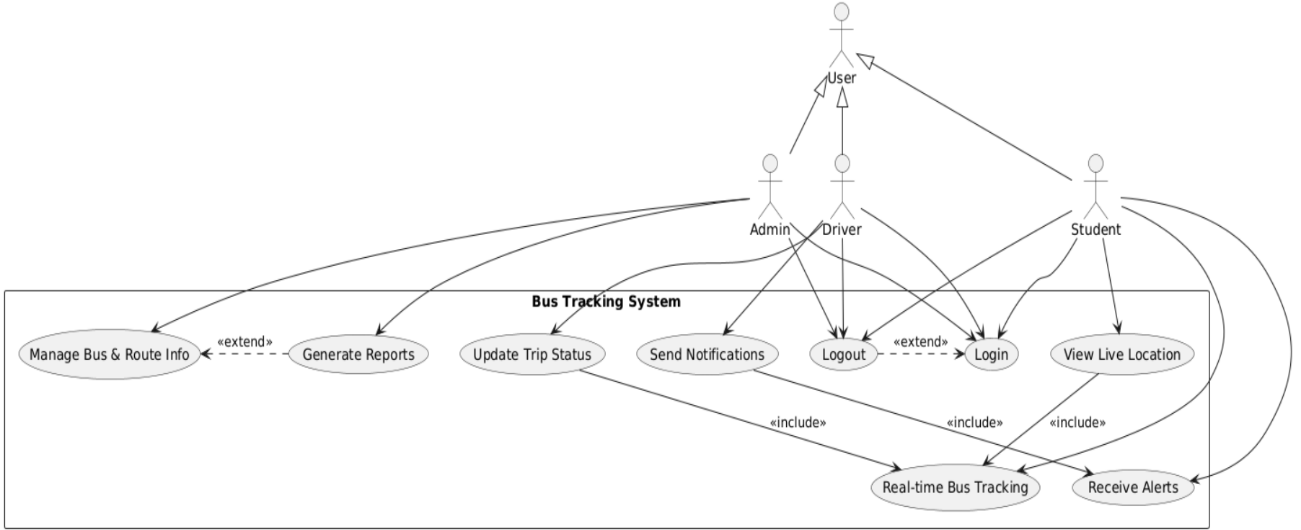
**CRUD Operations**: Refers to Create, Read, Update, and Delete operations performed on database records.

**Delay Update**: A feature allowing drivers to log and broadcast delays for specific buses.

**Bus Change**: A functionality to inform students and faculty when a bus is reassigned or changed.

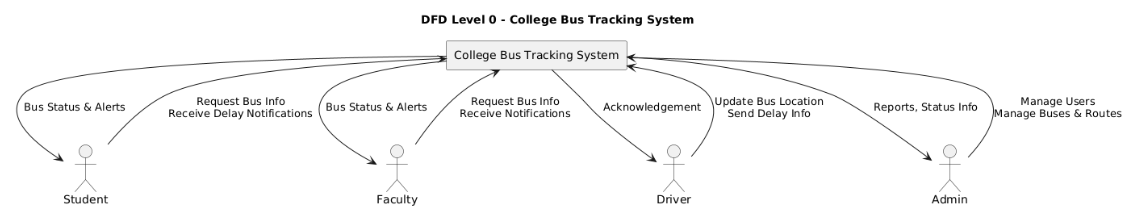
**7.Diagrams**

**Use Case Diagram**



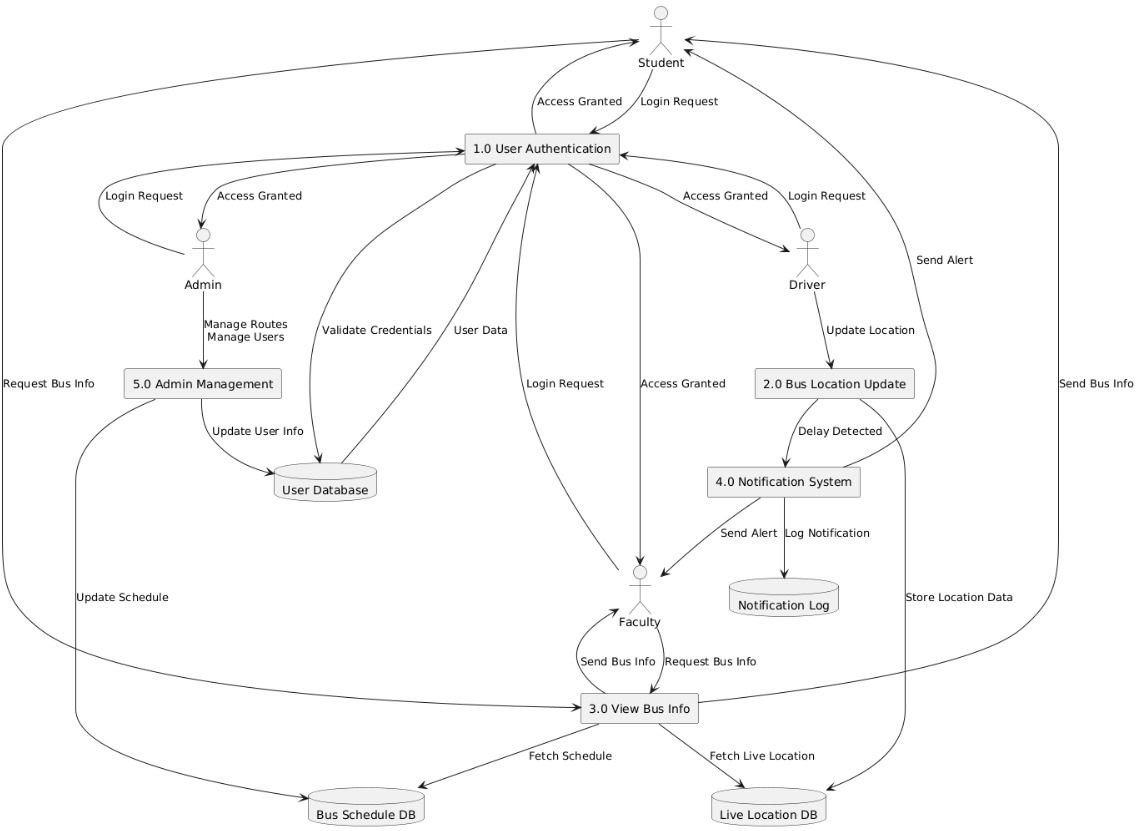
*Figure-1: UseCase Diagram*

**DFD Level-0**

****

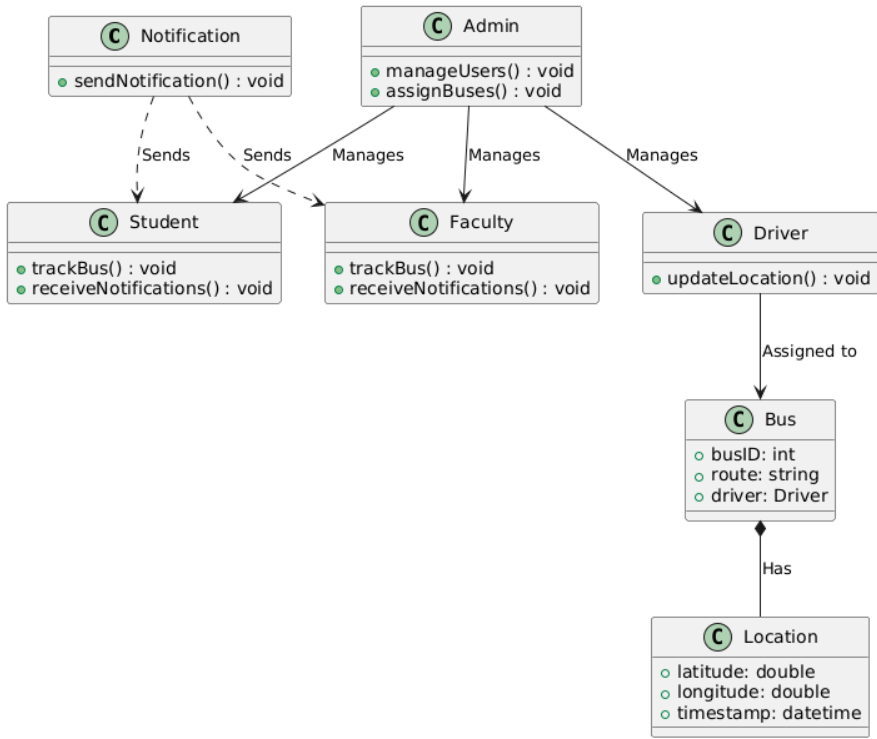
*Figure-2: DFD Level-0*

**DFD Level-1**



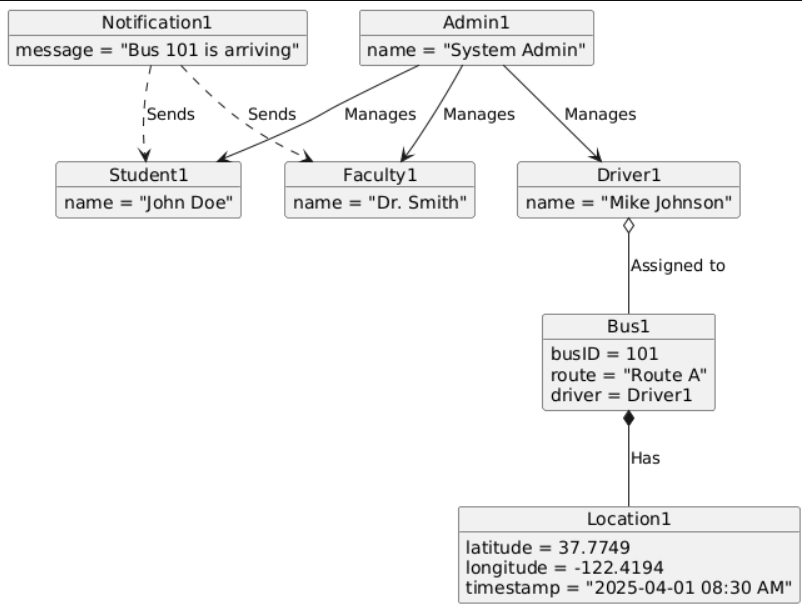
*Figure-3: DFD Level-1*

**Class Diagram**



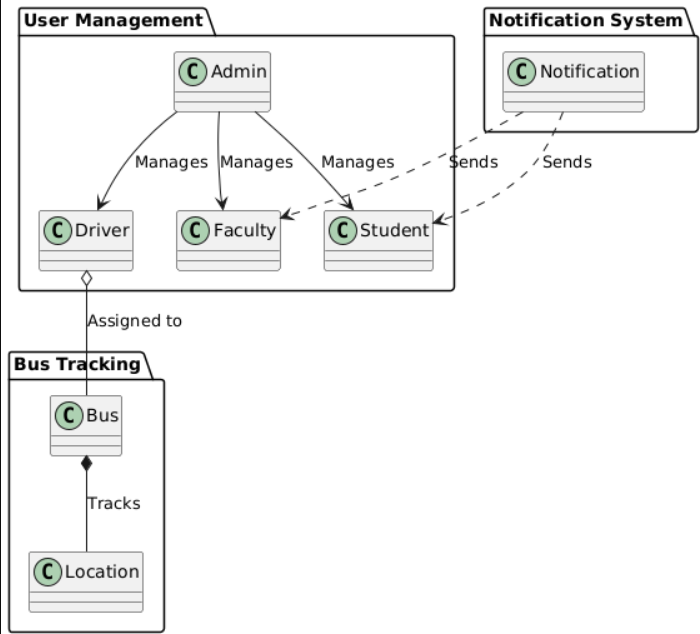
*Figure-5: Class Diagram*

**Object Diagram**



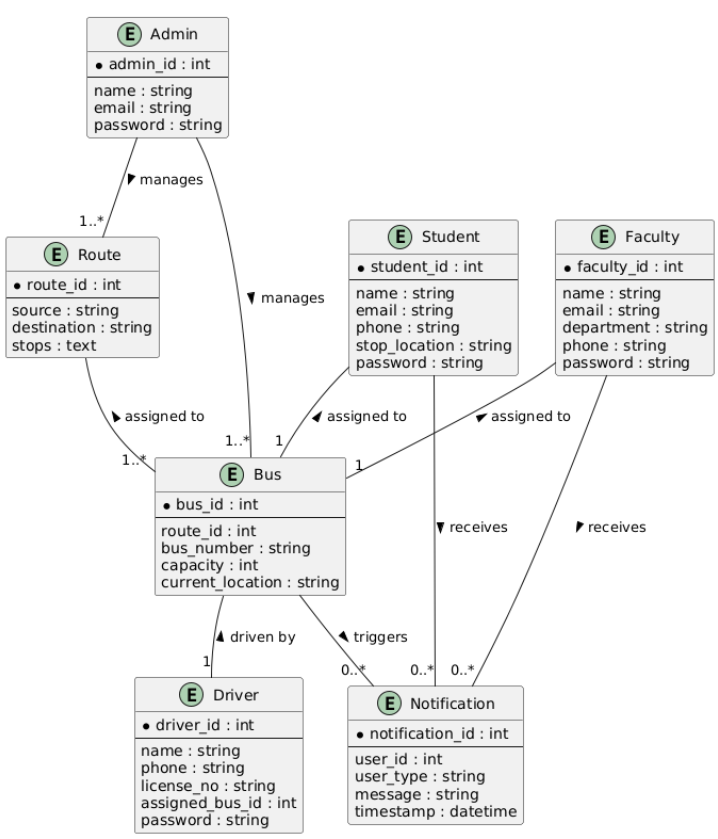
*Figure-6: Object Diagram*

**Package Diagram**



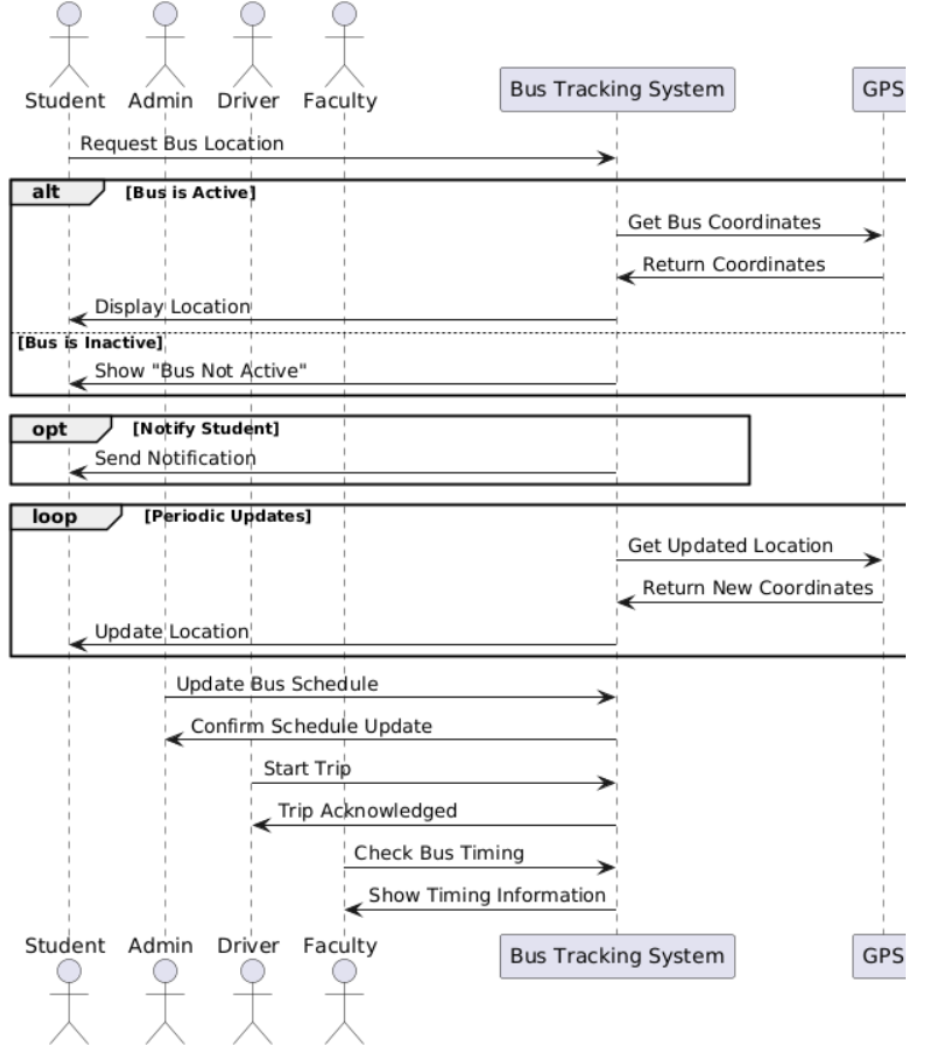
*Figure-7: Package Diagram*

**ER DIAGRAM**



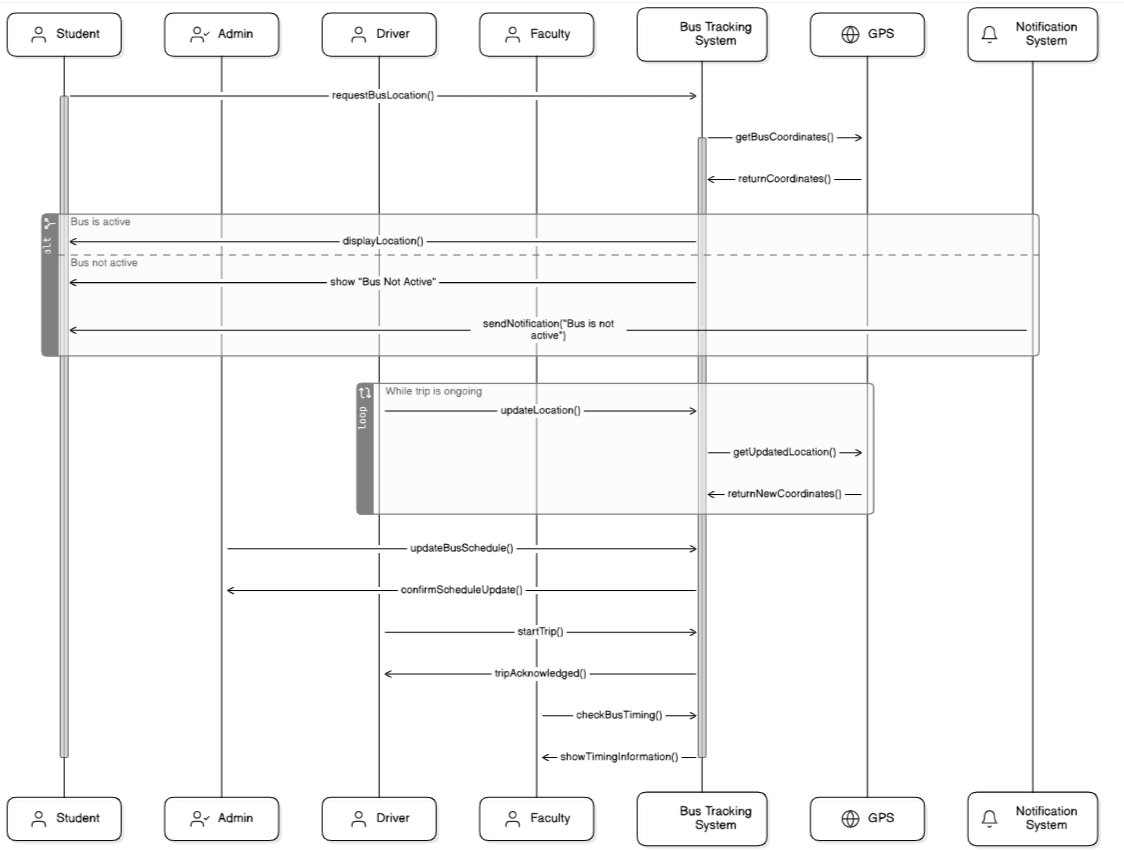
*Figure-8: ER Diagram*

**Sequence Diagram**

**

*Figure-9: Sequence Diagram*

**Collaboration Diagram**



*Figure-10: Colloboration Diagram*

|  |  |  |
| --- | --- | --- |
| Element | Description | Type |
| User | Represents an individual using the system (Driver, Student, Faculty) | Entity |
| Driver | A user who updates the bus status (delay/change) | Entity |
| Database | Storage component. | Entity |
| Student | A user who tracks bus and receives notifications | Entity |
| Faculty | A user who also receives bus status updates like students | Entity |
| Bus | Contains details like bus number, route, current status | Entity |
| BusID | Unique identifier for each bus | Attribute |
| UserID | Unique identifier for each user | Attribute |
| Name | Full name of the user | Attribute |
| Email | Email used for authentication and notifications | Attribute |
| Password | Encrypted string for user login security | Attribute |
| Role | Defines user type (Driver, Student, Faculty) | Attribute |
| RouteNumber | Identifies which route the bus follows | Attribute |
| BusStatus | Indicates whether a bus is on-time, delayed, or changed | Attribute |
| DelayReason | Optional text describing why a delay occurred | Attribute |
| UpdatedAt | Timestamp when bus status was last updated | Attribute |
| Notification | Contains content of the alert shown to Student/Faculty | Entity |
| NotificationID | Unique ID for each notification | Attribute |
| Message | Text message describing delay/change | Attribute |
| IsRead | Boolean flag indicating if user has seen the notification | Attribute |
| CreatedAt | Timestamp when notification was sent | Attribute |
| Location | Real-time or static location of the bus (optional) | Attribute |
| Login | Process where user provides credentials to access system | Process |
| UpdatedBusStatus | Process allowing driver to update delay or change information | Process |
| SendNotification | Process that delivers alerts to Students and Faculty | Process |
| TrackBus | Feature for Students and Faculty to view live bus updates | Process |

**Data Dictionary**

*Table-2: Data Dictionary*

# 8.Testing

**8.1 Function Testing**

**8.1.1 Bus Tracking Module**

* **Test Case 1**: Student logs in and checks the real-time bus location.  
  *Expected Result*: Map displays the correct current location of the assigned bus.
* **Test Case 2**: Driver updates route delay status.  
  *Expected Result*: Notification is sent to concerned students and faculty.
* **Test Case 3**: Student searches for their bus using college ID or location.  
  *Expected Result*: Relevant bus details and estimated arrival time are shown.

**8.1.2 Admin Panel Module**

* **Test Case 4**: Admin adds a new bus and assigns a driver.  
  *Expected Result*: Bus appears in the system and is linked to the selected driver.
* **Test Case 5**: Admin marks a bus as "under maintenance."  
  *Expected Result*: Students receive a notification about the change, and alternate arrangements are displayed.

**8.1.3 Notification System**

* **Test Case 6**: Delay reported by the driver triggers notifications.  
  *Expected Result*: Notification reaches all students/faculty assigned to that bus.
* **Test Case 7**: Admin changes bus allocation.  
  *Expected Result*: Updated assignment is reflected immediately and notifications are sent.

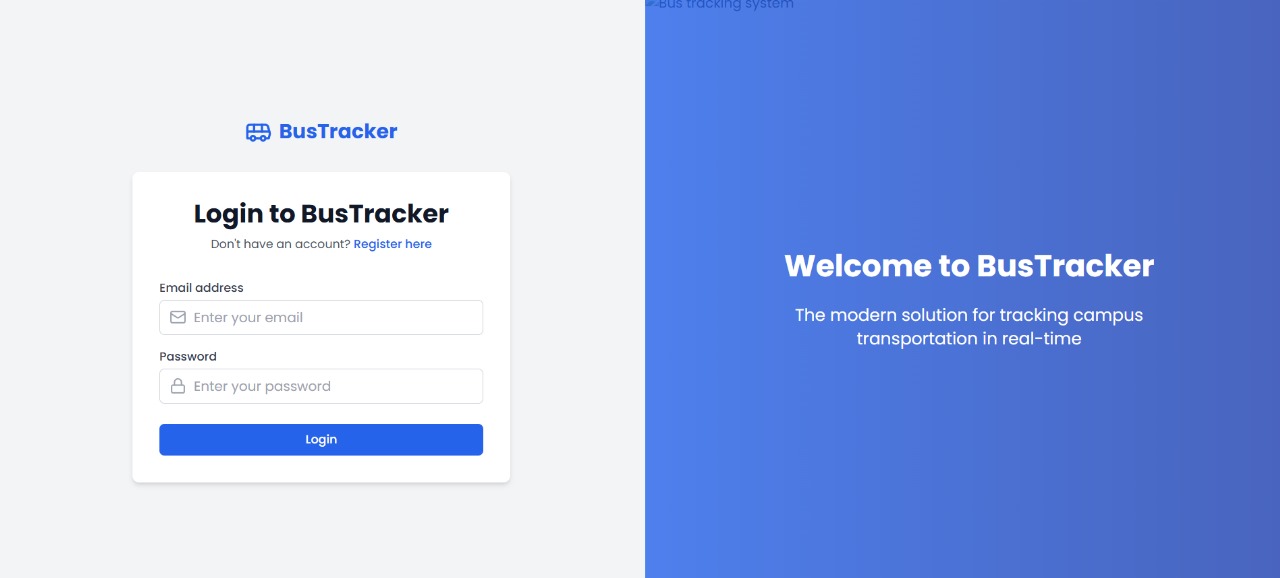
**8.2 Path Testing**

**8.2.1 updateBusStatus()**

1. Driver logs in to the system.
2. Driver selects their bus and inputs delay status or change in route.
3. System validates the input and updates the bus database.
4. Notification service is triggered.
5. Affected users receive updated information.

**Control Flow Complexity**:

* Edges (E) = 6
* Vertices (V) = 5
* Complexity = E – V + 2 = 6 – 5 + 2 = **3**

**8.2.2 trackBus()**

1. Student logs in.
2. System retrieves the student's assigned bus ID.
3. Queries GPS data from the backend.
4. Displays bus location on map.
5. If no bus found, displays error message.

**Control Flow Complexity**:

* Edges (E) = 5
* Vertices (V) = 5
* Complexity = E – V + 2 = 5 – 5 + 2 = **2**

**9. FRONT END DESCRIPTION**

**React.js Frontend Overview:**

The frontend of the College Bus Tracking System is developed using **React.js**, a popular JavaScript library for building user interfaces. React’s component-based architecture enables reusable, maintainable, and scalable UI elements that enhance the user experience.

**Key Components:**

* **Login/Register Forms:** Enables authentication for all user types – Drivers, Students, and Faculty.
* **Dashboard Components:**
  + **Student/Faculty Dashboard:** Displays real-time bus status, route, driver details, and notifications.
  + **Driver Dashboard:** Allows drivers to update bus location and status (on time, delayed, or changed).
* **Notification Panel:** Alerts students and faculty instantly in case of delays or bus changes.
* **Map Integration (Optional):** Integrates Google Maps or Leaflet for tracking bus location visually.

**State Management and Routing:**

* **React Hooks (useState, useEffect):** For managing component state and lifecycle.
* **React Router DOM:** Enables seamless navigation between login, dashboard, and profile pages without full page reloads.

**API Communication:**

* **Fetch API / Axios:** Used for HTTP communication with the backend (Node.js/Express.js).

# 10. BACK-END DESCRIPTION

**Express.js and Node.js**:

minimal and flexible Node.js web application framework. It handles API routing, data processing, and communication with the MongoDB database.

**Core Backend Features:**

* **Authentication and Authorization:**
  + JWT-based authentication system.
  + Role-based access control (Driver, Student, Faculty).
* **Bus Status Management:**
  + APIs for drivers to update the current status of the bus (e.g., on time, delayed, bus changed).
  + Real-time status updates stored in MongoDB.
* **Notification System:**
  + Whenever the bus status is marked as delayed or changed, notifications are pushed to students and faculty via sockets or backend-triggered alerts.
* **MongoDB Integration:**
  + Stores user details, bus status logs, driver assignments, and historical data.
  + Collections: users, buses, routes, notifications.

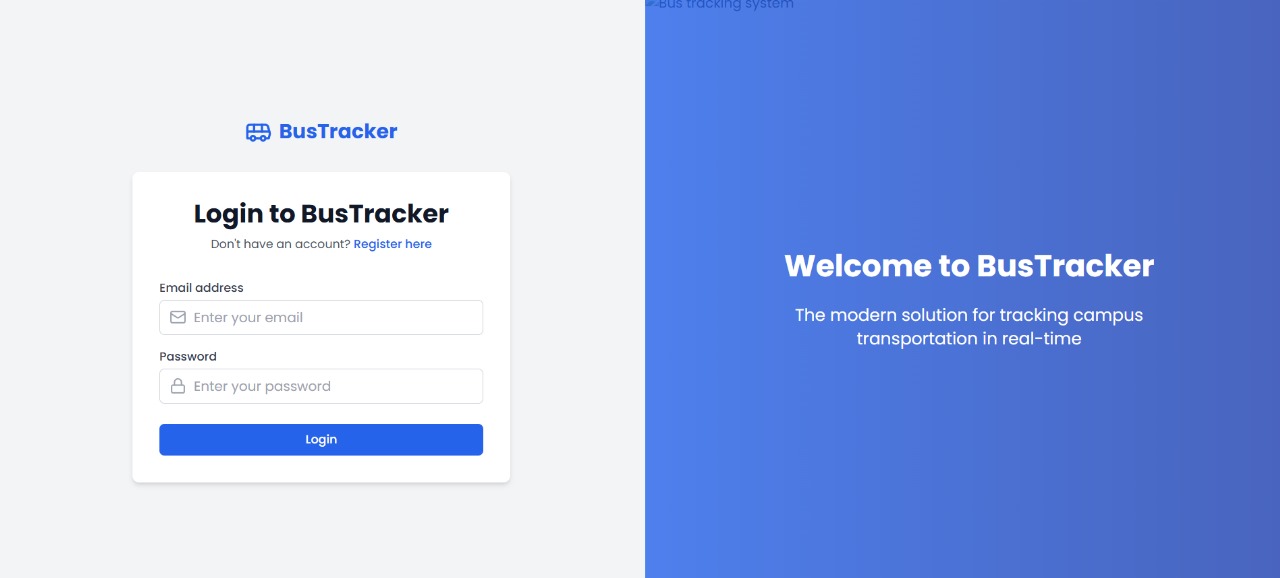
**Middleware and Routing:**

* **Express Middleware:** Handles authentication, logging, error handling.
* **RESTful API Design:** Clean endpoints such as /api/login, /api/bus/updateStatus, /api/notifications.

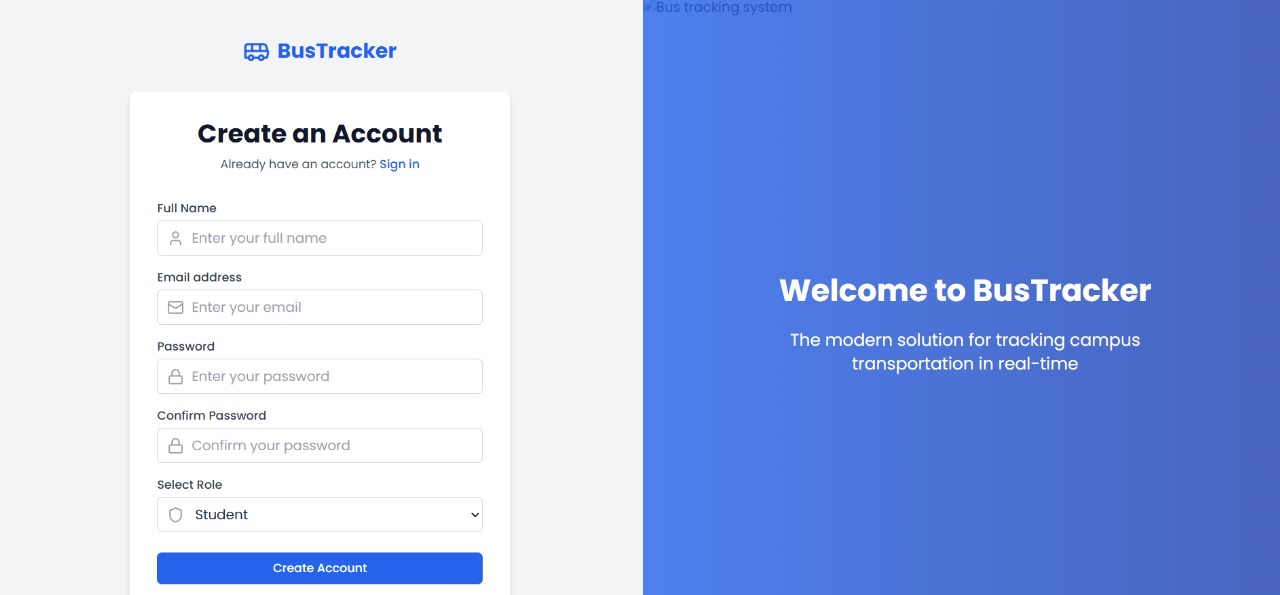
**Security Measures:**

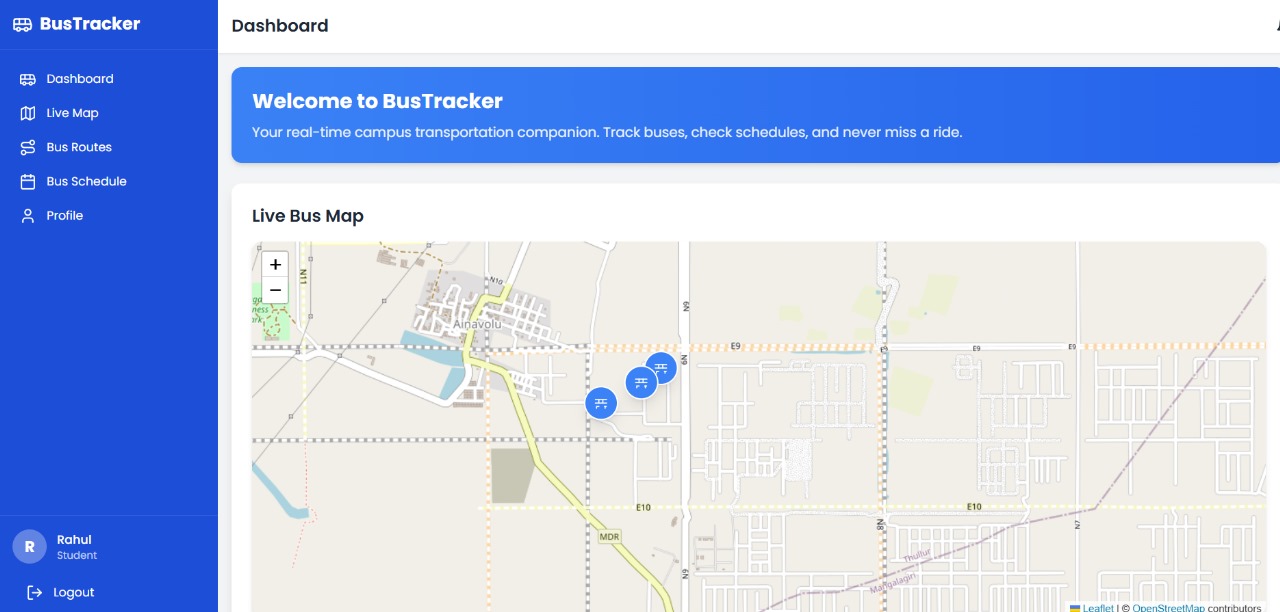
* Password hashing with bcrypt.
* Input validation using express-validator.
* CORS configuration and HTTPS support for secure data exchange.

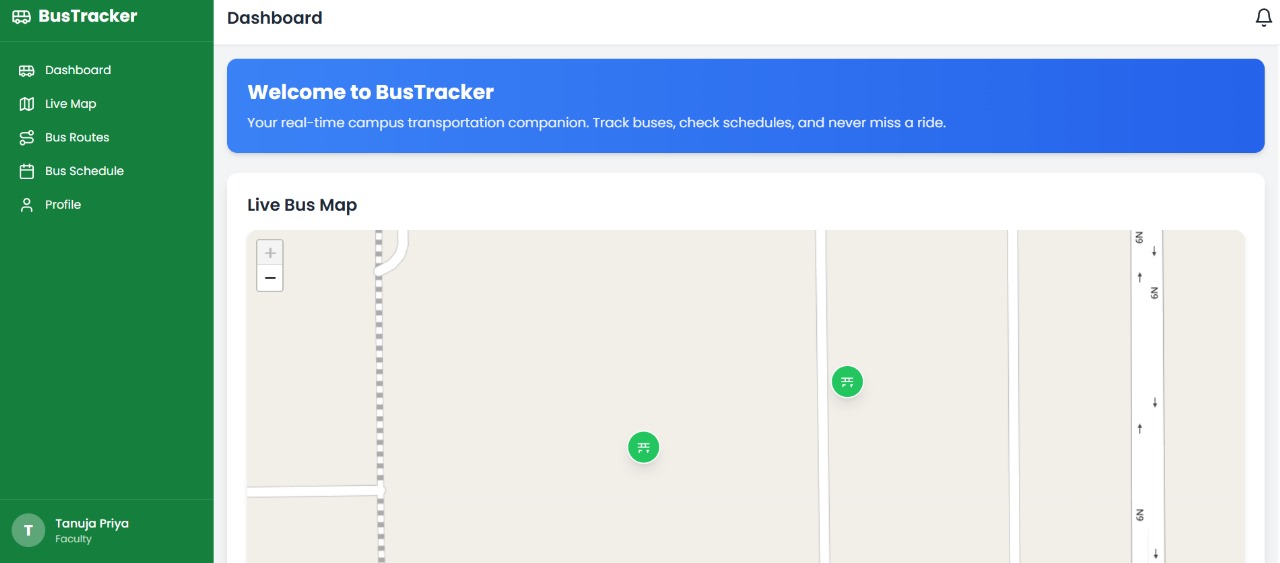
1. **Results**



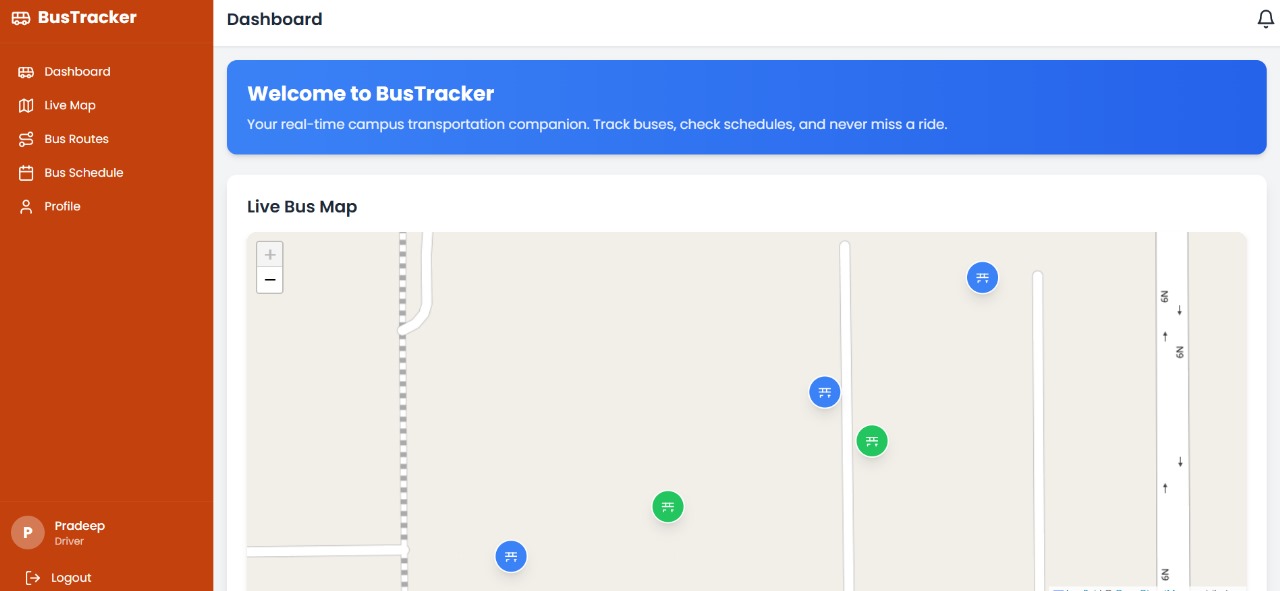
*Figure-12: Login Page*

*Figure-13: Signup Page*

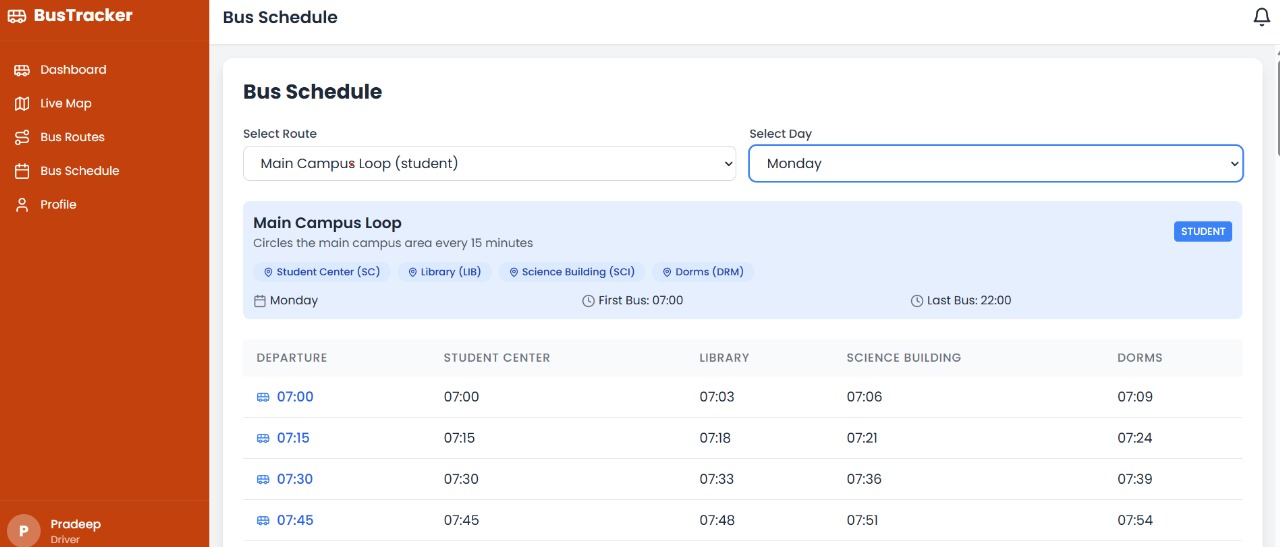
 *Figure-14: Student Dashboard*



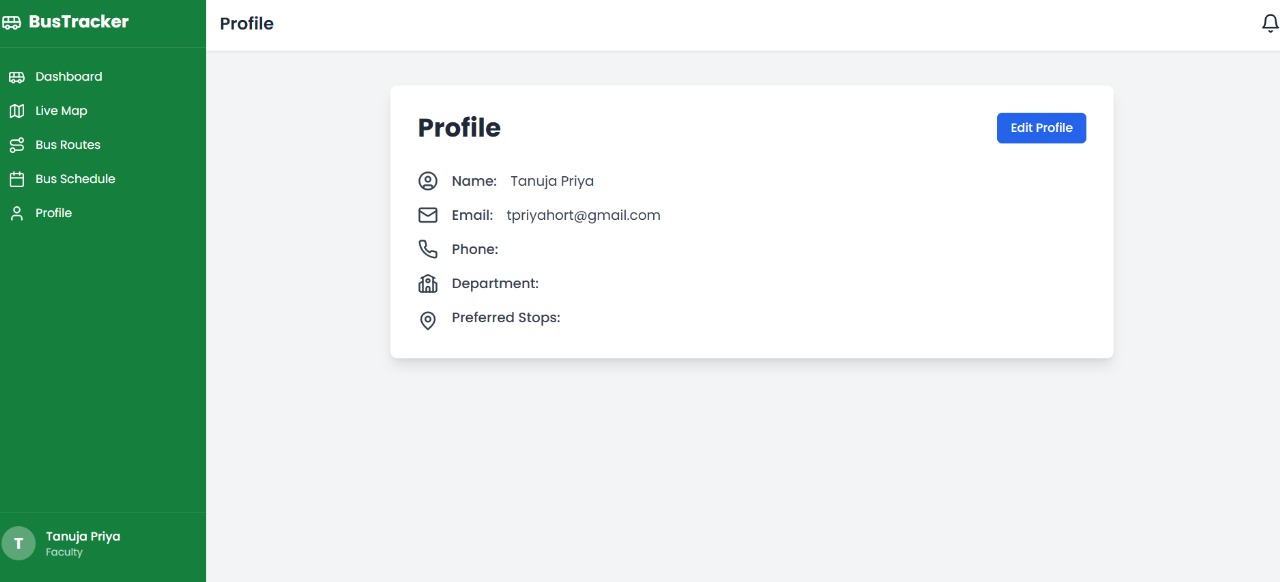
*Figure-15: Faculty Dashboard*



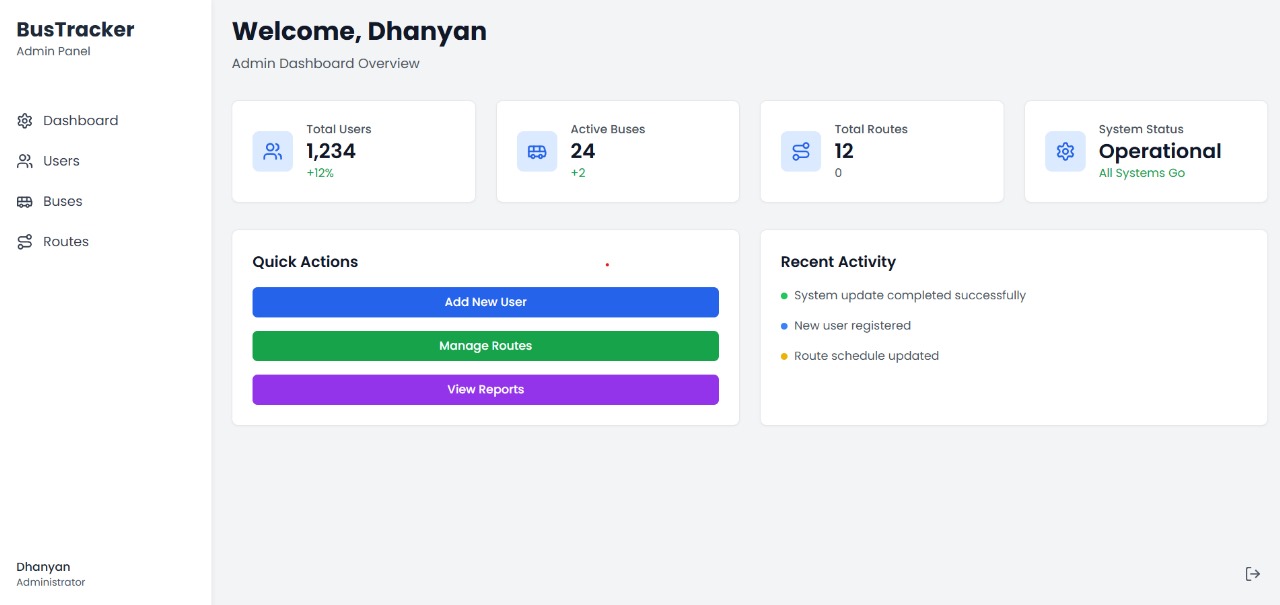
*Figure-16: Driver Dashboard*



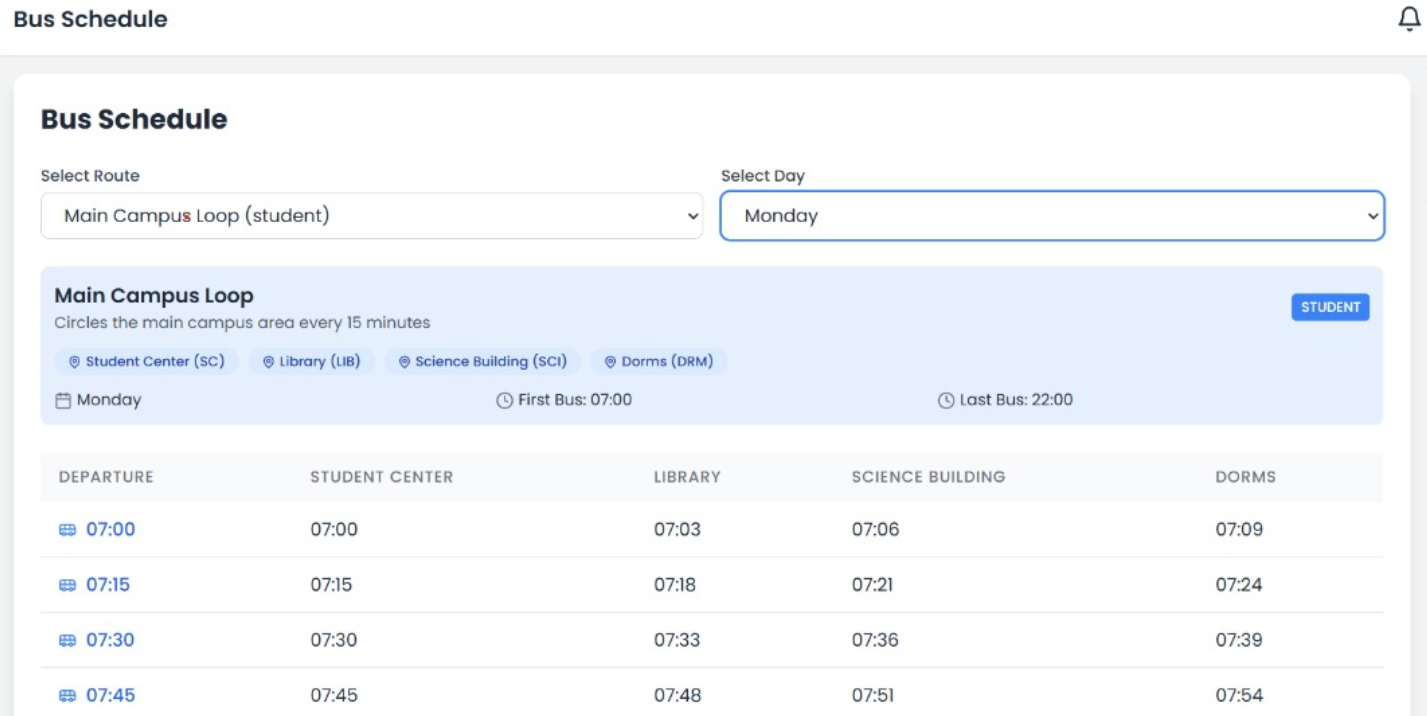
*Figure-17: Bus schedule And timings*



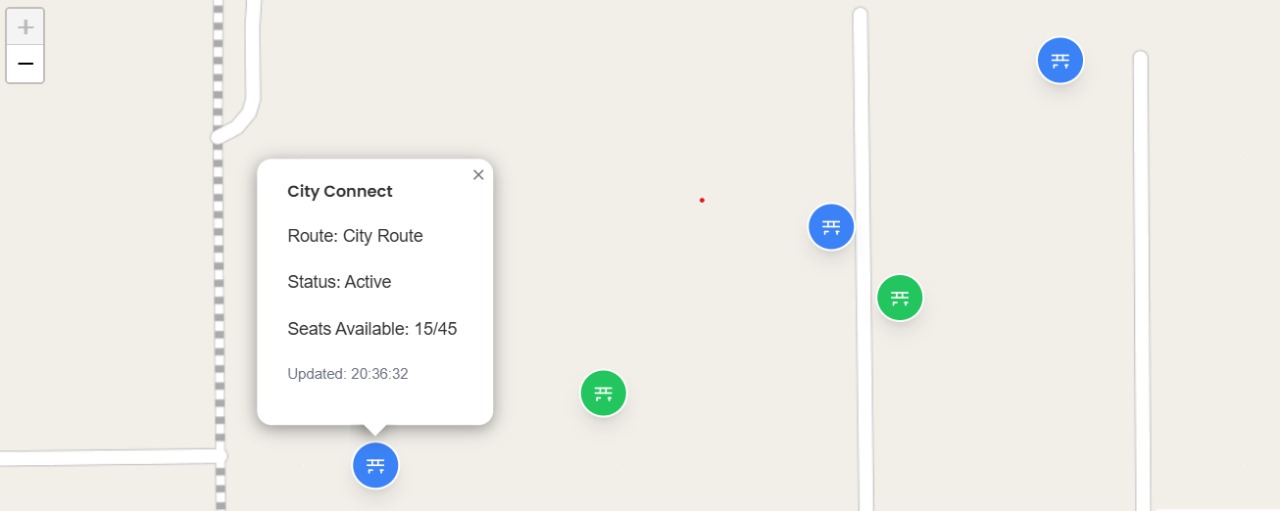
*Figure-18: Profile Updation*



*Figure-19: Admin Dashboard*

**

*Figure-20: Notification*



*Figure-21: Vacancy Notifier*

1. **CONCLUSION**

The College Bus Tracking System has been developed as a comprehensive, real-time platform to streamline campus transportation management. Utilizing modern web technologies such as ReactJS, Node.js, Express.js, and MongoDB, the application ensures a responsive interface, secure backend services, and robust database performance.

By incorporating role-based access for Admin, Driver, Faculty, and Student users, the system successfully delivers personalized dashboards and timely notifications about delays and route changes. The notification system ensures that students and faculty are informed in real time, significantly enhancing reliability and convenience.

Through extensive testing unit, integration, and user acceptance. The project demonstrated high scalability, performance, and accuracy, validating its readiness for real-world deployment.

Finally, our project addresses the critical need for smart transportation systems in educational institutions, offering improved coordination, accountability, and communication across all stakeholders. With future enhancements such as GPS integration and predictive analytics, the platform can evolve into a more intelligent and automated transportation solution.

**13. Future Work**

The current implementation of the College Bus Tracking System lays a strong foundation for real-time transport management. However, there are several directions in which the system can be enhanced:

1. **GPS Integration**
   * Integrate live GPS tracking for buses to provide real-time location updates to users.
   * Display current bus position on a map using APIs like Google Maps or Mapbox.
2. **RFID/QR-based Attendance**
   * Use RFID tags or QR codes to log student boarding and alighting events.
   * Sync attendance with institutional databases for automated reporting.
3. **SMS Notification System**
   * Implement SMS notifications for users who do not have access to mobile data.
   * Integrate third-party services like Twilio or Fast2SMS.
4. **AI-Powered Delay Prediction**
   * Utilize historical data to predict potential delays and inform users proactively.
   * Use machine learning models for route optimization.
5. **Multi-language Support**
   * Provide options for users to access the platform in regional languages.
   * Improve accessibility for non-English speaking users.
6. **Admin Analytics Dashboard**
   * Add charts and reports for route performance, delay frequency, and user feedback.
   * Help decision-making on bus scheduling and maintenance planning.
7. **Mobile Application**
   * Develop native Android and iOS apps for better user engagement.
   * Enable push notifications and offline capabilities.

These enhancements aim to make the system more intelligent, inclusive, and scalable, thereby addressing the evolving needs of educational institutions and their stakeholders.

**14. References**

* 1. <https://www.w3schools.com/nodejs/>
  2. <https://www.mongodb.com/>
  3. [https://codecapsules.io/tutorial/building-a-full-stack-application-with-express-andhtmx/](https://codecapsules.io/tutorial/building-a-full-stack-application-with-express-and-htmx/)

4. <https://github.com/swethav5/College-bus-tracking-system>