Mini-Project Report On

CareConnect (An Ambulance Dispatch Service)

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor Of Technology *in*

Computer Science & Engineering

By

Sahil Titto (U2003181)
Roy Rajesh (U2003177)
Shaun Tojan (U2003194)
Rony Mons (U2003175)

Under the guidance of
Mr. Ajith S

Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039

CERTIFICATE

This is to certify that the mini-project report entitled " ***CareConnect (An Ambulance Dispatch Service)"*** *is a bonafide work done by* ***Mr. Sahil Titto (U2003181), Mr. Roy Rajesh (U2003177), Mr. Rony Mons (U2003175), Mr. Shaun Tojan (U2003194)****, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.	**Ms. Anita John**	**Mr. Ajith S**
Professor And Head of the Department	Mini-Project Coordinator	Mini-Project Guide
Dept. of CSE	Asst. Professor	Asst. Professor
RSET	Dept. of CSE	Dept. of CSE
	RSET	RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal, RSET and **Dr. Preetha K. G.**, Professor And Head of the Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Care Connect".

We are highly indebted to our mini-project coordinators, **Ms. Anita John**, Assistant Professor, Department of Computer Science and Engineering and **Mr. Sajan Raj**, Assistant Professor, Department of Computer Science and Engineering, for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Mr. Ajith S**, Assistant Professor, Department of Computer Science and Engineering, for his patience and all the priceless advice and wisdom he has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Sahil Titto

Roy Rajesh

Shaun Tojan

Rony Mons

ABSTRACT

CareConnect is an innovative Critical Care Transport System designed to provide efficient online ambulance dispatch services. The system consists of three interfaces: User, Hospital Admin, and Ambulance Driver, each with specific functionalities to streamline the emergency response process. CareConnect aims to enhance the critical care transport process by leveraging modern web technologies, efficient communication APIs, and intelligent request management between users, hospital administrators, and ambulance drivers.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 User Interface

The User module of CareConnect enables individuals to access critical care services with ease. Users can log in securely using their phone numbers and a one-time password (OTP) verification system powered by Vonage API. Once authenticated, users are prompted to select their state and region, and using the FourSquare API, the system presents a curated list of hospitals within a 50 to 60-kilometer radius. From this list, users can make bookings at their preferred hospital and conveniently track the status of their request. The user's request is then seamlessly relayed to the Hospital Admin interface for further processing.

1.1.2 Hospital Admin Interface

The Hospital Admin interface empowers authorized personnel to manage and respond to user requests effectively. Hospital Admins can log in securely using their passwords and select their respective state and region. Leveraging the FourSquare API, the system dynamically generates a list of hospitals within the specified radius. Upon login, admins gain access to a comprehensive dashboard displaying Pending Cases, Active Cases, and Completed Cases.

Within the Pending Cases section, Hospital Admins can review requests submitted by users and exercise the option to accept or decline them. Accepted requests progress to the Driver Assigning Page, where admins can choose from a pool of Active Drivers (Engaged and Non-Engaged) or identify Inactive Drivers. The selected driver is then assigned the transport request, initiating the subsequent phase of the process.

1.1.3 Ambulance Driver Interface

Ambulance Drivers interact with CareConnect through their dedicated interface, which allows them to efficiently manage and execute transport requests. Similar to the previous interfaces, drivers can select their state and region, which triggers the retrieval of a relevant list of hospitals using the FourSquare API. After authenticating themselves with a password, drivers have the ability to set their status as Active or Sleep. When in Active mode, drivers receive transport requests from Hospital Admins and can initiate the trip promptly. Upon completing a trip, drivers can simply click the finish button to update the system accordingly.

1.2 Existing System

Existing systems that provide online ambulance dispatch services typically offer features like user authentication, location-based hospital selection, request tracking, and driver allocation. However, it's important to note that the specific implementations may vary.

Some potential shortcomings of existing systems in this domain could include:

1. **Lack of real-time data synchronization:** If the systems don't have efficient real-time data synchronization mechanisms, it may result in delays or discrepancies in tracking the status of requests or availability of drivers. Users, hospital admins, and drivers may not have the most up-to-date information, leading to potential inefficiencies.

2. **Poor user experience:** A user-friendly interface and intuitive user experience are crucial for the success of any online system. Existing systems may suffer from complex navigation, confusing workflows, or lack of responsiveness, which can negatively impact user satisfaction and adoption.

3. **Limited integration with external services:** While FourSquare API and Vonage API are utilized in your project, existing systems may lack integration with other essential services or APIs. This can restrict the functionality or limit the potential for expanding the system's capabilities in the future.

1.3 Problem Statement

The challenge is to create a web application which will provide assistance to patients in reaching their respective hospitals.

The existing emergency medical response systems often face challenges in efficiently dispatching ambulances to critical care situations. There is a need for an online platform that connects users in need of emergency medical assistance with available ambulances and hospitals within a specific radius. The platform should facilitate seamless communication and coordination between users, hospital administrators, and ambulance drivers to ensure timely and effective care.

1.4 Objectives

- **User Module**- The user module aims to provide a seamless experience for users accessing the CareConnect platform. Users will be able to log in using their phone numbers and verify their identity using the Vonage API for OTP services. After successful verification, users will be prompted to select their state and region. Using the FourSquare API, the system will display a list of hospitals within a 50 to 60 km radius of the selected region. Users can then choose their desired hospital and proceed to book it. Additionally, the system will enable users to track the status of their booking, ensuring transparency and real-time updates. User requests will be seamlessly passed to the Hospital Admin interface for further processing.

- **Hospital Admin Module** - The Hospital Admin interface serves as a control panel for hospital administrators, allowing them to efficiently manage incoming requests and allocate ambulance resources. Hospital Admins will have the ability to select their respective state and region, which will trigger the FourSquare API to retrieve a list of nearby hospitals. Upon logging in with a password, the Hospital Admin will be directed to a dedicated page displaying pending, active, and completed cases. By clicking on the pending cases, Hospital Admins can review requests from users and make informed decisions by accepting or declining them. Upon acceptance, the system will direct the Hospital Admin to the Driver Assigning Page, which presents

options for active drivers (engaged and non-engaged) as well as inactive drivers. From the available active (non-engaged) drivers, the Hospital Admin can select an appropriate driver for assignment. Once assigned, the request will be forwarded to the selected driver's interface.

- **Ambulance Driver Module** -The driver interface caters to the needs of ambulance drivers, ensuring efficient dispatching and seamless trip management. Similar to the previous interfaces, drivers can select their state and region, triggering the retrieval of a list of hospitals using the FourSquare API. After logging in with their password, drivers can set their status as either "Active" or "Sleep." In "Active" mode, drivers will be able to receive trip requests from the Hospital Admin interface. Upon accepting a request, drivers can initiate the trip and provide critical care transport services to the assigned hospital. Once the trip is completed, drivers can indicate the conclusion of the journey by clicking the finish button, allowing the system to update the status accordingly.

- **Technology Stack** - To build the CareConnect system, you have utilized MongoDB as the database for storing and retrieving data. ReactJS has been employed to create the user interfaces, providing an interactive and user-friendly experience. ExpressJS and NodeJS have been used to develop the backend server, facilitating the communication between the different interfaces and handling various operations, such as authentication, data retrieval, and request management.

1.5 Scope

The CareConnect project aims to develop an online ambulance dispatch website called Critical Care Transport System. The system consists of three main interfaces: User, Hospital Admin, and Ambulance Driver.

In the User module, users can log in using their phone numbers and verify their identity through OTP using the Vonage API. Once verified, users can select their state and region. Based on the selected region within a radius of 50 to 60 km, the system utilizes the FourSquare API to provide a list of hospitals. Users can choose and book the required hospital from the provided list. The system also allows users to track the status

of their booking, ensuring transparency and convenience. User requests are then passed on to the Hospital Admin interface for further processing.

The Hospital Admin interface enables Hospital Administrators to select their state and region. Similar to the User module, the system employs the FourSquare API to present a list of hospitals within the designated radius. Hospital Admins can log in using a password and are directed to a page displaying Pending Cases, Active Cases, and Completed Cases. By selecting the Pending Cases option, Hospital Admins can view the pending requests from users and choose to accept or decline them. Upon acceptance, they are directed to the Driver Assigning Page, where they can choose from a list of non-engaged Active Drivers. The selected driver is then assigned the request, and the information is passed on to the Driver interface.

In the Driver interface, drivers have the option to select their state and region. Similar to the previous interfaces, the FourSquare API is used to provide a list of hospitals within the specified radius. Drivers can log in using their passwords and set their status as Active or Sleep. When in Active mode, drivers can receive trip requests from the Hospital Admin. Upon accepting a request, they can start the trip and, once completed, mark it as finished by clicking the appropriate button.

Overall, the CareConnect project aims to streamline the process of dispatching critical care transport by connecting users, Hospital Administrators, and Ambulance Drivers through an intuitive online platform. It enhances the accessibility and efficiency of ambulance services, ensuring timely assistance and improving the overall healthcare experience for users in need.

Chapter 2

Literature Review

2.1 FourSquare Places API

Endpoints: The Places API lets you bring location context into your applications with the speed and reliability you need. Power location searches with rich details to help users find the best places near them through access to photos, reviews, tips, and more.

Place Search: Search for places in the Foursquare Places database using a location and querying by name, category name, telephone number, taste label, or chain name. For example, search for "coffee" to get back a list of recommended coffee shops. You may pass a location with your request by using one of the following options. If none of the following options are passed, Place Search defaults to geolocation using IP biasing with the optional radius parameter.

QUERY PARAMS:

query (string): A string to be matched against all content for this place, including but not limited to venue name, category, telephone number, taste, and tips. ll (string): The latitude/longitude around which to retrieve place information. This must be specified as latitude,longitude (e.g., ll=41.8781,-87.6298). radius (int32): Sets a radius distance (in meters) used to define an area to bias search results. The maximum allowed radius is 100,000 meters. Radius can be used in combination with ll or IP-biased geolocation only. By using radius, global search results will be omitted. categories (string): Filters the response and returns Foursquare Places matching the specified categories. Supports multiple Category IDs, separated by commas. chains (string): Filters the response and returns Foursquare Places matching the specified chains. Supports multiple chain IDs, separated by commas. exclude-chains (string): Filters the response and returns Foursquare Places not matching any of the specified chains. Supports multiple chain IDs, separated by commas. min-price (int32): Restricts results to only those places within the specified price

range. Valid values range between 1 (most affordable) to 4 (most expensive), inclusive. max-price (int32): Restricts results to only those places within the specified price range. Valid values range between 1 (most affordable) to 4 (most expensive), inclusive. open-at (string): Support local day and local time requests through this parameter. To be specified as DOWTHHMM (e.g., 1T2130), where DOW is the day number 1-7 (Monday = 1, Sunday = 7) and time is in 24-hour format. (No comments about seconds) Places that do not have opening hours will not be returned if this parameter is specified. open-now (boolean): Restricts results to only those places that are open now. Places that do not have opening hours will not be returned if this parameter is specified. ne (string): The latitude/longitude representing the north/east points of a rectangle. Must be used with the sw parameter to specify a rectangular search box. Global search results will be omitted. sw (string): The latitude/longitude representing the south/west points of a rectangle. Must be used with the ne parameter to specify a rectangular search box. Global search results will be omitted. near (string): A string naming a locality in the world (e.g., "Chicago, IL"). If the value is not geocodable, returns an error. polygon (string): A string containing the list of coordinates (separated with a comma) that define the edges of the polygon. Must have at least 4 coordinates and be considered a "closed" polygon. sort (string): Specifies the order in which results are listed. Possible values are: relevance (default), rating, distance. limit (int32): The number of results to return, up to 50. Defaults to 10. session-token (string): A user-generated token to identify a session for billing purposes. Learn more about session tokens. Get Place Details: Retrieve comprehensive information and metadata for a Foursquare Place using the fsq-id.

PATH PARAMS:

fsq-id (string, required): A unique string identifier for a Foursquare Place (formerly known as Venue ID). E.g., Foursquare HQ's fsq-id = 5a187743ccad6b307315e6fe QUERY PARAMS:

fields (string): Indicate which fields to return in the response, separated by commas. If no fields are specified, all Core Fields are returned by default. For a complete list of returnable fields, refer to the Response Fields page. session-token (string): A user-generated token to identify a session for billing purposes. Learn more about session tokens. Find Nearby Places: Utilize Foursquare's Snap to Place technology to detect where your user's device is and what is around them. The endpoint is designed to provide

POI tagging that supports a check-in use case. It will intentionally return lower-quality results not found in Place Search. It is not intended to replace Place Search as the primary way to search for top, recommended POIs.

QUERY PARAMS:

Indicate which fields to return in the response, separated by commas. If no fields are specified, all Core Fields are returned by default. For a complete list of returnable fields, refer to the Response Fields page. The latitude/longitude around which to retrieve place information. This must be specified as latitude,longitude (e.g., ll=41.8781,-87.6298). If you do not specify ll, the server will attempt to retrieve the IP address from the request and geolocate that IP address. A string to be matched against the venue name for this place. The number of results to return, up to 50. Defaults to 10. Authentication: The Places API uses API Keys to authenticate requests. Authenticating against the API is done by passing your API Key through an Authorization header parameter (see example headers).

Rate Limits: These rate limits pertain solely to the most current version of the Places API. If you are using the previous version of the Places API (which includes any end-points beginning with /v2/), please refer to our Rate Limits guide that outlines the rate limitations applicable to that version. Usage of the Places API is subject to rate limits.

Algorithms Used: While the exact algorithms used by Foursquare are not known, location-based APIs, in general, often rely on techniques such as:

Geocoding and Reverse Geocoding: Converting between addresses and geographic coordinates (latitude and longitude) to facilitate location-based searches.

Spatial Indexing and Data Structures: Efficiently organizing and indexing geographic data to quickly retrieve relevant information based on the proximity to a given location.

Ranking and Recommendation Algorithms: Determining the relevance and ranking of places based on factors like user reviews, popularity, proximity, and other relevant attributes.

Clustering and Categorization: Grouping similar places together based on shared characteristics or categories.

Distance and Proximity Calculations: Computing distances between two geographic points to identify places within a specific radius of a given location.

Real-Time Data Processing: Handling real-time updates and dynamic changes in the

dataset to keep the API information up to date.

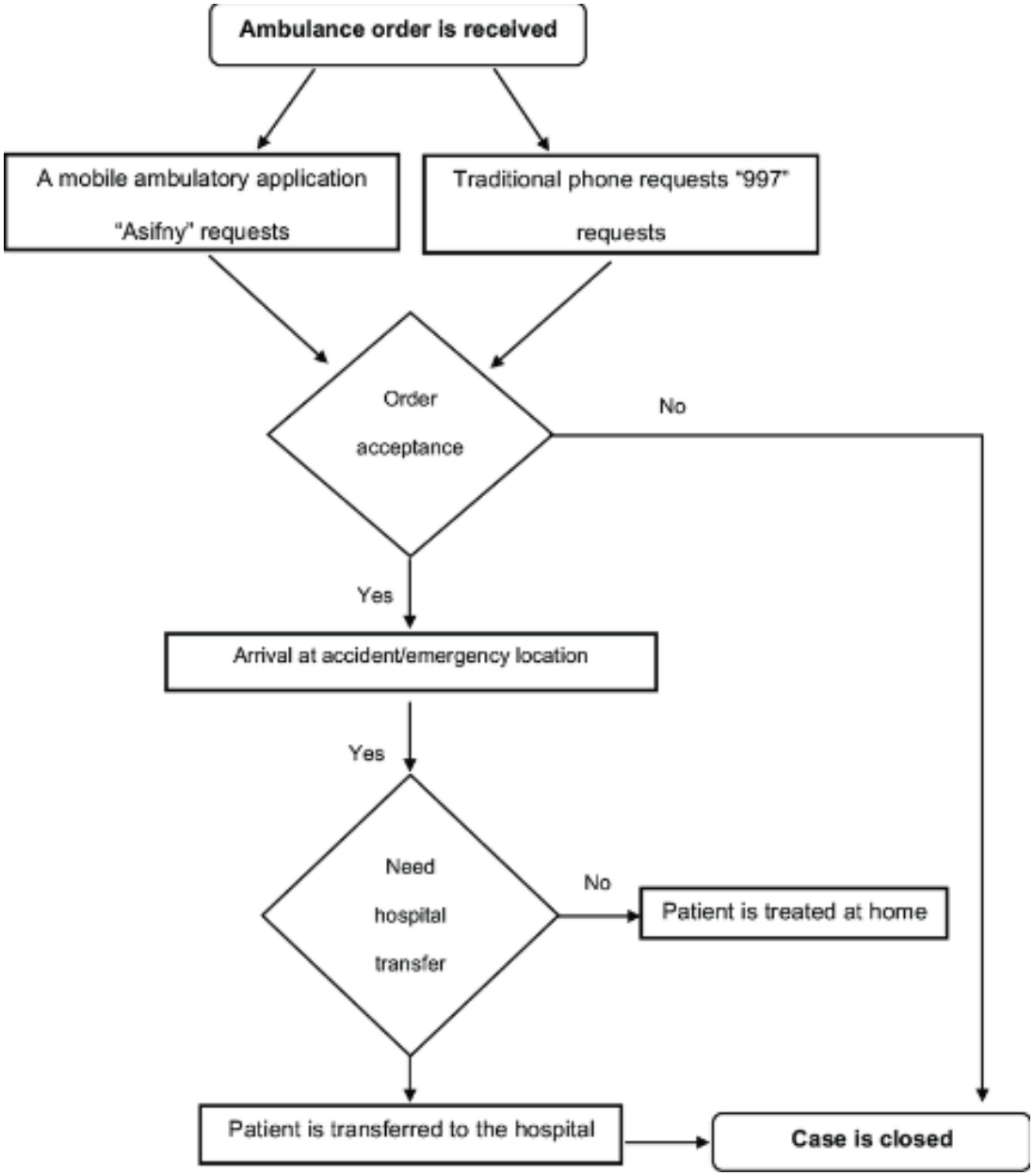

Figure 2.1: An existing Ambulance booking system

2.2 Vonage Messaging API

Messages API Technical Details The Messages API allows you to send and in some cases receive messages over SMS, MMS, Facebook Messenger, Viber, and WhatsApp. Further channels may be supported in the future.Major US carriers have announced their requirements for a new standard for application-to-person (A2P) messaging in the USA,

which applies to all messaging over 10-digit geographic phone numbers, also known as 10 DLC. If you are or are planning to send SMS/MMS traffic from a +1 Country Code 10 Digit Long Code into US networks, you will need to register a brand and campaign in order to get approval for sending messages. See the 10 DLC documentation for details.

The following diagram illustrates how the Vonage Messages API enables you to send messages via multiple channels from a single endpoint:

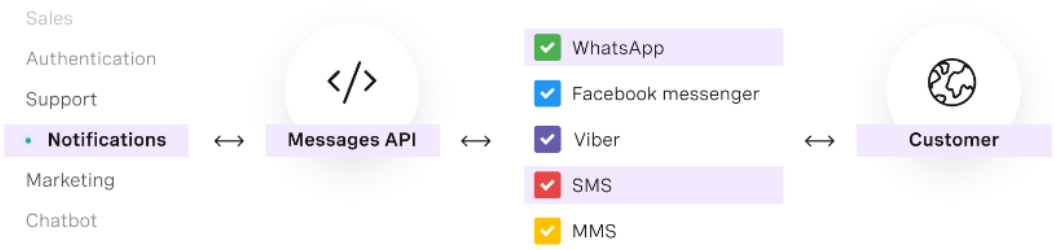

Figure 2.2: Working of Vonage API

Comparison

The existing methods has less features. We have come up with features of selecting hospitals and managed drivers efficiently.

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is a smart phone is expected to have the following features:

- Sufficient bandwidth allocation to accommodate the expected website traffic and data transfer requirements.
- Requirement of Internet connection for working of our website.
- A mobile phone to recieve OTP.
- A minimum Ram size of 4GB RAM in the device.

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The project is technically feasible since majority of the population are in possession of mobile phones. The website only requires minimum requirements to run on a device.

3.2.2 Operational Feasibility

The operations are built in a simple and easy to use manner for geriatric people, considering the availability of resources, compatibility with existing infrastructure, and stakeholder acceptance.

3.2.3 Economic Feasibility

The app can reduce the overhead of expense incurred by geriatric people in order to maintain physical assets essential for them to interact with society. The development of the app is also zero budget as it was built using free resources.

3.3 Hardware Requirements

The following are the system requirements to develop our Care Connect website.

- Processor: Intel Core i5
- Hard Disk: Minimum 50GB
- RAM: Minimum 8GB

3.4 Software Requirements

The following are the softwares used in the development of the app.

Operating System: Windows or Linux

3.4.1 Mongo Db

MongoDB is a NoSQL document-oriented database that can be utilized in your Critical Care Transport System project. MongoDB offers a flexible and scalable data storage solution, allowing you to store and retrieve data in a JSON-like format called BSON. It supports dynamic schema, making it suitable for evolving data structures, and provides powerful querying and indexing capabilities for efficient data retrieval. MongoDB's scalability and replication features can ensure high availability and performance as your project grows.

3.4.2 Express JS

Express.js is a popular web application framework that can be utilized in your Critical Care Transport System project. It is built on top of Node.js and provides a minimalist and flexible approach for developing web applications. Express.js simplifies the process of handling HTTP requests, routing, and middleware integration. With its modular architecture, you can easily add functionalities and middleware to handle authentication, session management, and data validation. Express.js enables efficient development by providing a robust set of tools and utilities, making it well-suited for building the backend of your web application.

3.4.3 React JS

React.js is a JavaScript library that can be utilized in your Critical Care Transport System project to build interactive and user-friendly front-end interfaces. React.js follows a component-based approach, allowing you to create reusable UI components and efficiently manage the application's state. It provides a virtual DOM (Document Object Model) for efficient rendering and updates, resulting in improved performance. React.js enables seamless integration with other libraries and frameworks, making it flexible for building complex and dynamic user interfaces. With its declarative syntax and extensive ecosystem, React.js facilitates rapid development and enhances the overall user experience of your web application.

3.4.4 Node JS

Node.js is a JavaScript runtime environment that can be utilized in your Critical Care Transport System project for server-side development. It allows you to run JavaScript code outside the browser, making it ideal for building scalable and high-performance web applications. Node.js leverages an event-driven, non-blocking I/O model, which enables handling a large number of concurrent requests efficiently. It provides a rich ecosystem of modules and packages through npm (Node Package Manager), facilitating rapid development by leveraging existing libraries and tools. With its lightweight and efficient architecture, Node.js is well-suited for building the backend of your web application, handling data processing, API integrations, and real-time communication.

3.4.5 Embedded JS

EJS is a template engine that can be used with Node.js and Express.js in your Critical Care Transport System project. EJS allows you to embed JavaScript code within HTML templates, making it easier to dynamically generate and render HTML content. It enables you to create reusable templates and inject dynamic data into them during runtime. With EJS, you can build dynamic web pages, display data from the server, and implement conditional logic or loops within your HTML templates. It enhances the flexibility and maintainability of your front-end code by separating presentation and logic.

Chapter 4

Methodology

4.1 Proposed Method

- Develop a web application that help users to book an ambulance.
- We plan to build individual interfaces for User,Hospital Admin and Driver.Each of these interfaces have a different login setup.
- User can login through their phone number and verify it through OTP for that we have used a free API Service called Vonage API.
- Application also contains features like showing nearest hospitals in the locality,Driver can set their status as Active/Sleep and the Hospital Admin can Accept or Decline the user request.

4.2 Procedure

The CareConnect system utilizes a combination of frontend, backend, and database components to facilitate its operations. Here's a breakdown of how the system works internally:

4.2.1 Frontend:

The frontend of the CareConnect system is built using ReactJS, a popular JavaScript library for building user interfaces. It consists of various components and pages that provide the user interface for different modules, including the user module, hospital admin module, and ambulance driver module. The frontend communicates with the backend server via API calls to fetch data and update the system.

4.2.2 Backend:

The backend of the CareConnect system is developed using Node.js and Express.js. It handles the logic and processes the requests from the frontend. The backend includes the following components:

- API Endpoints: The backend exposes various API endpoints that are consumed by the frontend for user registration, login, hospital selection, booking, and tracking. These endpoints handle the incoming requests, validate inputs, and interact with the database.

- User Management: The backend manages user authentication, registration, and login. It verifies the provided phone number and OTP using the Vonage API for secure authentication. User information is stored securely in the database.

- Hospital and Case Management: The backend facilitates hospital and case management for hospital administrators. It fetches hospital data from the FourSquare API based on the user's location. It also handles the creation, update, and tracking of cases, including pending, active, and completed cases.

- Driver Assignment: The backend enables hospital administrators to assign drivers to active cases. It presents a list of available drivers and allows administrators to select and assign them to specific cases. It updates the case status accordingly.

4.2.3 Database:

The CareConnect system uses MongoDB, a NoSQL database, to store and manage system data. The database stores information such as user details, hospital data, case details, and driver assignments. It provides a structured and efficient way to store and retrieve data necessary for the system's operations. The backend communicates with the database to fetch and update relevant information as needed.

4.2.4 External APIs:

The CareConnect system integrates with external APIs, such as the Vonage API for OTP verification and the FourSquare API for hospital location services. These APIs are utilized

for specific functionalities, such as verifying user phone numbers during registration and fetching nearby hospitals based on user location. The backend handles the communication with these APIs to retrieve the required data.

4.2.5 Workflow:

The workflow of the CareConnect system involves the following steps:

- Users register and log in to the system.
- Users select their desired hospitals based on location.
- Users book emergency medical transportation, providing relevant details.
- Hospital administrators receive pending cases and can accept or decline requests.
- Hospital administrators assign available drivers to active cases.
- Drivers log in and set their availability status.
- Drivers receive trip requests and can accept or decline them.

Throughout the workflow, the frontend interacts with the backend by making API calls to fetch and update data. The backend processes these requests, updates the database, and communicates with external APIs as necessary.

By effectively integrating the frontend, backend, database, and external APIs, the CareConnect system functions smoothly to facilitate the online ambulance dispatch process, ensuring efficient emergency medical transportation for users.

Chapter 5

System Design

5.1 Architecture Diagram

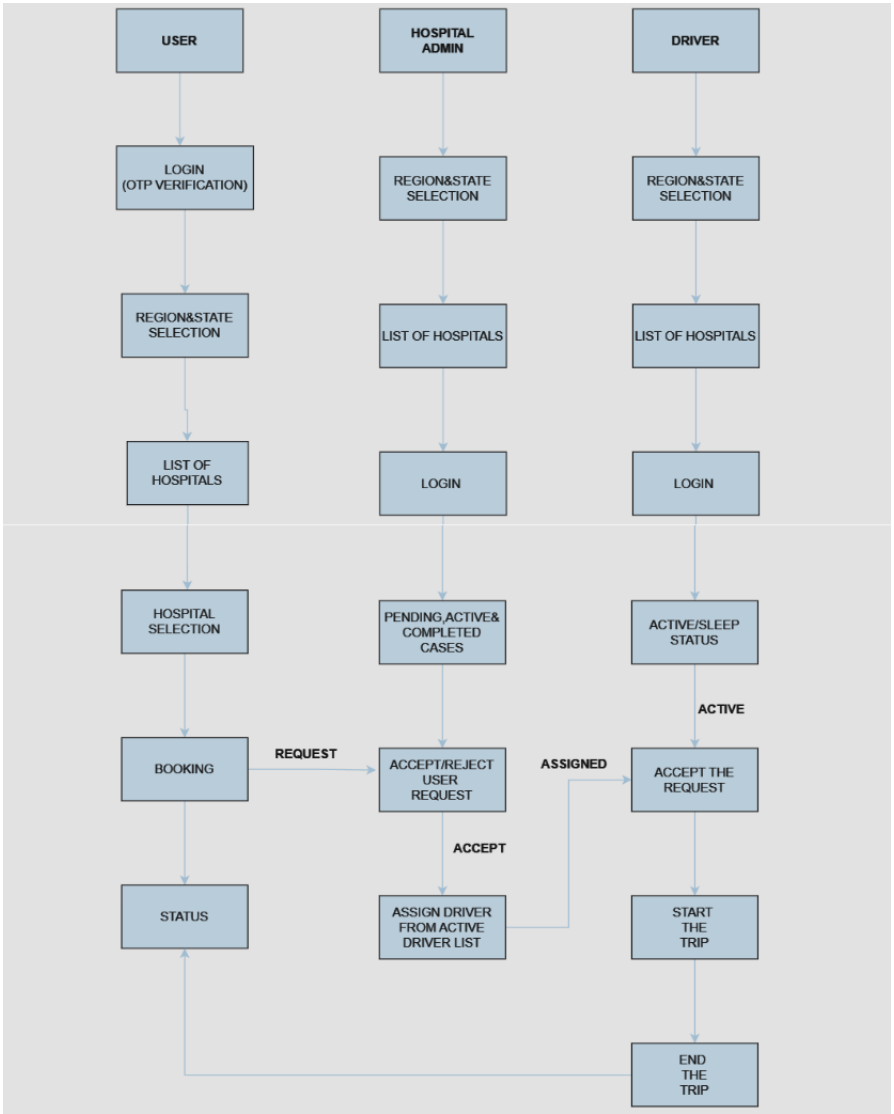

Figure 5.1: Architecture diagram

5.2 Sequence diagram

5.2.1 User Module

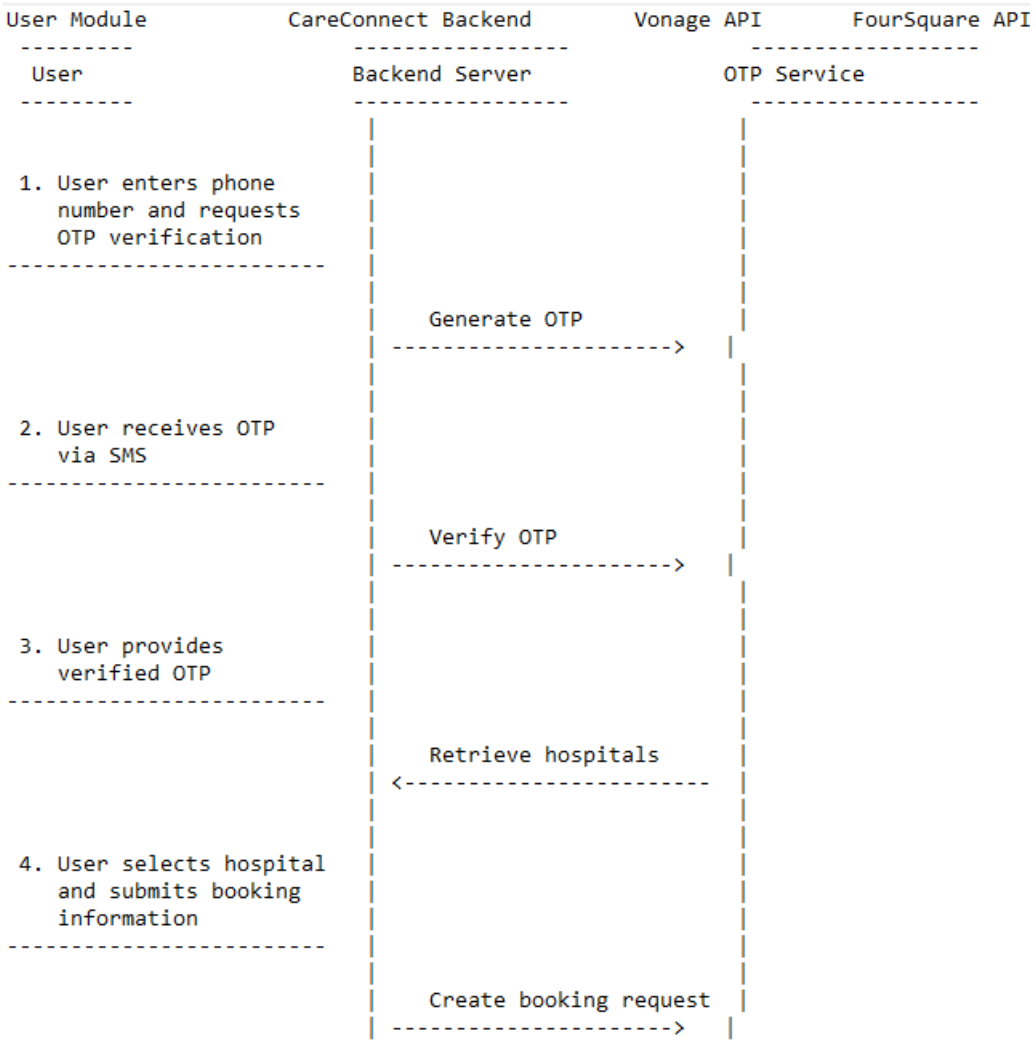

Figure 5.2: User Module

5.2.2 Hospital Admin Module

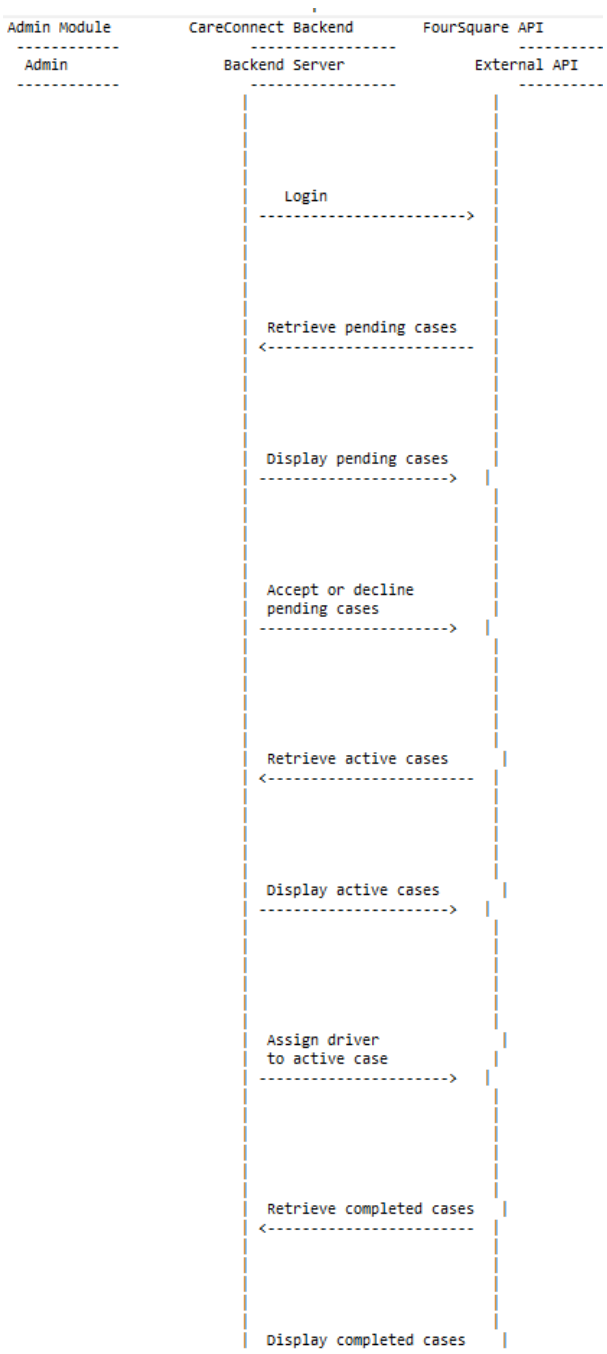

Figure 5.3: Hospital Admin Module

5.2.3 Driver Module

Figure 5.4: Driver Module

Chapter 6

System Implementation

6.1 Technology Stack

The CareConnect system is implemented using the following technologies: - Frontend: ReactJS for building the user interface and providing an interactive user experience. - Backend: Node.js and Express.js to develop the server-side logic and handle user requests. - Database: MongoDB for storing system data efficiently and facilitating data retrieval and management.

6.2 User Interface Implementation

The user interface is implemented using ReactJS. It provides users with a seamless experience for registration, login, hospital selection, booking, and tracking. The interface is designed to be intuitive, responsive, and user-friendly.

6.3 Backend Implementation

The backend server is developed using Node.js and Express.js. It handles user requests, communicates with the database, and integrates with external APIs for OTP verification and hospital location services. The backend implements the business logic of the CareConnect system and ensures data security and integrity.

6.4 Database Design and Implementation

The system data is stored in a MongoDB database. The database is designed to efficiently store user information, hospital details, case statuses, and driver assignments. The implementation includes the creation of appropriate collections, defining indexes for efficient querying, and ensuring data consistency.

6.5 User Module

6.5.1 User Registration

The user registration process allows users to create an account by providing their details, including phone number and basic personal information. Once the information is submitted, an OTP is sent to the provided phone number via the Vonage API for verification.

6.5.2 Login and Authentication

After successful registration, users can log in using their phone number and password. The system verifies the credentials and grants access to the user interface. Proper authentication mechanisms are implemented to ensure secure access.

6.5.3 Hospital Selection

Users can select their desired hospitals based on their state and region. The FourSquare API is utilized to fetch a list of hospitals within a specific radius. The hospitals are displayed to the user, who can choose from the available options.

6.5.4 Booking Hospital

Once a hospital is selected, users can book their required hospital for emergency medical transportation. The booking process involves submitting relevant details, such as the nature of the emergency and additional comments if necessary.

6.5.5 Tracking Booking Status

Users can track the status of their booking by accessing the system interface. They can view updates on the progress of their request, such as whether it is pending, accepted, or completed. Real-time notifications can be implemented to keep users informed about their booking status.

6.6 Hospital Admin Module

6.6.1 Login and Authentication

Hospital administrators can log in to the system using their credentials. Proper authentication mechanisms are implemented to ensure secure access to the admin interface.

6.6.2 Pending Cases

Hospital administrators can view and manage pending cases, which include requests from users awaiting acceptance. The admin interface displays relevant details, such as user information, nature of the emergency, and additional comments.

-

6.6.3 Active Cases

The active cases section provides hospital administrators with an overview of ongoing cases, including accepted requests awaiting transportation. Administrators can track the progress of active cases and manage driver assignments.

6.6.4 Completed Cases

Hospital administrators can review and close completed cases, ensuring that all necessary information and updates are properly recorded. The completed cases section provides an overview of all successfully transported patients.

6.6.5 Driver Assignment

Hospital administrators can assign drivers to active cases based on availability and proximity to the user's location. The interface provides options to select drivers from lists of active and non-engaged drivers.

6.7 Ambulance Driver Module

6.7.1 Login and Authentication

Ambulance drivers can log in to the system using their credentials. Authentication mechanisms ensure secure access to the driver interface.

6.7.2 Setting Driver Status

Drivers can set their status as active or sleep mode to indicate their availability for new trip requests. The status update allows hospital administrators to assign available drivers accordingly.

6.7.3 Accepting Trip Requests

Drivers receive trip requests from hospital administrators through the driver interface. They can review the details of the trip, including the user's location and emergency information. Drivers have the option to accept or decline the trip request based on their availability and proximity.

6.7.4 Completing Trip

Once a driver accepts a trip request, they can navigate to the user's location and transport them to the designated hospital. After successfully completing the trip, the driver can mark it as finished in the driver interface.

Chapter 7

Testing

This testing report provides an overview of the testing activities conducted for the Ambulance Booking System. The purpose of the testing phase is to ensure that the system functions as intended, meets the specified requirements, and performs reliably under various scenarios. The testing process includes functional testing, performance testing, usability testing, and security testing.

7.1 Testing Objectives

The objectives of the testing phase are as follows:

-Verify the functionality of each module (User, Admin, Ambulance Driver) according to the requirements.

-Validate the system's performance and responsiveness under different loads.

-Evaluate the usability of the system's interfaces and interactions.

-Identify and address any security vulnerabilities or risks.

7.2 Testing Approaches

The following testing approaches were employed during the testing phase:

7.2.1 Functional Testing

Functional testing was performed to verify the correct behavior of the system's functionalities. Test cases were designed to cover all possible user interactions and system responses. The key functional areas tested include:

-User login

-Ambulance booking and tracking

-Admin management of users and ambulance services

-Ambulance driver interactions with booking requests

7.2.2 Performance Testing

Performance testing was conducted to evaluate the system's performance under expected and peak loads. The objectives were to identify any performance bottlenecks, measure response times, and ensure that the system can handle the anticipated user load without significant degradation. Performance testing scenarios included:

-Simulating concurrent user bookings and monitoring response times

-Stress testing the system with high volumes of simultaneous requests

-Load testing to assess the system's behavior under sustained user activity

7.2.3 Usability Testing

Usability testing aimed to evaluate the system's user interfaces, navigation, and overall user experience. A group of representative users participated in the testing process and provided feedback on the system's ease of use, clarity of instructions, and intuitiveness. Usability testing activities included:

-Task-based testing to assess the system's ease of completing common user tasks

-Feedback collection through surveys and interviews

-Iterative refinement of the user interfaces based on user feedback

7.2.4 Security Testing

Security testing was performed to identify and address any vulnerabilities or risks in the system. The objective was to ensure the confidentiality, integrity, and availability of user data. Security testing activities included:

-Penetration testing to identify potential security weaknesses

-Assessment of authentication and authorization mechanisms

-Encryption and secure transmission of sensitive data

-Handling of error conditions and prevention of information leakage

7.3 Testing Results

The testing phase produced the following key results:

-Functional Testing: All functional test cases were executed, and the system demonstrated correct behavior and adherence to the specified requirements. No critical defects were identified.

-Performance Testing: The system performed well under normal and peak loads, with response times within acceptable limits. No performance bottlenecks were discovered.

-Usability Testing: Users provided positive feedback regarding the system's ease of use and intuitive interfaces. Some minor suggestions for interface improvements were collected and addressed.

-Security Testing: The system passed security testing with no significant vulnerabilities or risks identified. Recommendations for further enhancing security were implemented.

7.4 Conclusion

The Ambulance Booking System successfully passed the testing phase, demonstrating its reliability, functionality, performance, usability, and security. The system met the specified requirements and provided a positive user experience. The testing process identified and addressed any issues, ensuring the system's readiness for deployment.

Chapter 8

Result

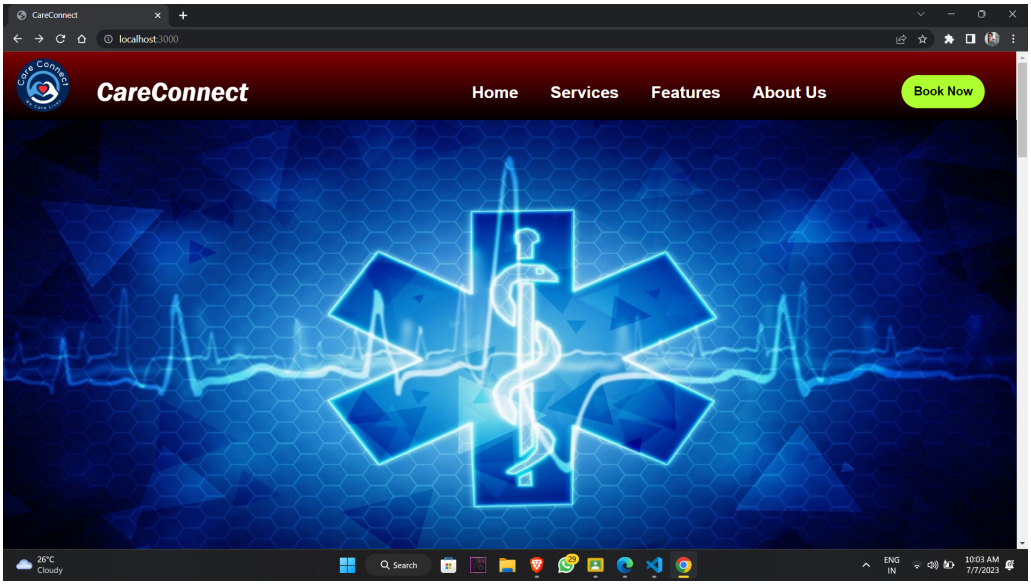

Figure 8.1: Main Menu interface

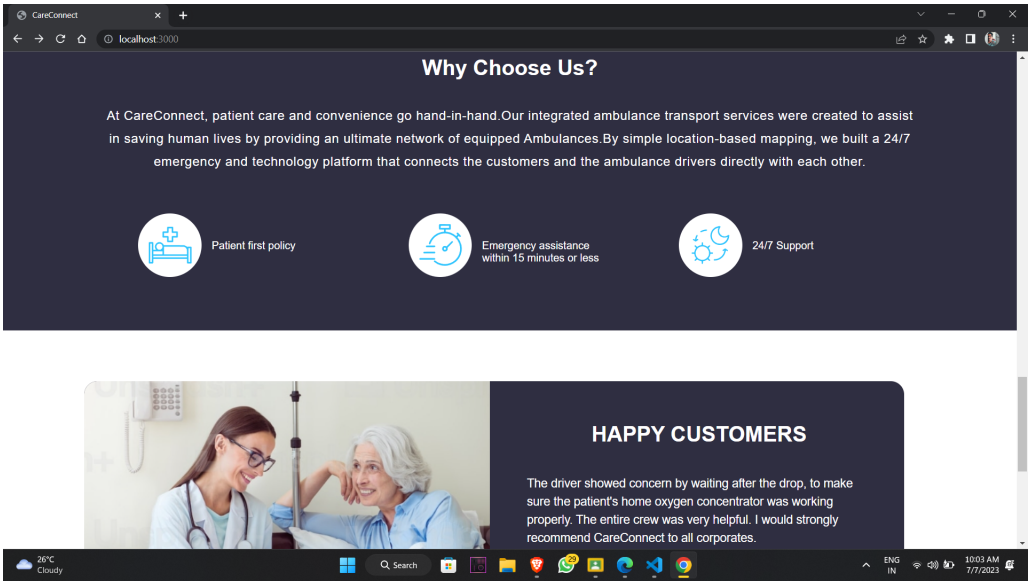

Figure 8.2: Main Menu interface

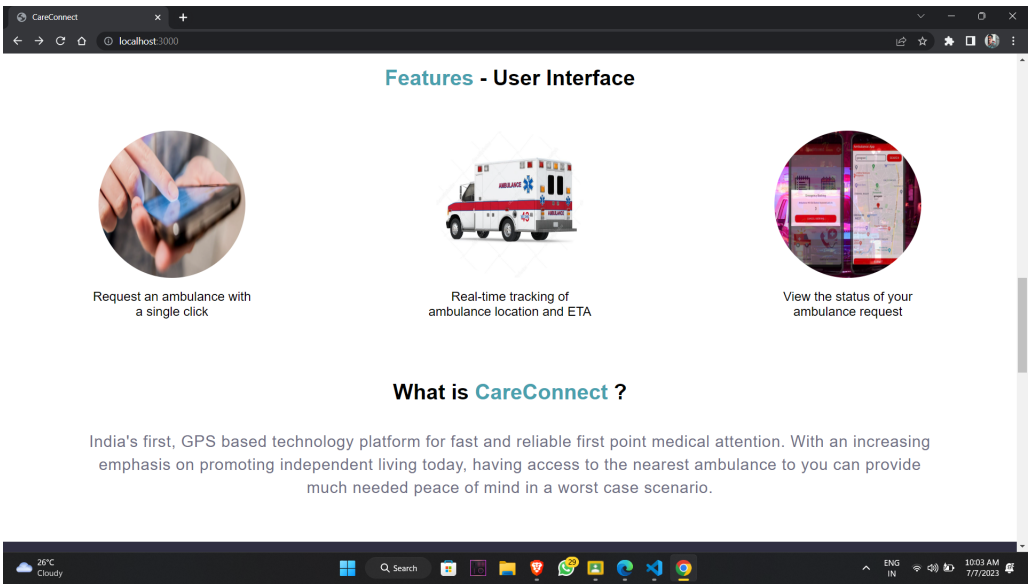

Figure 8.3: Main Menu interface

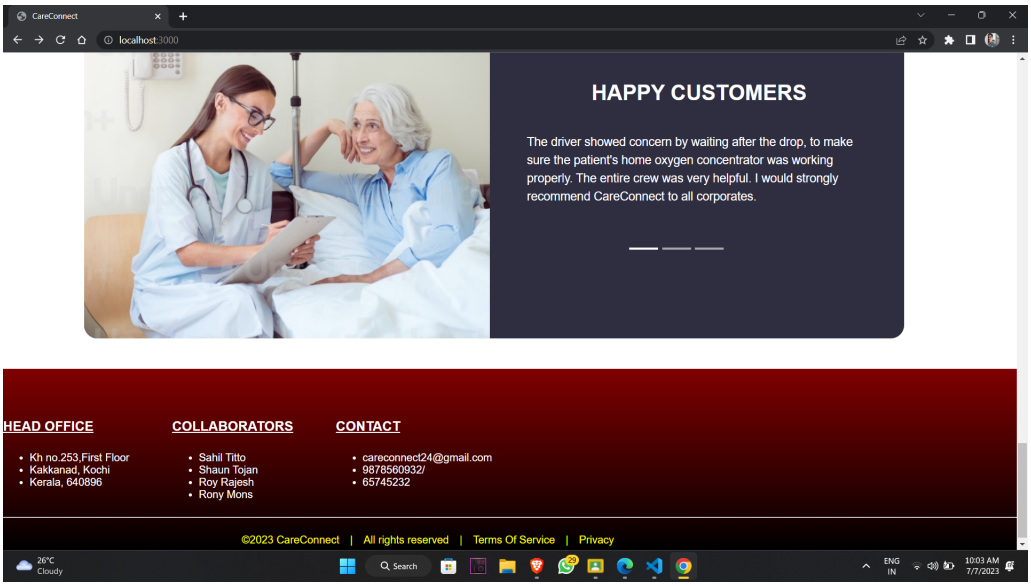

Figure 8.4: Main Menu interface

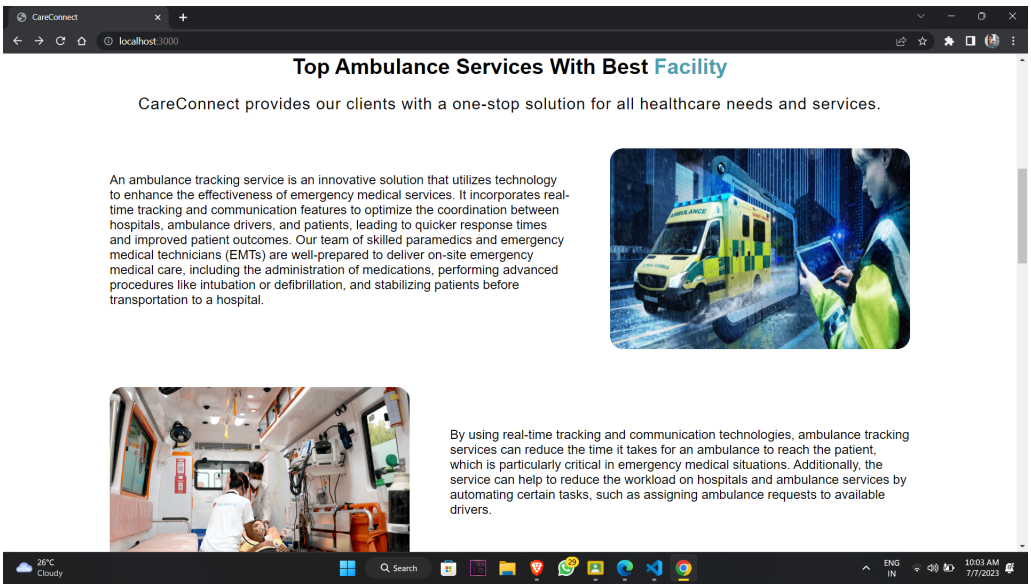

Figure 8.5: Main Menu interface

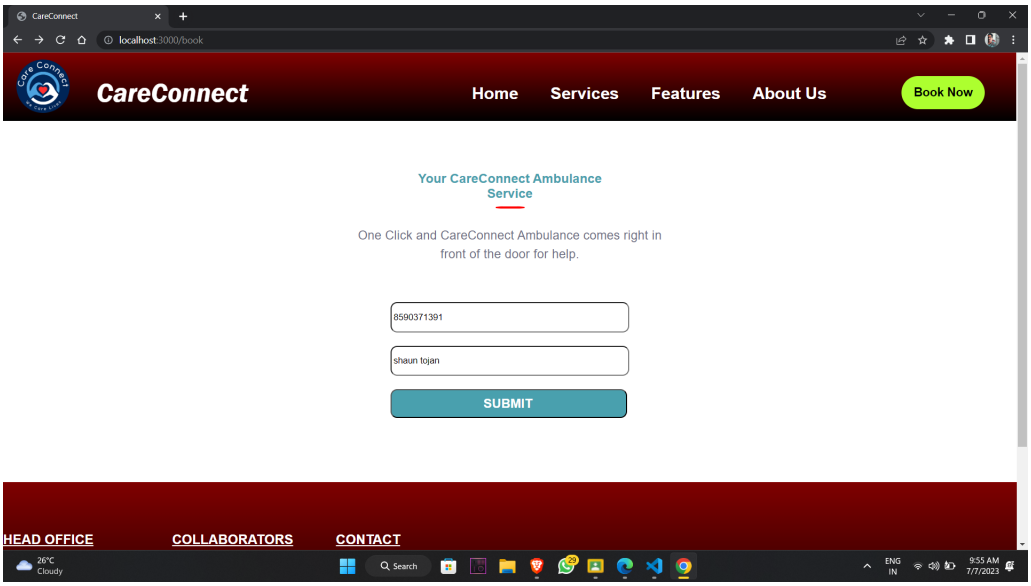

Figure 8.6: OTP Authentication

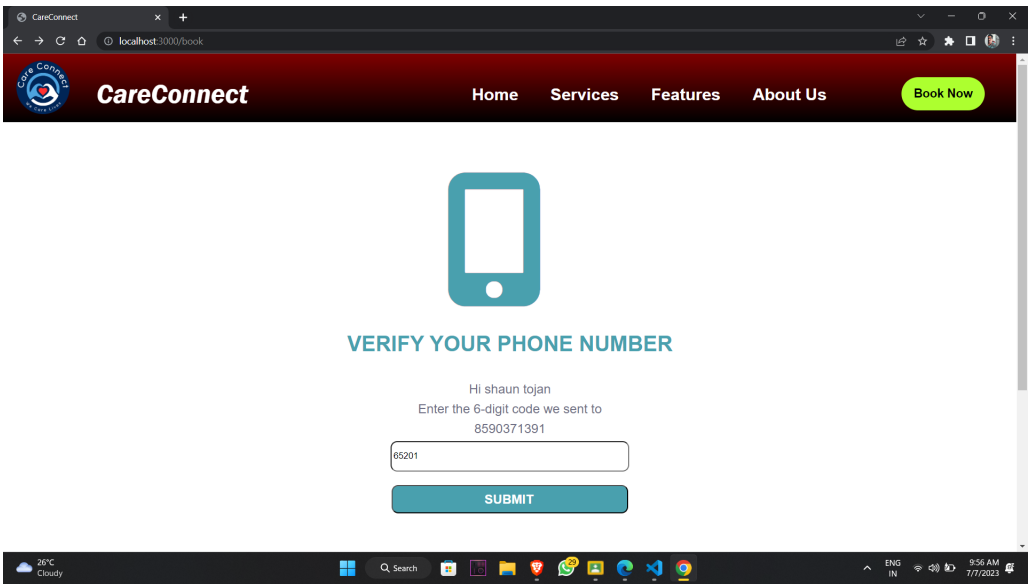

Figure 8.7: OTP Authentication

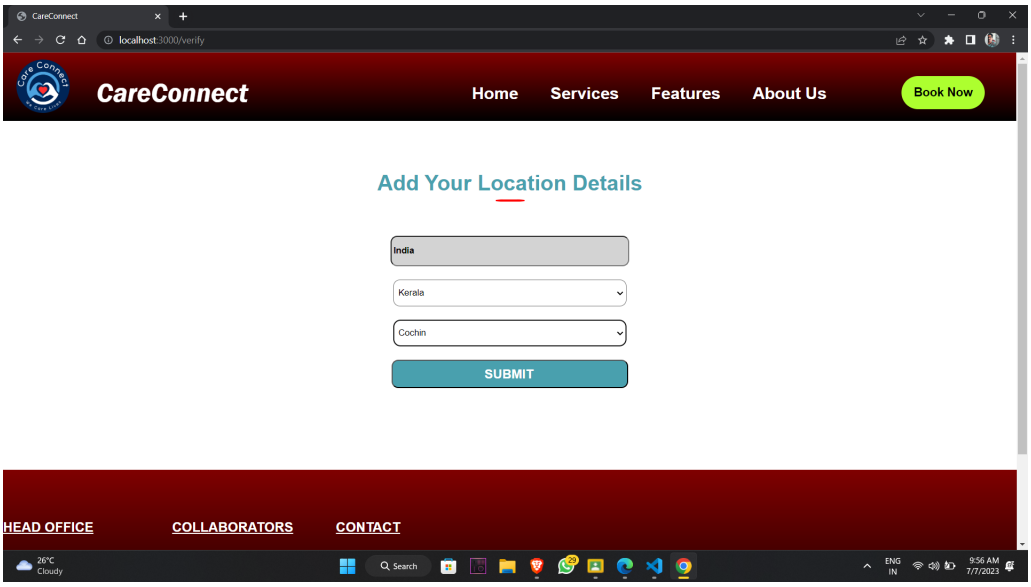

Figure 8.8: Location Details

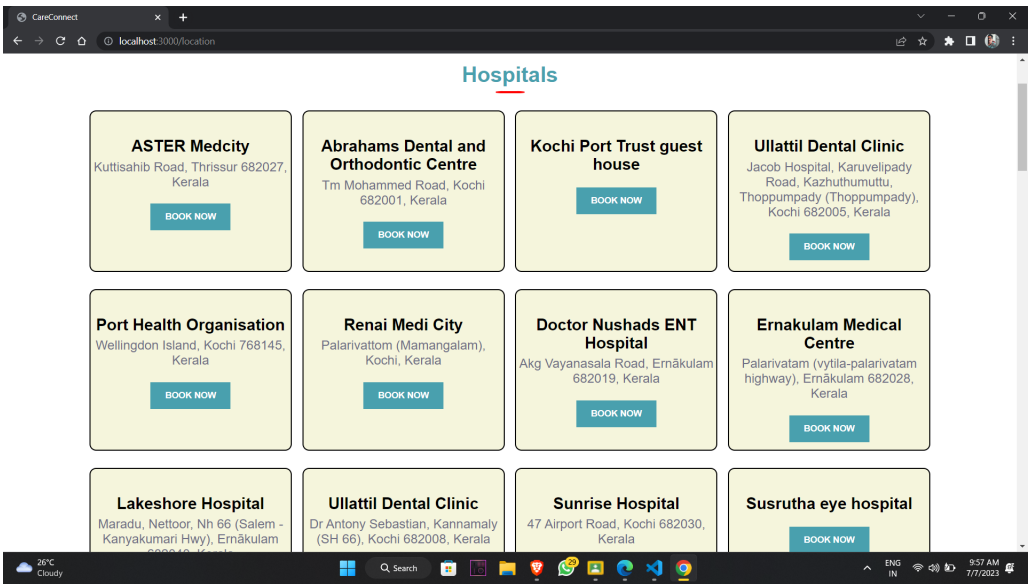

Figure 8.9: List of Hospitals

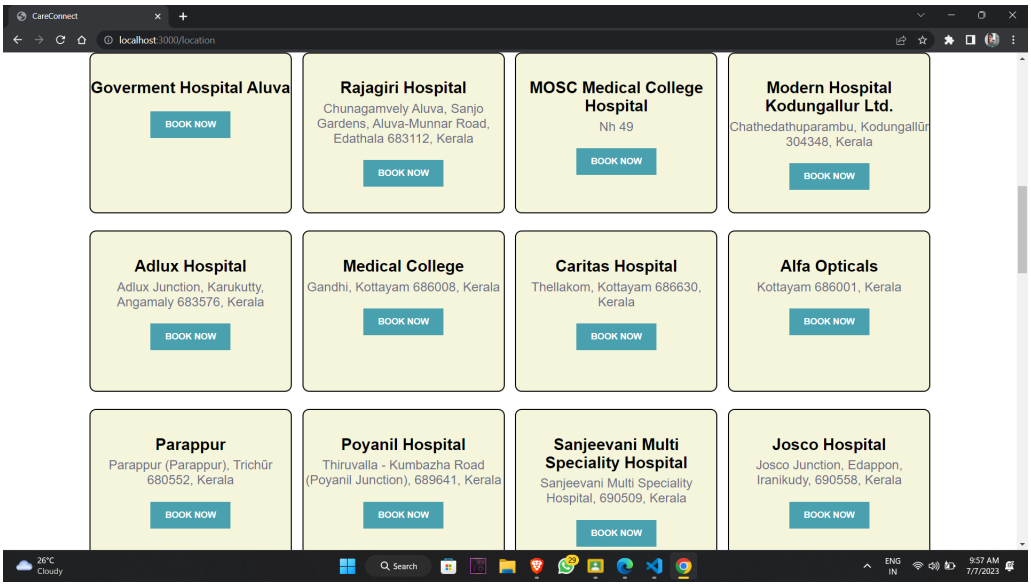

Figure 8.10: List of Hospitals

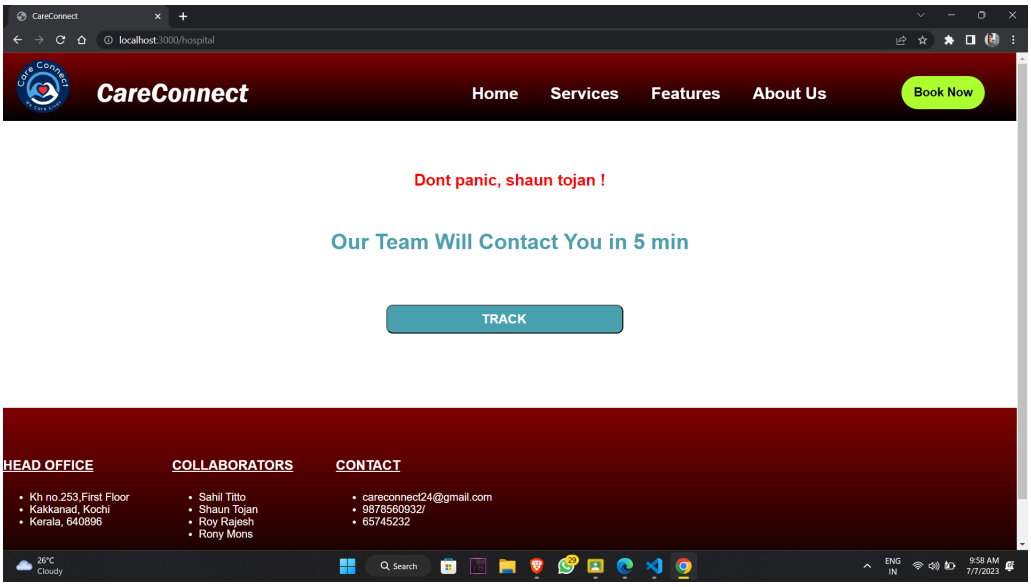

Figure 8.11: Booking Request Send

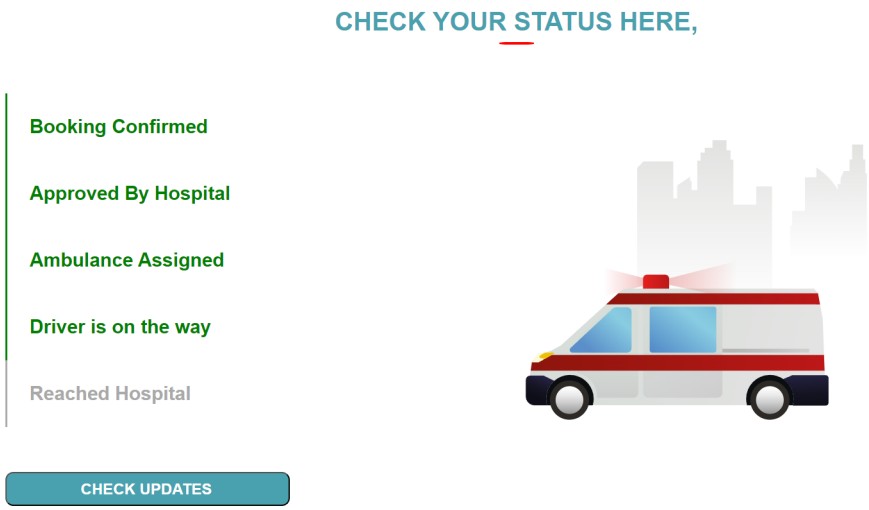

Figure 8.12: Booking Status

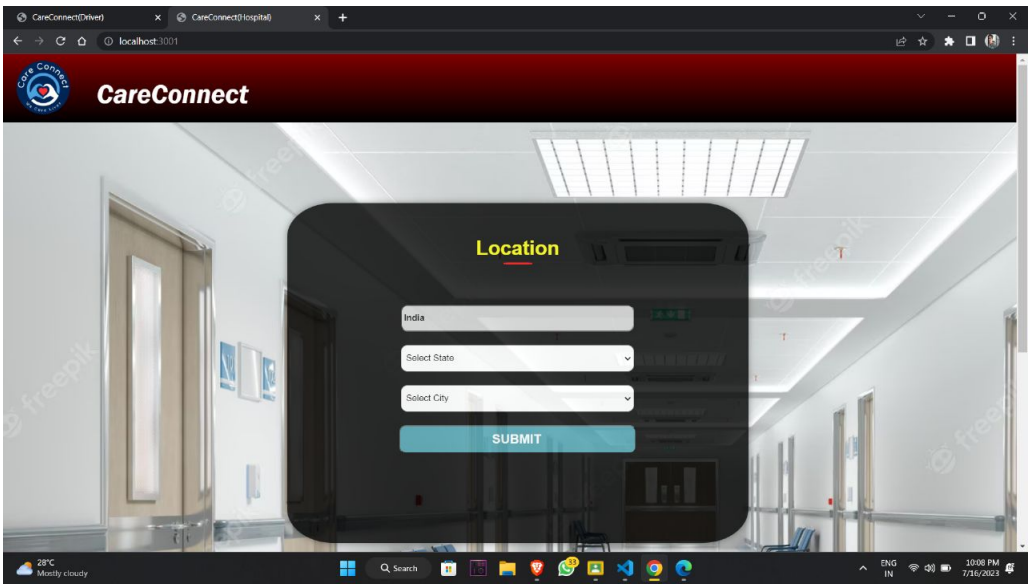

Figure 8.13: Location Selection

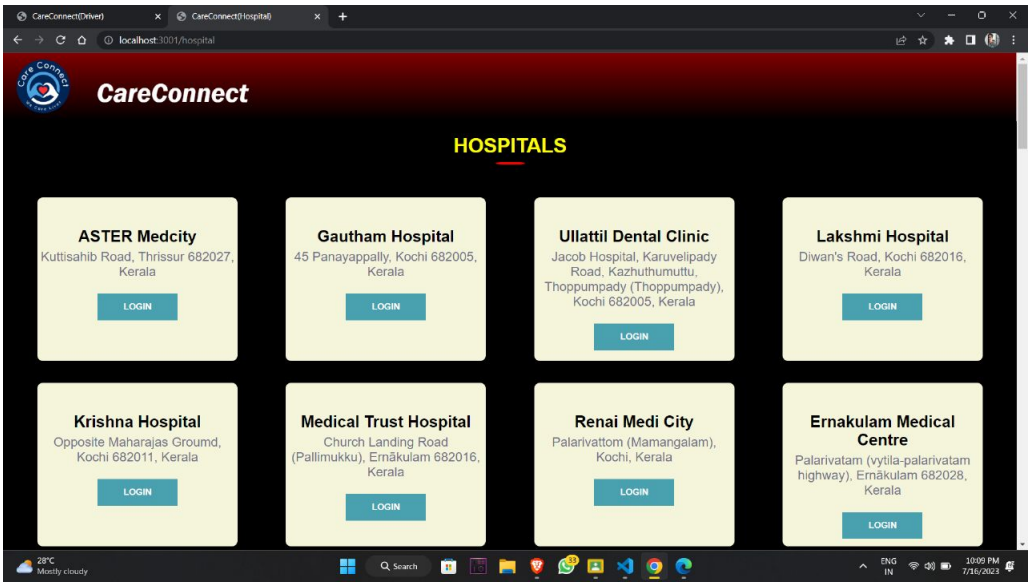

Figure 8.14: List of Hospitals

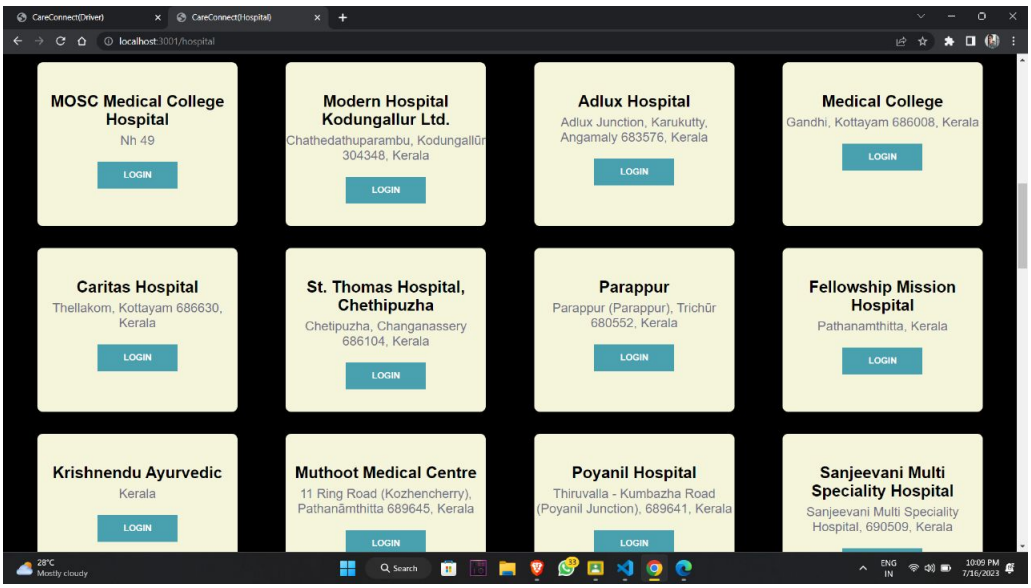

Figure 8.15: List of Hospitals

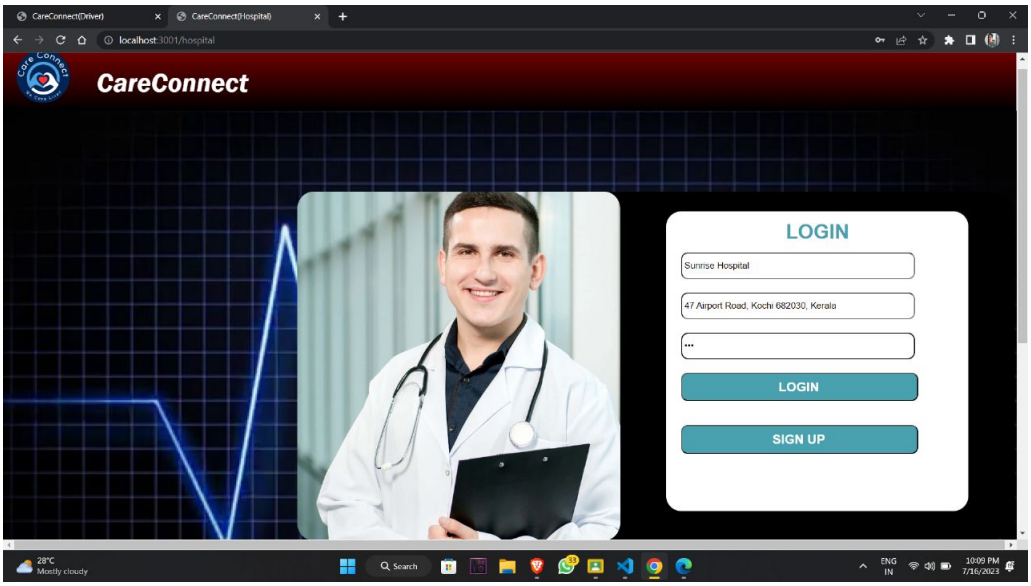

Figure 8.16: Hospital Admin Login

Figure 8.17: Status of cases

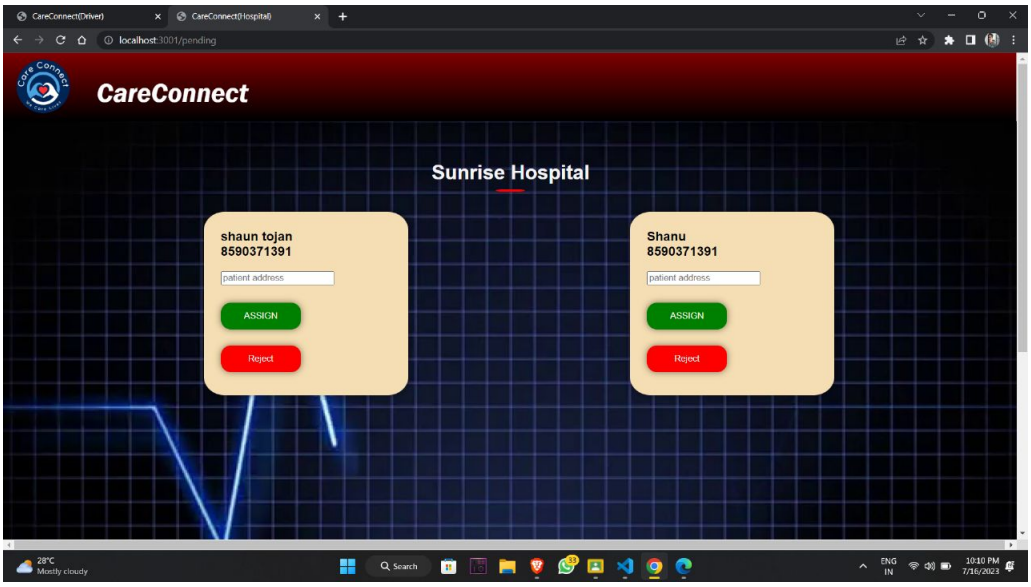

Figure 8.18: Assign/Reject Case

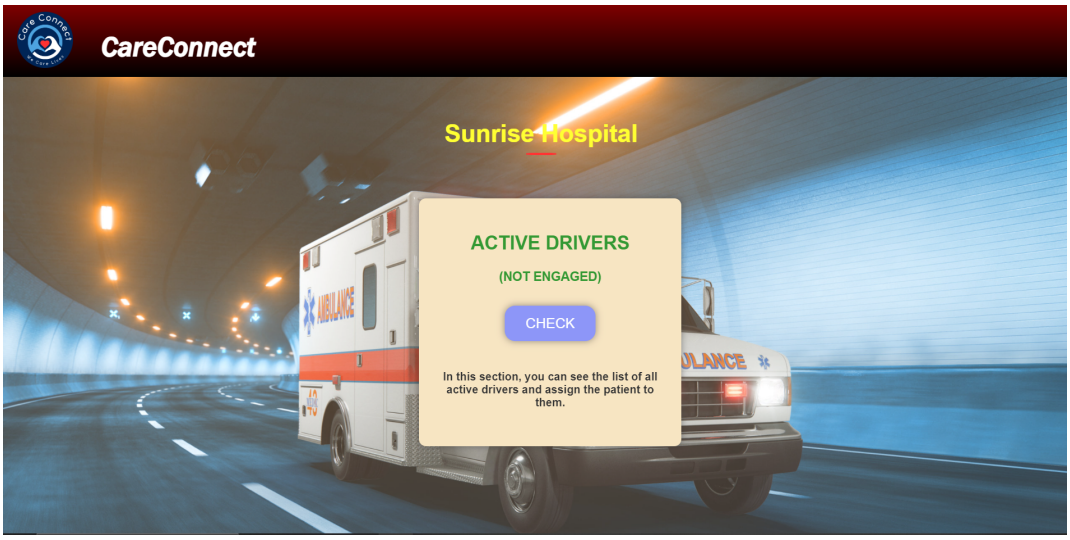

Figure 8.19: Driver Status

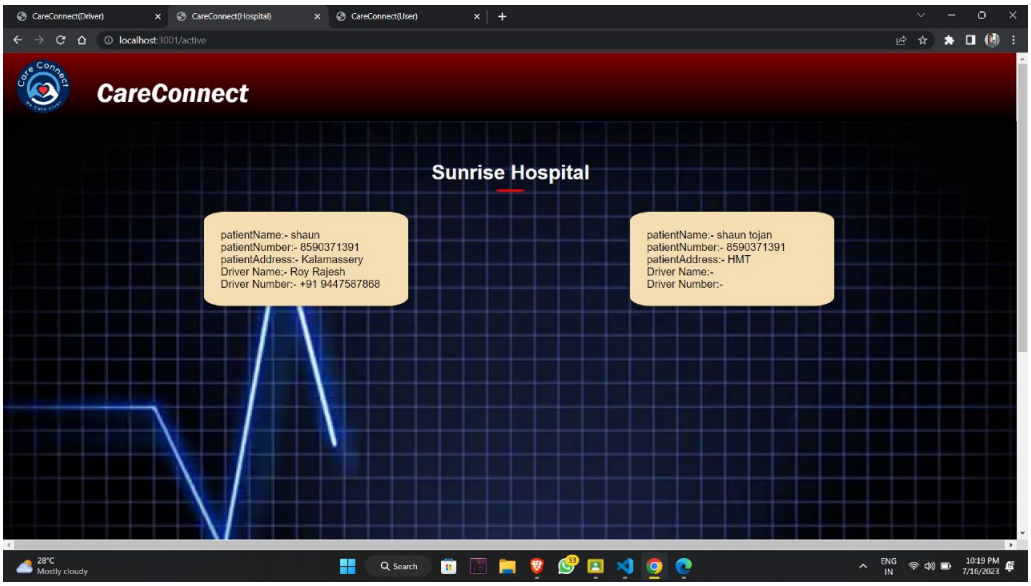

Figure 8.20: Assigned Case

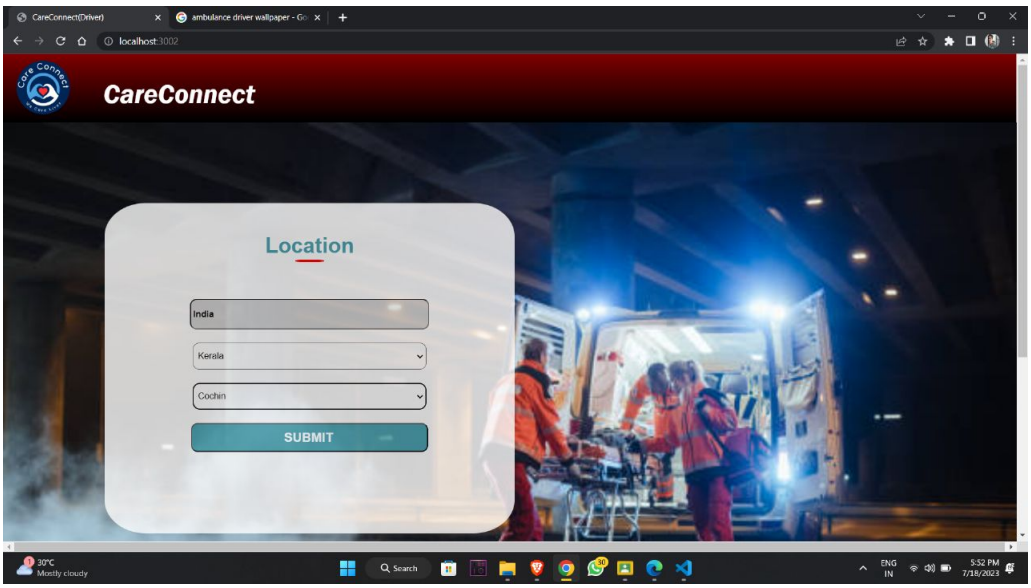

Figure 8.21: Location Selection

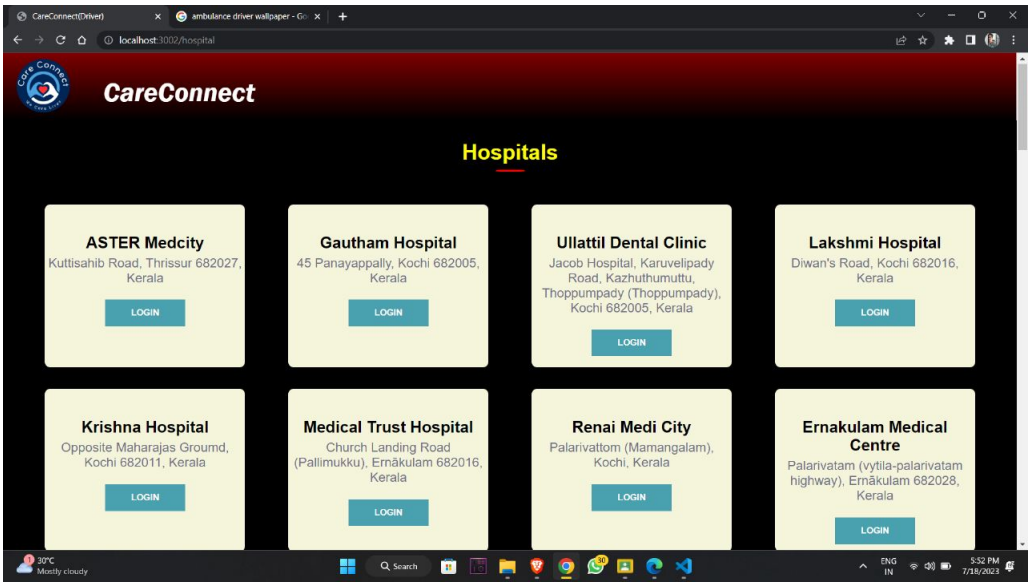

Figure 8.22: Hospital List

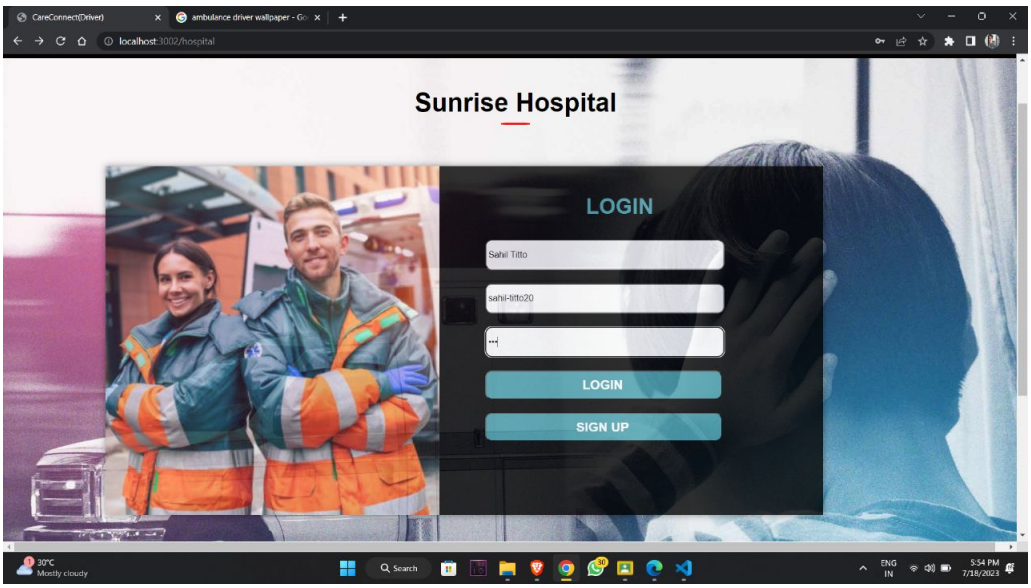

Figure 8.23: Driver Login

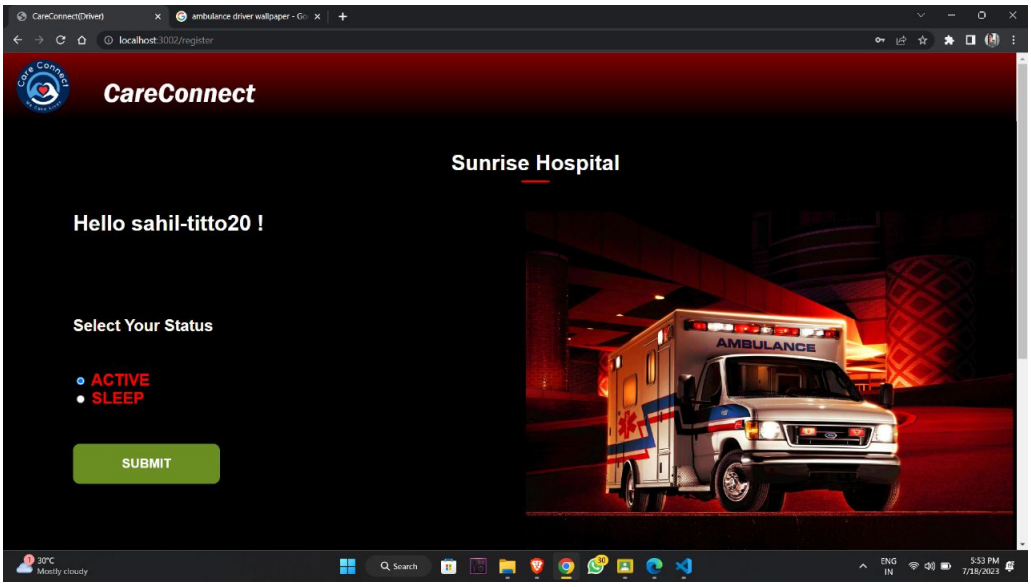

Figure 8.24: Driver Status

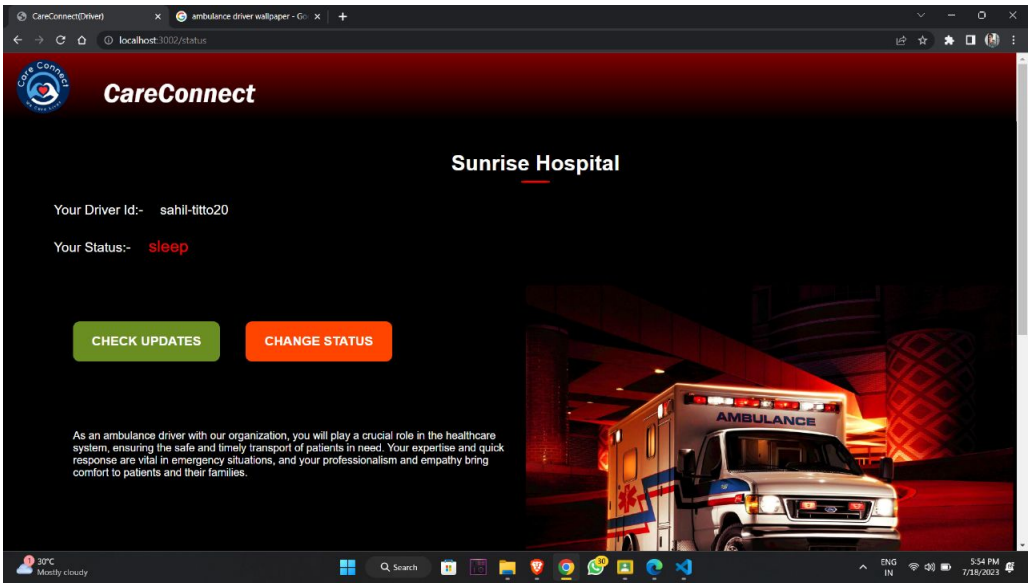

Figure 8.25: Driver Status 1: Sleep

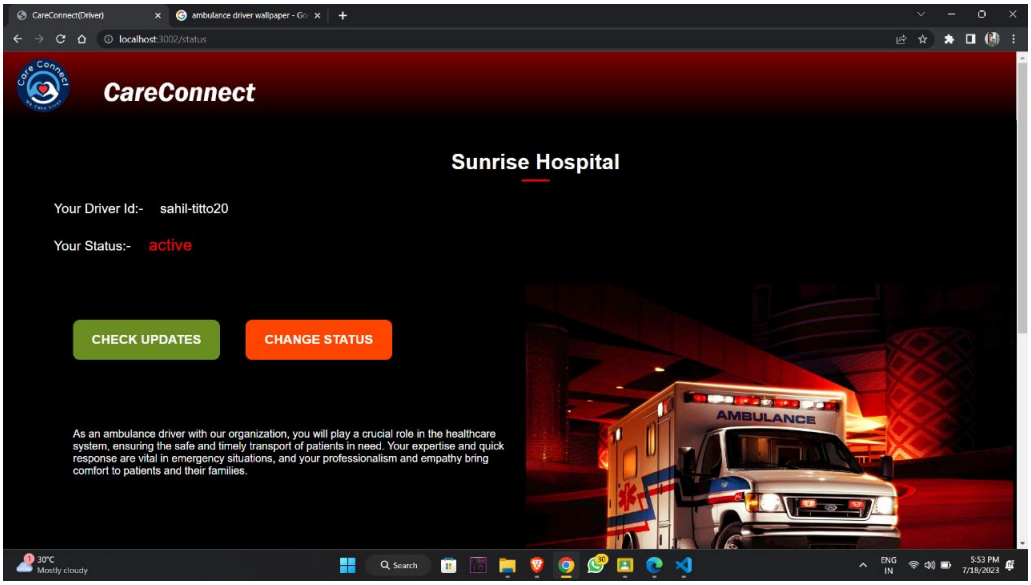

Figure 8.26: Driver Status 2: Active

Chapter 9

Risks and Challenges

1. Ensuring the security and privacy of user data.
2. Coordinating real-time updates between the modules for accurate availability
3. Addressing potential technical issues or system failures during emergencies.
4. Ensuring user-friendly interfaces and seamless user experiences across devices.

Chapter 10

Conclusion

Our project aims to address the challenges faced in critical care transport systems by developing an online ambulance dispatch website. By implementing the proposed system, we aim to improve the efficiency and effectiveness of the ambulance dispatch process, thereby enhancing patient care during medical emergencies.

Future plans are: Extend the project by creating a mobile application that allows users to request an ambulance directly from their smartphones. This would provide a more convenient and accessible way for individuals to request emergency medical assistance.

Expansion of the online ambulance dispatch system to a global scale, catering to different countries and regions, thereby improving emergency response worldwide.

Chapter 11

Glossary

1. **MERN:-** A collective term for MongoDB, ExpressJS, ReactJS and NodeJS.

2. **API:-** API stands for "Application Programming Interface." It is a set of rules and protocols that allows different software applications to communicate and interact with each other.

3. **OTP:-** OTP stands for "One-Time Password." It is a unique and temporary code generated for authentication purposes, usually sent to a user's mobile phone or email during the login process or when performing sensitive transactions.

4. **VONAGE API:-** Vonage API, now known as the Vonage Communications API, is a cloud communications platform that provides a set of APIs for integrating voice, video, messaging, and verification functionalities into applications.

5. **FOURSQUARE API:-** The FourSquare API is an application programming interface provided by FourSquare, a location-based social networking platform. It allows developers to access and interact with various location-based data and services provided by FourSquare.

6. **ReactJS:-** ReactJS is a popular JavaScript library for building user interfaces(UI). It allows developers to create reusable UI components and efficiently manage the state of web applications.

7. **ExpressJS:-** ExpressJS is a minimalist and flexible web application framework for Node.js that simplifies the process of building robust and scalable web applications and APIs.

8. **NodeJS:-** Node.js is an open-source, server-side JavaScript runtime environment that allows developers to execute JavaScript code outside of a web browser. It is

designed to be efficient and scalable, making it suitable for building various types of applications, including web servers, command-line tools, and real-time applications.

9. **SMS:-** SMS stands for "Short Message Service." It is a text messaging service that allows the exchange of short text messages between mobile devices, typically limited to 160 characters.

10. **DLC:-** A "Digit Long Code" refers to a sequence of numerical digits that is relatively long in length. It could be a phone number, an identification code, or any numeric sequence consisting of multiple digits.

11. **RAM:-** RAM stands for "Random Access Memory." It is a type of computer memory that allows data to be stored and accessed quickly by the computer's central processing unit (CPU).

12. **NoSQL:-** NoSQL, short for "Not only SQL," refers to a type of database management system that differs from traditional relational databases. It is designed to handle and store unstructured or semi-structured data and does not rely on a fixed schema.

13. **BSON:-** BSON, short for Binary JSON, is a binary-encoded data format used primarily for storing and transmitting documents in a compact and efficient way. It is designed to be lightweight and easy to parse, making it ideal for data storage and exchange in applications that require high performance and low overhead.

14. **EJS:-** EJS stands for Embedded JavaScript, and it is a simple templating engine that allows you to embed JavaScript code directly into HTML. It enables dynamic content generation on the server-side, making it easier to generate HTML pages with dynamic data. EJS is commonly used with Node.js and Express.js to create dynamic web applications.

15. **HTML:-** HTML stands for "Hypertext Markup Language." It is the standard markup language used to create and structure content on the World Wide Web.

16. **CSS:-** CSS, or Cascading Style Sheets, is a stylesheet language used to describe the presentation of a document written in HTML or XML. It defines how elements

in a web page should be displayed, including layout, colors, fonts, and other design aspects.

17. **DOM:-** DOM stands for "Document Object Model." It is a programming interface for web documents, representing the structure of a webpage as a tree of objects.

References

[1] Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node" by Vasan Subramanian.

[2] Full-Stack React Projects: Modern web development using React 16, Node.js, Express, and MongoDB" by Shama Hoque.

[3] Learning Full-Stack JavaScript Development: MongoDB, Node.js, React, and Redux" by Tim R. Rozday.

[4] MERN Quick Start Guide: Build web applications with MongoDB, Express.js, React, and Node" by Eddy Wilson Iriarte Koroliova.

[5] MERN Stack Tutorial - Full Course" by FreeCodeCamp: This tutorial covers the entire MERN stack, providing step-by-step instructions on building a complete application. It covers MongoDB, Express.js, React.js, and Node.js. Link: https://www.youtube.com/watch?v=mrHNSanmqQ4

[6] Build a MERN Stack Application - Full Tutorial" by Traversy Media: This comprehensive tutorial covers building a MERN stack application from scratch, including setting up the backend with Node.js, Express.js, and MongoDB, and building the frontend with React.js. Link: https://www.youtube.com/watch?v=7CqJlxBYj-M

[7] Build a COVID-19 Tracker with React.js, Node.js, and MongoDB" by JavaScript Mastery: While not specific to ambulance dispatch systems, this tutorial demonstrates building a full-stack application using the MERN stack to track COVID-19 cases. It provides valuable insights into integrating the technologies. Link: https://www.youtube.com/watch?v=khJlrj3Y6Ls

[8] MERN Stack React Node MongoDB CRUD Project" by Codevolution: This tutorial demonstrates building a basic CRUD (Create, Read, Update, Delete) application using the MERN stack, providing a foundation for developing more complex projects. Link: https://www.youtube.com/watch?v=mrHNSanmqQ4

Appendix A: Sample Code

USER Module: index.js

```
const express=require("express");
const bodyParser=require("body-parser");
const nodemailer=require("nodemailer");
const mongoose=require("mongoose");
const fetch = require('node-fetch');
const session = require('express-session');
const axios = require('axios');
const http=require("https");
const State=require("country-state-city").State;
const { Vonage } = require('@vonage/server-sdk');
const { City } = require("country-state-city");


require('dotenv').config();
const app=express();
app.set("view engine","ejs");
app.use(bodyParser.urlencoded({extended:false}));
app.use(express.static("public"));


app.use(session({
    secret: "nxKopVT2Hdgx0vqJ",
    resave: false,
    saveUninitialized: true
}));

mongoose.connect("mongodb+srv://admin:admin@ambulancedispatch.7yt2txc.mongodb.
net/test",{useNewUrlParser:true})
.then(() => console.log('connected'));

const userSchema=new mongoose.Schema({
    name:String,
    email:String
});
```

```
const RegisteredHospital=new mongoose.Schema({
    hospitalName:String,
    hospitalAddress:String,
    password:String,
    patient:[{
        patientName:String,
        patientNum:String,
        patientAddress:String,
        patientStatus:String,
        ambuTrack:String
    }],
    driver:[{
        driverName:String,
        driverNum:String,
        driverId:String,
        driverPass:String,
        driverStatus:String,
        patientAssign:String
    }]
});

const hospitallist=mongoose.model("hospitallist",RegisteredHospital);

const user=mongoose.model("user",userSchema);

const vonage = new Vonage({
    apiKey: "0c22d6fc",
    apiSecret: "nxKopVT2Hdgx0vqJ"
  })

app.get("/",(req,res)=>{
    res.render("home");
});
app.get("/service",(req,res)=>{
    res.render("service");
});
```

```
app.get("/features",(req,res)=>{
   res.render("features");
});
app.get("/aboutUs",(req,res)=>{
   res.render("aboutus");
});

app.get("/contactus",(req,res)=>{
   res.render("contactus");
});


app.get("/book",(req,res)=>{
   res.render("bookNow");
});


app.post("/message",(req,res)=>{
   const name=req.body.name;
   const email=req.body.email;
   const msg=req.body.msg;
   const transporter=nodemailer.createTransport({
      service:'gmail',
      auth:{
         user:process.env.NODEMAILER_EMAIL,
         pass:process.env.NODEMAILER_PASS
      },
      port:465,
      host:'smtp.gmail.com'
   });

   const mailOption1={

      from: process.env.NODEMAILER_EMAIL,
```

```
    to:`${email}`,
    subject:"Ambulance Tracker customer care",
    text:"Thanks For Contacting Us "+`${name}`
};

const mailOption2={
    from:process.env.NODEMAILER_EMAIL,
    to:process.env.SECOND_EMAIL,
    subject:`${name}`,
    text:"name:- "+`${name}`+"\n email:- "+`${email}`+"\n message:- "+`${msg}`
}

transporter.sendMail(mailOption1,(error,info)=>{
    if(error){
        console.log(error);
        res.send("error sending email");
    }
    else{
        res.send("email sent successfully");
    }
});

transporter.sendMail(mailOption2,(error,info)=>{
    if(error){
        console.log(error);
        res.send("error sending email");
    }
    else{
        res.send("email sent successfully");
    }
});



user.findOne({ email: email }).then(function(elem) {
    if (!elem) {
```

```
        const newUser = new user({
          name: name,
          email: email
        });
        newUser.save();
      }
    }).catch((err) => {
      console.log(err);
    });

  res.render("message");

});




app.post("/book", (req, res) => {
  var number = req.body.phone;
  var username = req.body.Name;

  // Check if SMS has already been sent for this session
  if (req.session.smsSent) {
    // SMS has already been sent, render the verify page without sending another SMS

    res.render("verify", { number: number, username: username, code: req.session.code
});
  } else {
    var code = Math.floor(Math.random() * 999999);
    const from = "Vonage APIs"
    const to = "+91" + number;
    const text = 'hlo ' + username + " this is your verification code:- " + code;

    async function sendSMS() {
      await vonage.sms.send({ to, from, text })
        .then(resp => {
```

```
                console.log('Message sent successfully');
                console.log(resp);
                // Set the flag and code in session to indicate that SMS has been sent
                req.session.smsSent = true;
                req.session.code = code;
                res.render("verify", { number: number, username: username, code: code });
            })
            .catch(err => {
                console.log('There was an error sending the messages.');
                console.error(err);
                res.render("error", { error: err });
            });
        }

        sendSMS();
    }
});


app.post("/verify",(req,res)=>{
    var userName=req.body.userName;
    var phoneNumber=req.body.phoneNumber;
    var enterCode=req.body.code;
    var code=req.body.realCode;
    var count=0;
    if(enterCode==code){
        var allState=(State.getStatesOfCountry("IN"));
        var allCities={};
        for(var i=0;i<allState.length;i++){
            var city=City.getCitiesOfState("IN",allState[i].isoCode);
            allCities[allState[i].name]=city;
        }
        var allCitiesString = JSON.stringify(allCities);

res.render("location",{allState:allState,allCitiesString:allCitiesString,userName:userNam
e,phoneNumber:phoneNumber});
```

```
    }
    else{
        count++;
        if(count==3){
            res.redirect("/book");
        }
        else{
            res.render("verify",{Username:userName,number:phoneNumber});
        }
    }
});


app.post("/location",async(req,res)=>{
    try{
    var latitude;
    var longitude;
    var userName=req.body.userName;
    var phoneNumber=req.body.phoneNumber;
    var state=req.body.state;
    var city=req.body.city;
    var apiUrl = "https://nominatim.openstreetmap.org/search";
    var params = {
        q: city + ", " + state,
        format: "json",
        limit: 1
    };

    var queryString = Object.keys(params).map(function(key) {
        return encodeURIComponent(key) + "=" + encodeURIComponent(params[key]);
    }).join("&");

    var url = apiUrl + "?" + queryString;

    var response=await fetch(url);
    const data=await response.json();
```

```
        if (data.length > 0) {
            latitude = data[0].lat;
            longitude = data[0].lon;

        } else {
            console.log("Coordinates not found for the specified location.");
        }

         function hospitalCall(){
            const options = {
                method: 'GET',
                hostname: 'api.foursquare.com',
                port: null,
                path:
'/v3/places/search?ll='+latitude+'%2C'+longitude+'&radius=100000&categories=15000&
limit=50',
                headers: {
                  accept: 'application/json',
                  Authorization:
"fsq3uht2DYTSaumoNnaX3suX0C98LuGM6QIIa1FqvRa7Y5o="
                }
              };

              const apiRequest = http.request(options, function (apiResponse) {
                let responseBody = '';

                apiResponse.on('data', function (chunk) {
                  responseBody += chunk;
                });

                apiResponse.on('end', function () {
                  const data = JSON.parse(responseBody);
                  const hospitals = data['results'];
                  const filteredHospitals = hospitals.map(hospital => {
                    return {
```

```
                    name: hospital['name'],
                    address : hospital['location']['formatted_address']
                  };
                });

res.render("hospital",{hospital:filteredHospitals,userName:userName,phoneNumber:phon
eNumber});
              });
            });

            apiRequest.end();
        }
        hospitalCall();


    }catch(error){
        console.log("An error occured: "+error);
    }

});




  app.post("/hospital",(req,res)=>{
      var hospitalName=req.body.hospitalName;
      var hospitalAddress=req.body.hospitalAddress;
      var userName=req.body.userName;
      var phoneNumber=req.body.phoneNumber;
      hospitallist.findOneAndUpdate(
       { hospitalName: hospitalName, hospitalAddress: hospitalAddress },
       { $push: { patient: { patientName: userName, patientNum:
phoneNumber,patientStatus:'pending',ambuTrack:"Booking Confirmed" } } },
       { new: true }
      )
```

```
        .then((updatedHospital) => {
          if (!updatedHospital) {
            res.send("Hospital is not registered");
          } else {

res.render("track",{userName:userName,phoneNumber:phoneNumber,hospitalName:hos
pitalName,hospitalAddress});
          }
        })
        .catch((error) => {
          console.log("Error updating pending case:", error);
          res.send("Error updating pending case");
        });

  });


  var assigndriverName = "Not Assigned Yet";
  var assigndriverNum = "Not Assigned Yet";
  var assigndriverId = "Not Assigned Yet";
app.post("/track", async (req, res) => {
    var userName = req.body.userName;
    var phoneNumber = req.body.phoneNumber;
    var hospitalName = req.body.hospitalName;
    var hospitalAddress = req.body.hospitalAddress;
    var ambuTrack;
    var patientId;

    try {
      const hospital = await hospitallist.findOne({ hospitalName: hospitalName,
hospitalAddress: hospitalAddress });

      if (!hospital) {
        console.log("Hospital not found");
      } else {
```

```
      const patient = hospital.patient.find((p) => p.patientName === userName &&
p.patientNum === phoneNumber);

       if (!patient) {
         console.log("Patient not found");
       } else {
         ambuTrack = patient.ambuTrack;
         patientId = patient._id.toString();

         if (ambuTrack === "Ambulance assigned") {
           const h1 = await hospitallist.findOne({ hospitalName: hospitalName,
hospitalAddress: hospitalAddress });

           if (!h1) {
             console.log("Hospital is not found");
           } else {
             const driver = h1.driver.find((d) => d.patientAssign === patientId);

             if (!driver) {
               console.log("Driver not found");
             } else {
               assigndriverName = driver.driverName;
               assigndriverId = driver.driverId;
               assigndriverNum = driver.driverNum;
             }
           }
         }
       }
     }

     res.render("status", {
       userName: userName,
       phoneNumber: phoneNumber,
       hospitalName: hospitalName,
       hospitalAddress: hospitalAddress,
       ambuTrack: ambuTrack,
```

```
        driverName: assigndriverName,
        driverNum: assigndriverNum,
        driverId: assigndriverId
      });
    } catch (err) {
     res.send(err);
    }
  });

app.listen(process.env.PORT || 3000);
```

Hospital Module: index.js

```
const express=require("express");
const bodyParser=require("body-parser");
const nodemailer=require("nodemailer");
const mongoose=require("mongoose");
const axios = require('axios');
const http=require("https");
const State=require("country-state-city").State;
const { City } = require("country-state-city");
const fetch=require("node-fetch");
const { ObjectId } = require('mongodb');
require('dotenv').config();
const app=express();
app.set("view engine","ejs");
app.use(bodyParser.urlencoded({extended:false}));
app.use(express.static("public"));

mongoose.connect("mongodb+srv://admin:admin@ambulancedispatch.7yt2txc.mongodb.
net/test",{useNewUrlParser:true});

const hospitaUserSchema=new mongoose.Schema({
    name:String,
    email:String
});
```

```
const RegisteredHospital=new mongoose.Schema({
    hospitalName:String,
    hospitalAddress:String,
    password:String,
    patient:[{
        patientName:String,
        patientNum:String,
        patientAddress:String,
        patientStatus:String,
        ambuTrack:String
    }],
    driver:[{
        driverName:String,
        driverNum:String,
        driverId:String,
        driverPass:String,
        driverStatus:String,
        patientAssign:String
    }]
});

const hospitalUser=mongoose.model("hospitalUser",hospitaUserSchema);
const hospitallist=mongoose.model("hospitallist",RegisteredHospital);

app.get("/",(req,res)=>{
    var allState=(State.getStatesOfCountry("IN"));
    var allCities={};
    for(var i=0;i<allState.length;i++){
        var city=City.getCitiesOfState("IN",allState[i].isoCode);
        allCities[allState[i].name]=city;
    }
    var allCitiesString = JSON.stringify(allCities);
    res.render("hospital-home",{allState:allState,allCitiesString:allCitiesString})
});

var latitude;
```

```
var longitude;
app.post("/",async(req,res)=>{
   try{
      var state=req.body.state;
      var city=req.body.city;
      var apiUrl = "https://nominatim.openstreetmap.org/search";
      var params = {
         q: city + ", " + state,
         format: "json",
         limit: 1
      };

      var queryString = Object.keys(params).map(function(key) {
         return encodeURIComponent(key) + "=" + encodeURIComponent(params[key]);
      }).join("&");

      var url = apiUrl + "?" + queryString;

      var response=await fetch(url);
      const data=await response.json();

         if (data.length > 0) {
            latitude = data[0].lat;
            longitude = data[0].lon;

         } else {
            console.log("Coordinates not found for the specified location.");
         }

      res.redirect("hospital");

      }catch(error){
         console.log("An error occured: "+error);
      }

});
```

```
app.get("/hospital",(req,res)=>{

    const options = {
        method: 'GET',
        hostname: 'api.foursquare.com',
        port: null,
        path:
'/v3/places/search?ll='+latitude+'%2C'+longitude+'&radius=100000&categories=15000&
limit=50',
        headers: {
          accept: 'application/json',
          Authorization: "fsq3uht2DYTSaumoNnaX3suX0C98LuGM6QIIa1FqvRa7Y5o="
        }
      };

      const apiRequest = http.request(options, function (apiResponse) {
        let responseBody = '';

        apiResponse.on('data', function (chunk) {
          responseBody += chunk;
        });

        apiResponse.on('end', function () {
          const data = JSON.parse(responseBody);
          const hospitals = data['results'];
          const filteredHospitals = hospitals.map(hospital => {
            return {
              name: hospital['name'],
              address : hospital['location']['formatted_address']
            };
          });
          res.render("hospital",{hospital:filteredHospitals});
        });
      });
```

```
    apiRequest.end();
});

app.post("/hospital",(req,res)=>{
   var hospitalName=req.body.hospitalName;
   var hospitalAdd=req.body.hospitalAddress;
   res.render("login",{hospitalName:hospitalName,hospitalAddress:hospitalAdd});
});


app.post("/message",(req,res)=>{
   const name=req.body.name;
   const email=req.body.email;
   const msg=req.body.msg;
   const transporter=nodemailer.createTransport({
      service:'gmail',
      auth:{
         user:process.env.NODEMAILER_EMAIL,
         pass:process.env.NODEMAILER_PASS
      },
      port:465,
      host:'smtp.gmail.com'
   });

   const mailOption1={

      from: process.env.NODEMAILER_EMAIL,
      to:`${email}`,
      subject:"Ambulance Tracker customer care",
      text:"Thanks For Contacting Us "+`${name}`
   };

   const mailOption2={
      from:process.env.NODEMAILER_EMAIL,
      to:process.env.SECOND_EMAIL,
```

```
    subject:`${name}`,
    text:"name:- "+`${name}`+"\n email:- "+`${email}`+"\n message:- "+`${msg}`
}

transporter.sendMail(mailOption1,(error,info)=>{
    if(error){
        console.log(error);
        res.send("error sending email");
    }
    else{
        console.log("email sent: "+info.response);
        res.send("email sent successfully");
    }
});

transporter.sendMail(mailOption2,(error,info)=>{
    if(error){
        console.log(error);
        res.send("error sending email");
    }
    else{
        console.log("email sent: "+info.response);
        res.send("email sent successfully");
    }
});



hospitalUser.findOne({ email: email }).then(function(elem) {
    if (!elem) {
      const newUser = new hospitalUser({
        name: name,
        email: email
       });
      newUser.save();
    }
```

```
      }).catch((err) => {
        console.log(err);
      });

    res.render("message");

});

app.get("/message",(req,res)=>{
  res.render("message");
});


var hospitalName;
var hospitalAddress;
app.post("/login",async(req,res)=>{
  function getvalue(){
    hospitalName=req.body.hospitalName;
    hospitalAddress=req.body.hospitalAddress;
  }

  await getvalue();
  hospitallist.findOne({ hospitalName:hospitalName ,
hospitalAddress:hospitalAddress,password:req.body.password }).then(function(elem) {
   if (!elem) {

res.render("login",{hospitalName:req.body.hospitalName,hospitalAddress:req.body.hospi
talAddress});
    }
    else{
       res.render("home",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
    }
  }).catch((err) => {
    console.log(err);
  });
});
```

```
app.post("/signup",(req,res)=>{
   hospitalName=req.body.hospitalName;
   hospitalAddress=req.body.hospitalAddress;
   res.render("signup",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
});

app.post("/register",(req,res)=>{
   hospitalName=req.body.hospitalName;
   hospitalAddress=req.body.hospitalAddress;
   hospitallist.findOne({ hospitalName:hospitalName ,
hospitalAddress:hospitalAddress}).then(function(elem) {
       if (!elem) {
          const newHospital = new hospitallist({
            hospitalName:req.body.hospitalName,
            hospitalAddress:req.body.hospitalAddress,
            password:req.body.password
           });
           newHospital.save();

res.render("home",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
       }
       else{

res.render("login",{hospitalName:req.body.hospitalName,hospitalAddress:req.body.hospi
talAddress})
       }
      }).catch((err) => {
       console.log(err);
      });
});

app.post("/pending",(req,res)=>{

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n(function(element){
```

```
        if(!element){
            res.send("hospital not found");
        }
        else{
            const
pending=element.patient.filter((patient)=>patient.patientStatus==="pending");

res.render("pending",{hospitalName:hospitalName,elem:pending,hospitalAddress:hospit
alAddress});
        }
    });
});

app.get("/pending",(req,res)=>{

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n(function(element){
        if(!element){
            res.send("hospital not found");
        }
        else{
            const
pending=element.patient.filter((patient)=>patient.patientStatus==="pending");

res.render("pending",{hospitalName:hospitalName,elem:pending,hospitalAddress:hospit
alAddress});
        }
    });
});



app.post("/active",(req,res)=>{

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n((element)=>{
         if(!element){
```

```
          res.send("hospital not found");
        }
        else{
          const active=element.patient.filter((patient)=>patient.patientStatus==="active");
          var driverName=[];
          var driverNum=[];
          for(var i=0;i<active.length;i++){
             const driver = element.driver.find((driver) =>
(driver.patientAssign)===active[i]._id.toString());
             if (driver) {
                driverName.push(driver.driverName);
                driverNum.push(driver.driverNum);
              }
          }

res.render("activeCase",{hospitalName:hospitalName,elem:active,hospitalAddress:hospit
alAddress,driverName:driverName,driverNum:driverNum});
        }
    }).catch((err)=>{
       res.send(err);
    });
});

app.get("/active",(req,res)=>{

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n((element)=>{
        if(!element){
          res.send("hospital not found");
        }
        else{
          const active=element.patient.filter((patient)=>patient.patientStatus==="active");
          var driverName=[];
          var driverNum=[];
          for(var i=0;i<active.length;i++){
```

```
            const driver = element.driver.find((driver) =>
(driver.patientAssign)===active[i]._id.toString());
            if (driver) {
               driverName.push(driver.driverName);
               driverNum.push(driver.driverNum);
             }
          }

res.render("activeCase",{hospitalName:hospitalName,elem:active,hospitalAddress:hospit
alAddress,driverName:driverName,driverNum:driverNum});
         }
   }).catch((err)=>{
      res.send(err);
   });
});

app.post("/completed",(req,res)=>{

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n(function(element){
      if(!element){
         res.send("hospital not found");
      }
      else{
         const
complete=element.patient.filter((patient)=>patient.patientStatus==="complete");

res.render("completeCase",{hospitalName:hospitalName,elem:complete,hospitalAddress:
hospitalAddress});
      }
   });
});

app.get("/completed",(req,res)=>{
```

```
hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n(function(element){
        if(!element){
            res.send("hospital not found");
        }
        else{
            const
complete=element.patient.filter((patient)=>patient.patientStatus==="complete");

res.render("completeCase",{hospitalName:hospitalName,elem:complete,hospitalAddress:
hospitalAddress});
        }
    });
});



app.post("/reject", async (req, res) => {
    var patientId=req.body.patientId;
    var hospitalName=req.body.hospitalName;
    var hospitalAddress=req.body.hospitalAddress;
    try {
        await hospitallist.updateOne(
            { hospitalName:hospitalName,hospitalAddress:hospitalAddress },
            { $pull: { patient: { _id: new ObjectId(patientId) } } }
        );
        console.log('Subdocument deleted successfully');
        res.redirect("pending");
    } catch (err) {
        console.error(err);
        res.status(500).send('An error occurred');
    }
});

app.post("/assign",async(req,res)=>{
```

```
  var patientAddress=req.body.patientAddress;
  var patientNum=req.body.patientNum;
  var patientName=req.body.patientName;
  var patientId=req.body.patientId;
  var hospitalName=req.body.hospitalName;
  var hospitalAddress=req.body.hospitalAddress;


hospitallist.findOneAndUpdate({hospitalName:hospitalName,hospitalAddress:hospitalA
ddress,"patient._id":patientId},{$set:{"patient.$.patientAddress":patientAddress,"patient.
$.ambuTrack":"approved by hospital"}}).then((hospital)=>{
   if(!hospital){
      res.send("hospital not found");
   }
   else{

res.render("assign",{hospitalName:hospitalName,hospitalAddress:hospitalAddress,patien
tId:patientId});
   }
  }).catch((err)=>{
   res.send(err);
  })

});


// get the active driver list

app.post("/activeDriver",(req,res)=>{
  var hospitalName=req.body.hospitalName;
  var hospitalAddress=req.body.hospitalAddress;
  var patientId=req.body.patientId;

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n((hospital)=>{
   if(!hospital){
```

```
        res.send("hospital not found");
    }
    else{
        const driver=hospital.driver.filter((driver)=>driver.driverStatus==="active");

res.render("activeDriver",{hospitalName:hospitalName,hospitalAddress:hospitalAddress,
patientId:patientId,driver:driver});
    }
   }).catch((err)=>{
    res.send(err);
   });

});


app.post("/assignDriver",(req,res)=>{
      var hospitalName=req.body.hospitalName;
      var hospitalAddress=req.body.hospitalAddress;
      var patientId=req.body.patientId;
      var driverId=req.body.driverId;


hospitallist.findOneAndUpdate({hospitalName:hospitalName,hospitalAddress:hospitalA
ddress,"driver.driverId":driverId},{$set:{"driver.$.driverStatus":"working","driver.$.patie
ntAssign":patientId}}).then((hospital)=>{
        if(!hospital){
            res.send("hospital not found");
        }
        else{

hospitallist.findOneAndUpdate({hospitalName:hospitalName,hospitalAddress:hospitalA
ddress,"patient._id":patientId},{$set:{"patient.$.patientStatus":"active","patient.$.ambuTr
ack":"ambulance assigned"}}).then((hospital)=>{
                if(!hospital){
                    res.send("2nd time hospital is not found");
                }
```

```
            else{
                res.redirect("/pending");
            }
        }).catch((err)=>{
            res.send(err);
        });
      }
    }).catch((err)=>{
      res.send(err);
    })
});


app.post("/WorkingDriver",(req,res)=>{
   var hospitalAddress=req.body.hospitalAddress;
   var patientId=req.body.patientId;

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n((hospital)=>{
    if(!hospital){
       res.send("hospital not found");
    }
    else{
       const driver=hospital.driver.filter((driver)=>driver.driverStatus==="working");

res.render("WorkingDriver",{hospitalName:hospitalName,hospitalAddress:hospitalAddre
ss,patientId:patientId,driver:driver});
    }
   }).catch((err)=>{
    res.send(err);
   });
});

app.post("InActiveDriver",(req,res)=>{
   var hospitalAddress=req.body.hospitalAddress;
   var patientId=req.body.patientId;
```

```
hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n((hospital)=>{
    if(!hospital){
       res.send("hospital not found");
    }
    else{
       const driver=hospital.driver.filter((driver)=>driver.driverStatus==="sleep");

res.render("InActiveDriver",{hospitalName:hospitalName,hospitalAddress:hospitalAddre
ss,patientId:patientId,driver:driver});
    }
   }).catch((err)=>{
    res.send(err);
   });
});




app.listen(process.env.PORT || 3001);
```

Driver Module: index.js

```
const express=require("express");
const bodyParser=require("body-parser");
const nodemailer=require("nodemailer");
const mongoose=require("mongoose");
const axios = require('axios');
const http=require("https");
const State=require("country-state-city").State;
const fetch=require("node-fetch");
const { City } = require("country-state-city");
const { ObjectId } = require('mongodb');
require("dotenv").config();
const app=express();
```

```
app.set("view engine","ejs");
app.use(bodyParser.urlencoded({extended:false}));
app.use(express.static("public"));


mongoose.connect("mongodb+srv://admin:admin@ambulancedispatch.7yt2txc.mongodb.
net/test",{useNewUrlParser:true});

const driverUserSchema=new mongoose.Schema({
   name:String,
   email:String
});

const RegisteredHospital=new mongoose.Schema({
   hospitalName:String,
   hospitalAddress:String,
   password:String,
   patient:[{
      patientName:String,
      patientNum:String,
      patientAddress:String,
      patientStatus:String,
      ambuTrack:String
   }],
   driver:[{
      driverName:String,
      driverNum:String,
      driverId:String,
      driverPass:String,
      driverStatus:String,
      patientAssign:String
   }]
});

const driverUser=mongoose.model("driverUser",driverUserSchema);
const hospitallist=mongoose.model("hospitallist",RegisteredHospital);
```

```
app.get("/",(req,res)=>{
    var allState=(State.getStatesOfCountry("IN"));
    var allCities={};
    for(var i=0;i<allState.length;i++){
        var city=City.getCitiesOfState("IN",allState[i].isoCode);
        allCities[allState[i].name]=city;
    }
    var allCitiesString = JSON.stringify(allCities);
    res.render("driver-home",{allState:allState,allCitiesString:allCitiesString});
});

var latitude;
var longitude;
app.post("/",async(req,res)=>{
    try{
        var state=req.body.state;
        var city=req.body.city;
        var apiUrl = "https://nominatim.openstreetmap.org/search";
        var params = {
            q: city + ", " + state,
            format: "json",
            limit: 1
        };

        var queryString = Object.keys(params).map(function(key) {
            return encodeURIComponent(key) + "=" + encodeURIComponent(params[key]);
        }).join("&");

        var url = apiUrl + "?" + queryString;

        var response=await fetch(url);
        const data=await response.json();

            if (data.length > 0) {
                latitude = data[0].lat;
```

```
            longitude = data[0].lon;
} else {
 console.log("Coordinates not found for the specified location.");
 }

 res.redirect("/hospital");

}catch(error){
console.log("An error occured: "+error);
 }
});

app.get("/hospital",(req,res)=>{

    const options = {
      method: 'GET',
      hostname: 'api.foursquare.com',
      port: null,
      path:
'/v3/places/search?ll='+latitude+'%2C'+longitude+'&radius=100000&categories=15000&
limit=50',
      headers: {
        accept: 'application/json',
        Authorization: "fsq3uht2DYTSaumoNnaX3suX0C98LuGM6QIIa1FqvRa7Y5o="
       }
     };

     const apiRequest = http.request(options, function (apiResponse) {
      let responseBody = '';

      apiResponse.on('data', function (chunk) {
        responseBody += chunk;
      });

      apiResponse.on('end', function () {
        const data = JSON.parse(responseBody);
```

```
        const hospitals = data['results'];
        const filteredHospitals = hospitals.map(hospital => {
          return {
            name: hospital['name'],
            address : hospital['location']['formatted_address']
          };
        });
        res.render("hospital",{hospital:filteredHospitals});
      });
    });

    apiRequest.end();
});

app.post("/hospital",(req,res)=>{
   var hospitalName=req.body.hospitalName;
   var hospitalAdd=req.body.hospitalAddress;
   res.render("login",{hospitalName:hospitalName,hospitalAddress:hospitalAdd});
});


app.post("/message",(req,res)=>{
   const name=req.body.name;
   const email=req.body.email;
   const msg=req.body.msg;
   const transporter=nodemailer.createTransport({
      service:'gmail',
      auth:{
         user:process.env.NODEMAILER_EMAIL,
         pass:process.env.NODEMAILER_PASS
      },
      port:465,
      host:'smtp.gmail.com'
   });

   const mailOption1={
```

```
    from: process.env.NODEMAILER_EMAIL,
    to:`${email}`,
    subject:"CareConnect customer care",
    text:"Thanks For Contacting Us "+`${name}`
};

const mailOption2={
    from:process.env.NODEMAILER_EMAIL,
    to:process.env.SECOND_EMAIL,
    subject:`${name}`,
    text:"name:- "+`${name}`+"\n email:- "+`${email}`+"\n message:- "+`${msg}`
}

transporter.sendMail(mailOption1,(error,info)=>{
    if(error){
        console.log(error);
        res.send("error sending email");
    }
    else{
        console.log("email sent: "+info.response);
        res.send("email sent successfully");
    }
});

transporter.sendMail(mailOption2,(error,info)=>{
    if(error){
        console.log(error);
        res.send("error sending email");
    }
    else{
        console.log("email sent: "+info.response);
        res.send("email sent successfully");
    }
});
```

```
    driverUser.findOne({ email: email }).then(function(elem) {
        if (!elem) {
          const newUser = new driverUser({
            name: name,
            email: email
          });
          newUser.save();
        }
       }).catch((err) => {
        console.log(err);
       });

    res.render("message");

});

app.get("/message",(req,res)=>{
    res.render("message");
 });


//  handling login page and signup page

app.post("/login",async(req,res)=>{
    var hospitalName;
    var hospitalAddress;
    var driverName;
    var driverId;
    var password;
    function getvalue(){
        hospitalName=req.body.hospitalName;
        hospitalAddress=req.body.hospitalAddress;
        driverName=req.body.driverName;
        driverId=req.body.driverId;
```

```
        password=req.body.password;
    }
    await getvalue();

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n(function(hospital){
        if(!hospital){
            res.send("hospital not found");
        }
        else{
            const driver=hospital.driver.filter((driver)=>driver.driverName===driverName
&& driver.driverId===driverId && driver.driverPass===password);
            if(driver.length==0){

res.render("login",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
            }
            else{

res.render("driverProfile",{hospitalName:hospitalName,driverId:driverId,hospitalAddress
:hospitalAddress});
            }
        }
    }).catch((err)=>{
        console.log(err);
    })
});

app.post("/driverProfile",(req,res)=>{
    var hospitalName=req.body.hospitalName;
    var hospitalAddress=req.body.hospitalAddress;
    var driverId=req.body.driverId;

res.render("driverProfile",{hospitalName:hospitalName,driverId:driverId,hospitalAddress
:hospitalAddress});
});
```

```
app.post("/signup",(req,res)=>{
    hospitalName=req.body.hospitalName;
    hospitalAddress=req.body.hospitalAddress;
    res.render("signup",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
});

app.post("/register",(req,res)=>{
   var hospitalName=req.body.hospitalName;
   var hospitalAddress=req.body.hospitalAddress;
   var driverName=req.body.driverName;
   var driverId=req.body.driverId;
   var password=req.body.password;
   var driverNum=req.body.driverNum;

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n(function(hospital){
    if(!hospital){
        res.send("hospital Not Found");
    }
    else{
        const driver=hospital.driver.filter((driver)=>driver.driverId===driverId);
        if(driver.length!=0){

res.render("signup",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
        }
        else{
            hospitallist.findOneAndUpdate(
                {hospitalName:hospitalName,hospitalAddress:hospitalAddress},
                { $push: { driver: { driverName: driverName, driverNum:
driverNum,driverId:driverId,driverPass:password,driverStatus:'sleep' } } },
                { new: true }
                ) .then((updatedDriver) => {
                if (!updatedDriver) {
                    res.send("Hospital is not registered");
                } else {
                    console.log("Driver updated successfully");
```

```
res.render("driverProfile",{elem:updatedDriver,driverId:driverId,driverName:driverName
});
                }
                })
                .catch((error) => {
                console.log("Error updating pending case:", error);

res.render("signup",{hospitalName:hospitalName,hospitalAddress:hospitalAddress});
                });
        }
    }

   }).catch((err)=>{
    console.log(err);
   });

});



app.post("/status", (req, res) => {
    var hospitalName = req.body.hospitalName;
    var hospitalAddress = req.body.hospitalAddress;
    var driverId = req.body.driverId;
    var status = req.body.status;

    hospitallist
      .findOneAndUpdate(
        {
          hospitalName: hospitalName,
          hospitalAddress: hospitalAddress,
          "driver.driverId": driverId
        },
        {
          $set: { "driver.$.driverStatus": status }
```

```
        },
        { new: true }
      )
      .then((updatedHospital) => {
        if (!updatedHospital) {
          res.send("Hospital not found");
        } else {

res.render("currentStatus",{hospitalName:hospitalName,hospitalAddress:hospitalAddress
,driverId:driverId,driverStatus:status});
        }
      })
      .catch((err) => {
        console.error(err);
        res.send(err);
      });
  });

  app.post("/currentStatus",async(req,res)=>{
    var driverId=req.body.driverId;
    var hospitalName=req.body.hospitalName;
    var hospitalAddress=req.body.hospitalAddress;
    var status;
    hospitallist.findOne({
        hospitalName:hospitalName,
        hospitalAddress:hospitalAddress,
    }).then((hospital)=>{
        if(!hospital){
            res.send("Hospital Not Found");
        }
        else{
            const driver=hospital.driver.filter((driver)=>driver.driverId===driverId);
            if(driver.length==0){
                res.send("driver not found");
            }
            else{
```

```
            status=driver[0].driverStatus;
            if(status!='working'){

res.render("currentStatus",{hospitalName:hospitalName,hospitalAddress:hospitalAddress
,driverId:driverId,driverStatus:status});
            }
            else{
               var patientName;
               var patientAddress;
               var patientPhoneNum;

hospitallist.findOne({hospitalName:hospitalName,hospitalAddress:hospitalAddress}).the
n((hospital)=>{
                  if(!hospital){
                     res.send("hospital not found");
                  }
                  else{
                     const driver = hospital.driver.find((driver) => driver.driverId ===
driverId);
                     if (!driver) {
                       res.send('Driver not found');
                     }

                     var patientId=driver.patientAssign;
                     console.log(hospital.patient[0]._id);
                     const patient = hospital.patient.find((patient) => patient._id.equals(new
ObjectId(patientId)));

                     if(!patient){
                        res.send("patient not found");
                     }
                     patientName=patient.patientName;
                     patientAddress=patient.patientAddress;
                     patientPhoneNum=patient.patientNum;

res.render("workingStatus",{hospitalName:hospitalName,hospitalAddress:hospitalAddre
```

```
ss,patientName:patientName,patientAddress:patientAddress,patientNum:patientPhoneNu
m,driverId:driverId,patientId:patientId});
                    }
                }).catch((err)=>{
                    res.send(err);
                })

            }
        }
    }
}).catch((err)=>{
    res.send(err);
});

});



app.post("/workingStatus",async(req,res)=>{
    var driverId=req.body.driverId;
    var hospitalName=req.body.hospitalName;
    var hospitalAddress=req.body.hospitalAddress;
    var patientId=req.body.patientId;


    await
hospitallist.findOneAndUpdate({hospitalName:hospitalName,hospitalAddress:hospitalA
ddress,"driver.driverId":driverId},{$set:{"driver.$.driverStatus":"active","driver.$.patient
Assign":""}});


hospitallist.findOneAndUpdate({hospitalName:hospitalName,hospitalAddress:hospitalA
ddress,"patient._id":patientId},{$set:{"patient.$.patientStatus":"complete","patient.$.amb
uTrack":"reached"}}).then(()=>{
```

```
res.render("currentStatus",{hospitalName:hospitalName,hospitalAddress:hospitalAddress
,driverId:driverId,driverStatus:"active"});
      }).catch((err)=>{
        res.send(err);
      })

});




app.listen(process.env.PORT || 3002);
```

Appendix B: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS622T.1-PO1	**HIGH**	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.1-PO2	**HIGH**	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS622T.1-PO3	**HIGH**	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS622T.1-PO4	**HIGH**	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS622T.1-PO6	**MEDIUM**	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS622T.1-PO7	**MEDIUM**	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS622T.1-PO8	**HIGH**	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS622T.1-PO9	**MEDIUM**	Identify technically and economically feasible problems by working as a team.
100003/CS622T.1-PO10	**MEDIUM**	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS622T.1-P011	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS622T.1-PO12	**HIGH**	Identify technically and economically feasible problems for long term learning.
100003/CS622T.1-PSO1	**MEDIUM**	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS622T.1-PSO2	**MEDIUM**	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS622T.1-PSO3	**MEDIUM**	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS622T.2-PO1	**HIGH**	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS622T.2-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS622T.2-PO3	**HIGH**	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS622T.2-PO4	**HIGH**	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS622T.2-PO5	**HIGH**	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS622T.2-PO6	**MEDIUM**	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS622T.2-PO8	**HIGH**	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS622T.2-PO9	**MEDIUM**	Identify and survey the relevant literature as a team.
100003/CS622T.2-PO10	**HIGH**	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS622T.2-PO11	**MEDIUM**	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS622T.2-PO12	**HIGH**	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS622T.2-PSO1	**MEDIUM**	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS622T.2-PSO2	**MEDIUM**	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS622T.2-PSO3	**MEDIUM**	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS622T.3-PO1	**HIGH**	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS622T.3-PO2	**HIGH**	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS622T.3-PO3	**HIGH**	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO4	**HIGH**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.3-PO5	**HIGH**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS622T.3-PO6	**MEDIUM**	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS622T.3-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PO8	**HIGH**	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS622T.3-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.3-PO10	**MEDIUM**	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS622T.3-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS622T.4-PO1	**MEDIUM**	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.4-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS622T.4-PO3	**MEDIUM**	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS622T.4-PO4	**MEDIUM**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS622T.4-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS622T.4-PO8	**HIGH**	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS622T.4-PO9	**HIGH**	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.4-PO10	**HIGH**	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS622T.4-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS622T.4-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS622T.4-PSO1	**MEDIUM**	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS622T.4-PSO2	**MEDIUM**	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS622T.4-PSO3	**MEDIUM**	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS622T.5-PO1	**HIGH**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.5-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PO3	**HIGH**	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS622T.5-PO4	**MEDIUM**	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.5-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO6	**MEDIUM**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO8	**HIGH**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO1	**MEDIUM**	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PSO2	**MEDIUM**	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.