**Project Documentation: Rapid Rescue Ambulance Booking System**

Hosted Link: **bitrebels.aptechgarden.com**

Github Repository Link: **https://github.com/syedkashanzikra/RapidRescue**

**1. Introduction**

**1.1 Background**

The **Rapid Rescue Ambulance Booking System** is a web application designed to optimize the process of requesting and dispatching ambulance services during medical emergencies. It leverages modern web technologies to provide users with a platform to quickly request ambulances, track their real-time location, and share essential medical information with Emergency Medical Technicians (EMTs) prior to their arrival. The system aims to reduce response times, enabling faster and more effective medical interventions.

**1.2 Objectives**

* **Primary Objective**: Provide a seamless and efficient web-based platform for real-time ambulance booking and dispatching during emergencies.
* **Secondary Objectives**:
	+ Enable users to provide essential medical information.
	+ Facilitate communication between dispatchers, EMTs, and users.
	+ Allow real-time tracking of ambulance locations.
	+ Empower ambulance drivers to update their status based on their current task (e.g., en route, arrived, transporting patient).

**1.3 Scope**

The **Rapid Rescue Web Application** includes the following functionalities:

* **User functionalities**:
	+ Ability to book ambulances.
	+ Real-time tracking of ambulance location.
	+ Provide medical information.
	+ Receive first-aid instructions.
	+ Provide feedback after the service.
* **Admin functionalities**:
	+ Assign and manage ambulances.
	+ Monitor real-time requests and ambulance statuses.
	+ Manage drivers and ambulances.
* **Driver functionalities**:
	+ Update their real-time status (e.g., en route, transporting patient).
	+ Access patient medical details.

*Note*: This system will not handle billing or insurance claims.

**2. Team Members**

The development of the Rapid Rescue Ambulance Booking System is carried out by a highly skilled team of full-stack developers with specialized roles:

1. **Syed Kashan Abbas** (1413911)- Full Stack Developer and Project Lead:
	* Manages overall project development.
	* Oversees architecture design, backend implementation, and system integration.
	* Leads the team to ensure timely delivery.
2. **Zayaan Zubair(**1422404**)** - Full Stack Developer and Flow Creator:
	* Creates the functional and logical flow of the application.
	* Works on both frontend and backend to ensure a seamless user experience.
	* Ensures that the system interaction is intuitive for users.
3. **Prem Kumar**(1444433)- Full Stack Developer, UI Developer, and Tester:
	* Focuses on developing the user interface (UI) and ensuring a visually appealing design.
	* Ensures that UI components follow best practices for user experience.
	* Conducts thorough testing of the system to ensure reliability.
4. **Ahsan Hussain(**1415099**)** - Full Stack Developer and Tester:
	* Responsible for backend logic implementation and optimizing database interactions.
	* Participates in testing the system to ensure functional requirements are met.
	* Validates system performance under various conditions.

**3. System Overview**

**3.1 System Architecture**

The Rapid Rescue system follows a **three-tier architecture**:

* **Frontend (Presentation Layer)**: Implemented using Razor views, HTML5, CSS3, Bootstrap 5, and JavaScript. This layer handles all user interactions, such as ambulance booking and real-time map integration.
* **Backend (Application Layer)**: Built using **ASP.NET Core MVC**, this layer manages the business logic, user management, and data processing.
* **Database (Data Layer)**: Managed by **MS SQL Server**, it stores essential data such as users, ambulances, drivers, and emergency requests. The application uses **Entity Framework** in a Code-First approach.

**3.2 Technology Stack**

* **Frontend**: Razor, Bootstrap 5, JavaScript, CSS3, HTML5.
* **Backend**: ASP.NET Core MVC.
* **Database**: Microsoft SQL Server.
* **Map API**: Google Maps API for real-time tracking of ambulances and users.

**4. Functional Requirements**

**4.1 User Module (Patient)**

* **Account Registration**: Users can create accounts using email addresses and passwords.
* **Profile Management**: Users can update personal information, including medical history and emergency contacts.
* **Emergency Request**: Users can book an ambulance by providing details such as pickup location and hospital information.
* **Real-time Tracking**: Users can track the location of assigned ambulances in real-time.
* **Medical Profile**: Users can optionally provide their medical history, which will be shared with EMTs.
* **Feedback**: After service completion, users can provide feedback.

**4.2 Admin Module (Dispatcher)**

* **Login**: Admins securely log into the dispatch system.
* **Ambulance Management**: Admins can add, modify, or remove ambulances from the system.
* **Driver Profiles**: Admins manage driver profiles and assign drivers to ambulances.
* **Dispatch Control**: Admins can view available ambulances and assign them to emergency requests.
* **Real-time Monitoring**: Admins track the status and location of all ambulances and view ongoing requests.
* **Notifications**: The system sends alerts for new emergency requests and status changes.

**4.3 Driver/EMT Module**

* **Login**: Drivers log in securely to access the system.
* **Patient Information**: EMTs can access patient medical details prior to arrival.
* **Status Updates**: Drivers update their status in real-time (e.g., "available", "on the way", "arrived", "transporting patient", "finished").

**5. Non-Functional Requirements**

**5.1 Security**

* **Authentication and Authorization**: Role-based access control ensures that only authorized users can access specific functionalities.
* **Data Privacy**: Patient medical records are securely stored to comply with privacy standards.
* **Encryption**: Sensitive data is encrypted in transit and at rest to prevent unauthorized access.

**5.2 Performance**

* **Load Handling**: The system can handle multiple concurrent requests without noticeable slowdowns.
* **Response Time**: The system aims for a sub-2-second response time for booking and tracking requests.

**5.3 Scalability**

* The system is designed to scale horizontally to accommodate an increasing number of users, ambulances, and requests as needed. This ensures that the system can grow as demand increases without compromising performance.

**5.4 Availability**

* The system should be available 24/7 with minimal downtime, given its role in emergency response. Downtime should be minimized through robust infrastructure, redundancy, and maintenance strategies.

**5.5 Compatibility**

* The web application is compatible with all major web browsers, including Chrome, Firefox, Safari, and Microsoft Edge. It is also mobile-responsive, allowing users to access the application from various devices, including smartphones and tablets.

**6. Database Design**

**6.1 Entities and Attributes**

Here is a breakdown of the key database entities and attributes used in the Rapid Rescue Ambulance Booking System:

1. **User Table**
	* user\_id (Primary Key): Unique identifier for each user.
	* first\_name: User's first name.
	* last\_name: User's last name.
	* email: User's email address for login.
	* phone\_number: User’s phone number.
	* password: User's hashed password.
	* address: User’s residential address.
	* date\_of\_birth: User's date of birth.
	* medical\_history: Optional medical details.
	* created\_at: Timestamp of account creation.
	* updated\_at: Timestamp of last profile update.
2. **Ambulance Table**
	* ambulance\_id (Primary Key): Unique identifier for each ambulance.
	* vehicle\_number: Ambulance registration number.
	* equipment\_level: Equipment type (basic or advanced).
	* status: Current status (available, on call, maintenance).
	* current\_location: Real-time GPS location coordinates (latitude, longitude).
	* driver\_id (Foreign Key): Reference to the driver operating the ambulance.
3. **Driver Table**
	* driver\_id (Primary Key): Unique identifier for each driver.
	* first\_name: Driver's first name.
	* last\_name: Driver's last name.
	* phone\_number: Driver's phone number.
	* license\_number: Driver’s professional license number.
	* ambulance\_id (Foreign Key): Reference to the assigned ambulance.
4. **Emergency Request Table**
	* request\_id (Primary Key): Unique identifier for each emergency request.
	* user\_id (Foreign Key): Reference to the user making the request.
	* ambulance\_id (Foreign Key): Reference to the assigned ambulance.
	* pickup\_location: GPS coordinates (latitude, longitude) of the user's location.
	* hospital\_destination: Address or coordinates of the hospital where the patient is being taken.
	* request\_status: Status of the request (pending, assigned, completed, canceled).
	* request\_time: Timestamp of when the request was made.
	* completion\_time: Timestamp when the request was completed.
5. **Admin Table**
	* admin\_id (Primary Key): Unique identifier for each admin.
	* username: Admin username for logging in.
	* password: Admin’s hashed password.
	* role: Specifies the role of the admin (e.g., dispatcher).
6. **Status Updates Table**
	* status\_update\_id (Primary Key): Unique identifier for each status update.
	* ambulance\_id (Foreign Key): Reference to the ambulance whose status is being updated.
	* driver\_id (Foreign Key): Reference to the driver making the update.
	* status: Current status (e.g., en route, arrived, transporting patient, completed).
	* timestamp: The time when the status update occurred.
7. **Feedback Table**
	* feedback\_id (Primary Key): Unique identifier for each feedback entry.
	* user\_id (Foreign Key): Reference to the user giving feedback.
	* rating: Rating provided by the user (e.g., 1-5).
	* comments: User comments or additional feedback about the service.
	* created\_at: Timestamp when the feedback was submitted.

**7. User Interface (UI) Design**

**7.1 User Interface Overview**

The **Rapid Rescue UI** is designed to be intuitive and user-friendly, providing easy access to critical features during emergencies. The UI is implemented using Razor views, Bootstrap 5, HTML5, and JavaScript, with mobile responsiveness for seamless use on various devices.

**7.2 Key UI Components**

1. **Homepage**
	* **Map Display**: Displays the user's current location using Google Maps API, along with a prominent “Book Ambulance” button.
	* **Booking Button**: An interactive button labeled "Book Ambulance" opens a form for booking.
2. **Booking Form**
	* **Personal Details**: Fields for users to enter their name, phone number, and address.
	* **Medical History (optional)**: Section for entering medical information such as known allergies, medical conditions, and emergency contacts.
	* **Location**: The user's current location is pre-populated using the Google Maps API, with an option to manually enter the pickup address.
3. **Real-time Tracking Page**
	* **Map with Ambulance Tracking**: Real-time map showing the user's location and the ambulance’s current location, with estimated time of arrival (ETA).
	* **Status Updates**: Display showing the ambulance's current status (e.g., en route, arrived).
4. **Admin Dashboard**
	* **Emergency Request Queue**: List showing all active emergency requests, including user location, request time, and status.
	* **Ambulance Management**: Admins can add, update, or remove ambulances, and manage their statuses.
	* **Driver Assignment**: Admins assign ambulances to specific requests and monitor their progress.
5. **Driver Dashboard**
	* **Patient Information**: Provides EMTs with the patient's information, including medical history relevant to the emergency.
	* **Status Update Interface**: Allows drivers to update their current status (e.g., en route, arrived at the scene, transporting).



**8. API Integration**

**8.1 Google Maps API**

The system integrates with the **Google Maps API** for real-time tracking of ambulances and users. The API handles the following functionalities:

* **Location Detection**: Automatically detects the user’s current location using GPS or IP geolocation.
* **Real-time Tracking**: Continuously updates the ambulance's location as it moves toward the user’s location.
* **Distance and ETA Calculation**: Displays the estimated arrival time based on the distance between the user and the ambulance.

**8.2 Status Update API**

The backend provides an API endpoint for ambulance drivers to update their status. This API allows drivers to send status updates such as:

* **"Available"**
* **"En route"**
* **"Arrived"**
* **"Transporting Patient"**
* **"Completed"**

Each status update triggers notifications to both the admin and the user, ensuring that all parties are informed in real-time.

**8.3 Notifications API**

A **Notifications API** will be responsible for sending push notifications, SMS, or email alerts based on the user's preferences. This will notify the user and admin about critical updates like:

* When an ambulance is dispatched.
* When the ambulance arrives at the pickup location.
* When the patient is being transported.
* Completion of service.

**9. System Deployment and Maintenance**

**9.1 Deployment Process**

* **Development Environment Setup**:
	+ Install **ASP.NET Core MVC** using Visual Studio for code development.
	+ Set up **Entity Framework Core** using the **Code-First approach**.
	+ Configure **MS SQL Server** (or another SQL-compatible database) for database management.
	+ Configure the **Google Maps API key** for geolocation services, ensuring accurate ambulance tracking.
* **Production Environment**:
	+ Deploy the web application on a server using **IIS** or a cloud hosting platform compatible with **ASP.NET Core** (e.g., Azure, AWS).
	+ Set up **continuous integration and deployment (CI/CD) pipelines** to automate the testing, building, and deployment process.
	+ Implement **automatic database migration** in production to apply schema changes without manual intervention.

**9.2 Database Initialization with Code-First Approach**

* **Entity Framework Core** manages database creation and migration. SQL scripts are not needed, as the database schema is generated directly from C# models.
* Use **Entity Framework migrations** to handle schema changes:
	+ **Initial Migration**:

**dotnet ef migrations add InitialCreate**

**dotnet ef database update**

* + **Subsequent Migrations**:

**dotnet ef migrations add MigrationName**

**dotnet ef database update**

**9.3 System Maintenance**

* **Automatic Backups**: Implement regular database backup strategies using **MS SQL Server** tools or cloud provider offerings.
* **Application Monitoring**: Use monitoring and logging tools such as **Application Insights**, **Log4Net**, or **Serilog** to detect issues and monitor performance metrics (response times, database load, etc.).
* **Security**: Keep all libraries, frameworks, and dependencies up to date with the latest security patches. Implement role-based access control to limit unauthorized access.
* **Database Migrations in Production**: Safely apply database schema updates in production using Entity Framework's Code-First migrations:

**dotnet ef migrations add YourMigrationName**

**dotnet ef database update**

* **User Support**: Establish a help desk or support system for user inquiries, and provide troubleshooting guides for users, admins, and drivers.

**10. Diagrams**

**10.1 Entity Relationship Diagram (ERD)**

The ERD depicts the relationship between key entities in the database, such as users, ambulances, drivers, and emergency requests.



**10.2 Flowchart**

****The flowchart visually represents the process flow, starting from the user's booking of an ambulance to the driver updating their status and the admin's interaction with the dispatch system.

**10.3 Data Flow Diagram (DFD)**

The DFD provides an overview of how data moves through the system between users, admins, drivers, and external services such as the Google Maps API.



**10.4 Sitemap**

The sitemap illustrates the overall structure of the website, showing the main pages and how they are connected. This includes user-facing pages such as the home page, booking form, and tracking page, as well as admin and driver dashboards.



**11. Testing and Quality Assurance**

**11.1 Testing Strategy**

* The **Rapid Rescue Ambulance Booking System** will undergo rigorous testing to ensure system reliability, security, and performance. The following tests will be conducted:
	+ **Unit Testing**: All modules will be individually tested to ensure each function performs as expected.
	+ **Integration Testing**: The interaction between different modules, such as the user interface, backend, and database, will be tested to ensure seamless integration.
	+ **Performance Testing**: Stress tests will be performed to verify that the system can handle a large volume of concurrent requests.
	+ **User Acceptance Testing (UAT)**: End-users will test the system to ensure it meets their needs and works as expected in real-world scenarios.

**11.2 Key Test Scenarios**

* **Booking an Ambulance**: Verify that the user can book an ambulance successfully, and the admin is notified of the request.
* **Real-time Tracking**: Test the real-time tracking feature to ensure that the user's and admin's views are updated with the ambulance's current location.
* **Status Updates**: Verify that the driver can update their status (e.g., en route, arrived), and that these updates are reflected in real time for the user and admin.
* **Authentication and Role-Based Access**: Test that only authorized users can access specific functionalities based on their roles (admin, driver, user).

**11.3 Performance Metrics**

* **Response Time**: Ensure that the system responds to booking and tracking requests within 2 seconds.
* **Concurrent User Handling**: Verify that the system can support multiple users booking ambulances and tracking vehicles simultaneously without performance degradation.

**12. Conclusion**

The **Rapid Rescue Ambulance Booking System** provides a critical service that connects patients with nearby ambulances in real-time, allowing for efficient and timely emergency response. By integrating technologies like ASP.NET Core, Entity Framework, and the Google Maps API, the system ensures a seamless user experience for patients, admins, and drivers.

With robust functionality, comprehensive testing, and scalability features, the Rapid Rescue system is positioned to meet the demands of real-world emergency medical services, improving response times and patient outcomes.