**FastX**

**Problem statement:**

The Online Bus Ticket Booking System is a web-based application that allows users to search for bus routes, view schedules, book tickets, and make payments online. It connects passengers with bus operators, simplifying the process of reserving bus seats, planning journeys, and managing bookings.

**Scope**

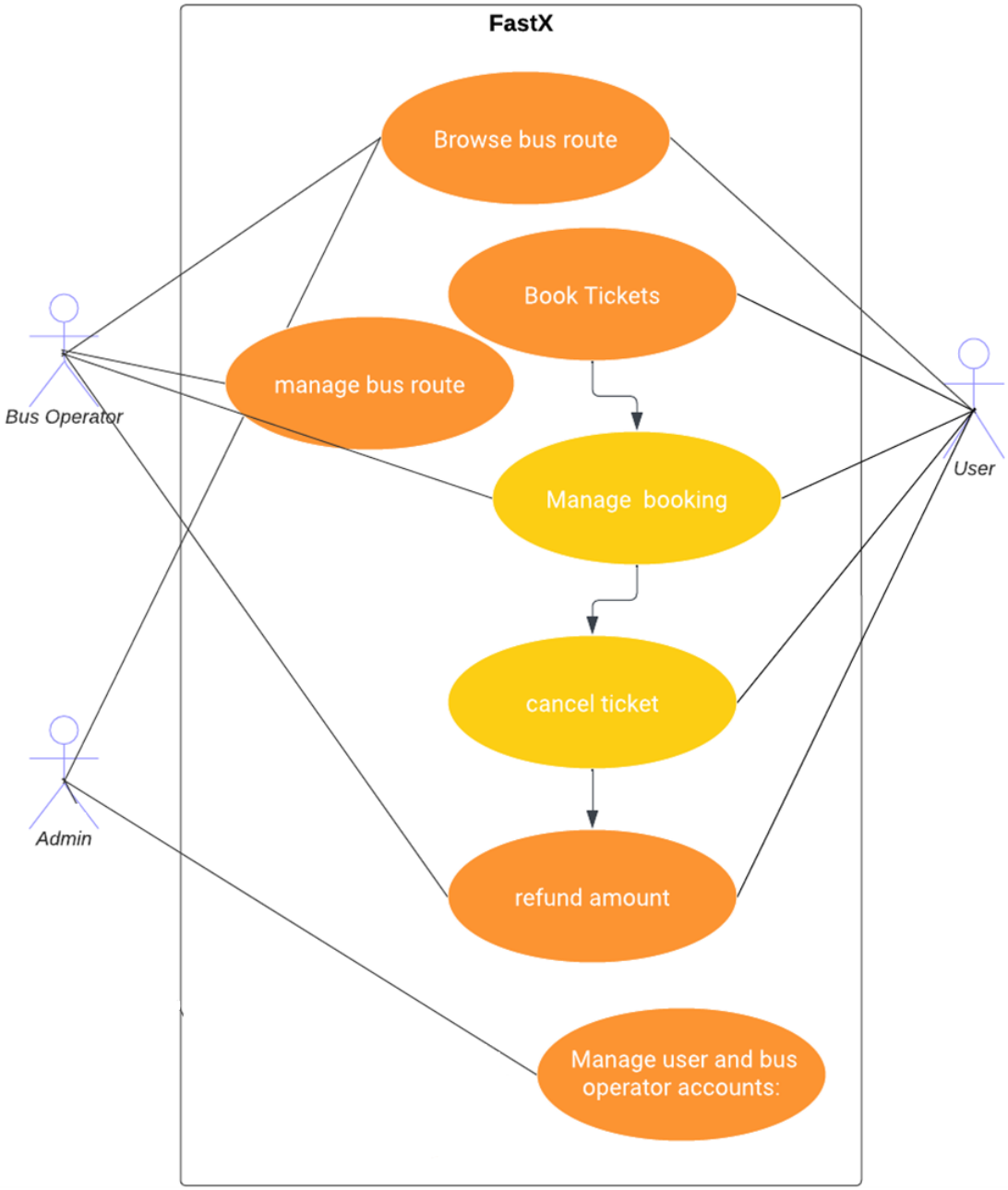
1. **User Registration and Authentication:** Provide user registration and secure login for passengers.
2. **Bus Route Search:** Enable users to search for bus routes based on origin, destination, and date.
3. **Route Details and Booking:** Display detailed route information, seat availability, and ticket prices. Allow users to select seats, make reservations.
4. **Ticket Confirmation:** Send booking confirmations with e-tickets to users.
5. **User Profiles:** Enable users to manage their profiles, view booking history, and cancel bookings.
6. **Admin Dashboard:** Offer bus operators/admins a dashboard to manage schedules, bus availability, and bookings.

**Technologies:**

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

**Reference Link:** https://www.makemytrip.com/

**Use case Diagram:**




**Use Cases:**

**Actor: User**

- Use Case: User Registration and login.
- Use Case: Search Bus.
- Use Case: Select Seats and Make Reservation.
- Use Case: Payment Processing.
- Use Case: Cancel Tickets.
- Use Case: View booking history.

**Actor: Bus Operators**

- Use Case: Log In as Bus Operator
- Use Case: Manage the Route
- Use Case: Manage Bookings
- Use Case: Logout

**Actor: Administrator**

- Use Case: Log In
- Use Case: User Management
- Use Case: Manage Bus Operator
- Use Case: Manage Bookings
- Use Case: Manage Routes
- Use Case: Logout

**System: Security and Authentication**

- Use Case: Authenticate User

**System: Database Management**

- Use Case: Store Bus routes and schedule Information.
- Use Case: Store Booking Information.
- Use Case: Store User and Bus Operator Information.

**Development Process:**

1. **User and Bus Operator Registration:**
    - User and Bus Operator can create accounts, providing personal details (name, gender, contact number, address, etc.)
    - The system validates the information and creates user profiles.
    - Users and Bus Operator log in using their credentials (username/email and password).
2. **User's Dashboard:**
    - Passengers can search for bus routes by specifying the date of journey, origin and destination.
        - To fill origin and destination location name should be typed and filled with auto suggestions.
        - To fill date of journey date should be selected from calendar. Date must selected from today and it should be validated.
    - The system displays routes, schedules, and fares, amenities (water bottle, blanket, charging point, tv) for the user search.

- Users have the option to select their preferred seats. available seats are allowed to select and booked seats are not available for booking.
- Users have the option to select more than one seat and price should be calculated as per number of seat selection.
- Users can book the tickets as they selected and can be viewed in manage booking.
- Passengers can view the history of booking and manage their booking history.
- They can cancel bookings and request refunds as needed.

3. **Bus operator Dashboard:**
    - Bus operators can add, edit, or remove bus routes and schedules. Bus route and schedules should have following details
        - Bus name, number, bus type(sleeper with and without AC, seat with and without AC), number of seat, origin, destination, timings, fare.
        - Bus amenities like water bottle, charging point, tv, blanket.
    - They can set fares and manage seat availability.
    - Bus operators can view the tickets booked by the user.
    - Bus operators can refund the amount of booked tickets which are cancelled by the user.
4. **Administrators Dashboard**
    - Administrators can manage(delete) user accounts and Bus operator accounts.
    - Admins can manage the booked tickets by the user.
    - Administrators can access a dashboard to manage bus routes.
5. **Security and Compliance:**
    - User authentication and authorization are enforced to ensure data privacy.

    **1. JWT Authentication:**

    JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

    - User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
    - Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
    - Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.

- Token Transmission: Send the signed token back to the client as a response to the login request.
- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

**2. JWT Authorization:**

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
  - Extract the token from the request header.
  - Verify the token's signature using the server's secret key.
- Payload Verification:
  - Decode the token and extract user information.
  - Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

**Logout:**

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

**Project Development Guidelines**

The project to be developed based on the below design considerations.

| | | |
|---|---|---|
| **1** | **Backend Development** | • Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services<br>• Use Java/C# latest features<br>• Use ORM with database<br>• perform backend data validation<br>• Use Swagger to invoke APIs<br>• Implement API Versioning<br>• Implement security to allow/disallow CRUD operations<br>• Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant.<br>• Any error message or exception should be logged and should be user-readable (not technical)<br>• Database connections and web service URLs should be configurable |

| | | |
|---|---|---|
| | | • Implement Unit Test Project for testing the API<br>• Implement JWT for Security<br>• Implement Logging<br>• Follow Coding Standards with proper project structure. |

**Frontend Constraints**

| | | |
|---|---|---|
| 1. | **Layout and Structure** | Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes. |
| 2 | **Visual Elements** | **Logo:** Place your application's logo at the top of the page to establish brand identity. |
| | | **Form Fields:** Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field. |
| | | **Buttons:** Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable). |
| | | **Error Messages:** Provide clear error messages for incorrect login attempts or registration errors. |
| | | **Background Image:** Consider using a relevant background image to add visual appeal. |
| | | **Hover Effects:** Change the appearance of buttons and links when users hover over them. |
| | | **Focus Styles:** Apply focus styles to form fields when they are selected |
| 3. | **Color Scheme and Typography** | Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text. |
| 4. | **Registration Page, add bus route, booking tickets by user** | **Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users. |
| | | **Validation:** Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors.<br>**Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission. |
| | **Registration Page** | **Password Strength:** Provide real-time feedback on password strength using indicators or text.<br>**Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords. |
| | | **Registration Success:** Upon successful registration, redirect users to the login page. |
| 5. | **Login Page** | **Form Fields:** Provide fields for users to enter their email and password. |
| | | **Password Recovery**: Include a "Forgot Password?" link that allows users to reset their password. |

| | | |
|---|---|---|
| 6. | **Common to React/Angular** | • Use Angular/React to develop the UI.<br>• Implement Forms, databinding, validations, error message in required pages.<br>• Implement Routing and navigations.<br>• Use JavaScript to enhance functionalities.<br>• Implement External and Custom JavaScript files.<br>• Implement Typescript for Functions Operators.<br>• Any error message or exception should be logged and should be user-readable (and not technical).<br>• Follow coding standards.<br>• Follow Standard project structure.<br>• Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets. |

**Good to have implementation features**

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application .
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.