RoadReady, a Car Rental Platform

Problem Statement:
Create a web-based, user-friendly, online solution for customers to rent cars, make reservations, manage bookings, and access detailed information about car availability, pricing, and features. The application will also offer a secure payment processing system, user reviews and ratings, and an admin dashboard for efficient management of car listings and reservations.

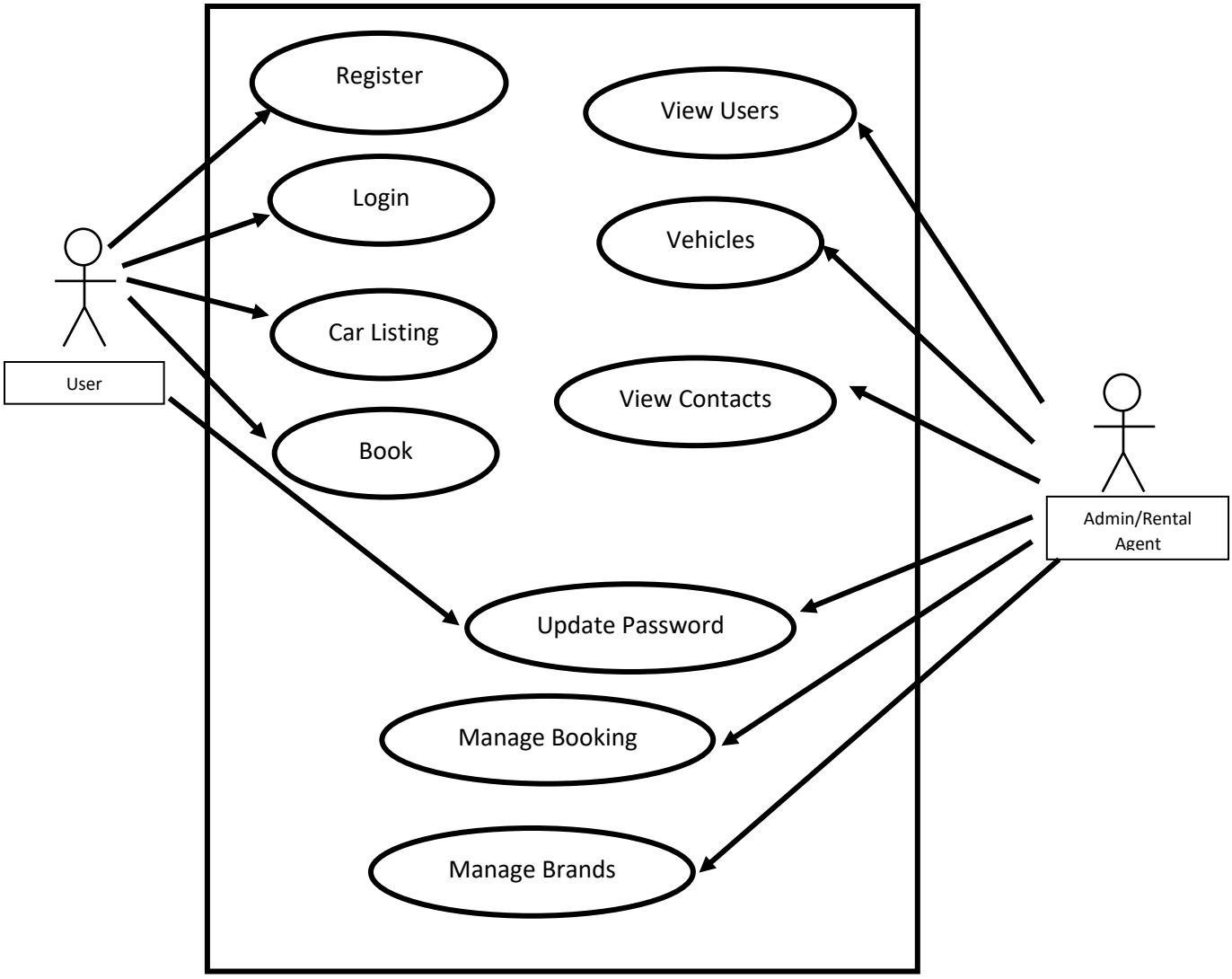
Scope:
The Car Rental Platform application aims to provide a user-friendly platform for customers to rent and manage cars. The application will have the following key features:

1. **User Registration and Authentication:**
    - Users can create accounts and log in using email or social media accounts.
    - Users can register by providing the following information First name, Last name, Email address, Password and Phone number.
    - Password reset and account recovery functionality.
2. **Car Search and Reservation:**
    - Users can browse available cars by make, model, location, and other criteria.
    - Users can view car details, including images, specifications, pricing, and availability.
    - Users can reserve cars for specific dates and times.
    - Users can reserve a car by selecting Pickup and drop-off dates and times, Optional extras (e.g., car seats)
3. **User Profile:**
    - Users can view and edit their profiles, including contact information and payment details.
4. **Booking Management:**
    - Users can view their current and past reservations.
    - Users can cancel or modify reservations.
5. **Payment Processing:**
    - Secure payment processing for reservations.
    - Support for various payment methods (credit cards, PayPal, etc.).
6. **Rating and Review:**
    - Users can rate and write reviews for their rental experiences.
7. **Admin Dashboard:**
    - Admins can manage user accounts, cars, and reservations.
    - Admins can access reporting and analytics on reservations and revenue.
    - Add, edit, and delete car listings.
    - Manage car availability and pricing.

Technologies:

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

Use Case Diagram:



Use Cases:

Actor: Admin/HR Manager

- Use Case: Manage Vehicles
- Use Case: View Reservations
- Use Case: Manage Customer Accounts
- Use Case: Generate Reports
- Use Case: Manage Admin Accounts

Actor: Customer

- Use Case: Browse Vehicles
- Use Case: Make a Reservation
- Use Case: View Reservation Details
- Use Case: Cancel Reservation
- Use Case: View Rental History

Development Process:

1. **Customer**:
   - Search for Vehicles: Customers can search for available rental vehicles based on location, date, and vehicle type.
   - Make a Reservation: Customers can select a vehicle, specify rental dates and times, and make a reservation.
   - Manage Reservations: Customers can view, modify, or cancel their existing reservations.
   - Check Vehicle Availability: Customers can check the real-time availability of vehicles at specific rental locations.
   - Provide Feedback: Customers can submit feedback and ratings for their rental experience.
   - View Billing and Payment History: Customers can review past rental bills and payment history.

2. **Administrator/Manager:**
    - Manage User Accounts: Administrators can create, modify, or deactivate user accounts, including customers and rental agents.
    - Configure Pricing and Promotions: Administrators can set pricing, discounts, and promotional offers.
    - Generate Reports: Administrators can generate reports on usage, revenue, and other key metrics.
    - Manage Feedback and Disputes: Administrators oversee and manage the feedback and dispute resolution process.
    - Vehicle Fleet Management: Administrators can add, remove, or update the fleet of rental vehicles.
3. **Rental Agent/Employee:**
    - Check-in and Check-out: Rental agents can facilitate the check-in and check-out process, including verifying the renter's identity, inspecting the vehicle, and completing necessary paperwork.
    - Manage Vehicle Inventory: Rental agents can update vehicle availability and status in the system.
    - Provide Vehicle Maintenance Alerts: Rental agents can report and request maintenance or repairs for rental vehicles.

4. **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:
    1. Extract the token from the request header.
    2. Verify the token's signature using the server's secret key.
- Payload Verification:
    1. Decode the token and extract user information.
    2. 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 • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • Use Swagger to invoke APIs. • Implement API Versioning • Implement security to allow/disallow CRUD operations. • 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. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable. • Implement Unit Test Project for testing the API. • Implement JWT for Security • Implement Logging • 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. **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. **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. • Implement Forms, data binding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files.

		• Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • 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.