# FullStack Project

## Project 25: Car Wash Booking System – ShineRide

A specialized booking solution for car wash and detailing centers. It allows vehicle owners to select wash packages, choose time slots, and track service status.

---

**User Authentication**

• **JWT-Based Login System**
Implement token-based authentication using JWT to support stateless, secure access for all users interacting with the system via frontend or API.

• **Role-Based Access Control (RBAC)**
Define user roles with controlled access to system features:

• Customer: Add vehicle details, book washes, and track progress.
• Washer/Staff: View assigned tasks, update status, and manage queue.
• Admin: Configure wash packages, pricing, and view revenue reports.

---

**Frontend Modules**

**1. Package Selection UI**

• Display service packages (Basic, Premium, Interior Detailing).
• Compare features and prices side-by-side.
• Add-on selection (Waxing, Engine Clean).
• "Book Now" workflow.

**2. Slot Booking and Vehicle Info**

• Form to enter vehicle make, model, and license plate.
• Select preferred date and time slot.

• Option for "Pick-up and Drop" service.
• Payment integration.

**3. Live Service Tracking**

• Visual progress bar (Queued -> Washing -> Drying -> Ready).
• Live photos upload by staff (optional feature).
• Estimated time of completion display.

**4. Staff Task Dashboard**

• List of cars scheduled for the day.
• Start/Stop timer for each service phase.
• Mark services as completed to trigger customer alerts.
• View vehicle-specific instructions.

**5. Admin Management**

• Manage wash bays and daily capacity.
• Update pricing and create discount coupons.
• View customer feedback and staff performance.

---

**Backend Modules (Spring Boot + SQL + NoSQL)**

**1. User Authentication Service (SQL)**

• REST APIs for login, logout, registration, and token management.
• Middleware for securing endpoints based on user roles.

**2. Service Package APIs (NoSQL)**

• CRUD endpoints for wash types and add-ons.
• Logic to calculate total cost and duration.
• Manage service availability based on equipment status.

**3. Booking and Queue APIs (NoSQL)**

• Handle appointment scheduling and bay allocation.
• Queue management logic (FIFO or Priority).
• Store vehicle history and frequency.

**4. Workflow Status APIs (NoSQL)**

• Update service states in real-time.
• Log timestamps for efficiency analysis.
• Trigger notifications on status change.

**5. Notification APIs (NoSQL)**

• SMS alerts when the car is ready for pickup.
• Email receipts after payment.
• Promotional messages to inactive customers.

---

**Deliverables**

• Secure authentication system using JWT with role-based access control.
• Service package configuration and selection interface.
• Real-time status tracking for customers.
• Staff interface for workflow management.
• Vehicle history logging and analytics.
• Responsive frontend built with , organized into modular components by role.
• RESTful API backend built with Spring Boot, connected to SQL for persistent storage.
• Exportable service reports in PDF format.
• Role-based routing and frontend route protection.
• Backend unit testing and API documentation using Swagger/OpenAPI.
• **GitHub repository link containing the complete project source code.**

---

# Deployment

The system is deployed using a combination of cloud services for hosting the frontend, backend, and databases. The deployment process is organized into four parts: frontend deployment, backend deployment, SQL database setup, and NoSQL database setup.

---

**1. Frontend Deployment (Vercel)**

The React frontend is deployed on Vercel for fast hosting, automatic builds, and seamless GitHub integration.

Steps:

• Push the frontend code to GitHub.

• Open Vercel and choose “Import Git Repository.”

• Select the repository and ensure it points to the frontend folder (if it's a mono-repo).

• Configure the build settings:

  • Build command: npm run build

  • Output directory: build or dist (based on your project setup)

• Add required environment variables such as:

REACT_APP_API_URL = <Backend API URL>

• Deploy the project and copy the live frontend URL provided by Vercel.

---

**2. Backend Deployment (Render using Docker)**

The Spring Boot backend is containerized using Docker and deployed on Render to provide a reliable runtime environment with integrated environment variable management.

**Steps:**

• Push the backend code and Dockerfile to GitHub.

• Create a new Web Service in Render and select Docker as the deployment option.
• Connect the GitHub repository and specify the backend directory.
• Add required environment variables:

- SPRING_DATASOURCE_URL (SQL Database)
- SQL_URI (SQL Atlas)
- JWT_SECRET
  • Deploy the service and use the generated backend URL for API access.

---

### 3. SQL Database Configuration (Aiven or Any Cloud SQL Service)

The SQL database is used for storing authentication data, user credentials, and other structured records.

**Steps:**
• Create a new SQL database instance using any cloud SQL provider (example: Aiven, Azure SQL, AWS RDS, or shared hosting).
• Copy the database connection string provided by the SQL service.
• Add the connection string to Render as:
SPRING_DATASOURCE_URL = <SQL connection string>
• Create required tables and configure credentials as needed.

---

### 4. NoSQL Database Configuration (MongoDB Atlas)

MongoDB Atlas is used for storing flexible data structures like student profiles, course details, attendance records, and chat messages.

Steps:

• Create a MongoDB cluster in MongoDB Atlas.

• Add a database user and assign the necessary permissions.

• Copy the MongoDB SRV connection string from the Atlas dashboard.

• Save the connection string in Render as:

MONGO_URI = <MongoDB SRV URL>

**5. Final Integration and Testing**

**Steps:**

• Update the frontend environment variable (REACT_APP_API_URL) with the Render backend URL.
• Redeploy the frontend on Netlify to apply the updated API configuration.
• Test the complete workflow, including authentication, data fetching, form submissions, and communication modules.
• Verify that the frontend, backend, SQL database, and NoSQL database are connected properly.
• Configure custom domains for both frontend and backend if needed.