RS-Express Bus Booking System Documentation

Table of Contents

Project Overview

RS-Express Bus Booking System is a comprehensive online platform for booking bus tickets in Sri Lanka. The system features both a customer-facing portal and an administrative dashboard for managing bookings, buses, routes, and users. The system includes a loyalty program, agent management system, and various booking options for users.

Core Functionalities:

- User registration and authentication
- Bus search and ticket booking
- Admin panel with role-based permissions
- Agent booking management
- Loyalty program for regular customers
- Bus route management
- Booking management and reporting
- Payment processing

Technology Stack

Backend:

- **Framework**: Laravel 10
- **PHP Version**: 8.1+
- **Authentication**: Laravel Sanctum
- **Database**: MySQL
- **ORM**: Eloquent

Frontend:

- **Framework**: React 18
- **State Management**: Context API
- **Routing**: React Router v6
- **Styling**: Tailwind CSS
- **HTTP Client**: Axios
- **Form Handling**: Manual form state management

Development Tools:

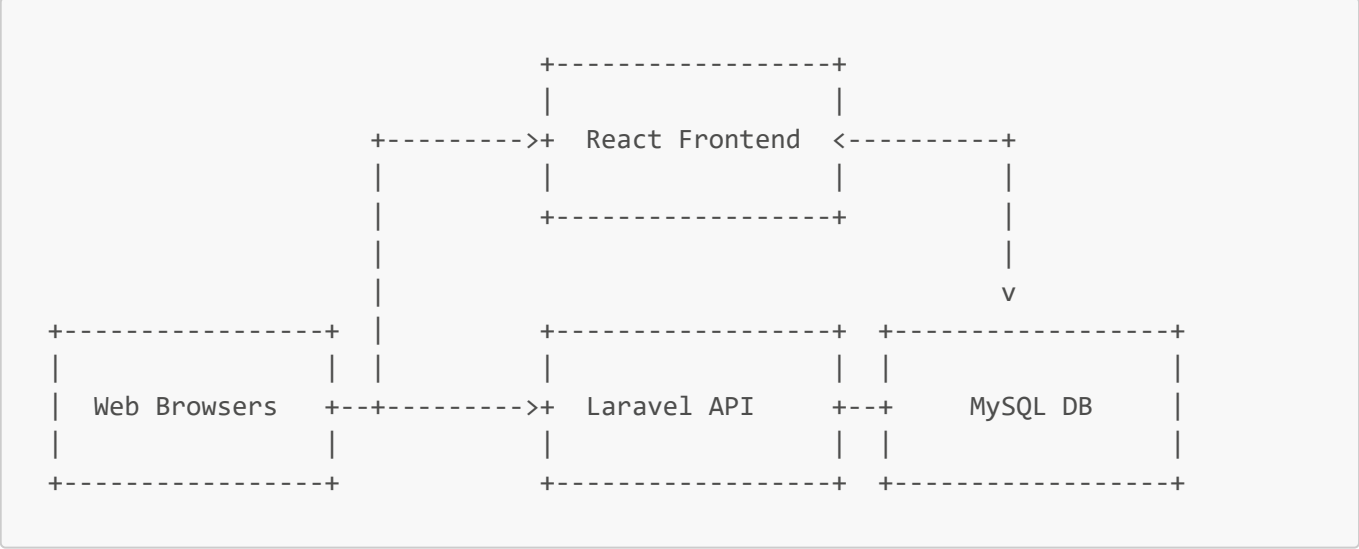
- **Version Control**: Git
- **Package Managers**: Composer (PHP), npm (JavaScript)
- **Build Tool**: Vite

Third-Party Integrations:

- Google Authentication

System Architecture

The RS-Express Bus Booking System follows a client-server architecture with a RESTful API backend and a Single Page Application (SPA) frontend:



- **Frontend**: A React SPA handling UI rendering and user interactions
- **Backend**: Laravel API providing endpoints for data operations
- **Database**: MySQL storing application data
- **Authentication**: Token-based authentication using Laravel Sanctum

Backend Structure

The backend follows Laravel's MVC architecture with additional service layers.

Key Directories:

- `app/Models/`: Contains all database models
- `app/Http/Controllers/`: API controllers handling requests
- `app/Http/Middleware/`: Request middleware including authentication
- `app/Services/`: Business logic separated into service classes
- `database/migrations/`: Database structure
- `database/seeders/`: Database seed data
- `routes/api.php`: API route definitions
- `routes/web.php`: Web route definitions
- `config/`: Configuration files

Key Models:

- `User.php`: User accounts (customers, admins, agents)
- `Booking.php`: Bus ticket bookings
- `BusRegister.php`: Bus registration information
- `BusRoute.php`: Bus route details
- `BusTrip.php`: Specific trip instances
- `Cancellation.php`: Booking cancellation records
- `LoyaltyMember.php`: Loyalty program membership

Authentication Flow:

The backend uses Laravel Sanctum for authentication:

1. User submits credentials (email/password)
2. Backend validates credentials and generates a token
3. Token is returned to the client
4. Client stores token and uses it for subsequent API requests
5. Sanctum middleware validates the token for protected routes

Frontend Structure

The frontend is a React SPA with multiple modules.

Key Directories:

- `src/components/`: Reusable UI components
- `src/pages/`: Page-level components
- `src/admin/`: Admin dashboard components
- `src/context/`: React Context providers
- `src/services/`: API service functions
- `src/utils/`: Utility functions
- `src/assets/`: Static assets like images

Key Components:

- `App.js`: Main application component and routing
- `components/Navbar.js`: Navigation header
- `pages/Login.js`: User authentication
- `pages/HomePage.js`: Landing page
- `components/BusCard/`: Bus search results
- `components/SeatBooking/`: Seat selection interface
- `admin/AdminRoutes.js`: Admin panel routing
- `admin/pages/`: Admin dashboard pages

Context Providers:

- `AuthContext.js`: User authentication state
- `PermissionsContext.js`: User permissions management

Authentication Flow:

1. User enters credentials on the Login page
2. Credentials are sent to the backend API
3. Upon successful authentication, token is stored in localStorage
4. User data is fetched and stored in AuthContext
5. Permissions are loaded based on user role
6. User is redirected to appropriate dashboard

Database Schema

Core Tables:

- **users**: User accounts with role information
  - Fields: id, name, email, password, role, etc.
- **bus_reg**: Bus registration details
  - Fields: id, bus_no, bus_type, seat_capacity, etc.
- **bus_routes**: Bus route information
  - Fields: id, route_name, start_point, end_point, etc.
- **bus_trips**: Specific trip instances
  - Fields: id, bus_id, route_id, departure_time, arrival_time, price, etc.
- **bookings**: Customer bookings
  - Fields: id, user_id, bus_id, seat_no, departure_date, price, status, etc.
- **guest_bookings**: Bookings made by non-registered users
  - Fields: id, name, phone, email, bus_id, etc.
- **cancellations**: Canceled booking records
  - Fields: id, user_id, bus_id, reason, etc.
- **loyalty_members**: Loyalty program membership
  - Fields: id, user_id, points, status, etc.

Relationships:

- Users can have multiple Bookings (one-to-many)
- BusRegister has many BusTrips (one-to-many)
- BusRoutes have many BusTrips (one-to-many)
- Users can have one LoyaltyMember record (one-to-one)

Authentication & Authorization

Authentication:

- Laravel Sanctum for token-based authentication
- CSRF protection for web routes
- Support for regular login and Google OAuth

Authorization:

- Role-based permissions system
- Roles include: admin, staff, manager, agent, user
- Each role has specific permissions defined in the database
- Permissions are loaded upon login and stored in context
- Admin panel access is restricted based on permissions
- UI elements are conditionally rendered based on permissions

Permission Structure:

Permissions are organized by modules, with each module having specific actions:

```
{
  "dashboard": {
    "view": true
  },
  "bookings": {
    "view": true,
    "create": true,
    "edit": true,
    "delete": false
  }
}
```

Key Features

User Management:

- User registration and authentication
- Profile management
- Role-based access control

Bus Management:

- Bus registration
- Route definition
- Trip scheduling
- Seat inventory management

Booking System:

- Bus search by route and date
- Seat selection interface
- Booking confirmation
- E-ticket generation
- Booking cancellation

Agent Portal:

- Agent registration
- Bulk booking capability
- Commission tracking
- Customer management

Admin Dashboard:

- Sales reporting
- Booking management
- User management
- Bus and route management
- System configuration

Loyalty Program:

- Point accumulation based on bookings
- Special discounts for loyalty members
- Membership tiers with different benefits

API Endpoints

Authentication:

- `POST /api/login`: User login
- `POST /api/signup`: User registration
- `GET /api/user`: Get authenticated user
- `POST /api/logout`: User logout

Users:

- `GET /api/users`: List all users
- `POST /api/admin/create-user`: Create new user
- `POST /api/user-details`: Update user profile
- `DELETE /api/user/{id}`: Delete user

Buses:

- `GET /api/buses`: List all buses
- `POST /api/buses`: Register new bus
- `GET /api/buses/{id}`: Get bus details
- `PUT /api/buses/{id}`: Update bus
- `DELETE /api/buses/{id}`: Delete bus

Routes:

- `GET /api/routes`: List all routes
- `POST /api/routes`: Create new route
- `GET /api/routes/{id}`: Get route details
- `PUT /api/routes/{id}`: Update route
- `DELETE /api/routes/{id}`: Delete route

Trips:

- `GET /api/trips`: List all trips
- `POST /api/trips`: Create new trip
- `GET /api/trips/{id}`: Get trip details
- `PUT /api/trips/{id}`: Update trip
- `DELETE /api/trips/{id}`: Delete trip

Bookings:

- `GET /api/bookings`: List all bookings
- `POST /api/bookings`: Create new booking
- `GET /api/bookings/{id}`: Get booking details
- `PUT /api/bookings/{id}`: Update booking
- `DELETE /api/bookings/{id}`: Cancel booking

Permissions:

- `GET /api/role-permissions/{role}`: Get permissions for a role

Component Hierarchy

Frontend Component Structure:

```
App
├── Navbar
│   └── UserProfileDropdown
├── Routes
│   ├── Public Routes
│   │   ├── HomePage
│   │   ├── Login
│   │   ├── SignUp
│   │   ├── BusList
│   │   └── SeatBooking
│   └── Protected Routes
│       ├── PassengerDashboard
│       └── AdminRoutes
│           ├── DashboardPage
│           ├── BusSchedulePage
│           ├── BusRouteManagementPage
│           ├── BusRegisterPage
│           ├── BusBookingPage
│           ├── FreezingSeatPage
│           ├── OnlinePaymentPage
│           ├── AgentPaymentPage
│           └── LoyaltyMembersPage
├── Context Providers
│   ├── AuthProvider
│   └── PermissionsProvider
└── Common Components
    ├── BusCard
    ├── SeatMap
    ├── FormComponents
    └── DataTables
```

Key Component Interactions:

1. **Authentication Flow**:
   - `Login.js` → `authService.js` → `AuthContext.js` → Redirect to dashboard
2. **Bus Booking Flow**:
   - `HomePage.js` → `BusList.js` → `BusCard.js` → `SeatBooking.js` → Confirmation
3. **Admin Dashboard Flow**:
   - `Login.js` → Check permissions → `AdminRoutes.js` → Specific admin page
4. **User Management Flow**:
   - Admin panel → User listing → User creation/editing → Update database

Workflow Diagrams

Booking Process:

1. User searches for buses by entering route and date
2. System displays available buses
3. User selects a bus and proceeds to seat selection
4. User chooses seats and provides passenger details
5. User makes payment
6. System confirms booking and generates e-ticket
7. Confirmation sent to user's email

Admin Management Process:

1. Admin logs in with credentials
2. System validates permissions
3. Admin accesses dashboard with relevant metrics
4. Admin can manage buses, routes, bookings, and users
5. Changes are saved to the database
6. System logs administrative actions

Deployment Guide

Prerequisites:

- PHP 8.1+
- Node.js 16+
- MySQL 8.0+
- Composer
- npm

Backend Setup:

1. Clone the repository
2. Navigate to the `Back-end(working)` directory
3. Copy `.env.example` to `.env` and configure database credentials
4. Run `composer install`
5. Generate application key: `php artisan key:generate`
6. Run migrations: `php artisan migrate`
7. Seed the database: `php artisan db:seed`
8. Start the development server: `php artisan serve`

Frontend Setup:

1. Navigate to the `bus(working)` directory
2. Run `npm install`
3. Copy `.env.example` to `.env` and set API URL
4. Run `npm run dev` for development or `npm run build` for production

Production Deployment:

1. Set up a web server (Apache/Nginx)
2. Configure the server to point to the `public` directory of Laravel
3. Set up appropriate server permissions
4. Configure SSL for secure connections
5. Set up a process manager like Supervisor for queue workers
6. Build the React frontend for production
7. Deploy the built files to the web server

This documentation provides a comprehensive overview of the RS-Express Bus Booking System. For specific implementation details, refer to the codebase and inline documentation.