



Car-Rental-Service

13th November 2024

Ayush Arya

Priyanshu

Anjali Chauhan

Pavani Arra



Table of Contents

1. Introduction
2. Architecture
3. Database Schema
4. Functionality
5. Code Structure
6. Snapshots
7. Setup and Deployment
8. Future Enhancements
9. Conclusion

Introduction

This **Car Rental Service** project is a full-stack web application designed to streamline car rental management. It consists of:

1. **Backend (Spring Boot - Java):** The backend, developed with Spring Boot, handles core functionalities such as authentication (using JWT for secure access), user and admin management, car bookings, and data interactions. It includes various controllers, DTOs (Data Transfer Objects), repositories, and configuration files, making it a structured, scalable, and secure service.
2. **Frontend (Angular):** The frontend, built with Angular, provides an interactive user interface. Key components include a login page, an admin dashboard, and booking pages. The frontend communicates with the backend via services, managing authentication and user interactions seamlessly.

Overall, this project is a comprehensive solution for managing car rentals, designed with security, modularity, and a smooth user experience in mind.

Architecture

The architecture of the **Car Rental Service** project follows a **modern, multi-tiered, and modular structure** with a clear separation of concerns. Here's an overview of its main layers and components:

1. Frontend Layer (Angular):

- **Framework:** Angular, a popular frontend framework, is used to build a dynamic and responsive user interface.
- **Components:** The UI is organized into various Angular components (e.g., login, admin-dashboard, post-car). Each component handles a specific part of the application's interface, such as user authentication, car listings, and booking details.
- **Services:** Services in Angular (e.g., `auth.service.ts`) manage API communication, state management, and business logic specific to the client-side.
- **Modules:** Angular modules (such as feature modules for different sections) help in organizing components and services, enabling lazy loading and improving performance.
- **Routing:** Angular's routing system allows navigation between views, ensuring a single-page application (SPA) experience.

- **Authentication:** The frontend likely handles token-based authentication by storing and attaching JWT tokens to HTTP requests to access protected routes.

2. Backend Layer (Spring Boot - Java):

- **Framework:** Spring Boot is used for creating a RESTful API backend that serves data and handles requests from the frontend.
- **Controller Layer:** The controllers (e.g., AdminController, AuthController, CustomerController) are responsible for receiving HTTP requests, processing them, and returning appropriate responses. Each controller is dedicated to a specific part of the application's functionality, like managing cars, handling user authentication, and processing bookings.
- **Service Layer:** Services contain the business logic of the application, handling processes like booking management, user authentication, and authorization.
- **Data Transfer Objects (DTOs):** DTOs are used to encapsulate and transport data between the backend and frontend layers without exposing internal data structures.
- **Repositories:** Repositories interact with the database, providing methods to access and modify data. They follow the **Data Access Object (DAO)** pattern, and are managed by Spring Data JPA.
- **Security Configuration:** Security is implemented using JWT-based authentication. The `JwtAuthenticationFilter` and `WebSecurityConfiguration` classes enforce secure access to API endpoints, allowing only authenticated users to interact with certain parts of the application.

3. Database Layer:

- **Database:** The application is likely using a relational database (such as MySQL or PostgreSQL) to store data related to cars, users, bookings, and other resources.
- **ORM (Object-Relational Mapping):** Spring Data JPA is used as the ORM to map Java objects to database tables and perform CRUD operations.

4. Authentication and Authorization:

- **JWT (JSON Web Token):** The application employs JWT for secure, stateless authentication. When a user logs in, the backend generates a JWT token, which the frontend stores and attaches to future requests to access protected resources.
- **Role-based Access Control:** Different user roles (such as admin and customer) likely determine access levels to certain endpoints and functionalities. For instance, admins can manage cars and bookings, while customers can only view and book cars.

5. Communication Between Frontend and Backend:

- **RESTful API:** The Angular frontend communicates with the Spring Boot backend via HTTP requests. The backend exposes RESTful endpoints that handle data-related operations like retrieving available cars, creating bookings, and managing users.
- **JSON Format:** Data is exchanged in JSON format, allowing structured and consistent data communication between the frontend and backend.

Database Schema

The screenshot displays a database management interface with two panels. The top panel shows SQL queries executed in a console:

```
1 • create database car_rental_db;
2 • use car_rental_db;
3 • show tables;
4
5
```

The bottom panel shows the results of the queries. The first query, `show tables;`, returned a list of tables in the `car_rental_db` database:

Tables_in_car_rental_db
bookacar
cars
users


The second query, `select * from bookacar;`, returned a single row of data:

id	book_car_status	days	from_date	price	to_date	car_id	user_id
1	0	7	2024-11-12 05:30:00.000000	5600000	2024-11-19 05:30:00.000000	1	2

```

5 • select * from cars;
6 • select * from users;

```

Result Grid									
Filter Rows:									
Edit: Export/Import: Wrap Cell Content:									
	id	brand	color	description	image	name	price	transmission	type
▶	1	Audi	Red	Car		puchku	800000	Automatic	Sports Car
*		NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```

6 • select * from users;

```

Result Grid					
Filter Rows:					
Edit: Export/Import: Wrap Cell Content:					
	id	email	name	password	user_role
▶	1	admin@test.com	Admin	\$2a\$10\$HrpHv/4dblI9rUtBtvB.H99boK0BEK/O...	0
	2	anjali Chauhan9462@gmail.com	Anjali Chauhan	\$2a\$10\$txpWepC/t3/vs1k3KssjR.Zj.QJr15F8QL...	1
*		NULL	NULL	NULL	NULL

Functionality

I. User (Customer) Functionalities

User Registration and Login:

- New users can register by creating an account.
- Existing users can log in with their credentials.
- JWT-based authentication secures user sessions.

Dashboard:

- Users can view available cars for rent, including details like model, make, year, name, image, brand and price per day.

Browse Available Cars:

- Users can view available cars for rent, including details like model, make, year and price per day.

Search Cars:

- Filters options may be available to help users find cars by brand, type, price range, or color.

Car Booking:

- Users can select a car, specify a rental period, and book the car if it's available.
- The application calculates the total price based on the rental duration.

Manage Bookings:

- Users can view their booking history, including current and past bookings.
- Options to cancel or modify a booking before the start date may be available

II. Common Functionalities

These functionalities are used by both customers and admins, enhancing the overall user experience.

- **Authentication & Authorization:**
 - Role-based access control with JWT ensures secure access to features. Customers have access to customer-specific features, and admins have access to admin features.
 - Password hashing and secure data handling protect user information.
- **Error Handling & Validation:**
 - Proper validation is performed on inputs (e.g., registration details, booking dates) to prevent invalid data entries.
 - Error messages and prompts help users understand issues, like invalid login attempts or unavailable booking dates.
- **Responsive and User-Friendly Interface:**

- The Angular frontend is optimized to work on various devices, providing a responsive and interactive experience for both customers and admins.
- **Logging & Monitoring:**
 - Logging important events (like logins, bookings, cancellations, and payments) provides insights and supports debugging.
 - Optional monitoring of API requests, errors, and performance helps improve system reliability.

Code Structure

Car-Rental-Spring:

1. `src/main/java/com/shounoop/carrentalspring`

The main package for the application, which contains sub-packages categorized by functionality.

- **configuration:**
 - Contains configuration files for security and CORS.
 - **JwtAuthenticationFilter.java**: Filter for handling JWT authentication.
 - **SimpleCorsFilter.java**: Configures CORS settings.
 - **WebSecurityConfiguration.java**: Configures Spring Security for the application.
- **controller:**
 - Holds the controllers that define the REST API endpoints.
 - **AdminController.java**: Handles requests related to admin functionalities.
 - **AuthController.java**: Manages authentication endpoints.
 - **CustomerController.java**: Manages customer-related requests.
- **dto (Data Transfer Objects):**
 - Classes used to transfer data between layers.
 - Examples include **AuthenticationRequest.java**, **AuthenticationResponse.java**, and **UserDto.java**, which handle authentication and user information data structures.
- **entity:**
 - Contains JPA entity classes representing database tables.
 - **BookCar.java**: Represents a booked car record.
 - **Car.java**: Represents a car entity.
 - **User.java**: Represents a user entity.
- **enums:**
 - Stores enumerations used within the application.
 - **BookCarStatus.java**: Enum to track booking statuses.
 - **UserRole.java**: Enum for user roles (e.g., ADMIN, USER).
- **repository:**

- Contains Spring Data JPA repositories for database operations.
- **BookCarRepository.java**, **CarRepository.java**, and **UserRepository.java**: Interfaces for CRUD operations on respective entities.
- **services:**
 - Contains the business logic layer, organized by module.
 - **admin:**
 - **AdminService.java**: Provides services related to admin operations.
 - **auth:**
 - **AuthService.java**: Handles authentication and authorization services.
 - **customer:**
 - **CustomerServiceImpl.java**: Implements customer-related services.
 - **jwt:**
 - **UserService.java**: Manages user-related services.
 - **UserServiceImpl.java**: Implementation of user services.
- **utils:**
 - Contains utility classes.
 - **JwtUtil.java**: Utility class for handling JWT token operations.
- **CarRentalSpringApplication.java**:
 - The main application class to run the Spring Boot application.

2. src/main/resources

- **application.properties**: Configuration file for application properties (e.g., database, security, etc.).

3. src/test/java/com/shounoop/carrentalspring

- Contains unit and integration tests.
- **CarRentalSpringApplicationTests.java**: Test class for testing the Spring Boot application.

4. Root Files

- **pom.xml**: The Maven configuration file for managing dependencies and build configurations.

Car-Rental-Angular:

- **src/app**
 - **auth**
 - auth.module.ts
 - **components**
 - login.component.ts
 - login.component.html
 - login.component.scss
 - login.component.spec.ts
 - signup.component.ts
 - signup.component.html
 - signup.component.scss
 - signup.component.spec.ts
 - **admin**
 - admin.module.ts
 - **components**
 - dashboard.component.ts
 - dashboard.component.html
 - dashboard.component.scss
 - dashboard.component.spec.ts
 - add-car.component.ts
 - add-car.component.html
 - add-car.component.scss
 - add-car.component.spec.ts
 - car-booking.component.ts
 - car-booking.component.html
 - car-booking.component.scss
 - car-booking.component.spec.ts
 - update-car.component.ts
 - update-car.component.html
 - update-car.component.scss
 - update-car.component.spec.ts
 - **customer**
 - customer.module.ts
 - **components**
 - book.component.ts
 - book.component.html
 - book.component.scss
 - book.component.spec.ts
 - dashboard.component.ts
 - dashboard.component.html
 - dashboard.component.scss
 - dashboard.component.spec.ts
 - my-bookings.component.ts
 - my-bookings.component.html
 - my-bookings.component.scss

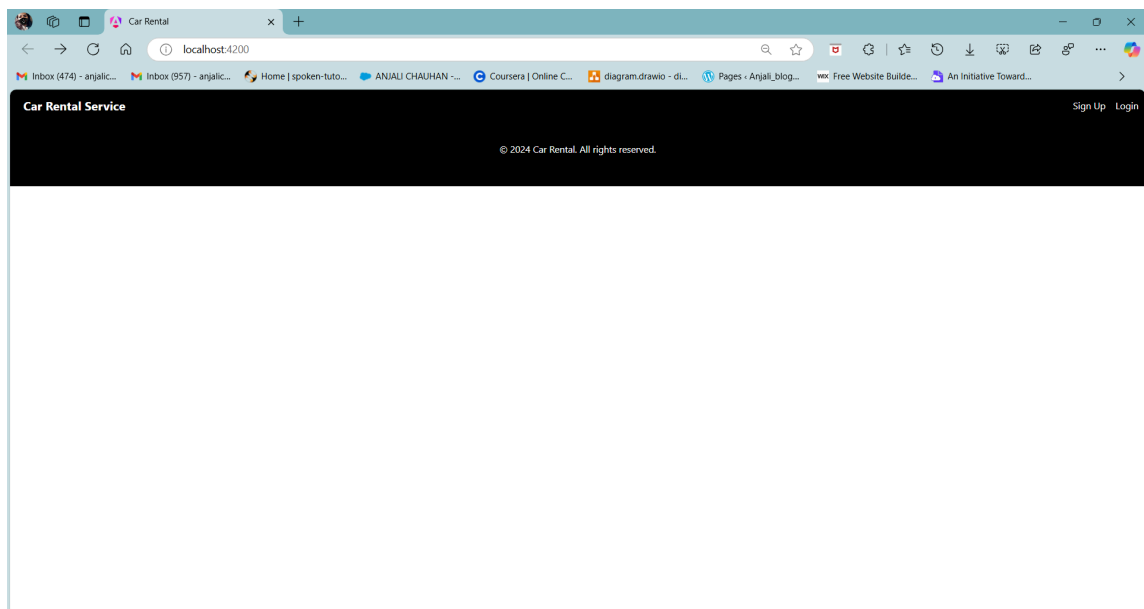
■ my-bookings.component.spec.ts

Additional Files/Folders:

- app.component.ts
- app.component.html
- app.component.scss
- app.component.spec.ts
- app.module.ts
- environments
- assets
- index.html
- main.ts
- polyfills.ts
- styles.scss
- tsconfig.app.json
- angular.json
- package.json

Snapshots

Home Page



Login Page

Car Rental Service

Sign Up Login

Login

Email Address

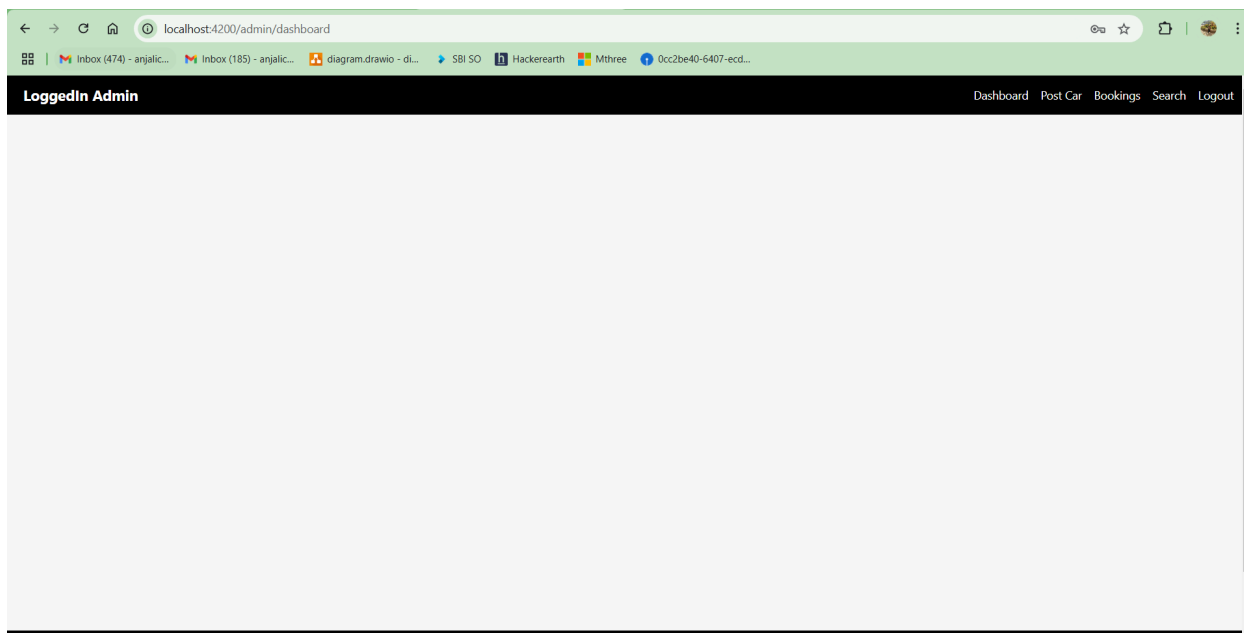
admin@test.com

Password

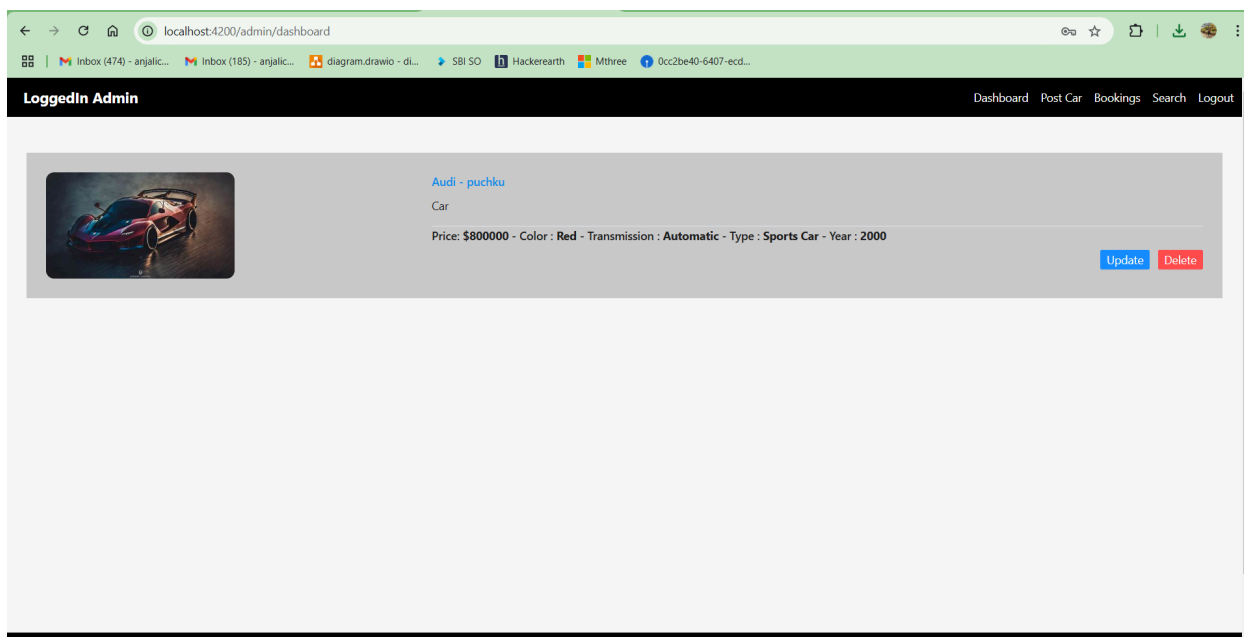
Login

You don't have an account? [Register Now](#)

Admin Dashboard Page



Bookings Page



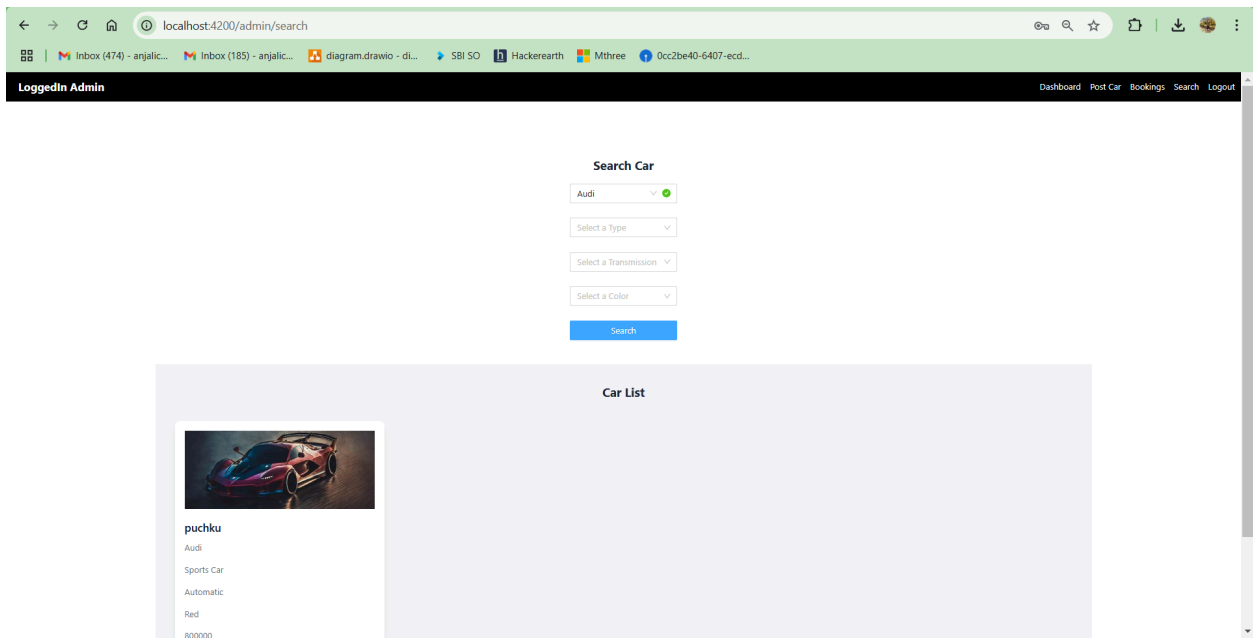
Post Car Page

The screenshot shows a web browser at the URL `localhost:4200/admin/car`. The page is titled "LoggedIn Admin" and has a navigation bar with links for "Dashboard", "Post Car", "Bookings", "Search", and "Logout". The main content area is titled "Post Car" and contains a form for adding a new car. The form includes the following fields:

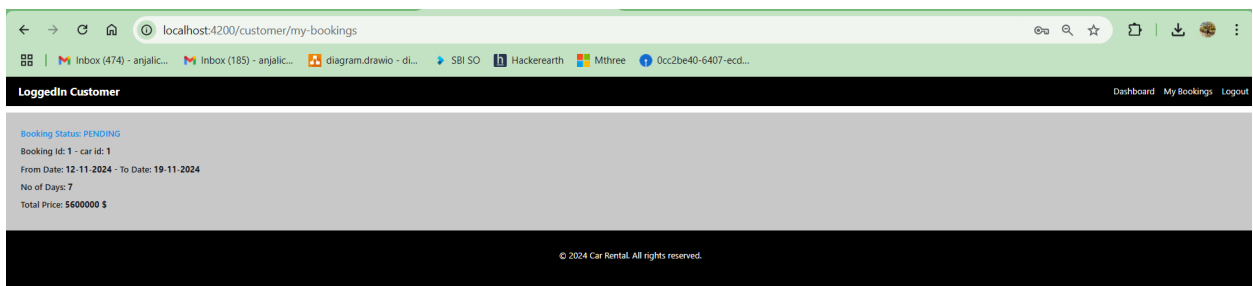
- Choose File** (button) and **No file chosen** (text)
- Select a Brand** (dropdown menu)
- Name** (text input)
- Select a Type** (dropdown menu)
- Select a Transmission** (dropdown menu)
- Select a Color** (dropdown menu)
- Modal Year** (text input)
- Price** (text input)
- Description** (text input with a rich text editor icon)

At the bottom of the form, there is a blue button labeled "Post Car".

Search Car Page



Customer Bookings Page



Registration Page

Car Rental Service Sign Up Login

Start Your Rental Adventure.

Email Address

Password

Confirm Password

FullName

Register

Already have an account? [Login](#)

Setup and Deployment

1. Download entire code in zip format , unzip it then go to your mySql and create a database called car_rental_db and select it

create database car_rental_db;

use car_rental_db;

2. Open the spring boot directory in your editor open its terminal and use maven to install all the dependencies by running the command

mvn clean install

3. Make sure that username and password of your my sql is " root " and if not then go to src then main and then resources folder their open application.properties file and change the username and password as per your my sql After that start the back end server
4. Open the angular directory in your editor then open its terminal and run install command to install all node dependencies npm install
5. Then run the angular project using

ng serve

6. go to localhost:4200 in your browser there click on login and login using given credentials to login as admin

Username: admin@test.com

Password: admin

Then you will see an empty dashboard go to upload cars there and upload one or two cars with whatever details and image , then those will be visible on you dashboard , then logout and register yourself through sign up using any email and pwd , then you will get to the customer dashboard from where you will be able to access the cars uploaded by admin and you can book them, also can see you bookings. Rest of the functionalities you will understand by toggling around.

Future Enhancements

Future enhancements for the car rental service project could include advanced search and filtering, real-time car availability, and dynamic pricing with discounts. A mobile app could enhance accessibility with geolocation for car pick-up/drop-off and push notifications. Security features like multi-factor authentication, GPS tracking, and multiple payment gateways would improve reliability and ease of use. An enhanced admin dashboard could provide insights and streamline inventory management. Additional features like user feedback, vehicle maintenance tracking, multilingual support, and insurance options would improve the user experience and make the platform more robust and user-friendly.

Conclusion

In conclusion, the car rental service project provides a strong foundation for managing rentals, customer interactions, and administrative tasks. With potential future enhancements such as advanced search options, dynamic pricing, mobile app support, and additional security and payment features, the platform can become even more user-centric and efficient. These improvements would not only enhance the user experience but also increase operational flexibility, making the service adaptable to various market needs and better positioned for growth.