Crazy Car Rental Application Report – Fergal Hughes 20054847

The system is a full-stack web application developed using React, NodeJS, Express, Bootstrap, and MySQL. The aim of the system is to allow customers to create an account, log in, browse available cars and make rental bookings through a cleat and responsive user interface. An administrator can access a protected dashboard to monitor rentals

This application follows a client-server architecture. The front end is built using React and React Router to enable multipage experience without page reloads. Communication between the front end and the back end is handled with Axios and Restful API endpoints. The back end is built with NodeJS and Express, which manages the business logic, validation and database interactions. A MySQL database is used to store all the persistent data.

See ERD folder for database design.

My approach

The database was designed first to ensure that entities and relationships were clearly defined before any application logic was implemented. This helped me establish a strong structure and reduce issues later. Primary keys and foreign keys were planned to enforce relationships between customers, cars and rentals to ensure good data integrity.

The application follows a clear separation of concerns. The front end is built using React and is responsible for the presentation, interaction, and basic client-side validation. It does not contain any business logic or direct database access. The back end is developed using NodeJS and express handles all the business logic, sever-side validation, and communication with the database. This separation improves maintainability, scalability, and security.

The overall architecture follows the client-server model. React communicates with the Express server using Axios to send HTTP requests to RESTful Api endpoints. Express process these requests, performs validation, and executes queries using the database. The database returns results to the server, which are then sent back to the React front end as a JSON response. This structure provides a clear flow of the data from the user to the database and back.

Authentication and Security

The system includes user authentication using JSON web tokens (JWT). Users create and account using email and password. Passwords are securely hashed using bcrypt before being stored on the database. During login, the password entered by the user is compared against the hash password in the database.
If authentication is successful, a JWT token is generated and returned to the front end. This token is stored in local storage and is included in protected API requests. Middleware is use on the server to verify the tokens and restrict access to protected routes such as the admin dashboard.
Role-based access control is implemented to differentiate between normal users and admin users. Admin users can access additional functionality such as viewing current rental. This improves security and ensures only authorised users can access sensitive information.

Front-End

The front end was developed using React functional components and React Router. State management is handled using React hooks such as useState and useEffect. The navigation bar dynamically updates based on the authentication state, showing different options depending on which type of user is logged in. Bootstrap was used to ensure the interface is responsive across different screen sizes. Forms include client-side validation to ensure required fields are completed before submission.

Back-End

The back end is structured using simple MVC pattern. The models handle the database queries. The controllers hand the request processing and the business logic. Routes define the API endpoints, and the middleware handles the authentication and authorization. Server-side validation ensure that required data is present before performing database operations. Errors are handled in the correct manner and meaningful JSON responses are returned to the client.

Conclusion

The Crazy Car Rental application demonstrates a full-stack implementation using modern web technologies. The system integrates front-end and back-end components effectively while maintaining a clear separation of concerns. Security is handled correctly and routes are protected for better integrity. The project demonstrates database design. RESTful API development, authentication handling and responsive front-end development within a client-server architecture