**Car Rental System**

|  |  |
| --- | --- |
| 1- **Abdullah Ismael Ahmed** | **Project lead, Team**  **management - testing and Front end development & ui /ux (Admin side)** |
| 2- **Mayar Mohamed Ebrahim** | **Front end development & ui /ux (client side)** |
| 3- **Ibrahim Hussien AbdelRazek** | **testing and debugging** |
| 4- **Kerlous Nasser Shehata** | **Back end development** |
| 5- **Mohab Marzouk Gaber** | **Front end developer ( integration)** |

Project Overview

Drifter is a full-stack web application designed to meet the comprehensive needs of a car rental company. It offers powerful management tools for handling rentals, car inventory, and maintenance, while also providing a user-friendly interface for clients to search, browse, and book cars.

Built using the MERN stack (MongoDB, Express.js, React, and Node.js), Drifter simulates a real-world car rental system inspired by "Ramy Rent" company . The application aims to streamline the company's operations, making the management of rentals and services more eﬃcient and the client booking experience smoother.

Project Goal :

The aim of the Drifter project is to help "Ramy Rent" transition away from traditional paper-based processes and embrace modern digital solutions. By adopting this web application, the company can keep pace with advancements in computing technology, enhancing their operational eﬃciency. Drifter enables the digital management of rental records, car

##### 2

maintenance logs, and client information, reducing manual paperwork and minimizing errors. This shift will not only optimize daily workﬂows but also position the company for future growth by making their services more accessible and streamlined.

“Ramy Rent” Requirements:

The requirements from "Ramy Rent" were centered around digitizing the car rental process and eliminating the cumbersome paperwork associated with it. The main objective was to have a system that not only stores all the relevant data in a structured manner but also allows for easy data management and export in various ﬁle formats for administrative purposes. The key requirements included:

###### Data Organization and Storage:

* + A reliable storage solution to keep all the information in a physical or cloud database, accessible whenever needed.
  + Ability to export data in different formats for administrative review or auditing purposes.
  + Ability to visually see the latest data and separate page for each speciﬁed requirement and **able to ﬁlter and sort and search through tables**

###### Data Separation:

* + **Cars:** Information for each car should include:
    - Model, year, last oil change date, kilometers driven, rental price, license plate, image, and owner's details.
  + **Rents:** Details about each rental transaction should cover:
    - Customer's name, phone number, identiﬁcation number, rented car, rent start and return dates, trust agreement status, total rental price, amount paid, and remaining balance.
  + **Maintenance:** Maintenance records should track:
    - Workshop name, car being serviced, description of the maintenance work, total cost, amount paid, and remaining balance.

##### 3

###### Customization and Expandability:

* + The solution was to be fully customizable, allowing for the addition of new features and functionalities in the future as the company's needs evolve.
  + It was designed as a generic software solution that could be adapted to suit any other car rental company, not just "Ramy Rent." The architecture was built to accommodate various workﬂows, making it ﬂexible enough for different businesses in the car rental industry.

# Application Architecture and Technology

The application is a full-stack MERN (MongoDB, Express, React, Node.js) app that leverages modern technologies to deliver a seamless user experience.

**Frontend**: The UI is built primarily using React.js, utilizing JSX and hooks for component logic. State management is eﬃciently handled through Redux, allowing for a clear separation of concerns. Each component imports necessary slices and pre-created functions from the Redux store, implementing event handlers within the returned JSX. This structure ensures effective UI updates while maintaining a clean integration with the backend.

**Backend**: The backend is developed using Express, a Node.js framework that manages routing and database interactions. It handles various HTTP methods (GET, POST, PUT, DELETE, PATCH) for data manipulation. MongoDB serves as the database, storing data securely and enabling real-time updates to the UI as data changes. This architecture ensures a responsive and dynamic application that eﬃciently meets user needs.

##### 4

Together, these elements create the magic wound behind this application, ensuring a responsive and dynamic platform that effectively meets user needs.

**Frontend Technologies**

* **React.js :** A JavaScript library for building user interfaces that simpliﬁes the development process by allowing developers to separate application logic into reusable components, each with its attached CSS.
* **Use:** React was the primary technology utilized to implement this application, leveraging its hooks and JSX to serve as the core building blocks for creating all components.

**React Router**: A part of React that enables dynamic routing for seamless navigation, allowing users to move between pages without reloading the entire application. It creates a network covering all pages, facilitating smooth redirection to any of them.

**Use**: React Router was primarily utilized in App.js to manage the outer structure for page navigation. By moving away from traditional <a> and href links, it revolutionizes web application development. Its integration with Redux allows it to wrap all pages with the store, providing necessary data from slices and applying required middleware.

###### Main Components:

* **BrowserRouter**: The primary router that uses the HTML5 history API to keep the UI in sync with the URL.
* **Provider**: Connects the Redux store to the React application.
* **Routes**: Deﬁnes the relationship between the URL path and the components to render.
* **Router**: The core component that enables the routing functionality.
* **Path**: Speciﬁes the URL path that should trigger the rendering of a speciﬁc component.
* **Element**: The component that is rendered when the route matches.
* **Index**: A special route that renders its element when the parent route path matches exactly. It was primarily used to ensure that the **Layout** component in the administration

5

section is visible for other routes within it, providing a consistent structure and navigation experience across those routes.



**Redux**: A core component that centralizes state management across the application, reducing code duplication and facilitating eﬃcient state handling. In this project, Redux plays a crucial role, especially in the administration section, by creating three main slices: **carSlice**, **rentSlice**, and **maintainSlice**.

The fundamental idea of Redux is to unify state management, allowing components to create localized, one-use states while managing shared states centrally.

**Caution: Redux is intended for general states that need to be accessed across multiple components.**

##### 6

Using Redux Toolkit, slices combine actions and reducers in a single entity, eliminating the need for separation. Within components, the useDispatch hook enables sending actions to the respective slices, which listen for these actions and execute the appropriate reducers. The reducer's role is to receive an action and the current state, returning the new state as needed.

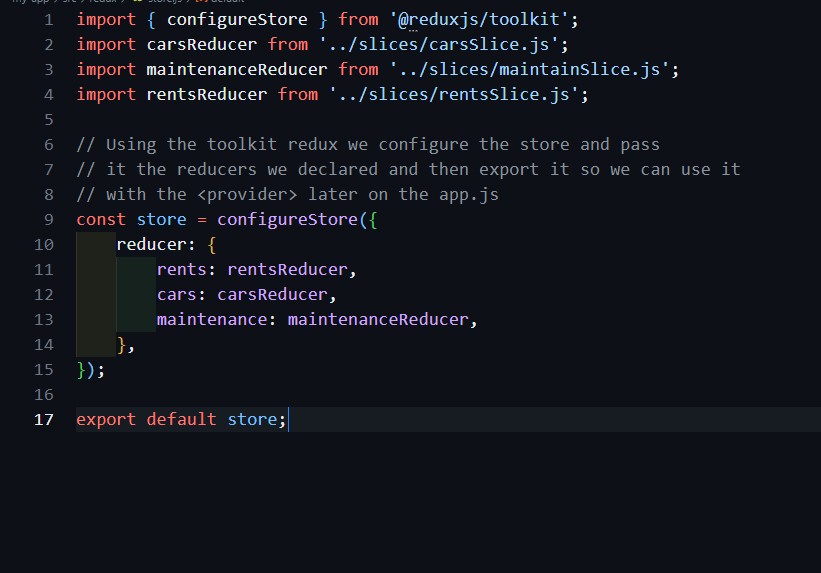
Components can then access the centralized state through the useSelector hook. All slices are linked to the store, allowing components to tap into shared state seamlessly. This powerful state management capability enhances the project's architecture, making Redux an invaluable tool in this application. I'm eager to discuss Redux further, as I've enjoyed working with and understanding its nuances!

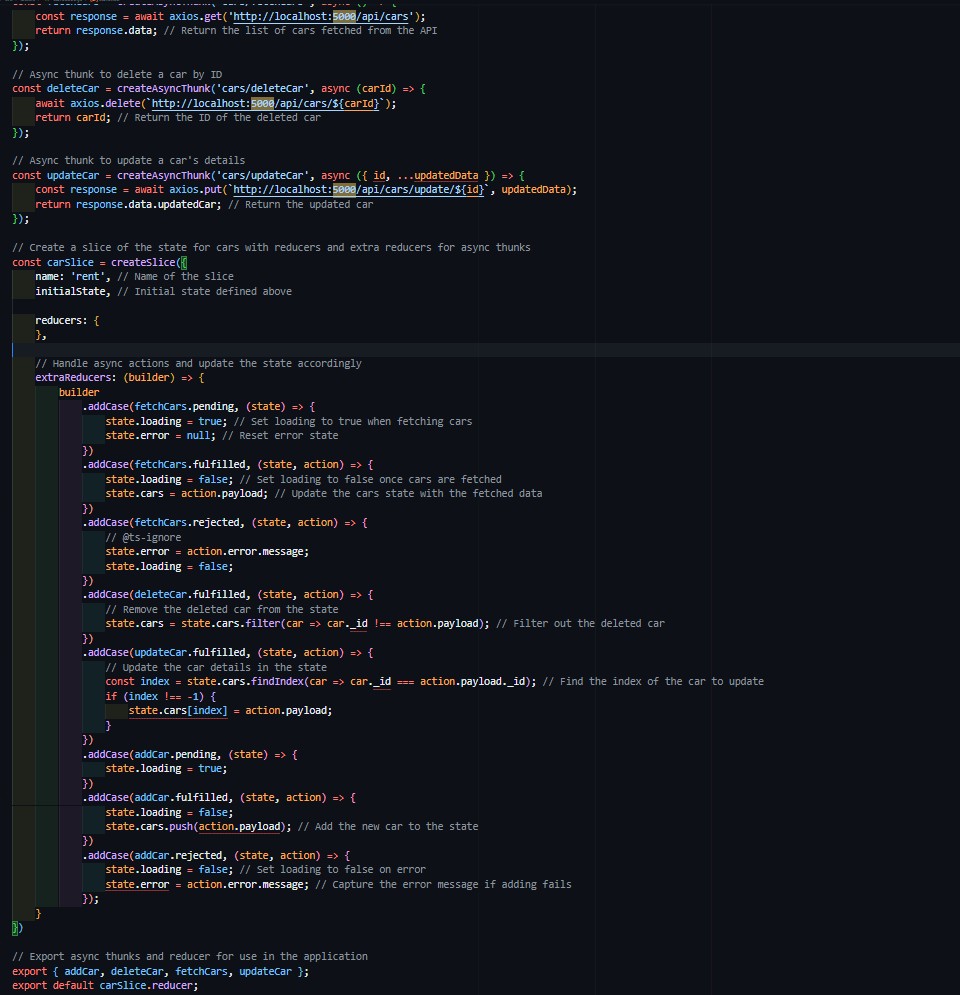
1. **conﬁgureStore Sets up the Redux store with good defaults and automatically applies middleware.**
2. **createSlice Creates a slice of the state with associated reducers and actions, reducing boilerplate code.**

In each slice you can deﬁne the name and the initial state and the reducers also

1. **createAsyncThunk Deﬁnes asynchronous actions and automatically dispatches lifecycle actions (pending, fulﬁlled, rejected).**

##### 7

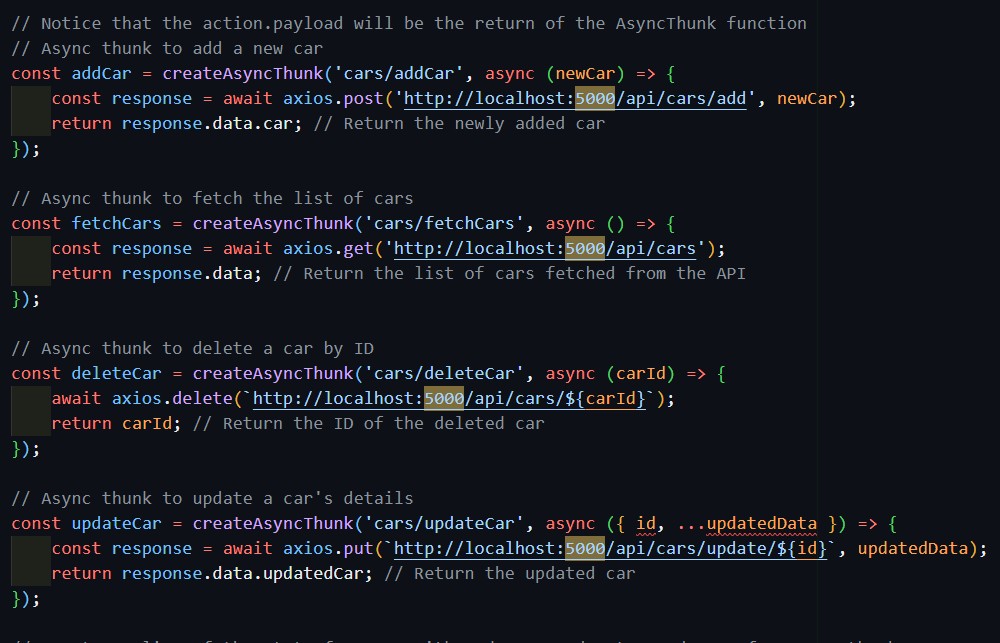




8

##### 9

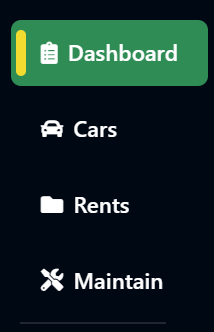
* + **Axios**: Promise-based HTTP client for API requests.
  + Use: in this application for its superior functionality compared to the native fetch API. Axios simpliﬁes the process of fetching, adding, updating, and deleting data by automatically handling JSON parsing, allowing for easier error handling through response interceptors, and supporting request cancellation and timeout options. This ensures eﬃcient communication with the server while seamlessly updating the Redux slices with the action payload



* + **Font Awesome**: Icon library for scalable vector icons

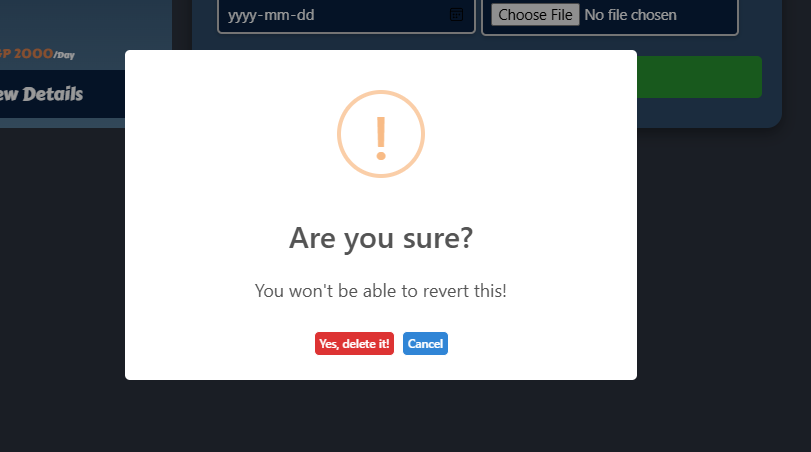
USE : Font Awesome was utilized in this project to enhance the user interface by incorporating a variety of icons, adding visual appeal and functionality to the components through its CDN integration, thus elevating the overall user experience.

##### 10



* + **SweetAlert2**: Creates customizable alert popups

USE : SweetAlert2 was implemented in this project to replace standard browser alerts, offering enhanced aesthetics and interactivity, thereby improving user engagement and providing a more polished experience within the application.

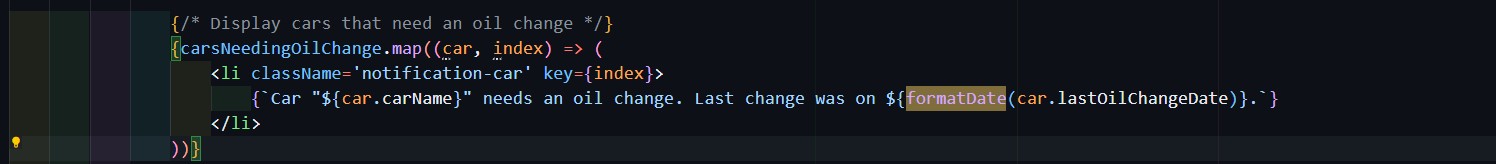
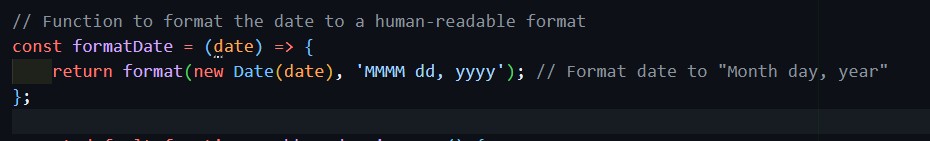


.

##### 11

* + **Date-fns**: Simpliﬁes date manipulation and formatting

USE : used mostly in forms validation to change received date into more valid format

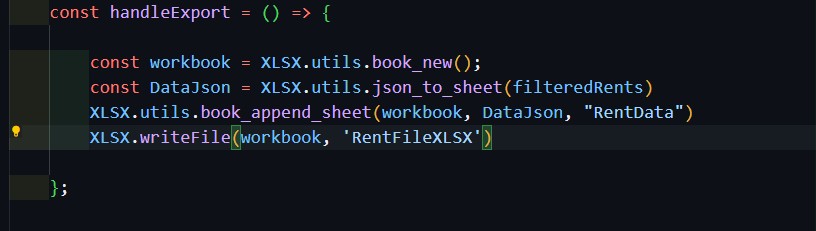


* + **JS Cookie**: Handles cookie management.

USE : it was used in the application store temporary data about logging in and out from the application

* + **xlsx**: Parses and writes Excel ﬁles.

USE : was used in the Rent components to export the table as a valid csv ﬁle in a table format so it can be printed or stored



* + **Jest**: A JavaScript testing framework that enables the creation of unit tests and integration tests for applications. **Use:** It was used to simulate user interactions and verify that components behave as expected during testing.

##### 12

* + **Redux-mock-store**: A utility for creating a mock Redux store in tests. **Use:** It was used to simulate the Redux store and dispatch actions during tests without needing a full Redux setup, allowing for eﬃcient state management testing.



* + **Web-vitals**: A library that measures key performance metrics of web applications. **Use:** It was used to track performance metrics such as loading speed, responsiveness, and overall user experience, helping to optimize the application's performance.

#### Backend Technologies

* + **Express:** A minimalist web framework for building APIs. **Use:** It was used to create server-side routes and handle incoming requests, providing a robust foundation for the backend. Mainly utilized to create routes using the Express.Router, it deﬁnes the logic for handling CRUD operations (Create, Read, Update, Delete).
  + Express enables middleware integration, allowing for functionalities like request logging, body parsing, and error handling. The framework uses a routing system that listens for HTTP requests and triggers corresponding handler functions, making it easy to structure

##### 13

and manage the application's ﬂow. In the server.js ﬁle, app.use() is used to set up middleware and deﬁne routes that respond to speciﬁc URL paths, facilitating the application's interaction with client requests and responses.





##### 14

**Mongoose:** An Object Data Modeling (ODM) library for MongoDB data modeling. **Use:** It was used to deﬁne schemas and interact with the MongoDB database, simplifying data manipulation and validation.

Mongoose allows developers to create models based on deﬁned schemas, which outline the structure and data types of documents within a collection. Key concepts include:

1. **Schemas:** Deﬁne the structure of documents, including ﬁeld types, validation rules, and default values.
2. **Models:** Created from schemas, models provide an interface for interacting with the corresponding MongoDB collections, enabling CRUD operations.
3. **Queries:** Mongoose supports a rich query language, allowing developers to easily retrieve and manipulate data using methods like .find(), .findById(), and

.update().

##### 15

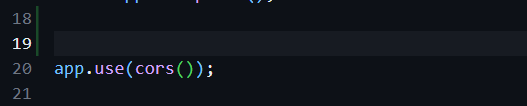
1. **Middleware:** Mongoose provides middleware (pre and post hooks) for performing operations before or after certain actions, like saving or removing documents.
2. **Population:** This feature allows for easy joining of documents across collections, enhancing the capability to work with related data.
   * **Mongoose Sequence:** A plugin that auto-increments ﬁelds for unique identiﬁers. It was used to automatically generate unique numeric identiﬁers for documents in the MongoDB database, simplifying record management.



* + **CORS:** A middleware that enables Cross-Origin Resource Sharing for secure requests. **Use:** It was used to allow or restrict resources on the server to be requested from another domain, enhancing security and usability.

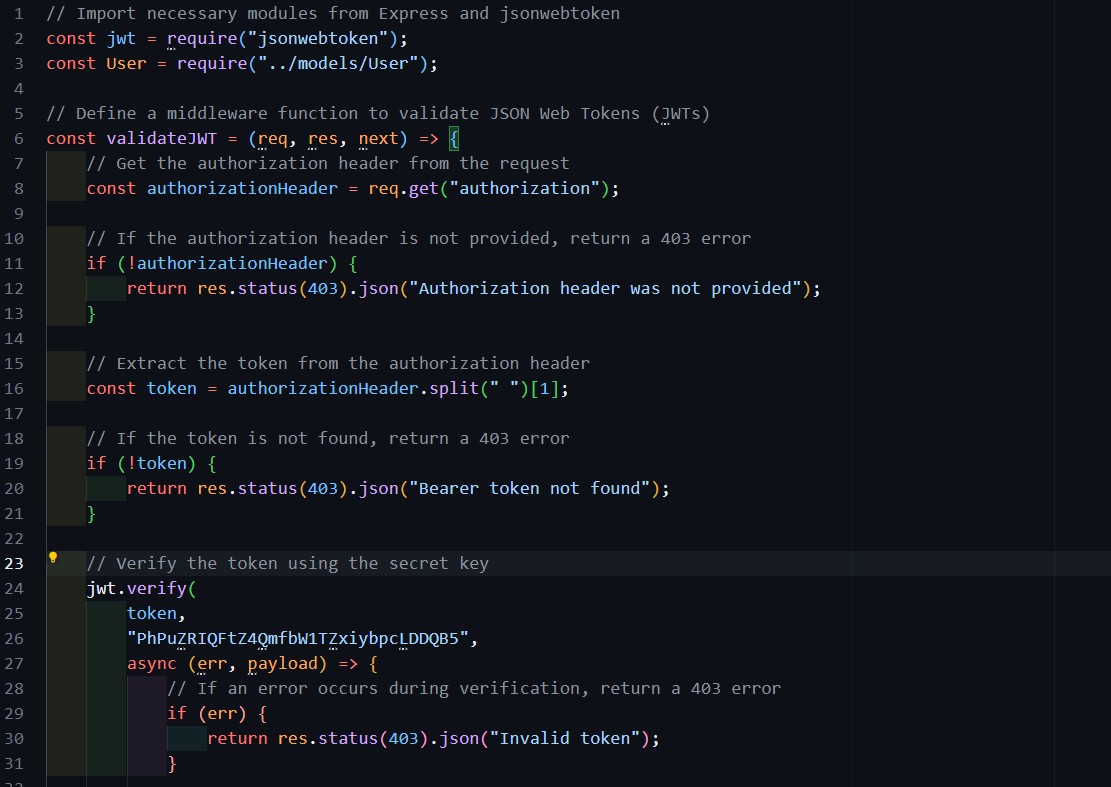
##### 16

Using CORS is essential because web applications often make requests to APIs hosted on different domains. By implementing CORS, developers can specify which origins are allowed to access their resources, helping to prevent malicious activities such as cross-site request forgery (CSRF) and ensuring that only trusted domains can interact with the server. This not only enhances security but also improves user experience by allowing legitimate cross-origin requests to function seamlessly.



* + **Jsonwebtoken:** A library that manages user authentication via tokens. **Use:** It was used to generate and verify JSON Web Tokens (JWT) for secure user authentication and session management.

JSON Web Tokens are a compact and self-contained way to represent claims between two parties. By using this library, the application can securely transmit information, ensuring that only authenticated users can access certain resources. It was used as a **middleware** to ensure the login that for admin and clients and ensure that different users has different authorities



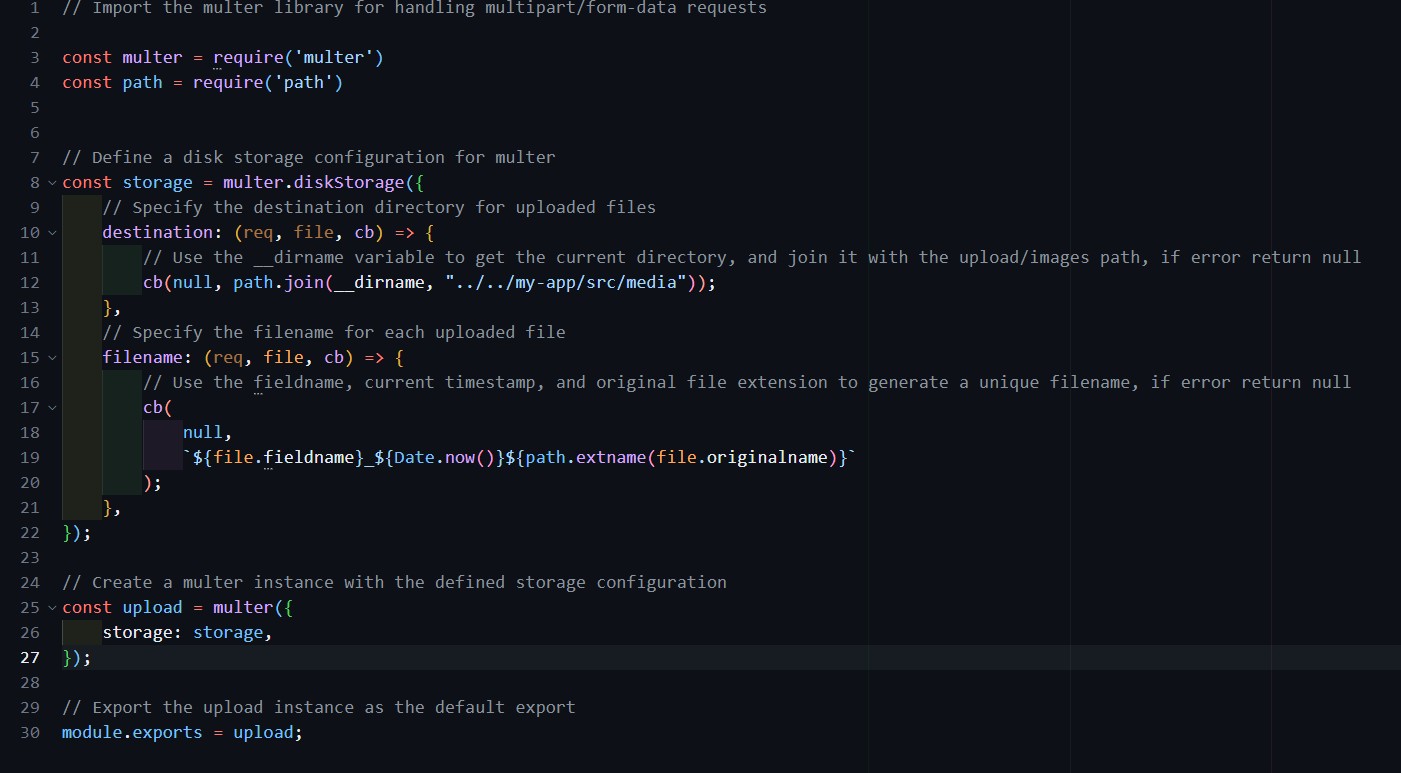
##### 17

* + **Bcrypt:** A library for hashing passwords for secure storage. **Use:** It was used to hash user passwords before storing them in the database, ensuring sensitive information remains protected. During the login process, the entered password is hashed and compared to the saved hashed version in the database, providing an additional layer of security against unauthorized access and data breaches. This approach prevents the exposure of raw passwords, safeguarding user credentials even if the database is compromised.



##### 18

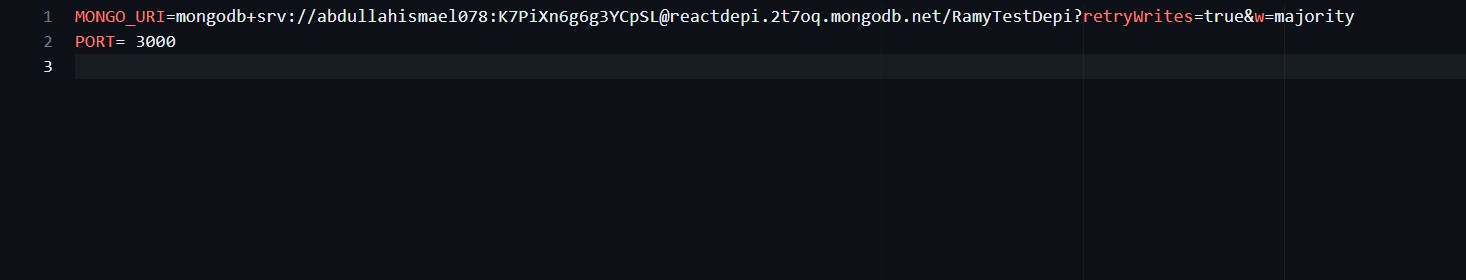
* + **Multer:** A middleware for handling ﬁle uploads. **Use:** It was used to manage multipart/form-data, enabling the upload of ﬁles and images in requests, facilitating the storage of user-uploaded content on the server. Multer simpliﬁes the process of handling ﬁle data in HTTP requests, allowing for easy integration into the application while providing options for ﬁle size limits, storage conﬁgurations, and ﬁle type ﬁltering. Mainly in the ‘upload’ folder that would handle any uploaded ﬁles and also

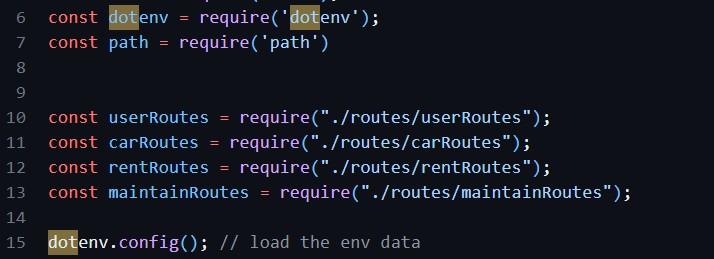


deleting them if they are deleted inside the app

##### 19

* + **Dotenv:** A module that loads environment variables from a .env ﬁle. **Use:** It was used to manage conﬁguration settings and sensitive information, such as database connection strings, in a secure manner, preventing hard-coded values in the source code and enhancing security. By centralizing environment-speciﬁc variables, Dotenv simpliﬁes application conﬁguration across different environments,





##### 20

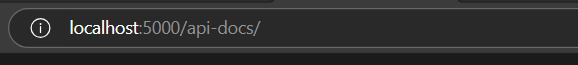
The architectural framework of this full-stack application exempliﬁes the effective integration of modern technologies and best practices in web development. By utilizing the MERN stack—MongoDB, Express, React, and Node.js—the application beneﬁts from a seamless ﬂow of data between the frontend and backend, ensuring a responsive user experience. Key libraries such as Redux, Axios, and Mongoose enhance state management, API interactions, and data modeling, while tools like SweetAlert2 and Font Awesome enrich user engagement through visually appealing interfaces.

The choice of middleware, including CORS, Bcrypt, and Multer, underscores a commitment to security and functionality, facilitating secure ﬁle uploads and user authentication. Environment management through Dotenv ensures that sensitive information is handled securely across various deployment environments. Overall, this architectural approach not only promotes maintainability and scalability but also aligns with the latest standards in web application development, setting a solid foundation for future enhancements and features.

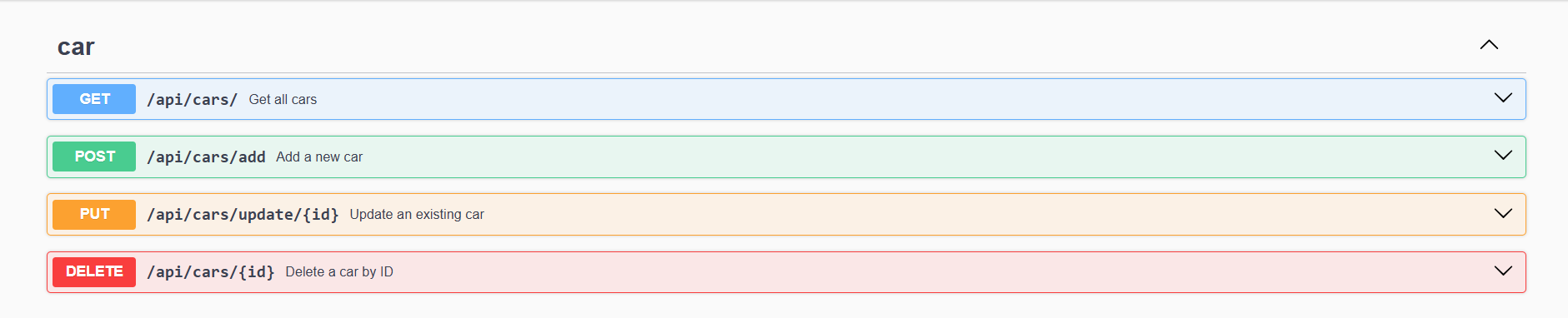
21

# Api

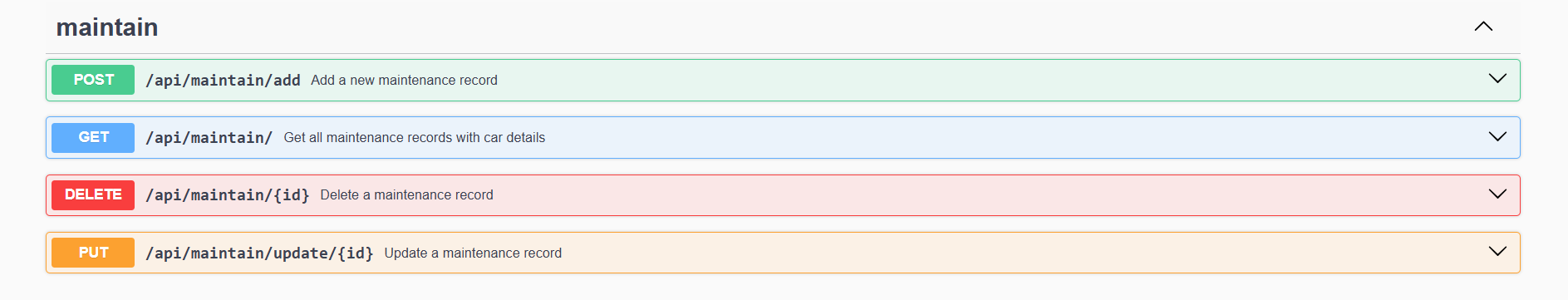
[**click here**](https://depi-react-final-83rt.vercel.app/swagger-ui.html)



**Car**



**Maintain**

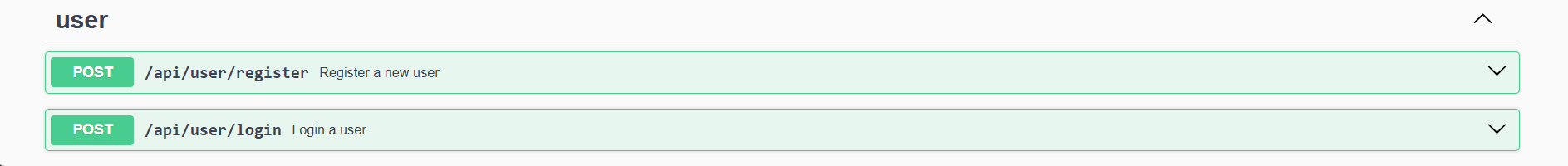


**Rent**



22

**User**



25

Testing

The testing process for Drifter involved using "Jest" due to its seamless integration with React, making it the primary tool for testing the application's core components. The testing covered 14 different components, including those on the admin and client sides, simulating real-world usage scenarios.

The tests focused on three key aspects:

* 1. **Rendering**: Ensuring each component renders correctly and displays the expected output.
  2. **Functionality**: Verifying that each component performs its intended function, such as validation, accurately.
  3. **Error Detection**: Identifying any uncaught errors during development to ensure a robust and stable application.

These tests helped guarantee that the components were working as intended and contributed to a smoother and more reliable user experience.

#### Main Keywords Explained:

* **describe**: Groups related tests together for a speciﬁc component or functionality.
* **it or test**: Deﬁnes an individual test case.
* **render**: Renders a component for testing.
* **fireEvent**: Simulates user actions (e.g., clicks, typing).
* **expect**: Asserts the expected outcome of a test.

#### Test Stages Overview:

* **Setup**: Import required modules and render the component.
* **Mocking**: Mock necessary functions like API calls and Redux actions (stores & slices ).
* **Simulating Actions**: Trigger events (e.g., button clicks, form submissions, triggering events).

##### 26

* **Assertions**: Check if the component behaves as expected (e.g., adding, removing updates state, makes API calls).

Sample Walkthrough :

it('handles delete confirmation correctly', async () => { render(

<**Provider** store={store}>

<**Maintain** />

</**Provider**>

);

// Simulate clicking the delete button fireEvent.click(screen.getByText(/Delete/i));

// Confirm delete action was dispatched

expect(store.getActions()).toContainEqual (deleteMaintenance(expect.any(String)));

});

1. **Render the Component**: The Maintain component is rendered, wrapped in a Redux

<Provider> to access the store.

1. **Simulate User Action**: The test simulates a user clicking the "Delete" button using

fireEvent.click.

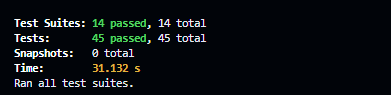
1. **Check Expected Behavior**: After the delete click, the test veriﬁes that the correct Redux action (deleteMaintenance) was dispatched.

This is the main concept of testing components is

###### Render the component → Simulate the needed event → Ensure the event behaves as expected

This way we were able to test the main components we have tested 14 components with total of 45 tests

27



**In general the project goal was achieved and the the user requirements were all implemented and tested and ready for real use based on usual inputs and outputs the Application ﬂows through the next**

**Inputs:**

###### User Inputs on the Frontend

* + - **User Information:** Details provided by customers, such as name, phone number, and National ID, while renting a car.
    - **Car Selection:** Customers choose a speciﬁc car from the available inventory, selecting based on preferences like car model, rental dates, and features.
    - **Rental Information:** Dates for renting and returning the car, mileage, and any other rental conditions.
    - **Login and Authentication:** User credentials such as email and password for logging into the system.
    - **Administrative Actions:** Admins can add, update, or remove cars, rentals, and maintenance records via the admin dashboard.
    - **File Uploads:** Images or documents related to car maintenance or rentals, uploaded by the admin.

###### Backend API Requests

* + - **Create, Read, Update, Delete (CRUD) Operations:** Requests sent to the backend to perform various operations, such as adding a new car, fetching rental records, or updating customer details.
    - **Authentication Tokens:** JSON Web Tokens (JWT) sent with requests to verify user sessions and protect API endpoints.

28

###### Conﬁguration Files and Environment Variables

* + - **Database Connection String:** The MongoDB Atlas connection URL speciﬁed in the .env ﬁle.
    - **Vercel Conﬁguration:** Deployment settings speciﬁed in the vercel.json ﬁle.
    - **API Endpoints:** URLs for communicating with the backend services.

#### Outputs:

###### Frontend User Interface

* + **Web Pages and Components:** The React application dynamically renders web pages, including the rental form, car listing, user dashboard, and admin panel, With all functions needed like exporting , ﬁltering , searching the tables
  + **Alerts and Notiﬁcations:** Pop-ups using SweetAlert2 to notify users of successful actions (e.g., car rented successfully) or errors (e.g., invalid login).
  + **Form Validations:** Error messages displayed when form inputs are invalid, such as missing required ﬁelds or incorrect data formats.
  + **Live Updates:** Real-time updates to the UI based on state changes (e.g., showing newly added cars or updating rental statuses).

###### Backend Responses

* + **JSON Data:** Responses to frontend requests containing data such as car lists, rental history, or user information.
  + **Success and Error Messages:** Feedback sent to the frontend to indicate whether actions were successful or if errors occurred (e.g., "Car added successfully" or "Invalid credentials").
  + **Authentication Tokens:** JWTs provided to authenticated users for session management.

###### Database Entries and Modiﬁcations

* + **Car Records:** Stored details about cars in the inventory, including make, model, plate number, and maintenance status.
  + **Rental Records:** Information about each rental, such as the car rented, dates, customer details, and payment status.
  + **User Information:** User registration data, hashed passwords for secure storage, and authentication tokens.

29

* + **Maintenance Logs:** Records of maintenance work performed on the vehicles.

###### Deployment and Logging Outputs

* + **Deployment Logs:** Vercel logs showing the status of deployments and any issues encountered.
  + **Server Logs:** Logs for monitoring API requests, errors, and server status.
  + **Performance Metrics:** Web vitals data collected to measure the application's performance.

Together, these inputs and outputs ensure the system functions correctly, enabling users to interact with the application and perform necessary operations while administrators manage the car rental services effectively.

# Deployment and Running Walkthrough

## - Consultation

We initially **collaborated with the stakeholders** at *Ramy Rent* company to understand their requirements and vision for the application. This involved gathering detailed insights into the features they needed, the desired functionality, and how they wanted various aspects of the system to work. We discussed user roles, workﬂows, and the key features they wanted to see in the car rental management system, including the rental processes, car maintenance tracking, and administrative functionalities.

These early discussions guided the development roadmap, helping us align the project's goals with the stakeholders' expectations. This alignment ensured that the ﬁnal product would meet their business needs and provide a user-friendly experience for both administrators and customers.

30

## - Constructing the Vanilla project

* + - <https://github.com/Blaxinoss/Depi>github repo
    - [Ramy Rent - Home (blaxinoss.github.io)](https://blaxinoss.github.io/Depi/) Vanilla project page hosted on gituhb

We began the development process with **UI Designing**, where we ﬁrst laid out the visual appearance and user interface structure using tools like Figma. This initial step ensured we had a clear vision of the user experience, helping us visualize the layout, colors, fonts, and interactive elements. Once the UI design was ﬁnalized, we proceeded to the next phase: **HTML and CSS Implementation**. This stage involved converting the design mockups into static web pages, focusing on creating responsive and visually appealing layouts.

After setting up the basic structure and styling, we moved on to **JavaScript Integration** to add interactivity to the site. This step allowed us to implement core functionalities such as form validation, dynamic content, and event handling, enhancing the overall user experience by making the web pages interactive and functional.

## - Starting the React project

After establishing a solid foundation with the static HTML, CSS, and JavaScript, we transitioned to **React** to take the project to the next level. We systematically converted the static web pages into dynamic **React components**, breaking down the interface into modular and reusable parts. This transition not only improved code maintainability but also enabled us to better manage the application's state and UI behavior.

Using **React's JSX** syntax, we were able to blend HTML and JavaScript seamlessly, allowing for more interactive and dynamic rendering of the content. Additionally, incorporating **React hooks** like useState and useEffect brought reactive data handling into the application, ensuring that the UI would automatically update in response to state changes. This modular approach made it easier to debug, extend, and scale the application, as each component was responsible for its own logic and presentation.

##### 31

The move to React also paved the way for integrating **Redux** later in the development process, enabling a centralized approach to state management across the entire application. This set the stage for a more powerful and responsive user experience, bringing the project closer to a modern, scalable web application.

## - Github Usage

Throughout the development journey, we consistently utilized **Version Control** tools like **Git and GitHub** to manage the codebase. This enabled us to keep a complete history of all changes, ensuring every commit was logged and could be reverted if needed. Using Git allowed us to branch off and experiment with new features without affecting the main codebase, while GitHub facilitated collaboration by providing a platform for code reviews and merging branches.

Ensuring that teammates can create branches and push it into the repository then request a pull request so every update needs to be reviewed before it can be merged with the master branch Version control

Also played a crucial role in tracking progress, addressing bugs, and rolling back changes if a feature introduced any instability. With GitHub, we maintained transparency in our development process, ensuring that all commits were well-documented, and that any version of the code could be restored, providing a safety net and providing a backbone to the project.

## - Running the Application before deploying

During the development phase, the application was primarily tested and debugged on localhost, allowing for a streamlined workﬂow and immediate feedback.

* For the backend, **Nodemon** was utilized to run the server using the command **nodemon server.js**. Nodemon is a powerful tool that automatically monitors changes in your application ﬁles and restarts the server when changes are detected. This signiﬁcantly enhances the development process by eliminating the need to manually restart the server after every modiﬁcation, allowing developers to focus on coding and testing new features without interruption.

##### 32

* On the frontend, the application was launched using **npm start**, which triggers the React development server, providing features like hot reloading for instant updates without losing the application state.
* For the database, the local **MongoDB Community Edition** was employed, enabling the application to interact with a NoSQL database seamlessly. Manual testing of API endpoints was performed using **Postman**, a widely used tool for API development that allows developers to send requests to the server and inspect responses easily. This ensures that all CRUD operations are functioning correctly and that the data ﬂow between the client and server is as expected.

Project Workﬂow

The project initially started with consultations, followed by UI design, HTML/CSS implementation, JavaScript integration, and ﬁnally moving into React development with Redux integration. Version control through Git and GitHub was used throughout the journey to manage the codebase, track changes, and enable smooth collaboration.

# Deployment and Running ( part 2 )

* <https://github.com/Blaxinoss/DepiReactFinal>github repository for the React App
* [React App (depi-react-ﬁnal-83rt.vercel.app)](https://depi-react-final-83rt.vercel.app/) React web application hosted on Vercel

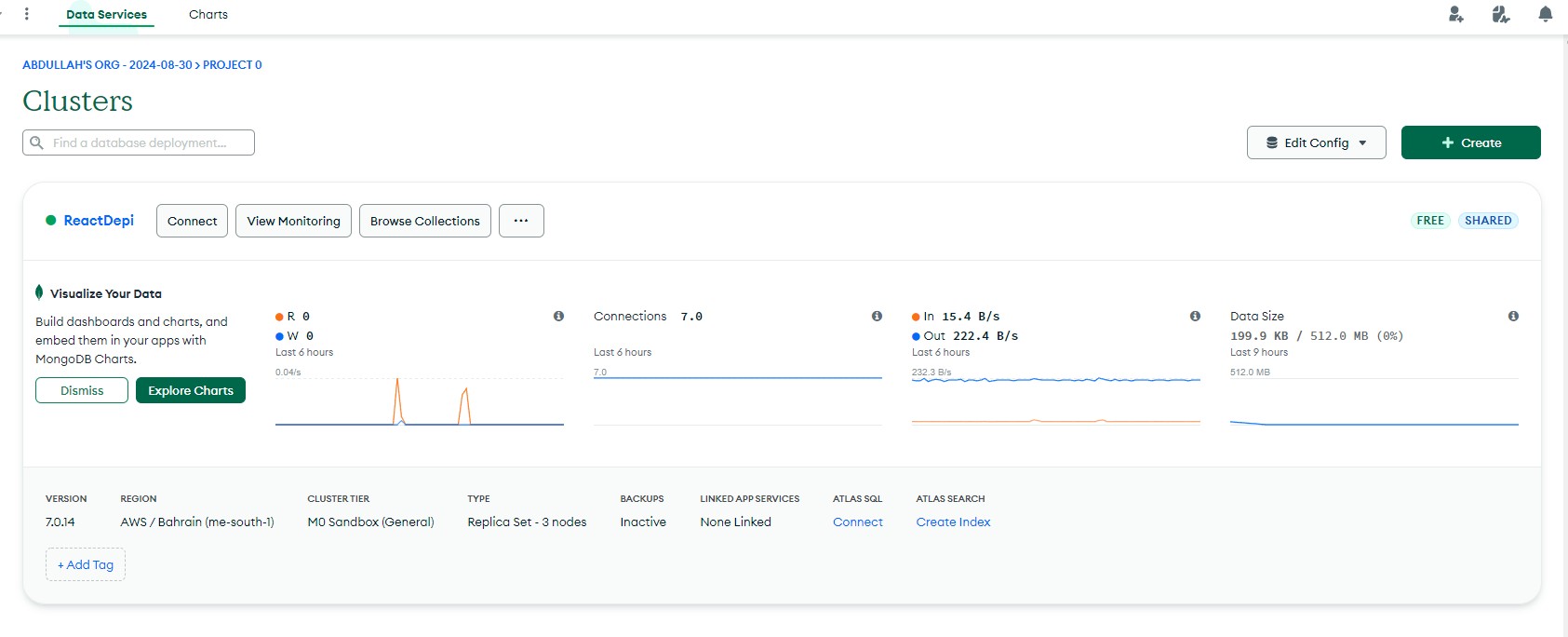
After completing the development phase, the next step was deploying the application to ensure it could be accessed and used by end-users in a production environment. Here’s how the deployment process unfolded:

###### Migrating the Database to MongoDB Atlas

To make the database accessible in a production environment, I migrated from the local MongoDB Community Edition to **MongoDB Atlas**, a cloud-based MongoDB service. The

33

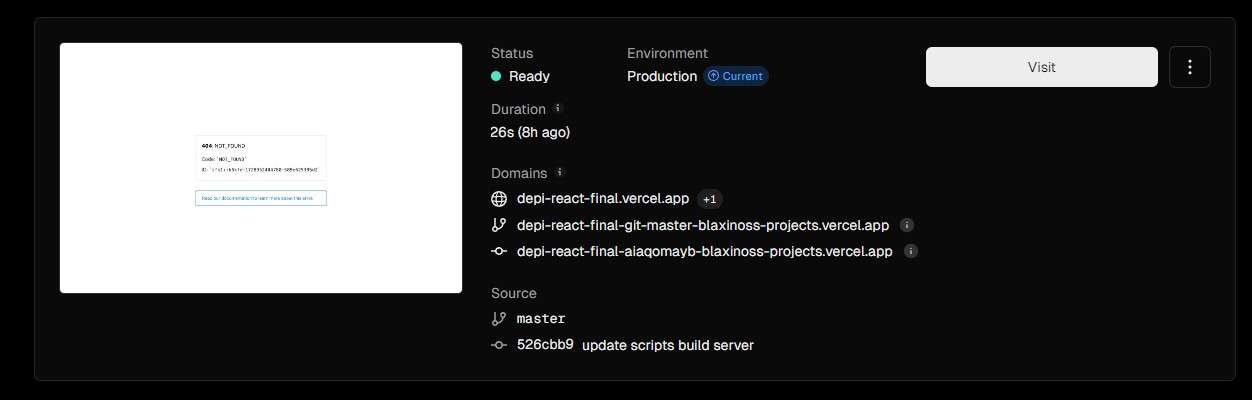
process involved creating a cluster in MongoDB Atlas and setting up a new database within that cluster. Once the database was ready, I obtained the connection URL and updated the .env ﬁle in the project to use this URL, ensuring that the application could securely connect to the cloud database.



###### Backend Deployment with Vercel

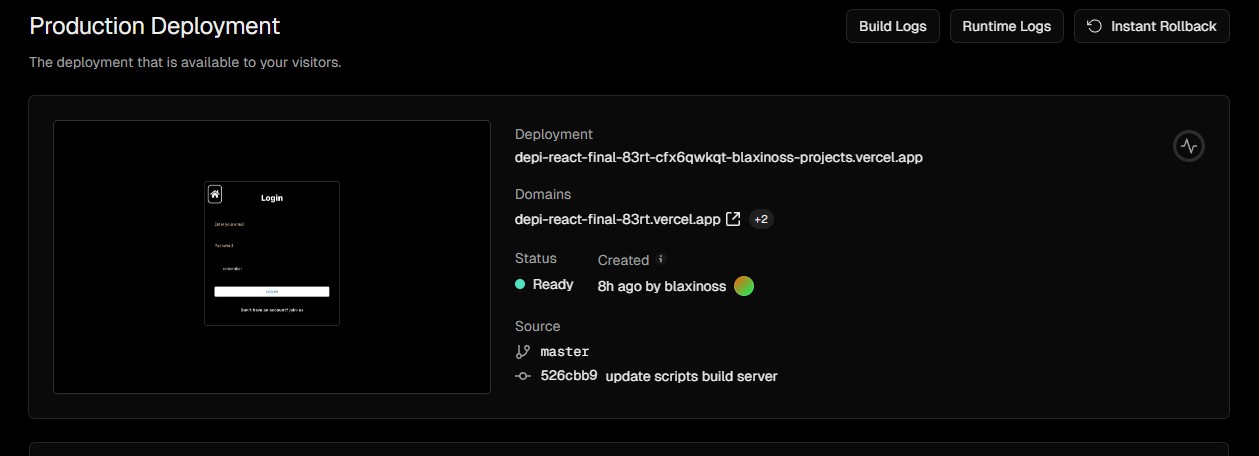
For hosting the backend, I chose **Vercel**, a popular platform for deploying serverless applications. I started by adding a **vercel.json** conﬁguration ﬁle to the project, which speciﬁed the deployment settings and route handling. The server-side code was then uploaded to Vercel, where the backend routes were hosted to provide API services for the application. Vercel's seamless integration with GitHub allowed for continuous deployment, so any updates made to the GitHub repository would automatically trigger a new deployment. This ensured the backend was always up to date.

##### 34



###### Frontend Deployment with Vercel

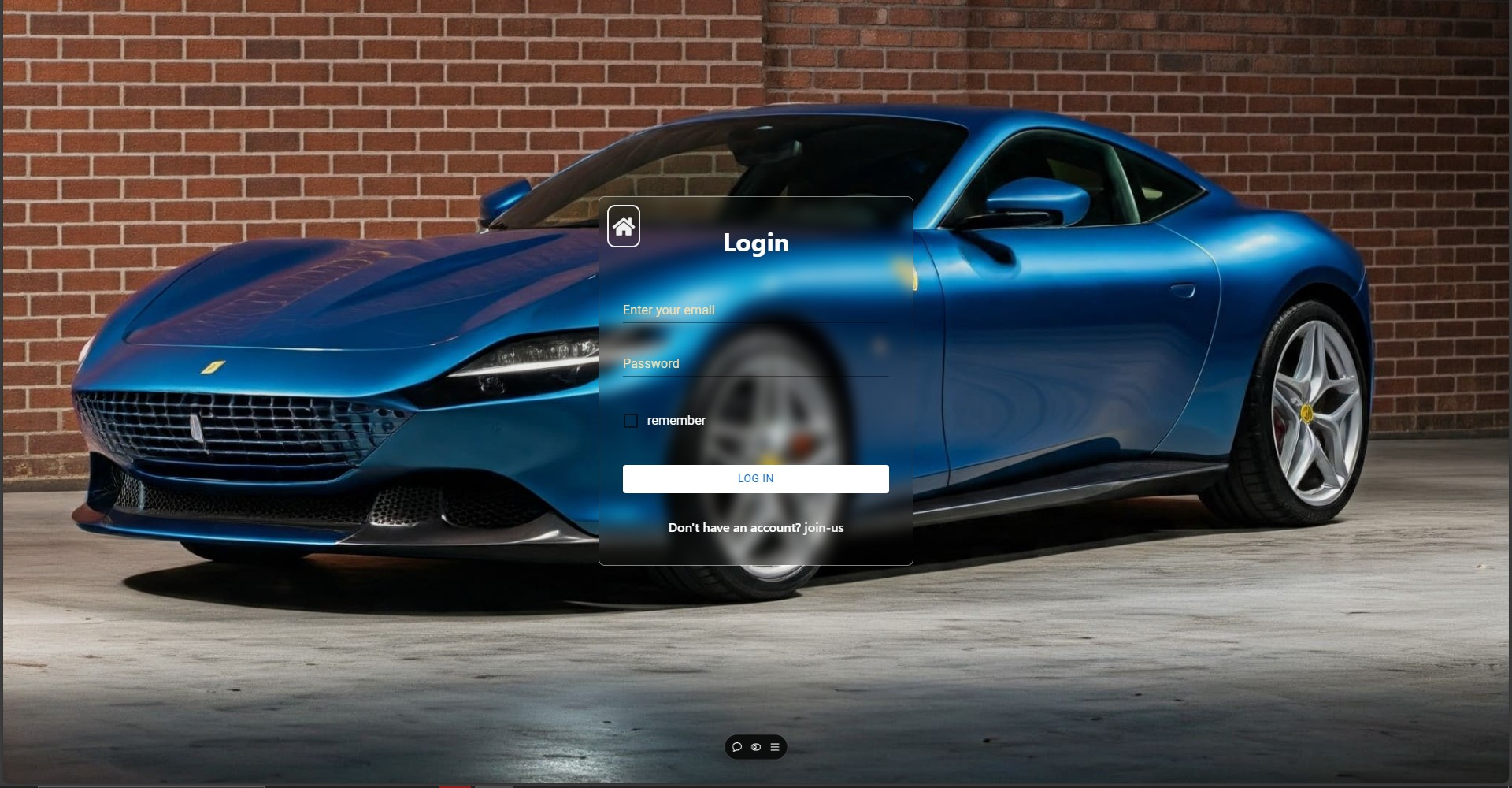
The **frontend** was also deployed using Vercel, allowing the React application to be hosted and accessed online. The deployment process again leveraged GitHub integration, where the front-end repository was linked to Vercel for automatic deployments on every commit. This streamlined approach ensured that any changes made to the frontend would quickly be reﬂected in the live application.



35

###### Connecting the Frontend and Backend

After deploying both the frontend and backend on Vercel, I conﬁgured the **frontend** to communicate with the **backend's** Vercel-hosted API routes by updating the API endpoints in the code. This ensured that all features dependent on the backend, such as data fetching and user authentication, were functioning correctly. Proper testing and veriﬁcation were carried out to conﬁrm that the entire stack was integrated seamlessly, resulting in a fully functional deployed application.



36

Now our web application is deployed and ready to go!

**Project Advantages:**

* **Meeting All Requirements:** The project successfully met all the requirements set by the stakeholders at "Ramy Rent," implementing the requested features and functionalities effectively.
* **Stakeholder Satisfaction:** The stakeholders' requests and vision for the car rental management system were thoroughly implemented, ensuring alignment with their business needs.
* **Redux Integration:** Integrating Redux for state management was a challenging yet one of the best experiences to have, a very powerful tool that needed a deep dive and can create a really interesting path to develop the whole application.. Although complex, it greatly improved the application's eﬃciency and provided a deeper understanding of state management in modern web applications.
* **Enhanced Skills in React and Hooks:** The project offered the opportunity to delve deeper into React, particularly with hooks like useState and useEffect, which are essential for managing component state and lifecycle in functional components.

##### 37

* **End-to-End Application Development:** Successfully developed, implemented, and connected the entire application, ensuring seamless integration across the frontend, backend, and database.
* **Automated Testing with Jest:** Gained valuable experience in automated testing using Jest, which provided insights into test-driven development and the beneﬁts of ensuring code reliability.
* **Real-World Application:** The program is designed for real-world use, where it will be maintained and enhanced over time, offering a practical and scalable solution for car rental management.
* **Customizability:** The application is fully customizable, ready for feature expansion, and adaptable to other car rental companies with similar needs, making it a generic solution for the industry.
* **Deployment and Hosting Experience:** Learned how to deploy and host different parts of the application, gaining practical knowledge in managing cloud infrastructure and server hosting.
* **Documentation Writing Skills:** Developed skills in writing documentation, which is essential for maintaining the project and facilitating future development.
* **Leadership and Team Management:** Acquired experience in leadership, task management, handling project pressure, and managing time effectively throughout the development process.

#### Challenges Faced Throughout the Project:

* **Understanding Redux:** Initially, it was challenging to understand and implement Redux for state management due to its complexity. However, overcoming this hurdle provided a solid foundation in managing application state.
* **React Component Structure:** Transitioning from a static web design to a dynamic React component structure required a shift in mindset to think in terms of reusable components and hooks.

##### 38

* **Backend and Frontend Integration:** Ensuring smooth communication between the backend API and frontend components was sometimes diﬃcult, especially with managing asynchronous data ﬂow.
* **Automated Testing:** Writing comprehensive tests and understanding how to simulate user interactions in Jest was challenging, especially for complex components.
* **Deployment Conﬁgurations:** Setting up the correct conﬁgurations for deployment, especially with integrating MongoDB Atlas, Vercel, and ensuring that environment variables were properly managed.
* **Version Control Workﬂow:** Managing different branches and merging pull requests eﬃciently required consistent coordination and communication with the team.
* **Team communication :** As a team leader there was a serious problem that could affect the project completion due to the neglect of some team members and their uncareness about doing or even asking about the tasks given to them.

#### How Challenges Were Overcome:

* **Gradual Learning Curve:** For Redux and React, a step-by-step approach was taken, starting with basic examples and progressively integrating more complex state management scenarios.
* **Testing Strategies:** Embraced a test-driven approach by writing smaller, more manageable tests ﬁrst, then expanding to cover more complex scenarios, which built conﬁdence in using Jest.
* **Deployment Trials:** Several iterations of deployment were performed in a development environment to troubleshoot conﬁguration issues before ﬁnalizing the production setup.
* **Version Control Best Practices:** Implemented best practices for using Git, such as branching strategies, pull request reviews, and proper commit messages to keep the codebase organized.



#### Shortcomings in the Project:

* **Initial Learning Curve:** There was a steep learning curve with some technologies, leading to initial delays in development as the team got accustomed to Redux, automated testing, and deployment practices.
* **Basic UI Design:** While the user interface was functional and met requirements, more effort could have been put into making it visually appealing and improving user

##### 40

experience. But as most of the team members had very low experience in ui designing and creating creative wireframes

* **Deployment Downtime:** There were minor issues with deployment downtime during the transition from local development to the live environment, which could be minimized with more robust deployment scripts and monitoring.
* **Performance Optimizations:** Some parts of the application could be further optimized, particularly in terms of state management and component rendering.
* **Client Side Downgrading :** The client side part needed to be further upgraded to make sure it arise with the expectations we had but again because of the overload tasks on some members and the neglection of some members it wasn’t as good as expected

Conclusion

The "Drifter" project represents a signiﬁcant achievement in modernizing car rental management through a comprehensive software solution. By eliminating paperwork and introducing a digital system, we provided stakeholders with a reliable platform for managing car rentals, maintenance, and customer interactions. The development process, from initial consultations to the deployment of a fully functional application, showcased our ability to translate stakeholder needs into a robust product.

Utilizing technologies such as React and Redux, we crafted a responsive and dynamic user experience that enhances both administrative eﬃciency

41

and customer satisfaction. The integration of automated testing with Jest ensured code reliability, contributing to a high-quality ﬁnal product.

Moreover, the application’s customizability makes it adaptable for other car rental businesses, demonstrating its potential as a scalable solution in the industry. Overall, the project was a great experience to have and a great idea to implement, establishing a strong foundation for future enhancements and continued growth in the car rental sector.

**Thank you.**