Weekly Report 4

CIS 453 M001

Group 10 - Hussein, Zichen, Chang, Ryan

Team Name / Members: Rental10 Squad

Phase 4: Technology Stack Justification

1. Programming Language & Frameworks

- **React.js (Frontend):** Chosen for its component-based architecture, which allowed for the creation of a modular and reusable Accordion Slider for the vehicle catalog. React's state management (useState, useEffect) ensures a fluid user experience by updating the UI without reloading the page.
- **Node.js & Express (Backend):** Using JavaScript on both ends allows for seamless integration. Express provides a lightweight and fast framework to build the REST API that serves vehicle data from the database to the frontend.
- **Axios:** Selected as the HTTP client to handle API requests. It simplifies the process of sending asynchronous requests to the backend and handling JSON data and potential network errors.
- **UUID Library:** Integrated for generating non-sequential, secure primary keys for transactions, preventing URL manipulation and enhancing system security for booking confirmations.

2. Database

- **MySQL:** As a Relational Database Management System (RDBMS), MySQL is ideal for managing structured data. It handles the relationships between vehicles, images, and rental records efficiently. It is a robust, industry-standard choice that is easily managed via phpMyAdmin.

3. Tools

- **Visual Studio Code (VS Code):** Used as the primary IDE due to its excellent debugging tools and integrated terminal, which allowed for running the server and client concurrently.
- **XAMPP / phpMyAdmin:** Utilized for local database hosting and visual management of SQL tables, facilitating rapid schema changes and data entry.

Prototype Implementation & Reflection

Core Functionalities Implemented:

- **Dynamic Showroom:** A fully functional vehicle catalog that fetches real-time data from MySQL.
- **Car Details Viewing:** Individual landing pages that utilize SQL JOINs to display technical specifications, high-resolution imagery, location data, and user reviews.
- **Dynamic Booking System:** An interactive calendar-based interface that fetches and blocks unavailable dates in real-time.
- **Transaction Logic:** Development of a robust backend process that creates bookings, records payments, and automatically updates vehicle availability status (e.g., 'available' to 'reserved') using SQL transactions.

1. Objectives for this Week:

- Finalize the "Showroom" core functionality and expand into the "Booking" phase.
- Establish a stable and secure connection between the React frontend and the Node.js backend.
- Implement advanced routing for dynamic vehicle pages (/details/:id and /booking/:id).
- Prepare the repository for final submission, including SQL export scripts and technical documentation.

2. Work Completed:

- **Data Layer:** Optimized SQL scripts to populate the fleet with 10 luxury vehicles and expanded the schema to support bookings and payments.
- **Frontend Development:** Completed the `CarDetails.js` and `Booking.js` components with responsive design and calendar logic.
- **API Integration:** Successfully tested the `/api/cars` and `/api/bookings` endpoints to ensure data flows correctly from the database to the UI.
- **Security & Identifiers:** Implemented the `uuid` library to ensure every rental transaction has a globally unique reference number for tracking and security.

3. Challenges Encountered:

- **Asset Linking:** Resolved issues with "broken" images caused by non-direct Unsplash links.
- **Database Connectivity:** Debugged initial "Connection Refused" errors by ensuring the MySQL service and Backend server were synchronized on the correct ports.
- **Asynchronous State Handling:** Managed the challenge of "loading states" to prevent the UI from breaking while waiting for complex SQL Joins (Vehicles + Locations + Reviews) to resolve.
- **Dependency Management:** Resolved environment-specific errors by documenting the mandatory installation of the uuid package for the new booking routes, addressing Node.js/npm version mismatches, and exploring Docker-based containerization as a future solution for environment consistency.

4. Team Contribution:

Chang:

- Designed and implemented the **Accordion Slider** UI using React.js.
- Developed the responsive CSS styling for the Catalog and implemented dynamic background image rendering.
- Managed the UI state logic to ensure smooth transitions between vehicle slides.
- Developed the **CarDetails** and **Booking** pages, implementing a complex calendar logic to prevent overbooking.

Hussein:

- Constructed the **REST API** using Node.js and Express.
- Developed the server-side routes to fetch vehicle data from the MySQL database.
- Configured **CORS** settings and middleware to ensure secure communication between the frontend and backend.
- Built the **Booking API** using SQL Transactions to ensure data integrity between the Bookings and Payments tables.

Ryan:

- Designed the **MySQL relational schema**, including the Vehicles and VehicleImages tables.
- Performed data sanitization and "SQL Dumps" to fix broken image URLs and ensure data integrity.
- Managed the local database deployment via XAMPP/phpMyAdmin and prepared the final .sql export script.
- Expanded the schema to include **Bookings, Payments, and Reviews** tables, establishing foreign key relationships.

Zichen:

- Managed the GitHub repository version control and helped finalize the project documentation.
- Created **a Docker/Docker Compose packaging** attempt in a separate branch to provide a reproducible local setup without impacting the main stable build.
- Documented the container run steps and verified `.env` configuration and cross-origin connectivity for local/containerized runs.
- Documented system challenges and provided technical justifications for the chosen stack.

5. Final Reflection: Challenges & Lessons Learned

The transition from conceptual design to a functional full-stack prototype involved a steep learning curve that extended well beyond basic coding skills. One of the most significant technical challenges was managing asynchronous data synchronization. In a relational system, we needed to ensure that frontend React components properly awaited complex SQL joins and aggregated vehicle specifications, reviews, and availability. This required a sophisticated approach to loading states and error boundaries to prevent application failures. Through this process, we learned that a fluid user experience depends not only on visual transitions, but also on robust management of the intermediate stages of data retrieval.

The development of the Booking Engine introduced us to the critical concept of atomic transactions. We recognized that, in a commercial context, a database must never exist in a partial state; for example, if a payment fails, the booking should not be recorded. Transitioning from basic data insertion to implementing SQL transactions and utilizing the UUID library marked a pivotal moment for our team. This shift transformed the project from a simple display system into a comprehensive business logic platform, demonstrating that data integrity and secure, non-sequential identifiers are fundamental to establishing user trust in digital commerce.

Ultimately, this phase highlighted the importance of system integration rather than isolated development. We discovered that an elegant frontend is ineffective without an optimized API, and a robust database remains inaccessible without correct CORS and environment configurations. This project has reinforced our understanding of the PERN stack as a unified ecosystem, where each layer, from the database schema to the client-side router, must be precisely coordinated to deliver a secure, responsive, and reliable service.

GitHub Repository:

https://github.com/hesto81c/CIS-453Project.git

(Note: Docker-based environment is on a separate branch for experimentation.)