**PeayPark Final Report**

Collaborators: Jnaria Wheeler, Mitchell Sollman, Connor Dial

**Table of Contents**

1. Introduction
2. Technology
3. Design
   1. Diagrams
   2. SupaBase Tables

3.3 JavaScript Modules

1. Deployment and Building
2. Known Issues
3. Appendices

**1. Introduction**

The Austin Peay State University (APSU) Parking lot application is created to make parking easier for students and staff on the APSU campus. This application will have real-time updates on a user-friendly interface. The app will allow users to view the parking availability across campus and see more detailed information about a single parking lot. Some information that will be provided about the parking lots will be what lots are most popular, what lots are running low on spaces, directions to each parking lot, and an option for users to report vehicles parked incorrectly. Additionally, the application will notify users when a parking lot is almost empty or about to reach full capacity.

Our application’s goal is to make the process of parking on campus more manageable and less stressful for everyone. By providing real-time updates on the applications, students or staff will not have to drive around campus hoping to find an empty parking spot. This application could possibly reduce traffic around the campus area by letting its users know where to go to find the empty parking space. The reporting option on the application will make sure everyone is held accountable and to think twice before parking in an invalid spot.

This system solves several problems through innovative solutions using features such as:

* **Real-Time Parking Updates**: Helps users determine availability instantly where otherwise they would have to give it a “Trial-and-Error” approach by physically going to each parking lot and searching for a parking space .
* **User Authentication**: Secure login and account creation processes ensure only authorized users access the system. It also allows for separation of accounts based on their designation. (Faculty, Guest, Student) It also allows for designated accounts to be given administrator privileges, allowing secure editing of parking lot data based on the discretion of the parking administrator(s).
* **Administration and Reporting**: Enables administrators to monitor parking lots and edit data pertaining to them..

The project is built with a combination of JavaScript/React Native for the front end and Supabase as the backend database and auth provider, as reflected in files such as PeayParkApplication/supabase.js and in the supporting diagrams.

**2. Technology**

The PeayPark project integrates several modern technologies, each chosen based on its robust characteristics and ease of integration. Below is an easy-to-read graph of our technologies we implemented in our project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Technology** | **Purpose in Project** | **Experience** | **Notes** |
| **React/React Native** | Building dynamic user-interface (UI) for pages like Admin, Login, Map, etc. | Moderate learning curve: extensive community support and AI explanations makes onboarding easier. | Initially proposed a hybrid web solution, but switching to React-native ensured improved mobile compatibility and a smoother user experience. |
| **SupaBase** | Acts as a backend-as-a-service, providing a cloud-hosted database and authentication. | Quick to set up and integrate, despite some initial configuration challenges. | Originally considered Firebase, but Supabase’s NoSQL-based approach and open-source nature proved more appealing. |
| **Expo** | Facilitates the integration between React Native and native functionalities, ensuring seamless cross-platform deployment. | Developer-friendly and simplifies the process of deploying mobile applications. | Used for rapid prototyping and building native apps. |

**Observations on the Technologies:**

* **Ease-of-Use**: React-native and Supabase both showed a fairly gentle learning curve. React allowed rapid creation of UI components (e.g. pages like AdminPage.js, LoginPage.js, and MapPage.js), while Supabase provided a straightforward REST-like API that integrated directly with the front-end code in parkingFunctions.js and supabase.js.
* **Technology Shift Rationale**: The team initially looked to other backend solutions such as FireBase but switched to Supabase because of its SQL-like support, ease of maintenance, and better community documentation. Overall, the chosen stack made the development process smoother, especially for rapid iteration and debugging during early development. The team also looked to use Google Maps API, but after discovering documentation on React Native for the MapView component, Google Maps API was no longer needed as this was taken care of by the component.

**3. Design**

**3.1 Diagrams**

The design of the PeayPark system is well-documented using several diagrams and drawings:

**Class Diagram**:

A diagram of a user interface

AI-generated content may be incorrect.

**File Location**: Documents/PeayPark Class Diagram.drawio

**Description**: This diagram maps out the main entities (like User, Admin, and ParkingRecord) used by the system, detailing their attributes and methods. It is an essential reference for understanding object interactions, such as the responsibility of handling login validations or updating parking statuses.

**Entity Relationship Diagram (ERD)**:

A diagram of a computer

AI-generated content may be incorrect.

**File Location**: Documents/ERDMARCH1updated01.drawio

**Description**: The ERD outlines the database schema, showing the relationships between tables such as Users, Parking Lot Table, and other supporting tables required for storing user data, parking availability, and reporting details. This ERD is critical for both the backend design and the Supabase integration.

**Block Diagram**:

A diagram of a software component

AI-generated content may be incorrect.

**File Location**: Documents/Peay Park Block Diagram.drawio.png

**Description**: It visually represents the overall system architecture. The block diagram details how the front end (React Native components) interacts with backend services (Supabase for database and auth) and how components such as the map view and reporting module fit into the larger system.

**3.2 SupaBase Tables**

**Parking Lot Table**

A screenshot of a black screen

AI-generated content may be incorrect.

**SupaBase Account Table**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Auth.Users() Table**

**A screenshot of a computer

AI-generated content may be incorrect.**

SQL query to generate tables easily and setup project in SupaBase.

-- Creating the Parking Lot Table

CREATE TABLE IF NOT EXISTS "Parking Lot Table" (

"ParkingLotID" varchar PRIMARY KEY,

"Latitude" varchar,

"Longitude" varchar,

"OpenHours" time,

"LotType" varchar,

"CloseHours" time,

"AvailableSpaces" int8,

"TotalSpaces" int8

);

-- Creating the Supabase Account Table

CREATE TABLE IF NOT EXISTS "SupaBase Account Table" (

"UserID" uuid PRIMARY KEY,

"UserType" text,

"Administrator\_Access" varchar,

"FirstName" varchar,

"LastName" varchar,

"ParkedLocation" text,

FOREIGN KEY ("UserID") REFERENCES auth.users(id) ON DELETE CASCADE

);

**3.3 JavaScript Modules**

The core functionality of the PeayPark application is implemented in several JavaScript files organized in the folder “PeayParkApplication”. Each JavaScript file serves a dedicated role which we will analyze:

**Main Entry & App Initialization**

**Index.js**

This is the simplest and smallest file we have in our application. The purpose of this file is that it serves as an entry point for the React Native application. It uses Expo’s registerRootComponent to register the main App component so that the app will run seamlessly in both Expo Go and native builds.

**App.js**

The purpose of this file is that it serves as the root application component that incorporates routing/navigation among different pages (e.g., LoginPage.js, AdminPage.js, MapPage.js, etc.). It implements the overall navigation logic and wraps individual pages which are later extended into their specific functions. It is central to managing the session, global state, and distribution of props among child components.

**User-Facing Pages**

Each page follows a similar structure, designed with clear separation of concerns:

A screenshot of a login screen

AI-generated content may be incorrect.**LoginPage.js**

The purpose of this file is to manage user authentication. This page includes form elements to capture login credentials and integrates with the authentication logic (likely supported by supabase.js). Some of its most notable features are error handling, form validations, and redirection to appropriate pages upon successful login.

A screenshot of a login form

AI-generated content may be incorrect.**createAccountPage.js**

The purpose of this file is to handle user account creation with forms for entering essential user details. It integrates with Supabase to add new users to the database while also handling form layout and validations.

A screenshot of a login

AI-generated content may be incorrect.A screenshot of a login

AI-generated content may be incorrect.**VerificationPage.js**

The purpose of this file is to provide an extra step for validating user credentials or account information which involves email verification. It ensures that the user is accurately identified before being granted further system privileges.

**forgotPasswordPage.js**

The purpose of this file is to allow users to click on the forgot password button at the bottom of the login page. Users are then directed to the forgot password page. This page allows users to enter their email address that is linked to their account to get sent a 6-digit code using Supabase’s reset Password function. Once the code is sent to their email, users will be directed to the verification page to input the verification code and continue the password reset.

**changePasswordPage.js**

The purpose of this file is to provide a page where users can securely update their passwords after completing the verification process. Users will be prompted to enter and confirm their new password. Upon success, Supabase updates its information, and users will then be directed to the login page to enter their new information.

**Administrative and Reporting Pages**

These files are designed for system administrators and reporting functionality:

**A screenshot of a phone

AI-generated content may be incorrect.A screenshot of a phone

AI-generated content may be incorrect.AdminPage.js**

The purpose of this file is to provide an interface for system administrators to manage users, monitor parking records, and perform maintenance tasks. It includes features such as listings of parking records, modifications for user credentials, and status updates.

**A screenshot of a phone

AI-generated content may be incorrect.ReportingPage.js**

The purpose of this file is to allows administrators or users to generate reports concerning parking usage and availability. This includes a picture and a description, which is then sent from the user’s email to an email address designated for the local parking authority.

A red rectangle with white background

AI-generated content may be incorrect.**SettingsPage.js**

The purpose of this file is to allow the user to logout of the account they are currently signed into and also access the administrator page and settings for adding, deleting, or modifying existing parking lots if they are designated in the SupaBase table to have elevated administrator access.

A map of a city

AI-generated content may be incorrect.A map of a city

AI-generated content may be incorrect.**MapPage.js**

The purpose of this file is to present a map-based view where users can see real-time available spots and navigational guidance. It utilizes dynamic fetching of data for displaying current availability. It is styled to integrate seamlessly with the rest of the application’s themes. It is where users will spend most of their time while using the application and serves as a home screen once logged in. It shows users all parking lots on campus, it includes information about each parking lot and allows users to filter through available lots as well as park or unpark in specific lots. This is done through crafted queries that are sent to our SupaBase database, managed in this application through the “supabase.js” file. All information retrieved on this page is stored in the "Parking Lot Table" and the "SupaBase Account Table".

**Supporting Functionality and Utilities**

**parkingFunctions.js**

The purpose of this file is to host utility functions for operations related to parking data such as updating available spaces and handling parking space reservations. It is where the parking functions that decrement and increment the parking spots in each parking lot are located. It contains the function updateData function which updates parking lot information by interacting with the Supabase backend. It contains the takeParkingSpace function which decrements the number available spaces displayed based on the parkingLotID, AvailableSpaces, and TotalSpaces. It also contains the leaveParkingSpace function which increments the number available spaces displayed based on the parkingLotID, AvailableSpaces, and TotalSpaces.

**supabase.js**

The purpose of this file is to configure and exports the Supabase client so that other parts of the application can interact with the backend easily. It contains the connection URL and API key. It serves as the central point for all database queries throughout the application.

**4. Deployment and Build**

Deploying and/or compiling the PeayPark application involves several clear steps that should be done in order after you have downloaded the files or cloned the repository on GitHub.

1. Open your IDE of choice that is capable of JavaScript and includes a terminal. For this, we will be using WebStorm, developed by JetBrains.
2. Navigate to the PeayParkApplication folder in terminal. This can be done by typing the command “cd PeayParkApplication” in terminal and pressing enter.
3. Install all relevant dependencies using Node Package Manager. This can be done by typing the command “npm install” in terminal and pressing enter.
4. Create SupaBase account for free and setup your tables exactly as the tables are in section 3.2 of this report. You can setup the tables easily by navigating to the SupaBase SQL editor and pasting the command found in section 3.2 of this report.
5. Configure SupaBase credentials. This can be done by updating the file “supabase.js” with your unique Supabase project credentials. These may be found in your SupaBase account settings. You will need to update the “supabaseURL” string and “supabaseAnonKey” string with your own that are connected to your SupaBase account.
6. Download Expo Go from the App Store if on iOS or the Google Play Store if on Android OS.
7. Run the project in WebStorm with Expo by typing the command “npx expo start” in terminal and pressing enter. This command will launch the Expo development server which will display a QR code in the terminal.
8. Scan the QR code using your mobile device with Expo Go installed which will automatically open and start the application from within the Expo Go application.

**5. Known Issues**

While the system has been developed to offer robust functionality, several known issues still remain:

**Race Conditions in Parallell Updates**:  
In parkingFunctions.js, simultaneous requests to takeParkingSpace may result in race conditions. A more transactional approach or some sort of locking mechanism is required to prevent data inconsistencies and errors when two people update their parked location in the same parking lot at the exact same time.

**Error Handling and Notifications**:  
Although error logs appear in the console (via console.error), there is limited user feedback on failed operations. Future improvements should incorporate user-friendly error messages on the front end.

**Incomplete UI/UX Elements**:  
Some pages might not fully reflect all design mockups. For instance, the SettingsPage.js may need enhanced styling and interactive components as well as the AdminPage.js and MapPage.js.

**Scalability Considerations**:  
While a free Supabase account works well for prototyping, load testing and potential scaling would likely reveal many challenges with the growing number of parking records.

**6. Appendices**

| **File Name** | **Description** |
| --- | --- |
| Documents/PeayPark Class Diagram.drawio | Class Diagram illustrating main entities and their interactions. |
| Documents/PeayPark ERD.drawio | Entity Relationship Diagram (ERD) for the database schema. |
| Documents/PeayPark Block Diagram.drawio.png | Block diagram showing system architecture. |
| PeayParkApplication/AdminPage.js | Admin interface for system management and monitoring. |
| PeayParkApplication/App.js | The root React component that sets up routing and global application state. |
| PeayParkApplication/LoginPage.js | Handles user login functionalities and authentication. |
| PeayParkApplication/MapPage.js | Displays and updates a map of parking spots in real time. |
| PeayParkApplication/ReportingPage.js | Allows for submission of detailed reports on parking usage and occupancy. |
| PeayParkApplication/SettingsPage.js | Provides configuration options and changeable preferences for the user. |
| PeayParkApplication/VerificationPage.js | Handles user account verification procedures. |
| PeayParkApplication/createAccountPage.js | Manages new user registrations and account creation workflows. |
| PeayParkApplication/forgotPasswordPage.js | Sends users to a page to enter their email to verify their account |
| PeayParkApplication/changePaswordPage.js | Users are able to enter their new password and confirm it as well |
| PeayParkApplication/index.js | Application’s entry point, integrating with Expo registration. |
| PeayParkApplication/parkingFunctions.js | Contains methods for updating and managing parking lot data via Supabase. |
| PeayParkApplication/supabase.js | Sets up the connection and configuration for Supabase as the backend service. |

**Styling and Layout**

**Visual Cues**: Throughout the application, color-coded alerts, rounded components, and responsive layouts have been implemented.

**Code Modularization**: Each component is kept in a separate file ensuring maintainability and testability. It also helps greatly in readability

**Conclusion**

The PeayPark application is a comprehensive parking management solution grounded in modern web technologies. Its architecture, from the user interface components to backend interactions via Supabase, provides a robust foundation that can be further extended for scalability and reliability. Although there are areas for improvement as previously noted in the known issues before this, the current design and development practices provide clear paths forward for future enhancements.

Developers working with this documentation should be able to quickly understand, build, and extend the system. The thorough inclusion of diagrams, technical details, and deployment steps ensures that anyone new to the project will catch up rapidly.