Alternative Plan: ParkEase with Spring Boot Backend

Objective:

The goal is to develop a **web-based dashboard** for real-time monitoring of parking spots using **Spring Boot** for the backend, with simulated parking spot data for testing. The system will be built in **Java** and will interact with **MySQL** for data storage and **React.js** for the frontend.

---

1. System Components

1.1 Hardware

- **ESP32 Microcontroller**: For detecting vehicle presence at parking spots.
- **Sensors**: Ultrasonic or Infrared sensors for occupancy detection.
- **Power Supply**: To ensure uninterrupted operation of the sensors and ESP32.

1.2 Software

- **Backend**:
  - **Spring Boot**: A Java-based framework to handle backend logic and API management.
  - **MySQL**: To store parking spot and floor data.
- **Frontend**:
  - **React.js**: For a dynamic, responsive user interface to display real-time parking data.
- **Protocols**:
  - **HTTP/REST**: To facilitate communication between the ESP32 and the backend.
  - **WebSocket**: For real-time updates to the dashboard.

2. Phase 1: Development Plan (Web App + Testing with Spring Boot)

Objective of Phase 1:

The primary goal of Phase 1 is to develop the **web-based dashboard** for real-time monitoring of parking spots, using **mock data** for testing. This phase will focus solely on the web app interface and backend API setup using **Spring Boot**.

2.1 Basic Flow

Step 1: Mock Data for Testing

- **Simulating the Parking Spot Data**:
  - As we don't have the actual ESP32 devices, we will simulate parking spot occupancy status.
  - A mock data generator will randomly simulate whether parking spots are vacant or occupied.

Step 2: Backend Setup (Spring Boot)

1. **Spring Boot Project Setup**:
   - Initialize the Spring Boot project using **Spring Initializr** or a preferred method.
   - Add required dependencies: Spring Web, Spring Data JPA, MySQL Driver.
2. **Database Models**:
   - Define entities for **Parking Spot** and **Floor**:
     - Floor entity: Stores information about parking floors.
     - ParkingSpot entity: Stores data on individual spots, their status, and associated floor.

3. **API Endpoints**:
   - **POST /api/update_status**: Accepts updates from the mock system to update the parking spot status.
   - **GET /api/get_status**: Returns the current status of all parking spots for frontend display.

4. **Service Layer**:
    - Implement services to handle the logic for updating and fetching parking spot statuses.

Step 3: Frontend Setup (React.js)

1. **React.js Dashboard**:
    - Set up a **React.js** project to display parking spot statuses in real-time.
    - Use **WebSocket** or **HTTP polling** to fetch real-time parking spot data from the backend.
2. **UI Elements**:
    - **Floor View**: Display parking floors in a grid layout.
    - **Status Indicators**: Color-coded icons or labels to show whether a parking spot is vacant or occupied.

Step 4: Testing the System

- **Simulated Data**:
    - Use a tool or script to generate random mock data for parking spot occupancy.
    - The system should behave as though it's receiving data from the ESP32 devices, updating the backend and frontend accordingly.
- **API Testing**:
    - Test the **POST** and **GET** API endpoints using **Postman** to ensure proper interaction between the frontend and backend.

2.2 Features in Phase 1

1. **Real-Time Monitoring**:
    - View the current status (vacant or occupied) of each parking spot across multiple floors.
2. **User Interface**:
    - A simple, intuitive dashboard displaying floors and parking spots with real-time status updates.
3. **Mock Data Simulation**:

- Simulate sensor data to test the full workflow of status updates and display.

---

3. System Workflow

3.1 Data Flow

1. **Simulated Data Input**:
   - Mock data is generated and sent to the backend via the **POST /api/update_status** endpoint, mimicking the updates from the ESP32 devices.
2. **Backend Processing**:
   - The Spring Boot backend updates the status of the parking spots in the database.
3. **Frontend Update**:
   - The frontend fetches updated parking spot status data from the backend and displays it in the dashboard.
4. **Real-Time Updates**:
   - The dashboard is updated live (via HTTP polling or WebSocket) to reflect changes in parking spot availability.

3.2 Interaction Between Components

1. **Mock Data Source → Backend (Spring Boot)**:
   - The mock data source sends updates (via HTTP) to the backend, updating parking spot statuses.
2. **Backend (Spring Boot) → Frontend (React.js)**:
   - The backend responds with updated data, which is displayed on the frontend dashboard.
3. **Frontend User Interface**:
   - Displays real-time parking spot availability based on backend data.

---

4. Future Enhancements

4.1 Hardware Integration (Phase 2):

- When the ESP32 devices are available, replace the mock data generation with actual sensor data being sent to the Spring Boot backend.

4.2 Additional Features:

- **Analytics**: Show peak usage times, most frequently occupied spots, etc.
- **Notifications**: Send alerts when parking lots are nearing full capacity.

---

5. Tools and Technologies

- **Frontend**: React.js, Tailwind CSS.
- **Backend**: Spring Boot, MySQL.
- **Data Simulation**: Python, Mock tool for generating parking spot data.
- **API Testing**: Postman for testing the backend API endpoints.
- **Version Control**: Git and GitHub for source code management.
- **Deployment**: AWS, Heroku, or similar cloud platforms for hosting the app.

---

6. Conclusion

The **ParkEase** system, using **Spring Boot** for the backend, will provide a comprehensive solution for managing parking in multi-floor lots. Phase 1 focuses on developing the **web-based dashboard**, simulating sensor data to test the system’s functionality, and ensuring a smooth integration between the frontend and backend. This approach allows for a seamless transition to the hardware integration phase once the ESP32 devices are available.