# **CenterPlate System Architecture Document**

## **1. Requirements**

### **Functional Requirements**

1. **User Profiles**: Users should be able to create accounts, manage profiles, set dietary preferences, and specify price ranges.
2. **Location Triangulation**: The system must be able to calculate midpoints between multiple users and suggest suitable venues.
3. **Group Voting System**: Users should be able to vote on restaurant or cuisine options to help make decisions as a group.
4. **Venue Recommendations**: The system must provide venue suggestions filtered by dietary restrictions, budget, and other user preferences.
5. **Notifications**: Users should receive notifications about updates such as votes, new suggestions, and reminders.

### **Non-Functional Requirements**

1. **Scalability**: The system should handle an increasing number of users, especially during peak times.
2. **Reliability**: System uptime must be high, especially during meal hours when usage will likely be at its peak.
3. **Performance**: Actions such as voting, fetching venue data, and midpoint calculations should be processed within acceptable time limits (<1s).
4. **Security**: User data must be encrypted, and authentication should be secure.
5. **Accessibility**: The app should be accessible, following best practices for users with disabilities.

## **2. Tech Stack**

### **Frontend**

* **Framework**: React Native with Expo (Free and cross-platform, suitable for Android and iOS development, with easy setup and debugging capabilities).
* **State Management**: Context API (Built-in with React, minimal learning curve).
* **UI Library**: NativeBase or React Native Elements (Open-source, pre-designed components). Additionally, consider using **React Navigation** for routing and **React Hook Form** for form handling to enhance usability and functionality.

### **Backend**

* **Framework**: Node.js with Express.js (Open-source, provides a scalable environment).
* **Database**: MongoDB Atlas (Free tier available).
* **API Integration**: REST API with Express.js (Free and easy to implement).

### **Mapping/Location**

* **Geolocation API**: OpenStreetMap with Leaflet.js (Free and suitable for location-based services).
* **Midpoint Calculation Algorithm**: Haversine Formula (Custom implementation).

### **Group Voting System**

* **Backend Integration**: Firebase Realtime Database (Free tier available for real-time updates).

### **Hosting & Deployment**

* **Backend Hosting**: Heroku (Free tier available).
* **CI/CD**: GitHub Actions (Free for public repositories).

### **Miscellaneous Tools**

* **Notifications**: Firebase Cloud Messaging (Free).
* **Authentication**: Firebase Authentication (Free tier available).
* **Analytics**: Google Analytics (Free).

## **3. Implementation Plan**

1. **Initial Planning & Setup**
   * Define system requirements and finalize the tech stack.
   * Set up GitHub repository, create folder structure, and configure the CI/CD pipeline.
   * Configure Firebase (Firestore, Realtime Database, Authentication, Cloud Messaging).
2. **User Profile & Authentication**
   * Implement user authentication using Firebase Authentication.
   * Develop the user profile functionality, allowing users to set preferences like dietary restrictions and budget.
3. **Location Triangulation & Midpoint Calculation**
   * Integrate OpenStreetMap and Leaflet.js for mapping functionalities.
   * Implement the midpoint calculation algorithm using the Haversine formula.
4. **Group Voting System**
   * Develop a backend voting system using Firebase Realtime Database for real-time updates.
   * Integrate the voting feature with the front end, allowing users to create and participate in votes.
5. **Venue Recommendations**
   * Fetch restaurant data using OpenStreetMap and filter it according to user preferences.
   * Develop recommendation logic that includes preferences, distance, budget, and availability.
6. **Testing & Deployment**
   * Test all functionalities, including authentication, voting, and location services.
   * Conduct stress testing and user acceptance testing.
   * Deploy to production using Netlify for frontend and Render/Heroku for backend.

## **4. Architecture Overview**

The architecture for CenterPlate can be divided into three major components:

1. **Frontend Layer**: The user-facing layer built with React Native, providing interfaces for user profiles, group voting, and venue recommendations.
2. **Backend Layer**: The server-side layer developed using Node.js and Express.js, handling API requests, business logic, and interactions with the database.
3. **Database & External Services Layer**: This layer includes MongoDB Atlas for data storage, as well as OpenStreetMap for geolocation services.

## **5. Data Flow**

1. **User Authentication & Profile Setup**
   * User data (profile, preferences) is saved to MongoDB Atlas upon signup.
   * Frontend requests authentication data from Firebase Authentication and verifies user identity.
2. **Midpoint Calculation & Location Services**
   * Users input their locations, which are sent to the backend.
   * Backend calculates the midpoint using the Haversine formula and fetches venue data from OpenStreetMap.
3. **Group Voting System**
   * Users create a voting session, which is saved in Firebase Realtime Database.
   * Participants cast their votes, and real-time updates are pushed to all users in the group session.
4. **Venue Recommendations**
   * Backend requests restaurant information from OpenStreetMap based on the midpoint.
   * Filters are applied (e.g., dietary preferences, price range), and the recommendations are sent to the frontend for user selection.