Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2025

Los Tragones
Social Foodies

Diana Rios
Perla Rivera
Thomas Valenciana
Ahmed Ibrahim
Joshua Harris

Revision History

Revision	Date	Author(s)	Description
1.0	Jan 27,2025	DR, PR, TV, AI, JH	document creation
1.5	Feb 28,2025	DR, PR, TV, AI, JH	complete draft
0.0	TBA	TBA	release candidate 1
0.0	TBA	TBA	official release
0.0	TBA	TBA	added design review requests

Contents

List of Figures

List of Tables

1 Introduction

The Social Foodies mobile application is designed to improve social dining experiences by connecting users within a 5-mile radius to meet and share meals. The app integrates elements of social networks with food exploration, enabling users to match others based on dining preferences, engage in real-time chat, and earn reward points for visiting participating restaurants. By merging aspects of dating apps with food discovery, Social Foodies provides an engaging platform for users to explore local dining spots while keeping social interactions.

This detailed specification document builds on the concepts of the Requirement Specification Document and the Architectural Design Document. The Requirement Specification Document defines the purpose of the app and key features, including the swipe-based matching mechanism, real-time chat, a rewards system, etc. The Architectural Design Document further elaborates on the apps scope, core functionalities, and technical requirements, detailing important system components such as user authentication, geolocation services, dynamic search, and rewards management. Together, these documents provide the background necessary to understand the design decisions and technical specifications presented in this document.

2 System Overview

The Social Foodies application is a mobile platform designed to connect users within a 5-mile radius for shared dining experiences. The system consists of three primary layers: **the User Interface, Matchmaking & Social Engagement**, and **Backend & External Services**.

- The **User Interface (UI) Layer** serves as the primary interaction point for users. It includes profile management, match discovery, and chat functionalities.
- The **Matchmaking & Social Engagement Layer** handles the matching algorithm, social interactions, and reward tracking.
- The **Backend & External Services Layer** manages core functionalities such as database operations, notification handling, and third-party services for rewards and location tracking.

The system follows a modular design to ensure scalability, maintainability, and seamless integration across all layers. Communication between components is facilitated via API calls and real-time data synchronization mechanisms.

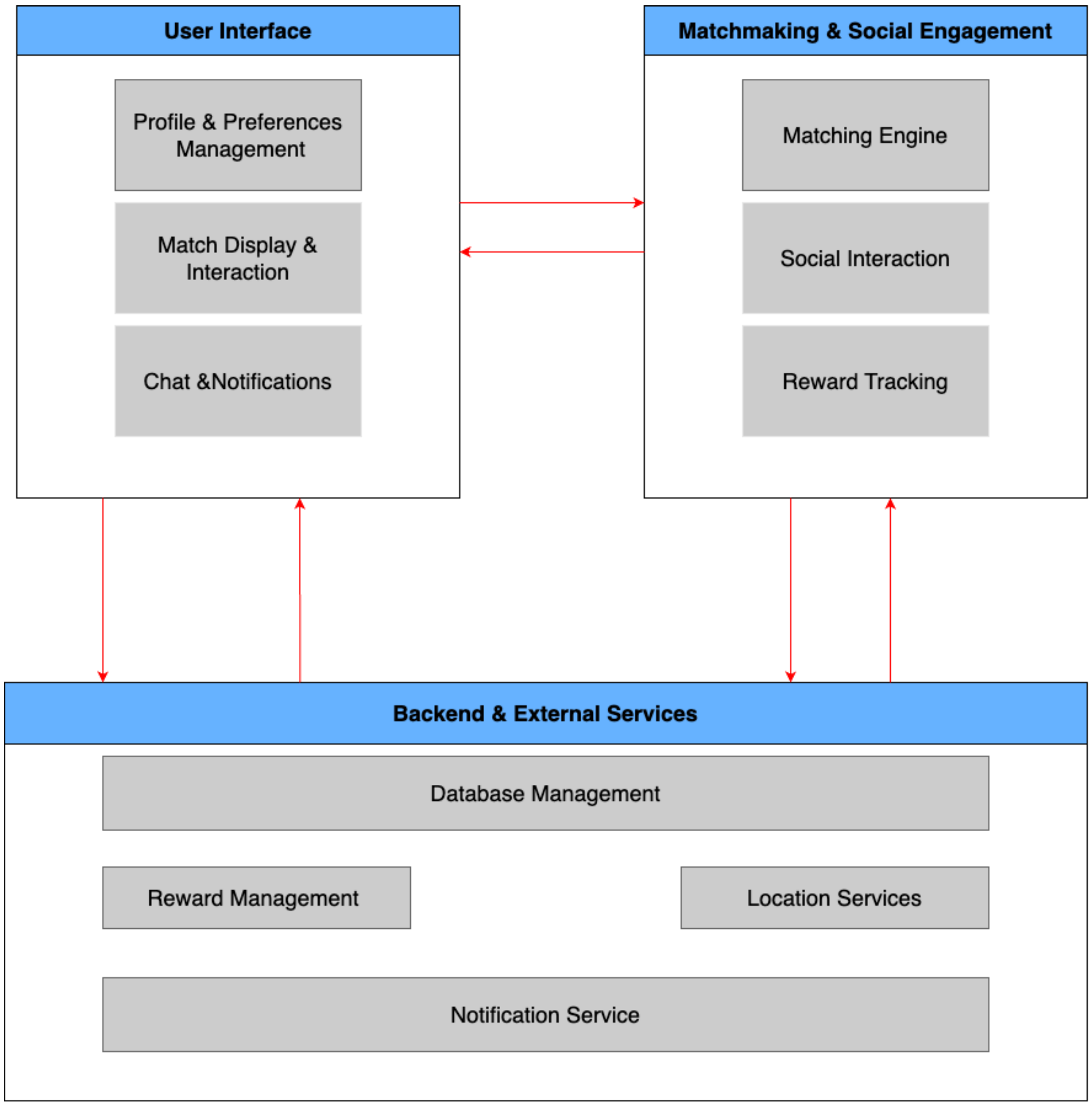

Figure 1: System architecture

3 X Layer Subsystems:User Interface

3.1 Layer Hardware

It is designed to run on mobile devices, including smartphones and tablets that support Android and iOS. The user interface components are adaptable, dynamically adjusting to different screen sizes and resolutions to provide smooth user experience across many devices.

3.2 Layer Operating System

Layer Operating System:

- The User Interface Layer is designed to run on the following mobile operating systems:
  - Android
  - iOS
- No additional OS-level modifications are required for this layer, as it operates within the native constraints of React Native.

3.3 Layer Software Dependencies

Layer Software Dependencies:

- The User Interface Layer relies on the following libraries, frameworks, and APIs:
- **Frontend Frameworks & UI Components**
  - **React Native** – Frontend framework for developing cross-platform mobile applications.
  - **React Native Components** – Used for UI elements like buttons, forms, and lists.
  - **Tailwind CSS / Material UI** – Used for styling the UI.
- **State Management & API Communication**
  - **React Context / Redux** – Handles global state management.
  - **Fetch API / Axios** – Communicates with Firebase and Google APIs.
- **Authentication & Database Access (via Backend APIs)**
  - **Firebase Authentication API** – Used for managing user login/logout.
  - **Firebase Firestore API** – Used for retrieving and updating user data.
- **Location & Mapping Services**
  - **Google Maps API** – Displays restaurant locations and user positioning.
  - **Google Places API** – Fetches restaurant details, such as reviews and addresses.
- **Development & Testing Tools**
  - **Git & GitHub** – Version control and collaboration for frontend development.
  - **Node.js & npm** – Provides the runtime environment and package manager.
  - **Prettier** – Code formatting tool to maintain clean and consistent code.

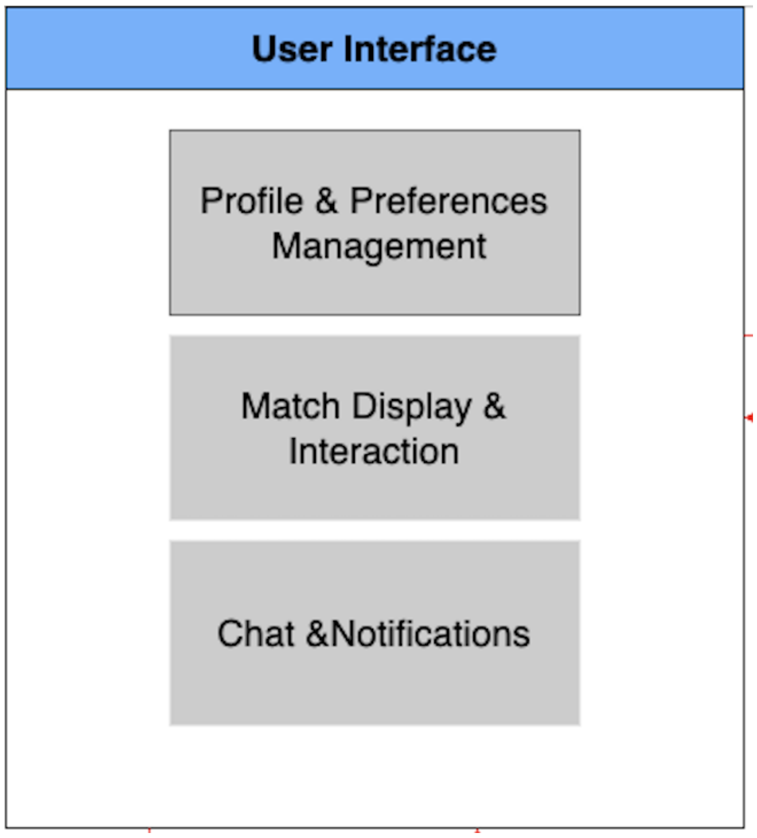

Figure 2: User Interface Diagram

3.4 Profile & Preferences Management

Purpose & Design: This subsystem allows users to create, edit, and manage their profiles, including setting personal preferences for matchmaking.

3.4.1 Subsystem Hardware

Since this is a frontend subsystem, it does not require specific hardware beyond the mobile device running the application.

3.4.2 Subsystem Operating System

This subsystem is designed to run on mobile platforms:

- **Android**
- **iOS**

It operates within the constraints of React Native, utilizing native mobile OS capabilities without requiring modifications.

3.4.3 Subsystem Software Dependencies

This subsystem relies on various software dependencies, including libraries and frameworks, to ensure functionality and seamless integration with backend services.

- **React Native UI Components** – Provides pre-built components for forms, profile displays, and settings.

- **Fetch/Axios** – Handles API requests to interact with Firebase Authentication and Firestore.

3.4.4 Subsystem Programming Languages

The subsystem is implemented using:

- **JavaScript** – Used for UI logic and state management in React Native.

3.4.5 Subsystem Data Structures

This subsystem utilizes the following data structures:

- **User Profile Object** – Stores user details, including name, age, bio, and preferences.
- **Preferences Object** – Stores user-selected matchmaking preferences.

3.4.6 Subsystem Data Processing

This subsystem handles various data processing tasks to ensure smooth functionality and data integrity.

- **Form Validation:** Ensures correct input formats for user details.
- **API Requests:** Communicates with Firebase Firestore for profile retrieval and updates.

3.5 Match Display & Interaction

Purpose & Design: This subsystem is responsible for displaying potential matches and handling user interactions such as liking, disliking, and messaging.

3.5.1 Subsystem Hardware

Runs on mobile devices.

3.5.2 Subsystem Operating System

This subsystem is designed to run on mobile platforms:

- **Android** – Utilizes native performance optimizations for smooth scrolling and animations when displaying match cards.
- **iOS** – Takes advantage of iOS-specific gesture handling for swiping interactions.

It operates within the constraints of React Native, leveraging platform-specific capabilities such as:

- **GPU-accelerated rendering** –Ensures smooth animations and transitions when swiping through matches.
- **Haptic Feedback** – Provides tactile responses when users like or dislike a match (available on supported devices).
- **Gesture Recognition** –Detects and responds to touch-based interactions, such as swipes and taps, to enhance user experience.
- **Push Notification Integration** – Enables real-time updates for new matches and incoming messages.

3.5.3 Subsystem Software Dependencies

This subsystem relies on the following libraries and frameworks for rendering match cards and handling user interactions:

- **React Native UI Components** – Used for displaying match cards, handling swipe gestures, and enabling interactive animations.
- **React Native Gesture Handler** – Enhances touch-based interactions such as swiping, tapping, and dragging for a seamless user experience.
- **React Native Reanimated** – Optimizes animations and transitions for a smooth, responsive UI when displaying potential matches.
- **Fetch/Axios** – Handles API requests to Firebase Firestore for retrieving and updating match data.

3.5.4 Subsystem Programming Languages

This subsystem is implemented using:

- **JavaScript** – Used for UI interactivity, event handling, and state management in React Native.

3.5.5 Subsystem Data Structures

This subsystem utilizes the following data structures to manage match information and user interactions:

- **Match Object** – Stores details of suggested matches, including:
  - Unique match ID
  - Name, age, and profile picture of the suggested match
  - Match compatibility score (if applicable)
  - Location data (if location-based matching is enabled)
- **Interaction Object** – Tracks user interactions with potential matches, including:
  - Liked/disliked status
  - Timestamp of the interaction
  - Message history (if a conversation is initiated)
  - Interaction status (e.g., pending, matched, unmatched)

3.5.6 Subsystem Data Processing

This subsystem handles various processing tasks related to match interactions and user preferences:

- **Event Handling for User Interactions:** Detects and processes actions such as:
  - Swiping left or right to like/dislike a match.
  - Tapping on a match profile for more details.
  - Initiating a conversation when a match is confirmed.
- **API Requests for Match Preferences:** Updates Firebase Firestore with user interactions, including:

- Storing liked and disliked matches in the user's profile.
    - Fetching new matches based on updated preferences.
    - Synchronizing match data across multiple sessions or devices.
- **Real-time Status Updates:** Ensures match interactions are updated instantly using:
    - Uses Firebase Firestore to instantly update match statuses as they change.
    - WebSockets or Firebase listeners to notify users of new matches.

3.6 Chat & Notifications

Purpose & Design: This subsystem enables real-time messaging and notifications for matched users.

3.6.1 Subsystem Hardware

This subsystem operates entirely on mobile devices and does not require any additional hardware.

3.6.2 Subsystem Operating System

This subsystem runs on mobile platforms and integrates real-time communication and notification services:

- **Android** – Utilizes background services to handle push notifications and maintain message synchronization even when the app is inactive.
- **iOS** – Integrated Apple Push Notification Service to deliver real-time chat notifications and ensures optimized handling of background message updates.

3.6.3 Subsystem Software Dependencies

This subsystem relies on the following libraries and frameworks to enable real-time chat functionality and notifications:

- **React Native UI Components** – Provides the chat interface, including message bubbles, input fields, and conversation lists.
- **WebSockets or Firebase Firestore's Real-Time Sync** – Ensures instant message delivery and synchronization across devices.
- **React Native Push Notifications** – Handles system alerts for new messages and updates, ensuring users are notified even when the app is in the background.
- **Firebase Cloud Messaging** – Enables cross-platform push notifications for both Android and iOS.

3.6.4 Subsystem Programming Languages

This subsystem is implemented using:

- **JavaScript** – Used for building the chat interface, handling real-time message updates, and managing user interactions in React Native.
- **TypeScript** – Provides static typing to improve code reliability and maintainability in chat-related components.

3.6.5 Subsystem Data Structures

This subsystem utilizes the following data structures to manage chat messages and notifications:

- **Chat Message Object** – Stores details of each message exchanged between users, including:
  - Sender ID
  - Receiver ID
  - Message content (text, media, or emojis)
  - Timestamp of when the message was sent
  - Read status (e.g., sent, delivered, read)
- **Notification Object** – Tracks notifications related to chat and user interactions, including:
  - Notification type (e.g., new message, match confirmation)
  - Associated user ID
  - Timestamp of the notification
  - Read status (e.g., unread, dismissed)

3.6.6 Subsystem Data Processing

This subsystem handles various processing tasks to ensure real-time messaging and notification delivery:

- **API Requests for Messaging:** Manages communication with Firebase Firestore to:
  - Send new chat messages and store them in the database.
  - Retrieve conversation history when users open a chat.
  - Update message statuses (e.g., sent, delivered, read).
- **Push Notification Triggers:** Ensures users receive timely alerts based on activity, including:
  - Sending push notifications when a new message is received.
  - Notifying users about unread messages after a set period.
  - Handling background notifications using Firebase Cloud Messaging (FCM) and Apple Push Notification Service (APNS).
- **Real-Time Message Sync:** Uses WebSockets or Firebase Firestore's real-time updates to:
  - Instantly update message lists when new messages arrive.
  - Ensure messages appear synchronously across devices without manual refresh.
  - Synchronize read receipts and typing indicators for an improved chat experience.

4 Y Layer Subsystems: Matchmaking & Social Engagement

The **Matchmaking & Social Engagement Layer** is responsible for connecting users based on their dining preferences and fostering social interactions within the Social Foodies app. This layer enables users to match with others, engage in conversations, and participate in a reward system for dining experiences. The subsystems within this layer handle user matching, chat functionality, and social engagement incentives.

4.1 Layer Hardware

The following devices interact with this layer:

- **User Devices:** Smartphones (Android, iOS) running the Social Foodies app.
- **Development & Testing Devices:** Laptops and desktops (Windows, macOS) used by developers to implement and test matchmaking algorithms and chat features.

4.2 Layer Operating System

The matchmaking and social engagement functionalities operate across multiple platforms:

- **User Devices:** Android and iOS smartphones interact with matchmaking and chat services via the mobile app.
- **Development Environments:** Windows, macOS, and Linux for coding, testing, and deploying matchmaking features.
- **Cloud Services:** Firebase and third-party APIs for real-time messaging and matchmaking computations.

4.3 Layer Software Dependencies

The **Matchmaking & Social Engagement Layer** relies on various software components to facilitate matching, chat functionality, and user interactions:

Matching & User Interaction

- Firebase Firestore : Stores user profile preferences and matches.
- Firebase Authentication:Manages user identity and authentication for matchmaking.
- Firestore Database: Stores user matches and ongoing conversations.

Chat Services

- Firebase Realtime Database:Enables real-time chat and instant message delivery.
- WebSockets: Supports real-time messaging for immediate responses.

Social Engagement & Rewards

- Firebase Cloud Functions : Triggers engagement-based rewards.
- Firebase Firestore : Tracks user participation, dining check-ins, and social interactions.

4.4 User Matching

This subsystem manages user matching based on dining preferences, proximity, and social engagement metrics. It uses a matchmaking algorithm to suggest compatible users who can dine together.

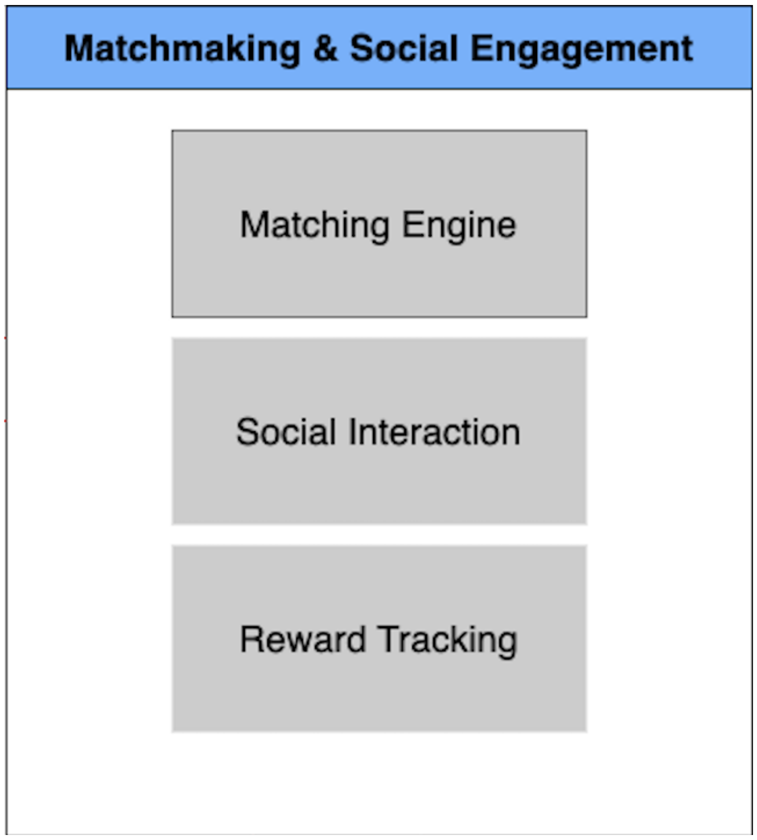

Figure 3: Matchmaking & Social Engagement Diagram

4.4.1 Subsystem Hardware

- No dedicated hardware; cloud-based matching services.
- User devices (smartphones) process and display matches.

4.4.2 Subsystem Operating System

- Cloud-based and platform-independent.
- User interactions occur on Android and iOS.

4.4.3 Subsystem Software Dependencies

- **Firebase Firestore :** Stores user profiles and preferences.
- **Google Maps API :** Determines user proximity within a 5-mile radius.
- **Firebase Functions :** Runs matchmaking algorithms periodically.

4.4.4 Subsystem Programming Languages

- **JavaScript (Node.js, React) :** Backend matchmaking logic.
- **NoSQL Queries :** Fetches and filters potential matches.

4.4.5 Subsystem Data Structures

- **User Object:** Stores user preferences, location, and match history.
- **Match Object:** Records matched users and pending invitations.

4.4.6 Subsystem Data Processing

- Retrieves users within a 5-mile radius.
- Compares preferences (cuisine, budget, dietary restrictions).
- Generates match suggestions.
- Updates match history after successful connections.

4.5 Chat & Social Interactions

This subsystem allows users to chat with their matches, plan dining meetups, and engage in conversations. It supports direct messaging and group chats.

4.5.1 Subsystem Hardware

- No dedicated hardware; chat is managed via Firebase Realtime Database.
- User devices (smartphones) handle message sending and receiving.

4.5.2 Subsystem Operating System

- Cloud-based and platform-independent.
- Chat interactions occur on Android and iOS.

4.5.3 Subsystem Software Dependencies

- **Firebase Realtime Database** :Stores and syncs chat messages.
- **WebSockets :** Supports instant message updates.

4.5.4 Subsystem Programming Languages

- **JavaScript (React, Node.js) :**Handles chat functionality.
- **NoSQL Queries :** Stores and retrieves messages.

4.5.5 Subsystem Data Structures

- **Message Object:** Stores sender, receiver, timestamp, and message content.
- **Chatroom Object:** Manages conversation history between matched users.

4.5.6 Subsystem Data Processing

- Stores chat messages in real-time.
- Syncs conversations across devices.
- Notifies users of new messages.

4.6 Social Engagement & Reward Tracking

This subsystem encourages user engagement through rewards and gamification. Users earn points for interacting with matches, checking in at restaurants, and participating in social dining experiences.

4.6.1 Subsystem Hardware

- No dedicated hardware; cloud-based tracking of engagement and rewards.

4.6.2 Subsystem Operating System

- Operates on Firebase Cloud Functions.
- User interactions occur on Android and iOS.

4.6.3 Subsystem Software Dependencies

- **Firebase Firestore :** Tracks reward points and redemption history.
- **Firebase Cloud Functions** :Automates point calculations.

4.6.4 Subsystem Programming Languages

- **JavaScript (React, Node.js) :** Handles reward logic.
- **NoSQL Queries :** Retrieves user engagement metrics.

4.6.5 Subsystem Data Structures

- **User Rewards Object:** Stores accumulated points.
- **Transaction History Object:** Logs rewards earned and redeemed.

4.6.6 Subsystem Data Processing

- Assigns points for match interactions.
- Tracks dining check-ins.
- Manages reward redemptions.

5 Z Layer Subsystems: Backend & External Services

5.1 Layer Hardware

The Backend & External Services layer for the Social Foodies app relies on the following hardware components to support its overall functionality:
Development and Testing Devices:

- Devices used by the team to build, test, and debug backend services ensure seamless functionality across all subsystems.
- Examples of devices used are laptops,desktops, or phones (running macOS,Windows, android, or ios).

Networking Hardware:

- Hardware required to facilitate communication between backend services, Firebase, and client applications during development and testing. Items used are Ethernet cables or wireless WIFI for development and testing devices.

5.2 Layer Operating System

Layer Operating System The Backend & External Services layer for **social foodies** app relies on the following operating systems to support its functionality:

Development and Testing Environments:

- These operating systems are used by team during the coding, testing, and debugging of backend services.
- **Examples:**
  - **Windows & Android:** Team members using Windows-based laptops or desktops or phones that run Android.
  - **macOS & IOS:** For developers using Apple devices like MacBooks or IOS devices such as iPhones.

Firebase Integration:

- Firebase is a cloud-based service that handles back-end functionality. The backend interacting with firebase needs the following compatible operating system to run firebase tools.
- **Examples:**
  - Linux, Windows, or macOS, depending on the user computer.

User Devices:

- The operating systems on end-user devices that interact with the back-end services through the Social Foodies mobile app.
- **Examples:**
  - **Android:** For users with Android phones.
  - **iOS:** For users with iPhones or IOS devices.

5.3 Layer Software Dependencies

Layer Software Dependencies The Backend & External Services layer for the social foodies app relies on the following software dependencies to support functionality such as reward management, location services, notification services, and database management:

Frontend & Backend Frameworks:

- **Firebase:** Provides tools for authentication, access to the Firestore database, and cloud storage.

Database & Data Storage:

- **Firebase Firestore :** database used to store user information, places, and visit history.
- **Firebase Authentication:** Manages user sign-ins and authentication processes.

Location & Mapping Services:

- **Google Maps API :** Used to retrieve restaurant locations, user, and mapping functionality.
- **Google Places API** : Allows for fetching location details such as restaurant reviews, addresses, and ratings.

Development & Testing Tools:

- **Git & GitHub** : Version control and collaboration for frontend and backend development.
- **Node.js & npm** : Provides the runtime environment and package manager for React development.
- **Prettier** : Code formatting tool to maintain code quality.

5.4 Location Services Subsystem

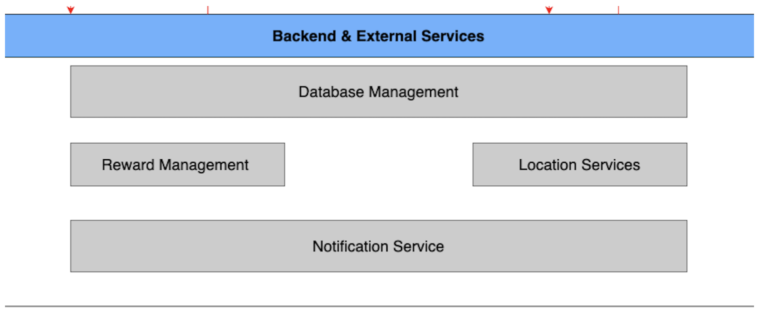

Figure 4: Backend & External Services Diagram

5.4.1 Subsystem Hardware

- **Devices :** Laptops or desktops used to access the internet
- **User Devices :** Phones of operating systems, such as iOS and Android

5.4.2 Subsystem Operating System

- **React, nodejs :** Due to the use of react and nodejs, this application can run on both Android and iOS.

5.4.3 Subsystem Software Dependencies

- **Npm :** Npm is the software used to run the software, both for coding and for testing purposes in a quick and timely manner.
- **Google Maps API :** Google Maps can create maps from a location that users will very likely be familiar with.
- **Google Maps Places API :** The Google Maps Places API can retrieve useful information for the use of matching and personal preferences of the users.
- **Google Maps Geolocation API :** The Geolocation API allows the Maps API to work from any location the User is currently at with no extra math or permissions involved.

5.4.4 Subsystem Programming Languages

- **HTML :** HTML is used to create the representation of the map itself.
- **TypeScript/Javascript :** TypeScript and Javascript are used in concert to create the extra functions needed by the rest of the application, such as the information retrieval and storage from Places API.
- **CSS :** The CSS is used to create stylings used in the map itself, such as fonts and things.

5.4.5 Subsystem Data Structures

- Data is sent to the database,and it should be grouped into a single structure full of information via the places API. Current data to be sent includes information fields such as a restaurants price level, number of ratings, and current overall rating.

5.4.6 Subsystem Data Processing

- No huge algorithms or processing is in this subsystem. Any functions are called directly from the API discussed in this section.

5.5 Database Management Subsystem

Purpose: This subsystem is responsible for storing and managing all application data, including user information, restaurant locations, visit history, and reviews. The system relies on Firebase Firestore, a database that allows real-time data , making it efficient for tracking user interactions and restaurant details.

5.5.1 Subsystem Hardware

This subsystem does not require dedicated hardware since Firebase Firestore is a cloud-based service. The following devices interact with this subsystem:

- **Devices:** Laptops or desktops used to access the firebase database.
- **User Devices:** Smartphones running the Social foodies app, which fetch and store data via Firebase.

5.5.2 Subsystem Operating System

- Firebase Firestore operates on Google cloud
- Firebase can be used on Windows, macOS, or Linux.

5.5.3 Subsystem Software Dependencies

- **Firebase Firestore SDK** : Provides database storage and retrieval functionality.
- **Firebase Authentication** : Handles user log-in and authentication.
- **Google Firebase Console**:Used for database monitoring.
- **Axios (React Library)** :Handles API requests to Firebase.

5.5.4 Subsystem Programming Languages

- **JavaScript (React)**:Used for interacting with Firestore via Firebase SDK.
- **NoSQL Queries** : Used to retrieve and manipulate data in Firestore.

5.5.5 Subsystem Data Structures

- **User Object**:Stores user information, including profile details and visit history.
- **Places Object** :Contains restaurant details such as name, location, rating, and number of visits.
- **Reviews Object** :Stores user-submitted ratings and comments for restaurants.
- **Visits Object** :Logs user check-ins and timestamps for restaurant visits.

5.5.6 Subsystem Data Processing

- **User Data Retrieval**: Queries user profiles and visit history from Firestore.
- **Location-Based Filtering**: Retrieves places within a 5-mile radius using Google Maps API.
- **Visit Tracking**: Increments visit counts when users check in at a restaurant.
- **Review Management**: Stores and retrieves user reviews and ratings for places.
- **User Authentication**: Validates and securely manages user logins through Firebase Authentication.

5.6 Notification Services Subsystem

Purpose: The Notification Services subsystem is responsible for sending real-time updates and alerts to users. It integrates with Firebase Cloud to deliver push notifications about new rewards, restaurant suggestions, and visit confirmations. This subsystem ensures timely communication between the app and its users.

5.6.1 Subsystem Hardware

This subsystem does not require dedicated hardware as Firebase Cloud handles notifications through cloud service. The following devices interact with this subsystem:

- **Devices**: Laptops or desktops used for testing push notifications.
- **User Devices**: Smartphones running the Social Foodies app, receiving notifications via the Firebase Cloud.

5.6.2 Subsystem Operating System

- Firebase is used to track and store rewards .The Operating system used here is the following Windows, macOS, or Linux.

5.6.3 Subsystem Software Dependencies

- **Firebase Firestore SDK** : stores the reward points and redemption history.
- **Firebase Authentication** : Ensures that reward points are linked to authenticated users.
- **Google Firebase Console** : Used for monitoring reward transactions and debugging.
- **Axios (React Library)** : Handles API requests to Firebase while retrieving and updating reward points.

5.6.4 Subsystem Programming Languages

- **JavaScript (React)** : Handles reward calculation, tracking, and redemption logic.
- **NoSQL Queries :** Used to retrieve and update user reward points in Firebase.

5.6.5 Subsystem Data Structures

- **User Rewards Object** :Stores information about a user reward points, including total points earned and redeemed.
- **Transaction History Object** :Keeps track of every reward, such as points earned from visits and points redeemed for rewards.
- **Redemption Object :**Tracks redeemed rewards, including the reward description and the number of points used.

5.6.6 Subsystem Data Processing

- **Reward Calculation**: Assigns points to users based on their check-ins and visit frequency.
- **Transaction Logging**: Updates the Firebase with point earnings and redemption.
- **Redemption Validation**: Ensures users have enough points before allowing redemption.
- **User Profile Synchronization**: Links earned rewards with user accounts and updates their total points.

5.7 Reward Management Subsystem

Purpose: The Reward Management subsystem is responsible for tracking and managing user reward points based on their visits. This interacts with Firebase to store reward points and redemption history, allowing users to accumulate points for going to locations and redeeming them for discounts or special offers.

5.7.1 Subsystem Hardware

This subsystem does not require dedicated hardware since all reward tracking is managed within Firebase. The following devices interact with this subsystem:

- **Devices**: Laptop or desktop.
- **User Devices**: Smartphones running on Social Foodie app, where users can view, earn, and redeem rewards

5.7.2 Subsystem Operating System

- Firebase Cloud operates on Google cloud infrastructure, the way it interact with Firebase is on Windows, macOS, or Linux.
- End-users receive notifications on Android and iOS devices.

5.7.3 Subsystem Software Dependencies

- **Firebase Cloud** : Handles push notifications to users.
- **Firebase Firestore SDK** : Stores notification preferences and history.
- **Axios (React Library)** : Handles API requests for sending notifications.
- **Google Firebase Console** : Used to configure and monitor notification services.

5.7.4 Subsystem Programming Languages

- **JavaScript (React)** : Manages sending and receiving notifications via Firebase.
- **NoSQL Queries** : Used for storing and retrieving notification preferences in Firebase.

5.7.5 Subsystem Data Structures

- **User Notification Preferences Object** : Stores user preferences for notification types and frequency.
- **Notification Object** : Logs notifications sent to users, including message content and timestamps.

5.7.6 Subsystem Data Processing

- **User Subscription Management**: Stores user preferences for receiving notifications.
- **Push Notification Trigger**: Sends notifications based on user actions, such as earning rewards or discovering new restaurants.
- **Notification Logs**: Records delivered notifications for reference.

6 Appendix A

References

[1] Google Developers. Google Maps Documentation, 2025. Accessed: 2025-02-05.

[2] Google Firebase. Firebase Documentation, 2025. Accessed: 2025-02-01.

[3] Meta Open Source. React Documentation, 2025. Accessed: 2025-02-05.

[1] [3] [2]