**Software requirements specification (SRS)**

**TEAM-04**

**Link to the SRS:** <https://github.com/Ramakotireddy9505/GDP-Team04-foodiefinds/wiki/Software-requirements-specification-(SRS)>

**Project Information**

# Project Charter:

## **Summary**

Foodie Finds is a restaurant discovery and dining recommendation application designed to assist users in locating nearby eateries, browsing reviews, and making reservations seamlessly.

## **Motivation**

Foodie Finds aims to enhance the client's workflow by providing a centralized platform for restaurant discovery and reservation management. By streamlining the process of finding and booking dining options, Foodie Finds will improve user satisfaction and operational efficiency.

## **Required Functionality**

For a detailed overview of the required functionality, please refer to the Functional Requirements Document.

1. **Task Management:** Manage orders, deliveries, and restaurant operations, assigning tasks to specific individuals or teams.
2. **Deadline Management:** Set deadlines for order preparation, delivery, or reservation confirmations to ensure smooth operations.
3. **Reliability:** Ensure that the platform is always accessible and functions properly to avoid disruptions in service.
4. **Scalability:** As more restaurants and users join the platform, it should be able to handle increased traffic without slowing down.
5. **Easy to Use:** A simple and intuitive interface for users, restaurants, and delivery personnel.
6. **Calendar View:** Allow restaurants to manage their availability for reservations and users to plan their orders ahead.
7. **Backend Stuff:** Secure user accounts, encrypted data storage, and reliable server infrastructure.
8. **Real-Time Updates:** Provide instant notifications for new orders, order status changes, and delivery updates.
9. **Security:** Implement strong login mechanisms, data encryption, and compliance with data protection regulations.
10. **Database:** Use a database that can efficiently handle the platform's requirements, whether standard SQL or a flexible NoSQL solution.

## **Roles and Responsibilities**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Role** | **Position** | **Contact Information** |
| Malaya Sugandhini | Developer | Project Manager | [S566637@nwmissouri.edu](mailto:S566637@nwmissouri.edu) |
| Spandana Pandi | Developer | Software Engineer | [S565959@nwmissouri.edu](mailto:S565959@nwmissouri.edu) |
| Ramakotireddy Ragipindi | Manager | Software Engineer | [S566606@nwmissouri.edu](mailto:S566606@nwmissouri.edu) |
| Akhila Mylavarapu | Developer | Software Engineer | [S566600@nwmissouri.edu](mailto:S566600@nwmissouri.edu) |
| Jahnavi chava | Quality Assurance | Team Member | [S566948@nwmissouri.edu](mailto:S566948@nwmissouri.edu) |

# About the developers

|  |  |  |  |
| --- | --- | --- | --- |
| **Developer** | **GitHub Username** | **Link to Profile** | **Strengths and Interests** |
| Malaya Sugandhini | [MalayaSugandhini](https://github.com/MalayaSugandhini) | [Profile](https://github.com/MalayaSugandhini) | - Proficient in front-end development technologies such as HTML, CSS, and JavaScript. - Experience in UX/UI design and creating user-friendly interfaces. |
| Ramakotireddy Ragipindi | [Ramakotireddy9505](https://github.com/Ramakotireddy9505) | [Profile](https://github.com/Ramakotireddy9505) | - Strong background in backend development with Node.js and Express. - Familiarity with database management systems like MongoDB and SQL. |
| Jahnavi chava | [Jahnavichava30](https://github.com/Jahnavichava30) | [Profile](https://github.com/Jahnavichava30) | - Skilled in full-stack development, with experience in both frontend and backend technologies. |
| Akhila Mylavarapu | [akhilamylavarapu](https://github.com/akhilamylavarapu) | [Profile](https://github.com/akhilamylavarapu) | - Interest in exploring new frameworks and technologies for web development. |
| Spandana Pandi | [SpandanaPandi](https://github.com/SpandanaPandi) | [Profile](https://github.com/SpandanaPandi) | - Skilled in Java, C#, Javascript, HTML, CSS, and .NET Technology. |

# Problem Statement:

# Foodie Finds

## **Introduction**

Finding and booking restaurants is often inefficient and frustrating due to flaws in the current systems.

## **Problem Statement**

Food delivery apps like Uber Eats lack key features that would make ordering food easier and more enjoyable. This makes the apps less appealing and harder for users to explore various food options.

## **Issues**

### **Fragmented Information**

Restaurant details like menus and reviews are scattered across multiple platforms, making it hard for users to make informed choices.

### **Complicated Reservations**

Booking tables is often complex and time-consuming, especially during busy times.

### **Lack of Personalization**

Current platforms don't offer personalized restaurant recommendations, leading to decision fatigue and less satisfaction.

### **Security Concerns**

Weak security measures for user data reduce trust and engagement.

### **Performance Issues**

Growing user demand leads to slow loading times and system glitches, hurting user experience.

## **Conclusion**

The current approach to restaurant discovery and reservations is inefficient and frustrating. We need to identify and document these problems carefully to foster innovation and improve restaurant technology.

# Design

# Use Cases

## **Use Case 1:** User Registration and Authentication

**Objective:** Allow users to register and log in for personalized features.

### Interactions

* **Input:** Personal info (name, email), password.
* **Processing:** Validate and securely store credentials, authenticate during login.
* **Output:** Successful registration/login grants access to personalized features.
* **Results:** Users can securely register and log in for personalized access.

## **Use Case 2**: Browsing and Adding Restaurants

**Objective:** Users find top-rated restaurants; admins add new restaurants and reviews.

### Interactions

* **Input:** Users browse/search for restaurants; admins add restaurants and reviews.
* **Processing:** Display listings/search results, add new entries.
* **Output:** User-friendly browsing and easy admin additions.
* **Results:** Users easily find top-rated restaurants; admins efficiently add new ones.

## **Use Case 3**: Restaurant API Connectivity

**Objective:** Fetch real-time availability and menu changes from restaurant APIs.

### Interactions

* **Input:** API communication for data retrieval.
* **Processing:** Fetch up-to-date availability and menus.
* **Output:** Real-time updates on tables and menus.
* **Results:** Users see the latest availability and menu info.

## **Use Case 4:** Personalized Restaurant Search

**Objective:** Users search for restaurants based on ratings and cuisines.

### Interactions

* **Input:** User search keywords and filter selections.
* **Processing:** Search algorithms match criteria.
* **Output:** Search results that match user preferences.
* **Results:** Users find restaurants based on personal preferences.

## **Use Case 5:** Admin Reviews

**Objective:** Admins add reviews for restaurants.

### Interactions

* **Input:** Admin selects restaurant and inputs reviews.
* **Processing:** Validate and store reviews securely.
* **Output:** Reviews appear in restaurant profiles.
* **Results:** Admins enhance user experience with added reviews.

## **Use Case 6:** Review Sorting

**Objective:** Users sort reviews by different criteria.

### Interactions

* **Input:** User selects sorting criteria.
* **Processing:** Reorder reviews accordingly.
* **Output:** Display sorted reviews.
* **Results:** Users find relevant reviews easily.

## **Use Case 7:** Reservation Booking

**Objective:** Users make restaurant reservations through the app.

### Interactions

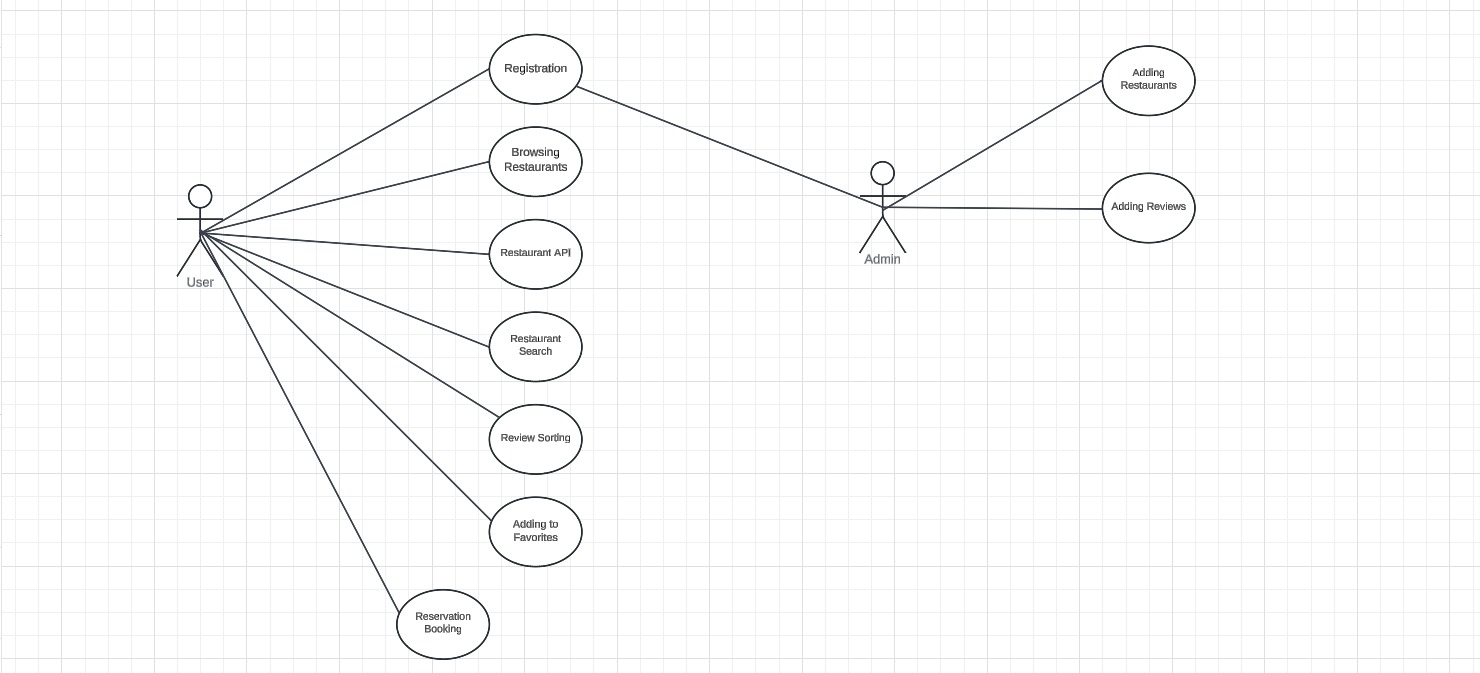
* **Input:** User selects restaurant, date, time, and party size.
* **Processing:** Display options, confirm and generate reservation.
* **Output:** Confirmation message/details, optional notification.
* **Results:** Users conveniently book reservations for a hassle-free dining experience.

## **Use Case 8:** Favorites Management

**Objective:** Allow users to manage their favorite restaurants or recipes.

### Interactions

* **Input:** User marks restaurants or recipes as favorites, views the favorites list, or removes items from it.
* **Processing:** Update and manage the favorites list based on user actions.
* **Output:** Display the updated favorites list to the user.
* **Results:** Users can easily manage their favorites, enhancing their experience with personalized lists.



# Functional Requirements

## 1. System Registration and Authentication

* The system SHALL validate user-provided personal information during registration, including name, email, and password.
* The system SHALL authenticate users during the login process.

## 2. Restaurant Exploration and Management

* The system SHALL allow users to navigate through the app to discover restaurants.
* The system SHALL enable users to search for restaurants based on ratings and preferred cuisines.
* The system SHALL facilitate admin input for adding restaurants and reviews.

## 3. Integration with Restaurant Data Sources

* The system SHALL communicate with restaurant APIs for data retrieval.
* The system SHALL retrieve up-to-date information on restaurant availability and menus.

## 4. Personalized Restaurant Search

* The system SHALL provide access to the restaurant search feature.
* The system SHALL allow users to enter search keywords and select filters.

## 5. Admin Review Management

* The system SHALL provide access to the restaurant management interface for admins.
* The system SHALL include admin-contributed reviews in the restaurant profiles.

## 6. Sorting of Restaurant Reviews

* The system SHALL provide access to the reviews section of a restaurant and allow users to select sorting criteria for reviews.

## 7. Restaurant Reservation Management

* The system SHALL offer a reservation section for users to book reservations at restaurants.
* The system SHALL allow users to select restaurant, date, time, and party size for the reservation.

## 8. Favorites Management

* The system SHALL provide users with the ability to mark restaurants or recipes as favorites.
* The system SHALL allow users to view a list of their favorite restaurants or recipes.
* The system SHALL enable users to remove restaurants from their favorites list.

# Negative Requirements

1. The system SHALL NOT allow users to book orders directly.
2. The system SHALL NOT include any advertisements.
3. Sensitive data, including personal information, SHALL be securely handled and protected.

# Foodie finds Non-functional Requirements List

## **Performance**

* Fast loading times for restaurant listings and profiles.
* Smooth user experience throughout the app.
* Quick response times for search queries and reservation requests.
* Optimal performance even under network congestion.

## **Security**

* Ensure only authorized users access the system securely.
* Protect user and admin data stored in Firebase Firestore collections with encryption.
* Regularly backup data and have a recovery plan in place.

## **Stability**

* Reliable server backend to ensure uninterrupted access to restaurant listings, reviews, and reservation data.
* High availability to prevent downtime during peak usage periods.
* Robust error handling to minimize disruptions and crashes.

## **Availability**

* Continuous availability of the application for users to access restaurant information and make reservations.
* Redundancy and failover mechanisms to maintain service availability in case of server failures or maintenance activities.
* Support for multiple platforms and devices to ensure accessibility for a wide range of users.

## **Scalability**

* Ability to handle increasing user traffic and data volume without degradation in performance or reliability.
* Scalable infrastructure to accommodate growth in user base and restaurant listings over time.
* Efficient resource utilization to optimize cost-effectiveness while scaling the system.

## **Compatibility**

* Compatibility with various devices, operating systems, and web browsers to reach a broader audience.
* Integration with external APIs and services to support real-time updates for menus, availability, and reservations.
* Adherence to industry standards and best practices to ensure interoperability with existing systems and technologies.

## **Usability**

* Intuitive user interface design to facilitate easy navigation and interaction for users of all skill levels.
* Accessibility features to support users with disabilities and diverse needs.
* Clear and concise messaging and instructions to guide users through the app's features and functionalities.

# Data Management Plan

**Data Management for Foodie Finds**

**1.Users Collection:**

•Each document in the "users" collection represents a user.

•Fields: UserID, Name, Email, Password, Role, Preferences.

**sample document of above collection :**

{

"UserID": "",

"Name": "",

"Email": "",

"Password": "",

"Role": "",

"Preferences": ""

}

**2.Restaurants Collection:**

•Each document in the "restaurants" collection represents a restaurant.

•Fields: RestaurantID, Name, Location, Cuisine, PriceRange, AverageRating, OpenHours, MenuItems, TableAvailability, Photos.

Sample document in collection:

{

"RestaurantID": "",

"Name": "",

"Location": "",

"Cuisine": "",

"PriceRange": "",

"AverageRating": "",

"OpenHours": "",

"MenuItems": "",

"TableAvailability": "",

"Photos": ""

}

**3.Reservations Collection:**

•Each document in the "reservations" collection represents a reservation.

•Fields: ReservationID, UserID, RestaurantID, Date, Time, PartySize.

sample document of above collection:

{

"ReservationID": "",

"UserID": "",

"RestaurantID": "",

"Date": "",

"Time": "",

"PartySize": ""

}

**4.Reviews Collection:**

•Each document in the "reviews" collection represents a review.

•Fields: ReviewID, UserID, RestaurantID, Rating, ReviewText, Date.

sample document of above collection:

{

"ReviewID": "",

"UserID": "",

"RestaurantID": "",

"Rating": "",

"ReviewText": "",

"Date": ""

}

**Initial Plan to Secure Data:**

**Access Restriction:** Use role-based access control (RBAC) to restrict access depending on user role. Admins will oversee restaurant and review features, while regular users will concentrate on restaurant discovery and reservations.

**Requirements to Data Storage:**

**System Registration and Authentication:** Store user registration and authentication data in the Users collection, which includes fields like name, email, and password. RBAC ensures that personalized features are only accessible to verified users with the proper roles.

**Restaurant Exploration and Management:** Maintain restaurant data in the Restaurants collection, such as name, location, cuisine, ratings, and menu items. Admins can edit this information through the admin interface, which is synced with the Restaurants and Reviews collections.

**Integration with Restaurant Data Sources:** Fetch real-time changes on restaurant availability and menus from external APIs and save them to the Restaurants collection.

**Personalized Restaurant Search:** Use user preferences from the Users collection to make personalized recommendations. Search engines match user criteria to restaurant data in the Restaurants collection to produce relevant results.

**Admin Review Management:** Securely store admin-contributed reviews in the Reviews collection, which are tied to specific restaurant and user IDs.

**Sorting of Restaurant Reviews:** Restaurant reviews can be saved in the Reviews collection and sorted according to user preferences.

**Restaurant Reservation Management:** The Reservations collection contains reservation information such as the restaurant, date, time, and number of guests. To streamline the booking process, select available restaurants and reservation options from the Reservations collection.

# List of Proposed Prototypes

**Project: Foodie Finds - Web Application**

**First Prototype: User Registration and Authentication**

Allow users and admins to log in and confirm their identity to access the application features.

**Features:**

1. The registration form includes validation checks for user information.
2. The login page has boxes for your username and password.

**Second Prototype: Browsing Restaurants and Adding Restaurants**

Users can easily find top-rated restaurants based on reviews, while admins can effortlessly add new restaurants with reviews.

**Features:**

1. Search functionality allows users to find restaurants based on ratings and preferred cuisines.
2. Admins have access to an interface to add new restaurants.
3. Admins can also add reviews for the newly added restaurants.

**Third Prototype: Connectivity to Restaurant APIs**

The app needs to connect with restaurant APIs to fetch real-time availability and menu changes.

**Features:**

1. The app integrates with restaurant APIs to fetch real-time data.
2. Users can view the latest information on restaurant availability and menus within the app.

**Fourth Prototype: Restaurant Search based on personal recommendations**

Users can search for restaurants based on ratings and preferred cuisines.

**Features:**

1. Users have access to the restaurant search feature within the app.
2. The application has to find restaurants that match the user's search criteria.

**Fifth Prototype: Adding Reviews (Admin Functionality)**

Admins can add reviews for restaurants they have added to the application.

**Features:**

1. Admins have access to a restaurant management interface within the app.
2. The interface provides functionality for admins to add reviews.

**Sixth Prototype: Review Sorting**

Users can sort restaurant reviews by different criteria.

**Features:**

1. Users have access to the reviews section of a restaurant within the application.
2. Reviews are reordered based on the user-selected sorting criteria.

**Seventh Prototype: Reservation Booking**

Users want to make reservations at restaurants through the application.

**Features:**

1. Easy access to the reservation area within the app.
2. Easy selection of restaurant, day, time, and group size.

**Eight Prototype: Firebase Database Integration**

The database will be the firebase which have different collections like users, Restaurants, Review etc.

**Features:**

1. Configure the Firebase project and integrate Firebase SDK into the FoodieFinds application.
2. Implement Firebase for user registration and authentication functionality.
3. Design a Firebase database structure to efficiently store and retrieve restaurant listings, reviews, reservations, and user data.

# Meeting Minutes

# Client Meeting ‐ 1

### **Who was present:**

1. Rama koti reddy Ragipindi
2. Akhila Mylavarapu
3. Malaya Sugandhin
4. Jahnavi Chava
5. Spandana Pandi

### **Meeting agenda:**

1. problem statement overview
2. Admin and User interaction to the application
3. How users and administrators will utilize the application and navigate its workflow.
4. Handle any questions or concerns brought up by the client or team members.

### **Specific questions asked (and who asked them):**

1. Question1: Could you give some insight into the project?

Asked By: Client

1. Question2: Do you possess alternative ideas concerning the project?

Asked By: Client

1. Question3: How will the user determine the best restaurant during their search process?

Asked By: Client

1. Question4: Where and in what manner will the data of both users and administrators be stored?

Asked By: Client

1. Question5: What challenges are anticipated in the project, and what strategies will be employed to address them?

Asked By: Client

### **Specific answers given (and who gave them)**

1. Question1: Could you give some insight into the project?

* Our web application allows users to easily book a table at their preferred restaurants. Users may easily browse through a wide selection of restaurants options, choose the goods they want. Users can also just book a table at the time and date of their choice, guaranteeing a wonderful meal. Our software seeks to give customers a simple method to take advantage of delicious cuisine and dining experiences whenever it's convenient for them.

Given By: Rama koti reddy Ragipindi

1. Question2: Do you possess alternative ideas concerning the project?

* We have implemented a role-based login system for users accessing our application. Upon logging in, users are directed to their respective home pages based on their role. Administrators, upon logging in, are redirected to the admin home page. Users have the capability to view and explore nearby restaurants based on their location.

Given By: Akhila Mylavarapu

1. Question3: How will the user determine the best restaurant during their search process?

* This update enables the system to adjust restaurant ratings when users leave positive reviews. Now, when diners share their positive experiences, the restaurant's overall rating will reflect this feedback accurately. This enhancement aims to improve the reliability of restaurant ratings, making it easier for users to find top-quality dining options on Foodie Finds.

Given By: Jahnavi chava

1. Question4: Where and in what manner will the data of both users and administrators be stored?

* The data will be stored in Firebase as collections corresponding to both users and administrators. This structure enables us to retrieve the data when users log in.

Given By: Malaya Sugandhin

1. Question5: What challenges are anticipated in the project, and what strategies will be employed to address them?

* To determine the number of available seats for reservation in Foodie Finds: Reservation System: Manage reservations with dates, times, and guest counts. Maintain tables with varying capacities. Availability Calculation: Query existing reservations for a given date and time. Calculate available seats by subtracting booked seats from total table capacity.

Given By: Spandana Pandi

### **Action items: what have you decided you will show the client in your next meeting:**

* Prototype Demonstration
* Development Plan
* Project Timeline
* Client Feedback

### **When is the next meeting (no more than two weeks away)**

* Next meeting is on - 05/08/2024.