GPS BASED RESTAURANT FINDING APPLICATION

Name: Abdul Rafay

Reg ID: 233679

Department: BS Software Engineering

Subject: Introduction to Software Engineering

Submitted to: Sir Syed Irfan Raza Naqvi

Table of Contents

Problem Statement:

"Existing restaurant finder apps lack real-time updates and personalized recommendations, hindering efficient dining discovery."

1. Introduction

1.1 Introduction:

The GPS-Based Restaurant Finder App project is initiated to address the growing need for a mobile application that simplifies the process of discovering nearby restaurants based on the user's GPS location. In response to the increasing reliance on mobile technology for navigation and decision-making, this application aims to provide a user-friendly solution for individuals seeking convenient and personalized restaurant recommendations.

1.2 Purpose and Scope:

This Software Requirements Specification (SRS) document serves as a comprehensive guide for the development team, outlining the functional and non-functional requirements necessary to build the GPS-Based Restaurant Finder App. It aims to provide a clear reference for developers, testers, and project managers, ensuring a shared understanding of the project goals and constraints. The scope of this document covers the initial version of the software, detailing the features, limitations, and expected outcomes.

1.3 Overview of the GPS-Based Restaurant Finder App:

The GPS-Based Restaurant Finder App will offer users the ability to effortlessly locate nearby restaurants using their device's GPS capabilities. With features such as location-based searches, customizable filters, map integration, and real-time updates, the application aims to provide a seamless and personalized experience for users. By allowing users to create profiles, save favorite restaurants, and receive timely notifications, the app seeks to enhance the overall restaurant discovery process. The development of this application aligns with the growing demand for innovative solutions in the mobile app market, offering a unique and valuable tool for individuals seeking dining options based on their preferences and location.

2. Stakeholder Identification

2.1 Business Stakeholders:

- Project Sponsor: Sir Syed Irfan Raza Naqvi
- Product Manager: Abdul Rafay
- Marketing Representative: Arslan Dilawar

2.2 End Users:

- General Users: Hassaan, Inayah
- Customers: Public

2.3 Developers:

- Software Developers: Abdul Rafay

2.4 Roles and Responsibilities of each Stakeholder:

2.4.1 Business Stakeholders:

- Project Sponsor: Approve major decisions, and provide financial support.
- Product Manager: Define product vision, and prioritize features.
- Marketing Representative: Communicate value proposition, and ensure market alignment.

2.4.2 End Users:

- General Users: Actively use the application, and provide feedback.
- Customers: Express specific needs and expectations, and participate in user testing.

2.4.3 Developers:

- Software Developers: Write code, and implement features.

3. System Overview

3.1 High-Level Description:

The GPS-Based Restaurant Finder App is designed to simplify the process of discovering nearby restaurants for users on the go. Targeted at individuals seeking convenient and personalized restaurant recommendations, the application leverages GPS technology to provide real-time information based on the user's current location. With user-friendly features and a seamless interface, the app aims to enhance the dining experience by offering quick and tailored restaurant suggestions.

3.2 Key Features and Functionalities:

The GPS-Based Restaurant Finder App incorporates a range of features to cater to the diverse needs of users. Key functionalities include location-based searches, customizable filters, map integration for visual directions, and the ability to create user profiles. Users can access real-time updates on restaurant information, make reservations directly through the app, and share their experiences on social media. The application prioritizes user engagement with features like in-app reviews, customizable alerts, and offline access to enhance the overall user experience.

3.3 System Architecture Overview:

The GPS-Based Restaurant Finder App is built on a modular and scalable architecture. The main components include the user interface for seamless interaction, a robust database for storing restaurant information, and integration with external APIs for map functionalities. The application's architecture facilitates efficient communication between components, ensuring quick and responsive user experiences. The technology stack comprises [mention technologies], providing a foundation for stability and future enhancements. Scalability considerations have been incorporated to accommodate a growing user base and expanding restaurant data. This section aims to give readers a holistic view of the application, setting the stage for more detailed requirements in subsequent sections of the SRS document.

4. Requirements

4.1 Functional Requirements:

1) Location-Based Search

- Users can initiate a search for nearby restaurants based on their current GPS location.
- The application should provide a detailed and intuitive process for users to access location-based search functionality.
- Users should be able to perform targeted searches by inputting the name of a specific restaurant.
- The search mechanism must be designed to retrieve and display accurate results based on the entered restaurant name.

2) Filtering Options:

- Users should have the ability to apply filters to refine search results based on various criteria, such as price range, cuisine type, and ratings.
- Specify the user interface elements and interactions that facilitate the application of filters.

3) Map Integration

- The application should integrate with maps to provide users with visual directions to the selected restaurant.
- The integration must be seamless and user-friendly, enhancing the overall navigation experience.
- The app should display restaurant locations on an interactive map.
- Specify how the map will present restaurant markers and any additional information when users interact with the map.

4) User Profiles

- Users should be able to create profiles within the app.
- Profiles should include features such as saving favorite restaurants and maintaining a search history.

5) Real Time Updates

- The application must provide real-time updates on restaurant information.
- Information to be updated in real-time includes opening hours, special offers, and any changes in the status of the restaurant.

6) Reservation Functionality

- Users should have the capability to make restaurant reservations directly through the app.
- Specify the steps involved in the reservation process and any confirmation mechanisms.

7) Social Media Integration

- The app should integrate with social media platforms to facilitate sharing of restaurant experiences.
- Specify the supported social media platforms, user interactions, and the information shared.

8) In-app Reviews
- Users should be able to leave reviews for restaurants within the app.
- Specify the review submission process, criteria for reviews, and how average ratings are calculated and displayed.

9) Customizable Alerts
- Users should have the ability to set and receive customizable notifications.
- Specify the types of notifications, user preferences, and the conditions under which alerts are triggered.

10) Offline Access
- The app should provide offline access features, allowing users to access certain functionalities without an internet connection.
- Specify which features are available offline and how the app handles data synchronization when the connection is re-established.

4.2 Non-Functional Requirements:

1) Performance
- Specify the expected response times for critical actions within the application, such as search queries and map interactions.
- Define performance benchmarks to ensure the application meets acceptable speed and responsiveness standards.

2) Scalability
- Outline how the system should scale to handle a growing number of users and increasing amounts of restaurant data.
- Define scalability requirements to accommodate future expansion without compromising performance.

3) Usability
- Specify criteria for an intuitive and user-friendly interface to enhance user experience.
- Include requirements for multilingual support to cater to a diverse user base.

4) Security

- Outline privacy measures to protect user data and ensure compliance with data protection regulations.
- Specify encryption standards and protocols to secure sensitive information during transmission and storage.

5) Reliability

- Define reliability requirements, including expectations for system uptime and availability.
- Specify measures to minimize downtime and ensure consistent service availability.

6) Cross-Platform Compatibility

- Ensure that the application is compatible with both iOS and Android platforms.
- Specify any platform-specific features or considerations.

7) Accessibility

- Include requirements for accessibility features, making the app usable by individuals with disabilities.
- Comply with accessibility standards such as WCAG (Web Content Accessibility Guidelines).

8) Data Backup and Recovery

- Specify how data should be backed up and at what frequency.
- Define procedures and requirements for data recovery in the event of system failures or data loss.

4.3 Domain Requirements:

1) Regulatory Compliance

- The application should comply with relevant regulations and standards governing the restaurant industry, including food safety regulations and licensing requirements.
- Compliance with data protection laws, such as GDPR (General Data Protection Regulation) or CCPA (California Consumer Privacy Act), to ensure the privacy and security of user data.

2) Payment Processing

- If the application supports in-app purchases or reservations requiring payment, it must comply with PCI DSS (Payment Card Industry Data Security Standard) for secure payment processing.
- Integration with secure payment gateways to facilitate transactions securely.

3) Restaurant Data Accuracy

- Ensure the accuracy and reliability of restaurant information, including business hours, addresses, and contact details.
- Regular verification and updating of restaurant data to reflect any changes in operation or offerings.

4) Licensing and Partnerships

- Compliance with licensing agreements and partnerships with restaurant owners or chains to access and display their information within the application.
- Clear terms of use and agreements outlining the responsibilities of both the application and the restaurants listed.

5) Geographic Coverage

- Consideration of geographic coverage and availability of restaurant data, ensuring comprehensive coverage in target regions or cities.
- Collaboration with local authorities or business associations to expand coverage and maintain up-to-date information.

6) Customer Feedback Handling

- Implementation of mechanisms to handle customer feedback, including complaints, suggestions, and inquiries regarding restaurant listings or services.
- Processes for addressing and resolving customer issues in a timely and satisfactory manner.

4.4 User Requirements:

1) Ease of Use

- The application should be intuitive and easy to navigate for users of all technical backgrounds.

- Users should be able to perform tasks such as searching for restaurants, making reservations, and leaving reviews without extensive training or guidance.

2) Personalization

- Users should have the ability to personalize their experience by saving favorite restaurants, setting preferences, and receiving tailored recommendations based on their past interactions.

3) Accessibility

- The application should be accessible to users with disabilities, incorporating features such as screen reader compatibility, adjustable font sizes, and alternative input methods.

4) Interactivity

- Users should be able to interact with the application seamlessly, with responsive feedback and intuitive controls for actions such as tapping, swiping, and entering text.

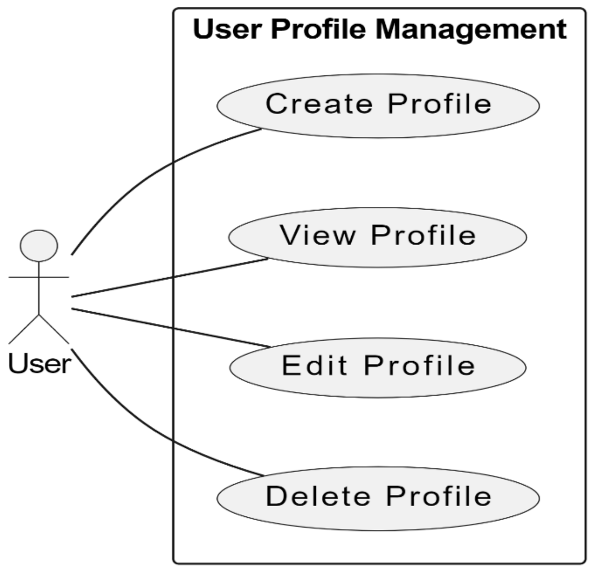
5. Design

5.1 User Profile Management Design:

5.1.1 Description:

- Data Structures:
    - User Profile: Contains user information such as username, email, password, saved restaurants, and search history.
- Information Flow:
    - When a user creates a profile, the provided information is stored in the user profile database.
    - Viewing, editing, and deleting functionalities interact with the user profile database to retrieve, update, or remove user data.
- Security Considerations:
    - User passwords should be securely hashed before storing them in the database to protect user privacy.
    - Access to user profiles and related functionalities should be restricted based on user authentication and authorization.

5.1.2 Use Case 1:

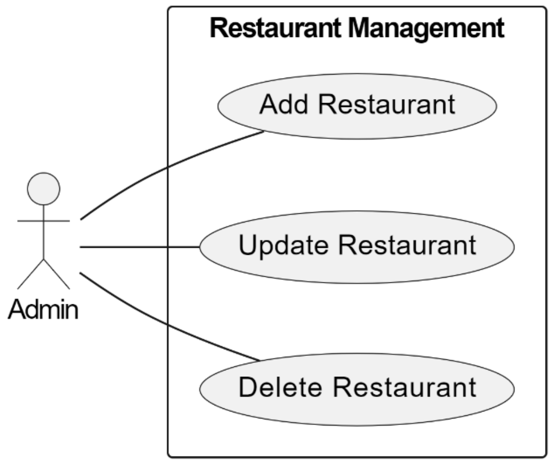


5.2 Restaurant Management Design:

5.2.1 Description:

- Data Structures:
  - Restaurant Information: Contains details such as restaurant name, location, cuisine type, ratings, reviews, and opening hours.
- Information Flow:
  - Admins can add, update, or delete restaurant information through an admin dashboard interface.
  - Restaurant data is stored in a separate database table and is accessible for viewing by both users and admins.
- Security Considerations:
  - Only authorized admins should have access to the restaurant management functionalities to prevent unauthorized modifications.
  - Input validation should be implemented to sanitize and validate restaurant data to prevent SQL injection or other security vulnerabilities.

5.2.2 Use Case 2:

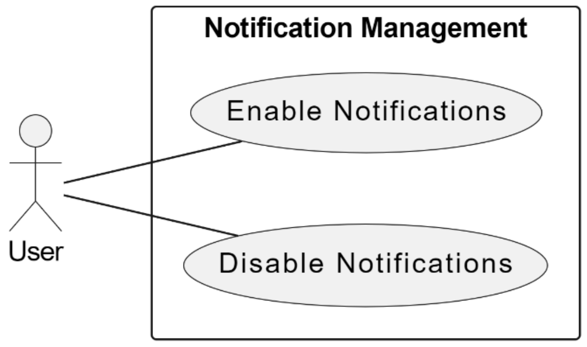


5.3 Notification Management Design:

5.3.1 Description:

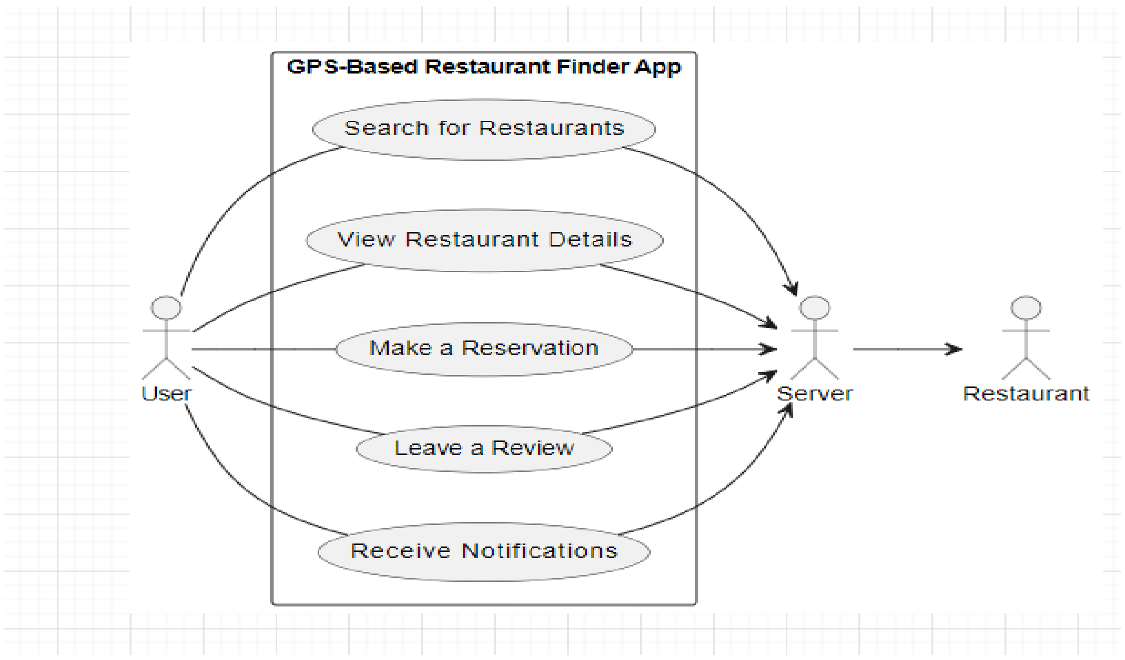
- Data Structures:
    - Notification Settings: Stores user preferences for enabling or disabling notifications.
- Information Flow:
    - Users can enable or disable notifications through their profile settings.
    - When events occur that trigger notifications (e.g., new restaurant added, reservation confirmation), notifications are sent to users who have enabled them.
- Security Considerations:
    - Notification delivery mechanisms should be secure to prevent unauthorized access to user data or interception of notifications.

5.3.2 Use Case 3:

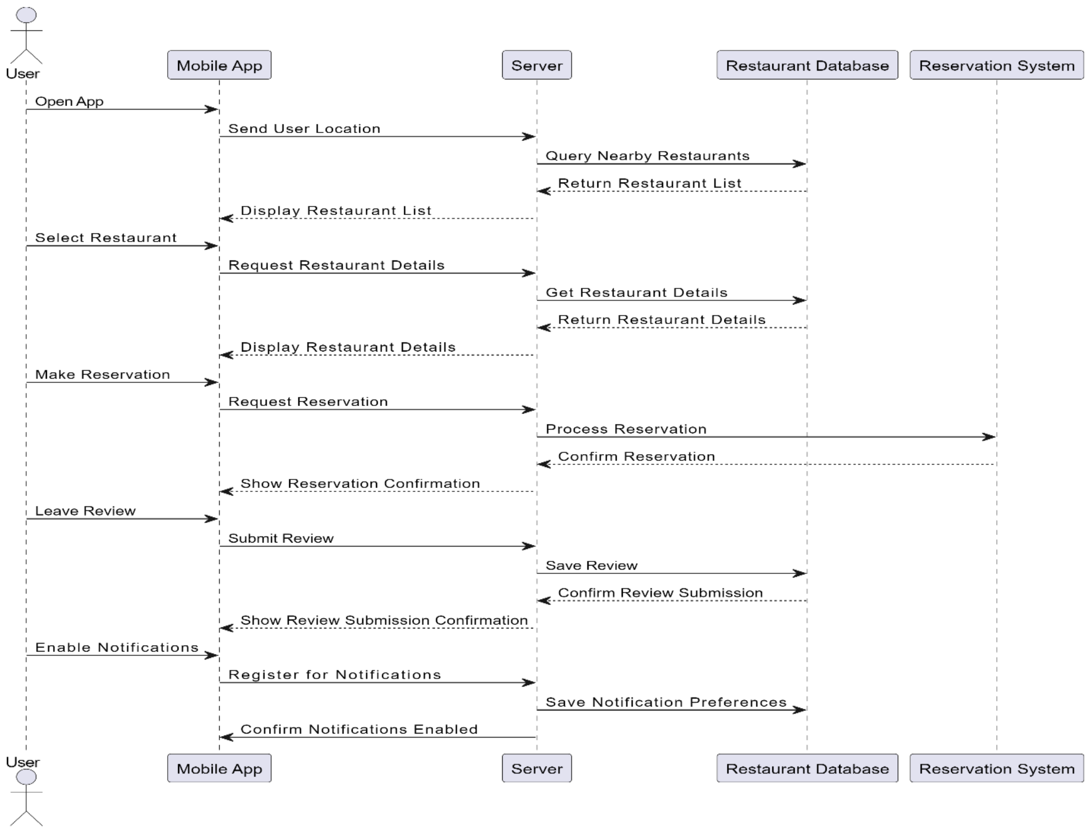


5.4 Representation:

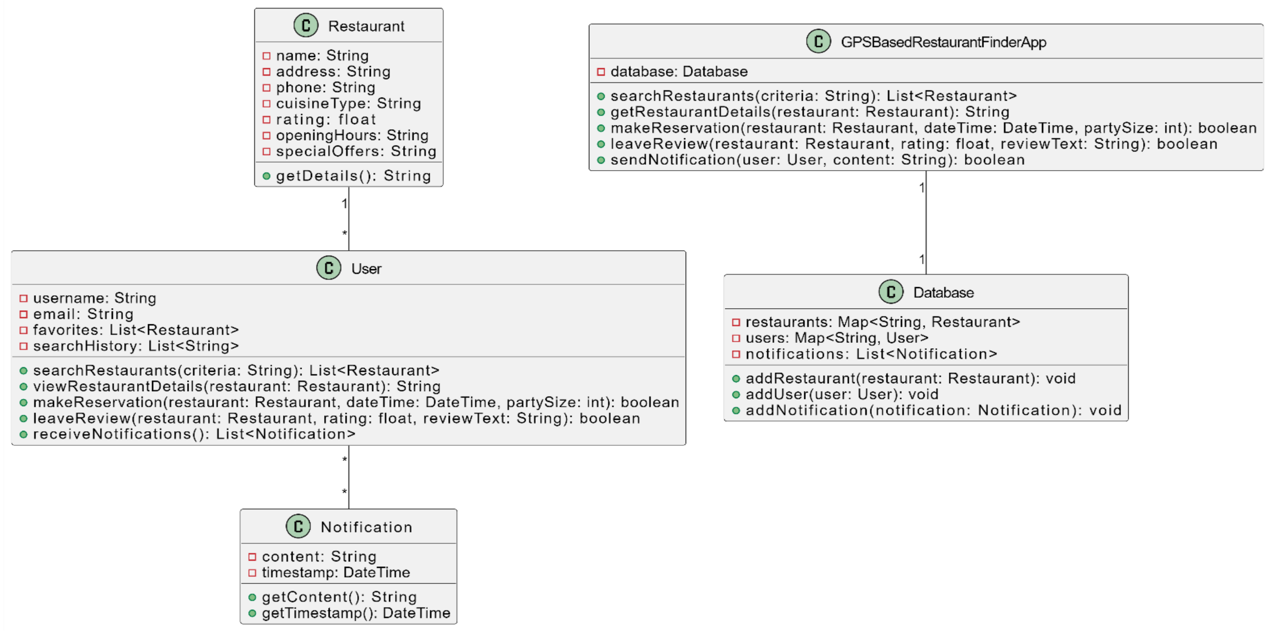
5.4.1 Using Use Case:



5.4.2 Using Sequence Diagram:



5.4.3 Using Class Diagram:


Restaurant
name: String
address: String
phone: String
cuisineType: String
rating: float
openingHours: String
specialOffers: String
getDetails(): String
GPSBasedRestaurantFinderApp
database: Database
searchRestaurants(criteria: String): List<Restaurant>
getRestaurantDetails(restaurant: Restaurant): String
makeReservation(restaurant: Restaurant, dateTime: DateTime, partySize: int): boolean
leaveReview(restaurant: Restaurant, rating: float, reviewText: String): boolean
sendNotification(user: User, content: String): boolean
1
*
1
1
User
username: String
email: String
favorites: List<Restaurant>
searchHistory: List<String>
searchRestaurants(criteria: String): List<Restaurant>
viewRestaurantDetails(restaurant: Restaurant): String
makeReservation(restaurant: Restaurant, dateTime: DateTime, partySize: int): boolean
leaveReview(restaurant: Restaurant, rating: float, reviewText: String): boolean
receiveNotifications(): List<Notification>
Database
restaurants: Map<String, Restaurant>
users: Map<String, User>
notifications: List<Notification>
addRestaurant(restaurant: Restaurant): void
addUser(user: User): void
addNotification(notification: Notification): void
*
*
Notification
content: String
timestamp: DateTime
getContent(): String
getTimestamp(): DateTime

6. Development

6.1 Which Model?

For the GPS-Based Restaurant Finder Application, considering the dynamic nature of the requirements, the need for frequent user feedback, and the emphasis on delivering value quickly, an **Agile software development methodology** would be highly suitable. Specifically, the Scrum framework within Agile could be effectively utilized.

6.2 Why using this Model?

1) Iterative and Incremental Development:

Agile methodologies, including Scrum, promote iterative and incremental development, allowing the project team to deliver small, functional increments of the software at regular intervals. This aligns well with the need for real-time updates and personalized recommendations in the GPS-Based Restaurant Finder Application.

2) Flexibility to Adapt to Changes:

Given the evolving nature of user preferences, market trends, and technological advancements, Agile methodologies offer flexibility to adapt to changes quickly. Through regular sprint cycles, the development team can incorporate feedback from stakeholders and adjust the product backlog accordingly.

3) Continuous Feedback Loop:
Agile methodologies emphasize frequent collaboration and communication between the development team and stakeholders. This facilitates a continuous feedback loop, ensuring that the delivered features meet user expectations and address the identified problem statement effectively.

4) Focus on Delivering Value:
Agile methodologies prioritize delivering value to the end-users early and frequently. By breaking down the project into smaller, manageable chunks, the development team can prioritize high-value features, such as real-time updates and personalized recommendations, ensuring that the most critical functionalities are developed first.

6.3 Advantages of the Model:

1) Flexibility:
Agile methodologies, such as Scrum, allow for flexibility in adapting to changing requirements and priorities. The iterative nature of Agile allows the project to respond quickly to feedback and new information.

2) Customer Involvement:
Agile methodologies emphasize regular collaboration with stakeholders, including customers and end-users. This involvement ensures that the delivered product meets their needs and expectations.

3) Early and Continuous Delivery:
Agile promotes the delivery of working software in small, incremental releases. This allows stakeholders to see tangible progress early in the project and provides opportunities for early feedback.

4) Improved Quality:
Agile methodologies incorporate practices such as continuous testing and frequent integration, which contribute to higher software quality. Issues can be identified and addressed early in the development process, reducing the likelihood of defects in the final product.

5) Adaptability:
Agile methodologies encourage teams to regularly inspect and adapt their processes to improve efficiency and effectiveness. This continuous improvement mindset fosters innovation and responsiveness to change.

6.4 Disadvantages of the Model:

1) Lack of Predictability:
Agile methodologies prioritize flexibility and responsiveness to change, which can sometimes result in uncertainty about project timelines and deliverables. Predicting the exact scope and schedule of the project can be challenging.

2) Resource Intensive:
Agile requires frequent collaboration, communication, and feedback, which can be resource-intensive, especially for large teams or distributed environments. It may require a significant investment of time and effort from team members and stakeholders.

3) Dependency on Customer Availability:
Agile methodologies rely heavily on customer involvement and feedback. If key stakeholders are not available or engaged throughout the project, it can hinder progress and decision-making.

4) Documentation Challenges:
Agile methodologies prioritize working software over comprehensive documentation. While this can lead to faster development cycles, it may result in a lack of detailed documentation, which can be challenging for new team members or future maintenance efforts.

5) Suitability for Large-Scale Projects:
Agile methodologies are well-suited for small to medium-sized projects with a moderate level of complexity. However, they may face challenges when applied to large-scale projects with multiple dependencies and extensive regulatory requirements.

7. Deployment

7.1 Deployment Strategy:

1) Continuous Deployment:

- Implement continuous deployment practices to ensure that updates and improvements are delivered to users rapidly and continuously.
- Automate the deployment pipeline to minimize manual intervention and streamline the release process.

2) Rolling Deployment:

- opt for a rolling deployment strategy where updates are gradually rolled out to subsets of users, allowing for minimal disruption and easy rollback in case of issues.
- By rolling out updates incrementally, you can monitor the impact on a smaller user base before deploying to the entire user population.

3) Feature Flags:

- Utilize feature flags to enable or disable specific features or functionalities dynamically.
- Feature flags allow you to control the release of new features, perform A/B testing, and mitigate risks by selectively enabling features for targeted user groups.

4) Canary Release:

- Consider implementing canary releases to deploy new versions of the application to a small percentage of users initially.
- Monitor the performance and stability of the new release in the canary environment before gradually expanding the release to a wider audience.

7.2 Environment Setup:

1) Cloud Infrastructure:

- Leverage cloud infrastructure providers such as AWS, Azure, or Google Cloud Platform for scalable and reliable deployment.
- Cloud platforms offer flexibility, scalability, and built-in services that can support the real-time nature of your application.

2) High Availability:
- Ensure high availability of the deployment environment to minimize downtime and ensure uninterrupted access to the application.
- Use load balancers, auto-scaling groups, and redundancy measures to distribute traffic and mitigate single points of failure.

3) Database Setup:
- Set up databases in the deployment environment to store restaurant information, user profiles, and application data.
- Configure database replication, backups, and failover mechanisms to ensure data integrity and availability.

7.3 Security Considerations:

1) Data Encryption:
- Implement data encryption mechanisms to protect sensitive information such as user credentials, payment details, and personal data.
- Use SSL/TLS protocols for secure communication between clients and servers to prevent eavesdropping and data tampering.

2) Access Controls:
- Configure access controls and permissions to restrict unauthorized access to the application and its resources.
- Implement role-based access control (RBAC) to enforce least privilege principles and ensure that users only have access to the features and data they need.

8. Testing

8.1 Functional Testing:

1) Search:
- Test different scenarios such as searching by restaurant name, cuisine type, location, and filtering options (e.g., price range, ratings).
- Verify that the search results are accurate and relevant to the user's query.

2) Reservation:
- Test the entire reservation process, including selecting dates, times, party sizes, and entering contact information.
- Verify that users receive confirmation messages upon successful reservation submission.

3) Map Integration:
- Verify that restaurant locations are accurately displayed on the map.
- Test the functionality of map interactions such as zooming, panning, and tapping on restaurant markers.

4) User Profile:
- Test the creation, editing, and deletion of user profiles.
- Verify that saved favorites, search history, and other profile data persist across sessions.

5) Real-Time Updates:
- Validate that user receive timely updates on restaurant information, including changes in opening hours, special offers, and closures.
- Test the system's ability to push notifications to users' devices in real-time.

8.2 Non-Functional Testing:

1) Performance:
- Conduct load testing to simulate various levels of user activity and measure the application's response time and throughput.
- Test the application's performance under peak load conditions to ensure it can handle high traffic volumes.

2) Usability:
- Conduct usability testing with representative users to gather feedback on the application's ease of use, navigation, and overall user experience.
- Identify areas for improvement based on user feedback and observations.

3) Security:
- Perform security assessments such as penetration testing and vulnerability scanning to identify potential security risks.
- Test authentication mechanisms, data encryption, and secure communication protocols to ensure the confidentiality and integrity of user data.

4) Reliability:
- Test the application's stability and robustness by subjecting it to various stress conditions such as high load, network interruptions, or device resource constraints.
- Monitor system logs and error reports to identify and address reliability issues such as crashes or unexpected behavior.

8.3 User-Acceptance Testing:
- Involve end users in testing the application to validate that it meets their requirements and expectations.
- Collaborate with user representatives to define test scenarios and acceptance criteria based on the problem statement and user needs.

9. Maintenance

9.1 Maintenance Strategy:
1) Bug Tracking and Resolution:
- Establish a system for tracking and prioritizing bug reports and issues identified by users or through monitoring tools.
- Implement a process for investigating, diagnosing, and resolving bugs promptly to minimize disruptions to users' experience.

2) Regular Updates and Enhancements:
- Schedule regular updates and enhancements to address user feedback, add new features, and improve existing functionalities.
- Prioritize updates based on user needs, market trends, and business goals to ensure continuous improvement of the application.

3) Performance Optimization:

- Monitor the performance of the application regularly and identify areas for optimization, such as improving response times, reducing load times, and optimizing resource utilization.
- Implement performance improvements iteratively to maintain or enhance the overall user experience.

9.2 Reporting Mechanisms:

1) Usage Analytics:

- Implement analytics tools to track user engagement, behavior patterns, and usage trends within the application.
- Generate reports on key metrics such as active users, session duration, popular search queries, and feature usage to gain insights into user preferences and behaviors.

2) Feedback Analysis:

- Collect and analyze user feedback through surveys, ratings, and reviews to understand user satisfaction, identify pain points, and uncover opportunities for improvement.
- Generate reports summarizing user feedback trends, common issues, and feature requests to inform product decisions and prioritize development efforts.

3) Bug and Issue Tracking:

- Maintain a centralized repository for tracking bugs, issues, and enhancement requests reported by users or identified through monitoring tools.
- Generate reports on bug resolution status, issue trends, and resolution times to track progress, measure team efficiency, and ensure accountability.

10. References:

1. "Mobile Application Requirements for a Restaurant Recommendation System" by A. M. A. Majeed, A. B. M. S. Abdullah, and S. Y. A. Bakar (2020)
2. "Requirements Elicitation for a Mobile Restaurant Navigation App" by S. Bandyopadhyay and A. K. Majumder (2019)
3. "A Study on Requirements Engineering for Mobile Restaurant Apps" by R. Kaur and N. Kaur (2017)
4. "Software Requirements Specification for a Restaurant Management System" by S. Sharma and P. Singh (2016)
5. "Requirements Engineering for Mobile Applications: A Use Case Driven Approach" by S. K. Bhavani and K. R. Ezhilmaran (2015)
6. "Mobile Restaurant Apps: A Review of User Requirements and Features" by L. Zhou, Y. Chen, and W. Dong (2014)
7. "Designing a Mobile Restaurant App Using User-Centered Design Principles" by H. C. Chen and Y. C. Hsu (2013)
8. "Usability Evaluation of Mobile Restaurant Apps" by J. Lin and Y. C. Hsu (2012)
9. "A Survey of Mobile Restaurant Apps: Features and Functions" by M. H. Khan, S. Z. Bhuiyan, and M. A. Hoque (2011)
10. "Mobile Restaurant Apps: A User Needs Analysis" by C. K. Tan and M. J. Khoo (2010)