Food Delivery App

General Issue

New app designed to enable users to order food from local restaurants for delivery or pickup. The app must:

- Allow users to create an account and save payment information for future orders (with integration capabilities with various payment gateways), providing access to order history for easy reordering.
- Feature a user-friendly interface.
- Track the user's location to:
    - Offer restaurant recommendations and display nearby options.
    - Estimate delivery times.
- Provide real-time order tracking.
- Integrate with restaurant inventory management systems.
- Include reviews and ratings for both restaurants and food items.
- Implement a notification system to keep users informed about order confirmations, delivery updates, and promotional offers.

To take a decision we consider:

1. Create a food delivery application that allows users to order food on mobile devices supporting iOS and Android systems.
2. Utilize a UI framework in React Native to create a user-friendly interface, ensuring an attractive user experience across all platforms.
3. Always keep in mind the performance and scalability of the app in terms of speed and maintenance.
4. Determine permissions for location, storage, notifications, and network access, always ensuring the privacy of user data and compliance with data security regulations.
5. Assess the integration of additional frameworks such as Google Maps to incorporate mapping and localization services, thus enhancing the overall performance and functionality of the application. While this may increase development complexity and reliance on third-party dependencies, it also provides advanced features and enhance the user experience.

Scenario

Decision Title: Location tracking approach for a food delivery application

Context: Users location access is a crucial factor for food delivery application to accurately provide restaurant suggestions and give estimated delivery durations.

Options Considered:

1. Utilizing phone GPS capabilities

2. Integrating with location-based services

Decision: We will utilize the GPS system on users' devices to track their location

Rationale:

1. GPS provides high accuracy in location tracking, ensuring precise recommendations and delivery estimates.
2. Utilizing the built-in GPS functionality minimizes the need for additional hardware or software, streamlining the development process and reducing costs.
3. Integrating GPS capabilities within the app allows users to control location sharing settings directly, enhancing privacy.

Consequences:

1. GPS is not available in all circumstances, such as on subways or in areas with poor satellite reception.
2. Utilizing GPS leads to increased battery and resources consumption on user devices, potentially impacting device performance.

Follow-Up Actions:

1. Develop and test the GPS-based location tracking feature to ensure accuracy and minimal battery consumption.
2. Implement user settings for controlling location sharing and privacy preferences.
3. Monitor user feedback and app performance to optimize battery usage and resource utilization.

Decision Title: Real-Time Order Tracking for the food delivery application

Context: The app needs to provide real-time order tracking capability to allow users to monitor the status and progress of their orders.

Options Considered:

1. Implementing a WebSocket-based architecture
2. Utilizing server-sent events (SSE)

Decision: We will implement a WebSocket-based architecture using Node.js and Socket.IO for real-time updates.

Rationale:

1. WebSocket enables bi-directional communication between the client and server, providing instant updates on order status.
2. WebSocket consumes low server resources, resulting in low server load and improved scalability.
3. Users receive live updates on their orders.

Consequences:

1. Implementing WebSocket requires additional server-side infrastructure and expertise, which may increase development complexity.
2. Some older browsers or network configurations may not fully support WebSocket.
3. Possibilities of connection such as connection drops or timeouts.

Follow-Up Actions:

1. Develop and deploy the WebSocket-based real-time order tracking system.
2. Conduct thorough testing to ensure reliability and scalability under various load conditions.
3. Monitor server performance and user feedback to identify and address any issues promptly.

Decision Title: Payment gateway options for the food delivery application

Context: The app needs to be integrated with a payment gateway to facilitate secure transactions.

Options Considered:

1. PayPal
2. Stripe
3. Square

Decision: We have decided to integrate Stripe as the payment gateway for the food delivery app.

Rationale:

1. Stripe has robust security measures, such as tokenization, and fraud detection tools. Ensuring the protection of sensitive customer data and secure transactions.
2. Stripe supports a wide range of payment methods, including credit/debit cards, digital wallets, facilitating diverse customer preferences.
3. Stripe offers competitive transaction fees and no hidden charges, allowing for cost-effective payment processing for businesses.

Consequences:

1. Integrating Stripe into the app requires development effort and resources, including API implementation, testing, and deployment.
2. The app will be depending on Stripe's services, any changes or disruptions to Stripe's platform may affect transaction processing and user experience.

Follow-Up Actions:

1. Implement Stripe's APIs and test payment functionality within the app.
2. Ensure compliance with Stripe's security requirements to safeguard customer data.
3. Monitor transaction performance and user feedback after integration.
4. Stay informed about updates and developments from Stripe to leverage new features or optimizations for the app's payment processing system.

Decision Title: Integration method between food delivery application with restaurant inventory management system

Context: To ensure the accuracy of restaurant menus, the app needs to be integrated with restaurants' inventory management system.

Options Considered:

1. API-based Integration
2. Web Scraping Techniques

Decision: We have decided to opt for API-based integration to synchronize data between the food delivery app and restaurants' inventory management systems.

Rationale:

1. API-based integration allows for direct communication between the app and the restaurant's inventory management system, ensuring real-time and accurate data.
2. APIs provide structured endpoints for accessing and updating information, reducing the complexity of data extraction and interpretation compared to web scraping techniques.
3. API-based integration typically involves authentication mechanisms and access controls, ensuring secure data exchange between the app and the restaurant's system.
4. APIs offer flexibility for customizing data exchange protocols and handling specific requirements or functionalities unique to the application and the restaurant's inventory management system.
5. This method facilitates scalability to accommodate future enhancements or modifications of the synchronization process.

Consequences:

1. API-based integration relies on the availability and accessibility of suitable APIs provided by the restaurants' inventory management systems.
2. Implementing API-based integration requires development effort and coordination between the app development team and the restaurants' technical teams to ensure compatibility and functionality.
3. Ongoing maintenance and monitoring are necessary to address any changes or updates to the APIs or system configurations, ensuring continued compatibility and data synchronization.

Follow-Up Actions:

1. Identify and evaluate the availability and suitability of APIs provided by restaurants' inventory management systems for integration with the application.
2. Develop and test API integration for data synchronization, ensuring compatibility and reliability.
3. Collaborate with restaurants to establish API access and authentication mechanisms, addressing any security or compatibility concerns.
4. Monitor data synchronization processes and address any issues promptly to maintain menu accuracy and reliability.

Decision Title: Review and Rating system for restaurants and food items on food delivery application

Context: Food delivery application needs to have review and rating system to assist users on making inform decisions when it comes to selecting restaurants or menu to order.

Options Considered:

1. Developing a Review and Rating System
2. Utilizing Third-Party Review and Rating APIs

Decision: We have decided to develop a custom review and rating system within the food delivery app to collect, display, and moderate user-generated content.

Rationale:

1. Developing a custom system allows for seamless integration with the app's existing architecture and user interface, providing full control over the design and functionality of the features.
2. Developing a custom system enables the implementation of personalized features and interactions tailored to the specific needs and preferences of users, enhancing engagement and satisfaction.
3. With a custom system, the app has full control over content moderation processes, allowing for the enforcement of community guidelines and the prevention of spam or inappropriate content.
4. Operating a custom review and rating system grants ownership and control over user-generated content and data.
5. A custom system can be designed with scalability in mind, accommodating future growth and evolving requirements without dependency on external services or APIs.

Consequences:

1. Developing a custom review and rating system creates significant development effort, time and resources.
2. Operating a custom system requires ongoing maintenance and support to address issues, feature enhancements, and adapt to changing user needs.

Follow-Up Actions:

1. Define requirements and specifications for the custom review and rating system, including user interface design, functionality, and moderation policies.
2. Implement backend logic and database structures to support user-generated content storage, retrieval, and moderation.
3. Develop user interfaces for submitting and displaying reviews and ratings.
4. Conduct thorough testing and validation of the review and rating system to ensure functionality, reliability, and adherence to moderation policies.
5. Monitor user feedback and engagement with the review and rating features, iteratively refining and enhancing the system based on user insights and platform analytics.

Decision Title: Order history access for the food delivery application

Context: The app needs to be able to provide users their order history record, allowing them to review and reorder the items they have ordered.

Options Considered:

1. Relational Database
2. NoSQL Database

Decision: We will utilize a relational database for storing and retrieving order history data.

Rationale:

1. A relational database provides a structured and organized way to store order history data, including information such as order details, timestamps, and user preferences, facilitating efficient data retrieval and analysis.
2. This method offers powerful querying capabilities, allowing for complex queries and filtering criteria.
3. Relational databases have scalability to accommodate increasing volumes of order history data and user interactions as the app's user base grows.
4. Relational databases integrate seamlessly with existing backend systems and development frameworks, leveraging established technologies and expertise within the development team.

Consequences:

1. Designing an optimized database schema for order history storage requires careful consideration of data relationships, indexing strategies, and normalization principles, potentially increasing development complexity.
2. Relational databases may encounter performance bottlenecks when handling large volumes of transactions or complex queries.
3. Operating a relational database demands ongoing maintenance tasks such as database backups, software updates, and performance monitoring to ensure optimal performance and data reliability.
4. Migrating order history data between different database versions or platforms may pose challenges and downtime risks, requiring careful planning and execution to minimize disruption to app functionality.

Follow-Up Actions:

1. Design and implement a relational database schema for storing order history data, considering factors such as data normalization, indexing strategies, and performance optimization.
2. Develop backend APIs and database interactions for storing and retrieving order history data, ensuring secure and efficient data access for the app's frontend components.
3. Conduct comprehensive testing and validation of the order history storage and retrieval mechanisms to ensure functionality, reliability, and performance.
4. Implement monitoring and alerting mechanisms to track database performance metrics and detect potential issues.

Decision Title: Notification system for the food delivery application

Context: Notification system is needed for food delivery application to keep users updated on their order status, delivery updates and promotional offers

Options Considered:

1. Push Notifications via Firebase Cloud Messaging (FCM)
2. In-App Notifications

Decision: Push notifications via Firebase Cloud Messaging (FCM) is the most suitable notification system in the food delivery app.

Rationale:

1. Push notifications give real-time delivery of updates to users' devices, ensuring timely notifications.
2. FCM supports both Android and iOS platforms, enabling consistent notification delivery across a wide range of devices and operating systems.
3. FCM allows for personalized and targeted notifications based on user preferences, order history, and location, enhancing the relevance and effectiveness of notifications.

Consequences:

1. Users need to grant permission for push notifications.
2. While push notifications provide high reliability and delivery rates, there may be occasional delays or delivery failures due to network issues or device-specific limitations.
3. Ensuring the relevance and timeliness of push notifications is crucial to avoid user annoyance. This requires careful targeting and segmentation of content based on user preferences and behavior.
4. Implementing push notifications via FCM requires integration with the app's backend server and client-side application code, as well as adherence to FCM's API requirements and best practices.

Follow-Up Actions:

1. Integrate Firebase Cloud Messaging (FCM) with the app's backend server to enable push notification delivery.
2. Implement client-side logic and user interface components for push notification display and interaction within the app.
3. Define notification triggers and message templates for various events such as order confirmations, delivery updates, and promotional offers.
4. Conduct testing and validation of push notification functionality to ensure reliable delivery and responsiveness across different devices and network conditions.