LAU
الجامعة اللبنانية الأميركية
Lebanese American University

DineSwift

Food Delivery System

Date: 2/4/2025

By Tarek Atassi

A Report submitted to Dr. Haidar Harmanani as the required project for capstone

This page intentionally left blank.

Preface

This document describes the general architecture and basic requirements for the development of the **DineSwift** food delivery system. It functions as an agreement between the contractor and the client regarding the precise features that will be included in the finished software. It also helps system designers figure out implementation specifics and possible limitations. This preliminary document, which lays out the parameters for the entire production process—including maintenance—is encouraged to be reviewed by all parties.

Table of Contents

Contents

1. Introduction

DineSwift is a food delivery platform designed to connect customers with local restaurants and streamline the ordering and delivery process. The app provides a convenient way for users to browse restaurant menus, place orders, and track deliveries in real time.

Restaurants can manage their listings, update menus, and process incoming orders, while delivery drivers receive optimized routes to ensure timely deliveries.

Admins will monitor and secure the whole system to make sure all is running smoothly and securely.

DineSwift includes several key features to enhance the user experience:

- **A user-friendly interface** for easy navigation and order placement
- **Restaurant and food images** to assist in decision-making
- **Detailed menu descriptions** with pricing and customization options
- **Secure processing** for a smooth checkout and experiance

The platform is designed to improve efficiency, reduce manual workload, and enhance customer satisfaction by automating several critical processes.

How DineSwift Automates Food Delivery

DineSwift automates key operations to optimize the food delivery workflow:

- **Restaurant Management:** Restaurants can update menus, manage availability, and receive orders instantly.
- **Order Processing:** The system automatically chooses the nearest available drivers to assigns orders to, reducing delays.
- **Delivery Optimization:** Drivers receive real-time updates and the most efficient routes for fast delivery. Users can see live tracking and status of all their orders.
- **Customer Relationship Management:** User order history is stored to track past purchases and improve service.
- **Admin Controls:** Admins can manage all user types, monitor all system activities, track orders, and even adjust delivery fees for drivers and commission rates for each restaurant.

Benefits of Automation in DineSwift

By automating these processes, DineSwift provides:

- **Increased Efficiency:** Reduces the need for manual intervention, speeding up operations.
- **Cost Savings:** Lowers operational costs for restaurants and delivery services.

- **Improved Accuracy:** Minimizes order errors and delivery mistakes.

- **Enhanced Customer Satisfaction:** Ensures a smooth and fast ordering experience. And allows customer to discover all local restaurants registered with **DineSwift.**

- **Improve scalability:** Will make managing a big number of restaurants delivery system way easier.

With its automated system and user-friendly design, **DineSwift** aims to revolutionize the food delivery experience by providing a reliable, efficient, and scalable solution for restaurants, drivers, and customers alike.

2. E-Commerce Glossary

A. Non-Technical Glossary

User App

- **Home Page**: Displays active restaurants near the user for quick access.

- **Restaurant Page**: Shows restaurant menu, including food images, descriptions, and prices.

- **Cart**: Holds selected food items and ensures checkout is restricted to one restaurant at a time.

- **Checkout Process**: Steps taken by users to finalize an order, including selecting items, confirming delivery details, and viewing total costs.

- **Order Tracking**: Feature that allows users to track order status in real-time, from preparation to delivery.

- **Profile**: Section where users manage personal details and saved addresses.
- **Support**: Provides users with a way to contact customer support for assistance.

Restaurant App

- **Dashboard**: Control center for restaurant owners to manage orders, menu items, and business details.
- **Restaurant Manager**: Tool allowing restaurants to add, modify, or remove menu items and update restaurant details.
- **Order Management**: Feature that enables restaurants to accept/reject orders, assign drivers, and update order statuses.
- **Item Archive**: Stores deleted menu items, allowing restaurants to restore them if needed.
- **Support**: Allows restaurant owners to reach out for help regarding app functionalities.

Driver App

- **Requests**: Section where drivers view available delivery requests and accept/reject them.
- **Order Manager**: Enables drivers to track accepted deliveries and update their status.
- **Availability Status**: Feature allowing drivers to set their availability and share their location with restaurants.
- **Account Management**: Section where drivers can modify their account details.
- **Support**: Provides users with a way to contact customer support for assistance.

Admin Web App

- **User, Driver & Restaurant Management**: Enables admins to approve/reject restaurants and drivers, block/unblock users, and monitor accounts.
- **Order Management**: Allows admins to oversee all orders, including active, completed, and canceled orders.
- **Commission & Delivery Fee Tracking**: Admins can adjust restaurant commission rates and delivery fees.
- **Support**: Admins can view and respond to support messages from users and restaurants.

Order Lifecycle

- **Order Placement**: Process where a user selects food items and submits an order.
- **Order Acceptance**: Restaurant confirms the order and begins preparation.
- **Driver Assignment**: Restaurant assigns an available driver for delivery.
- **Order Completion**: The driver delivers the order, and the restaurant confirms its completion.
- **Order Cancellation**: The order is canceled by either the restaurant or admin due to unavailability or other reasons.

Payment & Financials

- **Cash-on-Delivery (COD)**: Payment method where users pay in cash upon receiving the order.

- **Payment Tracking**: Admin feature that logs cash collected by drivers and restaurant earnings.
- **Restaurant Commission**: The percentage of revenue the restaurant pays to the platform for each order.

Delivery & Logistics

- **Live Tracking**: Users can track their order's location in real time via Google Maps.
- **Driver Availability**: Drivers set their status as online/offline to receive delivery requests.
- **Route Optimization**: System selects the best driver for the delivery based on distance and traffic conditions.

Search & Filtering

- **Restaurant Search**: Users can search for restaurants by name or category.
- **Food Item Search**: Users can search for specific food items across restaurants.

Analytics & Reporting

- **Admin Dashboard Analytics**: Displays total orders, revenue, and active users.
- **Order History**: Allows admins to generate reports on completed, pending, and canceled orders.

B. Technical Glossary

Authentication & Authorization

- **User Authentication**: Process of verifying users via email and password before allowing access to the app.
- **Authorization via Firestore Rules**: Ensures users, restaurants, and drivers can only access permitted data.
- **Access Control**: Restricts unauthorized access to sensitive information based on user roles.

Database & Backend Services

- **Firestore**: NoSQL cloud database used for storing app data, including users, orders, and restaurant details.
- **Firebase Authentication**: Secure authentication system handling user logins.
- **Firebase Functions**: Cloud-based serverless functions that process app logic, such as order updates and notifications.

Order & Delivery System

- **Real-Time Database Updates**: Ensures order status, driver availability, and restaurant activity are updated instantly.
- **Order Status Tracking**: System that logs changes in order status, such as "pending," "preparing," and "delivered."

- **Location Tracking**: Uses Firebase to store live driver locations and update restaurants and users in real time.
- **Google Maps API**: Integrates mapping and live tracking features for deliveries.

Security & Data Protection

- **Firestore Security Rules**: Defines access control for different user roles to prevent unauthorized actions.
- **Two-Factor Authentication (2FA)**: Additional security layer requiring users to verify their identity via OTP (optional future feature).
- **Data Encryption**: Ensures user and payment data is securely stored and transmitted.

Performance Optimization

- **Cloud Scaling**: Automatically increases Firebase resources to handle high traffic loads.
- **Load Balancing**: Distributes traffic efficiently to prevent slowdowns.
- **Caching**: Temporary storage of frequently accessed data to speed up app performance.

Payment & Financial System

- **Payment Processing**: System handling cash-on-delivery transactions and payment tracking.
- **Revenue Analytics**: Feature that tracks restaurant commission earnings and driver delivery fees over time.

Search & Filtering System

- **Firestore Queries**: Optimized queries for fast restaurant and item searches.
- **Full-Text Search**: Allows users to search for restaurants and food items efficiently.

Analytics & Reports

- **Order Insights**: Displays key performance metrics, including total orders, revenue, and user activity.
- **Financial Reports**: Generates monthly and weekly earnings reports for drivers and restaurants.

Image Uploading System

- **imgBB API Integration** – The app uses imgBB for hosting images, allowing restaurants to upload logos and food pictures. The system returns a URL for storing in Firestore.
- **Asynchronous Uploads** – Users can continue app interactions while images are uploaded in the background.
- **Storage Optimization** – External hosting minimizes Firestore storage costs and improves app performance.

3. System Requirements

A. Functional Requirements:

1. User Account Management System (All Apps)

1.1 User Registration:

- Users shall be able to create an account by providing their full name, email address, password, and optionally their phone number.
- The registration process will be designed to ensure a smooth and user-friendly experience.

1.2. Account Information Updates:

- Users shall be able to modify their details later on, including:
    - Name
    - Email address
    - Phone number (if applicable)
    - And other
- This feature will be available in the account settings of all apps.

1.3. Password Change:

- Registered users shall be able to change their passwords by either entering the old password or using email recovery to reset it.
- The password change process will be secure, ensuring a trusted login environment.

---

2. Secure Login (All Apps)

2.1. Authentication Process:

- All apps will implement a secure login system for users, restaurants, drivers, and admins using email and password.
- The authentication process will ensure strong security using Firebase authentication.

2.2. User Credentials:

- Users, restaurants, drivers, and admins shall provide their email and password for login.

2.3. Account Protection:

- Enhanced account protection measures will ensure a secure login environment for all apps, utilizing Firebase Auth for secure authentication.

---

3. Password Recovery and Account Security (All Apps)

3.1. Password Complexity Requirements:

- Passwords shall be at least 8 characters long and include:
    - One uppercase letter
    - One lowercase letter
    - One number
    - One special character
- Predictable passwords should be avoided.

3.2. Password Recovery:

- Users will be able to recover their passwords through email verification, ensuring their identity is confirmed.
- Users can also update their recovery email for added security.

---

4. Account Type Categorization (All Apps)

4.1. User Account Categories:

- Users shall be categorized into three types based on their roles:
    - **User Accounts (Customer App)**: Customers placing orders.
    - **Driver Accounts (Driver App)**: Drivers assigned to deliver orders.
    - **Restaurant Accounts (Restaurant App)**: Restaurant owners managing orders and menu items, and assigning orders to nearby drivers.
    - **Admin Accounts (Admin App)**: Admins managing the platform and overseeing all functionalities.

---

5. Product and Order Management (Restaurant App)

5.1. Restaurant Product Management:

- Restaurant accounts shall be able to:
    - Manage their product listings (add, edit, remove).

- Process customer orders, including order confirmation and driver management. System will automatically show nearby available drivers.
- Access order and inventory management tools. And manage availability of items and of the restaurant itself to be open for orders.

5.2. Order Management:

- Restaurants will manage customer orders, assign drivers, track delivery and manage statuses.

---

6. Driver Features (Driver App)

6.1. Order Pickup and Delivery:

- Drivers shall be able to:
  - View orders assigned to them.
  - Pick up and deliver orders based on customer addresses.
  - Update the delivery status in real-time.

6.2. Driver Tracking:

- The app shall provide real-time tracking for drivers, allowing restaurants and customers to monitor the delivery progress.

---

7. Admin Control and Management (Admin App)

7.1. Admin Privileges:

- Admin accounts shall have the ability to:
    - Approve or reject restaurant and driver accounts.
    - Block users.
    - Monitor system activity and orders.
    - Modify delivery fees and commissions for restaurants.

7.2. Reporting and Analytics:

- Admins will have access to detailed reports on orders, user activity, restaurant performance, and more.

---

8. Shopping and Order Features (User App)

8.1. Browsing and Ordering:

- Customers will be able to:
    - Browse restaurant listings.
    - Add items to a shopping cart.
    - Place orders and view order history.

8.2. Order Status and Notifications:

- Customers will receive updates on their order status, including estimated delivery time.

---

9. Payment and Checkout (User App)

9.1. Payment Methods:

- Customers will have the option to pay "cash upon delivery."

9.2. Order Confirmation:

- After checkout, customers will receive a confirmation with a unique order ID.

---

10. Support

10.1. Customer Support:

- All Users shall be able to contact support through a "Contact Us" form and receive responses within 24 hours from admins via call or email.

B. Non functional requirement :

1. User Interface and Accessibility:

- **User Interface Design:** The user interface for all four apps (Admin web-based, Driver, User, and Restaurant Android apps) shall be intuitive, easy to navigate, and user-friendly. The Admin app will be optimized for desktop and laptop browsers,

while the Android apps will be designed for smooth mobile navigation on various devices.

- **Cross-Platform Compatibility:** The Android apps shall work consistently across different versions of Android, ensuring compatibility with a wide range of devices. The Admin app shall be fully functional across modern browsers, including Chrome, Firefox, Safari, and Internet Explorer.

- **Responsive Design:** All apps shall adopt responsive design principles to ensure that the layout adapts efficiently to varying screen sizes and orientations, providing a seamless experience for users on mobile phones, tablets, and desktops.

2. Performance and Scalability:

- **Quick Page Load Times:** The Admin web app shall load quickly across various browsers, while the Android apps shall be optimized for fast launch and smooth transitions. All apps shall implement techniques like optimizing images, reducing unnecessary scripts, and caching for better performance.

- **Scalability:** Firebase will manage the backend, ensuring that the system can scale effectively to handle increased traffic and concurrent users. The system shall support at least 500 concurrent users without performance degradation, leveraging Firebase's infrastructure for scaling.

- **Graphics and Media Optimization:** All apps will optimize graphics and media files for faster load times, employing modern formats like WebP for images and compressing media files. The apps will also use lazy loading to load images or videos as needed to improve performance.

3. Reliability:

- **Low Downtime:** The system shall have minimal downtime, as Firebase manages the infrastructure and ensures high availability. The system should deliver continuous service even during high-traffic periods.
- **Continuous Service:** Firebase's cloud infrastructure ensures reliable service, capable of handling surges in traffic and maintaining uptime.
- **High Availability:** The system shall ensure high availability for both the Admin web app and the Android apps, with Firebase providing real-time database synchronization to ensure data consistency across all platforms.

4. Data Backup and Recovery:

- **Firebase Managed Backups:** Firebase handles data backup automatically, ensuring data integrity and availability. Data backups will be performed daily to prevent data loss in case of system failure.
- **Recovery Strategy:** Firebase’s cloud infrastructure includes disaster recovery features, providing quick recovery from failures. The system shall have defined processes in place to restore data swiftly.
- **Data Protection:** Firebase will ensure sensitive user data is encrypted both at rest and in transit. All sensitive information will be protected according to best practices for data privacy and security.

5. Security:

- **Firebase Security Rules:** Firebase Firestore security rules will be set up to restrict access to only authorized users. Users will only have access to the data they are permitted to view, ensuring that each user type (Admin, Driver, User, Restaurant) can only access relevant data.

- **Authentication and Authorization:** Firebase Authentication will manage user authentication and ensure that only authorized users can log in. The Admin app will have stricter access control, while Driver, User, and Restaurant apps will have specific roles and permissions.

- **Data Integrity:** Firebase ensures data integrity by using transactional capabilities to prevent data corruption. Firebase's security rules will also prevent users from tampering with data or making unauthorized changes.

- **Real-Time Database Security:** Firebase's real-time database ensures that the data is always consistent and up-to-date across all apps, with Firebase security rules preventing unauthorized access.

6. Compliance and Legal Adherence:

- **Legal Compliance:** The system shall fully comply with relevant e-commerce regulations, including payment processing and consumer protection laws. Firebase will ensure the platform adheres to privacy laws such as GDPR for users in applicable regions.

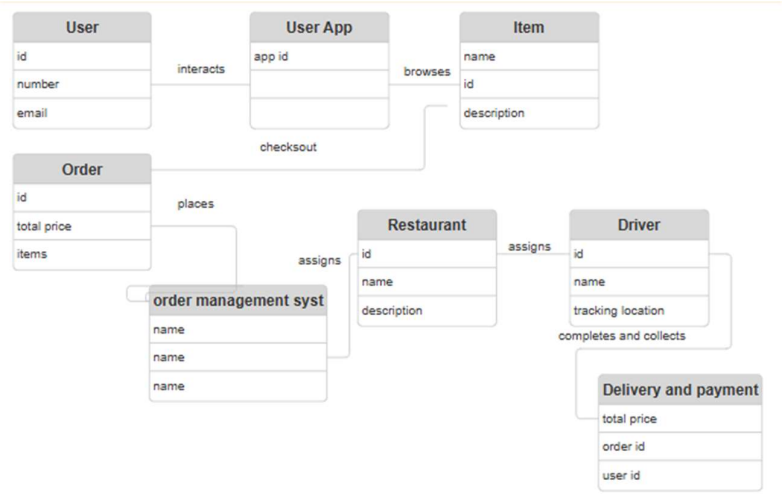
- **Industry-Specific Standards:** If applicable, the system will meet industry-specific standards for data protection and payment security (e.g., PCI DSS for handling payments).

- **Consumer Protection:** The platform will follow regulations to protect consumer rights, ensuring that transactions are secure and user data is kept private.
- **Security and Trust:** By adhering to Firebase's security protocols and encryption standards, the system will foster trust with users, mitigate legal risks, and ensure the secure operation of the platform.

4. System Model

ER diagram:

The following diagram shows a system model of the food delivery system:



The interaction between the various entities involved in the ordering process is depicted in the ER diagram. For instance, the order management system generates a new order record and forwards it to the restaurant for processing when a customer places an order. After that,

the order is assigned to a restaurant. The restaurant manages order and assigns a driver. The driver delivers the items and collects payment. The restaurant and driver complete the order.

An effective tool for comprehending the various entities and their interactions during the ordering process is the ER diagram. It can also be used to pinpoint possible process improvement areas. For instance, if the system for processing orders is sluggish the process bottlenecks can be found using the ER diagram.

This is an illustration of a possible practical application for the ER diagram:

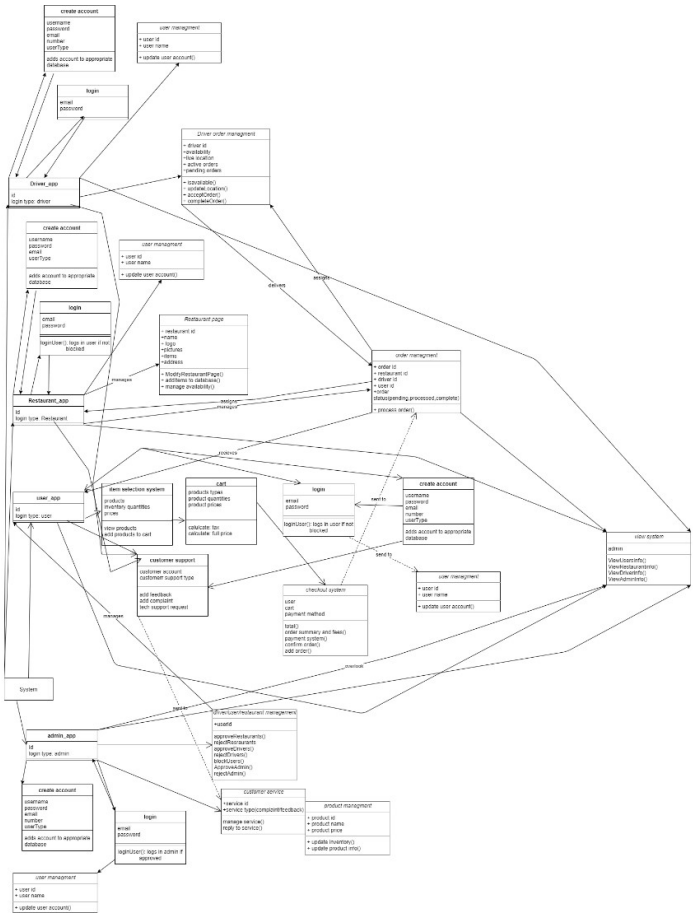
A client goes to the App and browses restaurant and items. After making a selection, they put the item in their cart. Once their shopping is done, they go to the checkout and provide/choose their shipping/address details.

After receiving the order from the user, the order management system makes a new order record. The order record is subsequently forwarded to the assigned restaurant for processing. The Restaurant prepares the order and manages the order. The restaurant assigns a driver. The Driver delivers and collects payment.

The restaurant can manage the items delete create or make items unavailable if out of stock. The user will have access to live tracking with the driver and live updates on their order from restaurant and driver. The admin will be able to oversee all the info to make sure all is running smoothly.

UML activity diagram:

Below is a simple UML activity diagram that may help summarise the processes



Subsystem

Database schema:

Collections & Fields

1. User Accounts (user_accounts)

- uid (string) → Unique user ID
- email (string)
- name (string)
- phoneNumber (string)

- createdAt (timestamp)
- isBlocked (boolean)

2. Admin Accounts (admin_accounts)

- uid (string)
- email (string)
- name (string)
- phoneNumber (string)
- status (string) → e.g., "accepted"
- createdAt (timestamp)

3. Driver Accounts (driver_accounts)

- uid (string)
- email (string)
- name (string)
- phoneNumber (string)
- status (string) → e.g., "accepted"
- availability (string) → e.g., "available"
- createdAt (timestamp)

- location (map)
    - latitude (number)
    - longitude (number)
- locationTimestamp (timestamp)

4. Restaurant Accounts (restaurants_accounts)

- uid (string)
- email (string)
- phoneNumber (string)
- name (string)
- status (string) → e.g., "approved"
- createdAt (timestamp)

5. Restaurants (restaurants)

- restaurantId (string)
- userId (string) → Owner's ID
- name (string)
- description (string)
- address (string)

- commission (number)
- createdAt (timestamp)
- location (map)
    - latitude (number)
    - longitude (number)
- logo (string) → Image URL
- restaurantPicture (string) → Image URL

6. Addresses (addresses)

- addressId (string)
- userId (string)
- addressName (string)
- building (string)
- street (string)
- city (string)
- location (map)
    - latitude (number)
    - longitude (number)

- createdAt (timestamp)
- deleted (string) → e.g., "yes" or "no"

7. Orders (orders)

- orderId (string)
- userId (string)
- restaurantId (string)
- status (string) → e.g., "completed"
- totalPrice (number)
- timestamp (timestamp)
- address (map)
    - addressId (string)
    - addressName (string)
    - building (string)
    - street (string)
    - city (string)
    - location (map)
        - latitude (number)

        - longitude (number)

- commissionAmount (number)
- deliveryFee (number)
- driverId (string)
- driverStatus (string)
- name (string) → User's name
- phoneNumber (string)
- items (array)
    - Each item contains:
        - itemId (string)
        - quantity (number)

8. Items (items)

- itemId (string)
- userId (string)
- name (string)
- description (string)
- price (number)

- discount (number)
- availability (string) → e.g., "yes"
- section (string) → e.g., "pasta"
- picture (string) → Image URL
- hide (string) → e.g., "no"
- isDeleted (string) → e.g., "no"
- createdAt (timestamp)
- updatedAt (timestamp)

9. Finance (finance)

- deliveryFee (number)

10. Support (support)

- messageId (string)
- userId (string)
- message (string)
- timestamp (timestamp)

System Components Aligned with Schema

1. Application Server Component

Function:

- Handles core backend functionalities, including user authentication, order processing, and restaurant/driver management.
- Manages API requests and ensures smooth interaction between the frontend and the Firestore database.

Uses Collections:

- user_accounts
- orders
- restaurants_accounts
- driver_accounts

Input:

- User login and authentication requests.
- Order placement, updates, and status changes.
- Profile updates from users, restaurants, and drivers.

Actions and Output:

- Validates user authentication.
- Places new orders and updates statuses.
- Manages restaurants and drivers, ensuring seamless coordination.
- Confirms successful operations or returns appropriate error messages.

---

2. Authentication & Authorization Component

Function:

- Handles user authentication and session management.
- Enforces role-based access control using Firebase Authentication and Firestore security rules.

Uses Collections:

- user_accounts
- admin_accounts
- driver_accounts
- restaurants_accounts

Input:

- User credentials (email/password).

- Authorization checks for various actions.

Actions and Output:

- Validates user credentials and grants access.
- Ensures restricted access based on roles (Admin, User, Driver, Restaurant).
- Blocks unauthorized actions and provides authentication feedback.

---

3. Order Management Component

Function:

- Manages the full order lifecycle from placement to delivery confirmation.
- Ensures real-time order tracking and notifications.

Uses Collections:

- orders
- user_accounts
- driver_accounts
- restaurants

Input:

- Order placement requests.

- Status updates from restaurants and drivers.

Actions and Output:

- Records order details, including user, restaurant, driver, and total price.
- Updates order status dynamically (e.g., Placed → Accepted → Out for Delivery → Completed).
- Sends real-time updates to users and drivers.

---

4. Delivery Management Component

Function:

- Assigns orders to drivers based on availability and location.
- Tracks driver movements and estimated delivery times.

Uses Collections:

- driver_accounts
- orders

Input:

- Driver location updates.
- Delivery assignment requests from restaurants.

Actions and Output:

- Assigns deliveries to available drivers based on proximity.
- Tracks driver movements via GPS and updates order statuses.
- Ensures real-time coordination between drivers, restaurants, and users.

---

5. Payment Processing Component

Function:

- Handles payment tracking for cash-on-delivery transactions.

Uses Collections:

- orders

Input:

- Payment confirmation from drivers.

Actions and Output:

- Updates payment status in the order database.
- Confirms transaction success and logs payment details.
- Ensures financial records are consistent with deliveries.

---

6. Search & Discovery Component

Function:

- Enables users to search for restaurants and menu items.
- Provides filters and ranking for search results.

Uses Collections:

- restaurants
- items

Input:

- User search queries.

Actions and Output:

- Fetches and returns relevant search results.
- Filters results based on keywords, restaurant availability, and item categories.
- Enhances user experience with fast and efficient search queries.

---

7. Admin Management Component

Function:

- Provides admin users with full control over system operations.

- Allows monitoring and moderation of restaurants, drivers, and user activities.

Uses Collections:

- admin_accounts
- user_accounts
- restaurants_accounts
- restaurants
- driver_accounts
- orders

Input:

- Admin actions (approving/rejecting accounts, modifying system settings, monitoring orders).

Actions and Output:

- Approves or rejects restaurant and driver applications.
- Blocks or unblocks users as necessary.
- Generates system-wide reports and analytics.

---

8. Web Application Component

Function:

- Provides an intuitive web interface for administrators.
- Allows real-time management of system entities.

Uses Collections:

- admin_accounts
- restaurants
- restaurants_accounts
- user_accounts
- driver_accounts
- orders
- finance

Input:

- Admin-initiated requests for account approvals, order tracking, and financial adjustments.

Actions and Output:

- Displays system data via a dashboard.
- Enables modification of restaurant commissions, delivery fees, and user permissions.

- Provides full visibility into platform operations.

8. Image Uploading System

- **imgBB API Integration**: The app uses imgBB as a third-party service for image hosting. When users upload images (such as restaurant logos or food pictures), the system sends them to imgBB, which returns a URL for storage in Firestore.

- **Asynchronous Uploads**: Ensures image uploads do not slow down the app, allowing users to continue other actions while images are processed in the background.

- **Storage Optimization**: Since images are hosted externally, this reduces Firestore storage costs and improves app performance.

9. Customer support System

- allow all user types to fill a contact form which will fill the support database
- the admin shall be able to read the messages and reply as soon as they can

5.System Architecture

A. Layered Architecture for Food Delivery App

1. Presentation Layer (Frontend - Mobile App)

This layer handles the user interface and user experience. Your app will have different UIs for users, restaurants, drivers, and admins:

- **User App**
    - Home Page: Displays available active restaurants near the user.

- Restaurant Page: Shows restaurant menu, food images, descriptions, prices.
- Search: Allows users to search for restaurants or specific food items.
- Cart Page: Displays selected food items with total price. Filters cart by restaurant to ensure one restaurant is checked out at a time.
- Checkout Page: Displays order and receipt and final price with all additional fees.
- Order Tracking Page: Displays order progress (order accepted, being prepared, out for delivery, delivered). And live tracking when order is out for delivery.
- Profile Page: Users can edit personal details, saved addresses(on the appbar).
- Support Page: Allows users to send messages for support on any issue they may face.

- **Restaurant App**
  - Dashboard: Display options for the restaurant. Restaurant manager, Order manager, profile, and item archive.
  - Restaurant Manager: Allows restaurants to add, update, or remove food items. Allow restaurant to modify restaurant info, pics, logo, and address. Allow restaurant to manage, delete, modify, hide, make unavailable, or delete items. Allow restaurant to be active or inactive to not take orders when not active.
  - Order Management: Restaurants can accept or reject orders and assign drivers(and see distance of driver), and restaurants can update order status and complete orders. Also restaurant can view driver status after assigning the

driver. Restaurant can view order history divided into active, completed, and canceled orders.

- Item archive: restaurant can retrieve deleted items onto the menu.
- Profile: Modify account info (email, password…) (modify account info not restaurant info).
- Support Page: Allows users to send messages for support on any issue they may face.

- **Driver App**
  - Dashboard: show options: availability status, requests, order manager, account management.
  - Requests: Shows available delivery requests for orders. Driver can accept order so the order is now in the drivers order manager, or driver can reject order.
  - Order manager: view accepted orders update the driver status on the order, and complete or cancel orders (canceled orders will have to be re assigned to new driver by the restaurant when the driver status is canceled).
  - availability status: allow driver to be available and share live location to database so restaurants can see the available locations and distances.
  - Account manager: Modify account info (email, password, number…)
  - Support Page: Allows users to send messages for support on any issue they may face.

- **Admin Web App**
    - Dashboard: Displays options: User management, Driver management, Restaurant management, order management, account management, admin management.
    - Driver & Restaurant Management & User & admin Management: Allows admins to approve/reject new restaurants and drivers and block/unblock users. Also allows admin to view orders for this specific user/driver/restaurant. Allow admin to view generated commission in a filtered time period(week month 3 month 6 month year) of every restaurant, and same for the driver but delivery fee generated for every driver. Admin can adjust the commission % of every restaurant in the restaurant manager. Admin can adjust general the delivery fee for drivers in the driver manager. Admin can approve/reject admin accounts.
    - Order management: monitor all active orders and statuses of the orders and all their info. View order history completed or canceled orders and their details.
    - Account manager: Modify account info (email, password, number…).
    - Support: view support messages sent by users or restaurants or drivers.

2. Application Layer (Backend - Firebase & APIs)

This layer handles the business logic and data processing, ensuring smooth authentication, order placement, delivery management, and user interactions.

A. Authentication & Authorization

Manages logins and user access control using Firestore Authentication.

1. **User Authentication & Login**
    - Email & password authentication for users, restaurants, drivers, and admins.
    - Each user type can only access their respective app.
    - Blocked users or non-approved accounts (restaurant/driver/admin) are restricted.
    - Uses Firebase **ruleset code** for access permissions. Example:

```
allow read: if request.auth != null && (
  // User can always read their own document if it exists and is not blocked
  request.auth.uid == resource.data.uid &&
  resource.data.isBlocked == false ||
```

2. **Authorization via Firestore Rules**
    - Prevents unauthorized access to collections and documents.
    - Users can only read/write their **own** orders, addresses, and profile and other.
    - Restaurants can only access their own **menu, orders, and settings and other**.

- Drivers can only access **delivery requests assigned to them and only collections they need**.
- Admins have full access to **manage users, restaurants, drivers, and orders and everything else**.
- Prevents creation of incomplete or wrong document fields via Firstore ruleset code. For ex:

```
allow read: if request.auth != null && (
  // User can always read their own document if it exists and is not blocked
  request.auth.uid == resource.data.uid ||


  // Admins can read any user account

exists(/databases/$(database)/documents/admin_accounts/$(request.auth.uid))
  &&
get(/databases/$(database)/documents/admin_accounts/$(request.auth.uid)).data.status == 'accepted'
);
  // Allow create for newly authenticated users to create their document
   allow create: if request.auth != null &&
```

request.resource.data.keys().hasAll(['uid', 'name', 'email', 'createdAt', 'status']) &&

request.resource.data.uid == request.auth.uid &&

request.resource.data.status == "pending" && // Ensure new accounts are not blocked by default

request.resource.data.name is string &&

request.resource.data.email is string &&

request.resource.data.createdAt is timestamp;

B. Order Management System

Handles order lifecycle: **Placement → Acceptance → Delivery → Completion**

1. **Order Placement**
    - Users select food items, add them to the cart, and place an order.
    - Orders are stored in Firestore with status: pending.
    - Restaurant receives a **real-time orders**
2. **Restaurant Order Management**
    - Restaurant can accept or reject the order.
    - If accepted, the restaurant updates status to preparing.

- Once ready, restaurant assigns a driver.

3. **Driver Assignment & Tracking**
    - Nearby available drivers receive **order requests**.
    - Driver **accepts or rejects** the delivery request.
    - If accepted, status updates to out for delivery.
    - If rejected, the restaurant assigns a different driver.
4. **Order Delivery & Confirmation**
    - Driver marks order as delivered once dropped off.
    - Restaurant **confirms completion**.
    - Order status updates to completed.
5. **Order Cancellation**
    - Restaurants and admins can cancel orders if items are unavailable or any other reason.
    - Drivers can reject delivery requests.

C. Delivery Management System

Ensures **efficient driver assignment** and **live tracking**.

1. **Driver Availability**

- Drivers set their availability (online / offline).
- Live location tracking updates **Firebase** every few seconds.
- Nearby restaurants see available drivers.

2. **Live Order Tracking**
   - Uses **Google Maps API** to show driver location.
   - Users see real-time tracking of their orders.
   - Driver location and **ETA and route are calculated dynamically**.
3. **Route Optimization**
   - Firebase queries the best **driver for the delivery** based on distance.
   - Waypoints are set for optimized **pickup & drop-off routes**.

D. Payment Processing

1. **Cash-on-Delivery (COD)**
   - Users pay **directly to the driver** at delivery.
   - Restaurants mark orders as paid once confirmed.
2. **Payment Tracking for Admins**
   - Admins track **cash collected by drivers**.
   - Reports generated for **restaurant commissions**.

F. Search & Filtering

1. **Restaurant & Item Search**
    - Users search for **restaurants** by name.
    - Users search for **specific food items**.

G. Analytics & Reporting

1. **Admin Dashboard Analytics**
    - View **total orders, revenue, active users, and deliveries**.
    - Track **restaurant commission earnings**.
    - Monitor **driver earnings from delivery fees**.
2. **Order History & Insights**
    - View **completed, pending, and canceled orders**.
    - Generate **monthly or weekly financial reports for drivers and restaurants**.

3. Data Layer (Firestore Database & Storage)

Stores **Stores structured data needed for the app:**

- **User Accounts (user_accounts)**: Stores user profiles
    - Contains: uid, email, name, phoneNumber, createdAt, isBlocked
- **Admin Accounts (admin_accounts)**: Stores admin profiles

- Contains: uid, email, name, phoneNumber, status, createdAt

- **Driver Accounts (driver_accounts)**: Stores driver details, including their status and availability
  - Contains: uid, email, name, phoneNumber, status, availability, createdAt, location (map with latitude and longitude), locationTimestamp

- **Restaurant Accounts (restaurants_accounts)**: Stores restaurant profiles
  - Contains: uid, email, name, status, createdAt, phoneNumber

- **Restaurants (restaurants)**: Stores restaurant details such as the name, address, and location
  - Contains: restaurantId, userId (owner's ID), name, description, address, commission, createdAt, location (map with latitude and longitude), logo, restaurantPicture

- **Addresses (addresses)**: Stores user addresses
  - Contains: addressId, userId, addressName, building, street, city, location (map with latitude and longitude), createdAt, deleted

- **Orders (orders)**: Tracks orders and assigns drivers
  - Contains: orderId, userId, restaurantId, status, totalPrice, timestamp, address (map with addressId, addressName, building, street, city, location), commissionAmount, deliveryFee, driverId, driverStatus, name, phoneNumber, items (array with itemId, quantity)

- **Items (items)**: Stores information about products available for order
    - Contains: itemId, userId, name, description, price, discount, availability, section, picture, hide, isDeleted, createdAt, updatedAt
- **Finance (finance)**: Stores financial data, such as delivery fees
    - Contains: deliveryFee
- **Support (support)**: Stores customer support messages
    - Contains: messageId, userId, message, timestamp

C. System Evolution & Scalability

To future-proof your delivery app, consider:

- **Mobile App Development**: Improve UI/UX with animations, dark mode, and more payment options.
- **AI-Based Order Prediction**: Suggest dishes based on user preferences & order history.
- **Multi-City Expansion**: Adapt database structure to support multiple locations.
- **Enhanced Security**: Two-factor authentication, IP restriction for admin access.
- **Cloud Scaling**: Auto-scale Firebase resources based on demand
- **Add a review system**: add review and feedback system for restaurants and drivers
- **Improve customer support and communication**: improve customer support system with live chats. Add live chats with drivers to improve communication.

INDEX