PROJECT REPORT

Role-Based Hierarchy Management System

Prepared By:

Python Development Team

- **Project Lead**: Arshad Shaikh
- **Team Members**:
    1. Rohit Tiwari
    2. Niraj Nikam
    3. Sateesh Sonawane
    4. Ram Waghmare

Prepared For:
Open Future Technology **Date: 27/12/2024**

Project Overview

The **Hierarchy Management System** is built to manage a multi-tier role structure, including **Admin**, **Manager**, **Super Distributor**, **Distributor**, and **Kitchen**, with clearly defined functionalities and permissions for each role. The **Admin** role has the highest level of access, enabling them to manage all roles, update permissions, and oversee operations across the system. **Managers** oversee daily operations within their respective domains, ensuring smooth functioning. **Super Distributors** handle distribution responsibilities, supporting the Manager's tasks. **Distributors** manage the delivery process and track order status, while **Kitchens** focus on food item management, order processing, and sales tracking.

Each role has access restrictions to sensitive data. For example, **Kitchens** can manage food items, track daily sales, and calculate royalties once sales targets are met. **Admin** roles can modify royalty percentages, ensuring full control over financial distributions. The system allows seamless transitions between roles, such as when an Admin is reassigned, ensuring data integrity and continuity. Roles are structured to maintain a balance between operational efficiency and security, ensuring that users have the appropriate permissions based on their role level.

Scope of the Report:

This document outlines the project's technical specifications, architecture, and functionalities, highlighting the development process, milestones, and final deliverables.

Objectives and Scope

Project Objectives

The primary objectives of this project are:

- To streamline hierarchical role management in organizations.
- To ensure secure data access with role-specific permissions.
- To provide real-time tracking and analytics for orders, sales, and royalties.
- To improve operational efficiency and reduce manual intervention.

Scope of the Project

The scope includes the following:

1. **Development and Deployment**: Building the system using Flask, MySQL, and modern frontend technologies.
2. **Multi-Role Support**: Admin, Manager, Super Distributor, Distributor, and Kitchen roles.
3. **Data Management**: Efficient tracking of orders, royalties, and notifications in a secure database.
4. **Accessibility**: Accessible on desktops and mobile browsers for ease of use.
5. **Customization**: Scalability and customization options for future enhancements.

System Architecture

High-Level Architecture

The system follows a **3-tier architecture**, ensuring separation of concerns:

1. **Presentation Layer** (Frontend):
   - Accessible through a web browser with role-based dashboards and an intuitive UI.
2. **Application Layer** (Backend):
   - Handles business logic, role-specific functionality, and API endpoints developed in Flask.
3. **Data Layer** (Database):
   - MySQL database stores all critical information, including user data, roles, orders, and sales.

Modules Overview

1. **Authentication and Authorization**:
   - Secure login, registration, and role-based access control.
2. **Role Management**:
   - Admin, Manager, Super Distributor, Distributor, and Kitchen with unique functionalities.
3. **Order Management**:
   - Efficient tracking and status updates for orders.
4. **Sales and Royalty Management**:
   - Automates sales tracking and royalty calculations.

Technical Stack

Technologies Used

1. **Code Editor**:
    - **Visual Studio Code (VS Code)** for writing and debugging the project.
2. **Programming Language**:
    - **Python** for backend logic and API development.
3. **Framework**:
    - **Flask** for rapid web application development.
4. **Database**:
    - **MySQL** for storing and managing project data.
5. **Frontend Technologies**:
    - **HTML, CSS, JavaScript** for creating user interfaces.
6. **Backend Tools**:
    - **SQLAlchemy**: Object-Relational Mapping (ORM) to interact with MySQL.
    - **Flask-SocketIO**: For real-time notifications and communication.
7. **Version Control**:
    - **Git** for collaboration and source code management.

1. Code Editor: Visual Studio Code (VS Code)

- **VS Code** is the primary code editor used for writing, editing, and debugging the project code. It offers features like IntelliSense, integrated Git support, debugging tools, and a large extension marketplace, making it ideal for both frontend and backend development.
- It is equipped with extensions that support Python, Flask, and MySQL development, improving the developer experience with syntax highlighting, code completion, and error detection.

2. Programming Language: Python

- **Python** is the core programming language used to develop the backend logic of the system. Its simplicity, readability, and vast ecosystem of libraries make it ideal for building scalable, maintainable web applications. Python is particularly effective in implementing business logic, role management, data handling, and integrating third-party services.

3. Framework: Flask

- **Flask** is a lightweight web framework that facilitates rapid development of web applications. It is used to manage the routes, HTTP requests, user authentication, and session management in the **Hierarchy Management System**. Flask's flexibility allows easy integration with various libraries and tools, making it suitable for implementing the multi-role access control system.

- It also supports RESTful API development, making it ideal for building the backend logic and integrating with the frontend.

4. Database: MySQL

- **MySQL** serves as the relational database for storing data related to user roles, permissions, food items, orders, and sales. It is an efficient, robust, and widely-used database management system that ensures data integrity, consistency, and scalability.
- The **Hierarchy Management System** uses MySQL to store tables for each role (Admin, Manager, Super Distributor, Distributor, Kitchen) along with their associated data, such as sales records, orders, and profile information. MySQL provides support for complex queries, indexing, and transactions to manage this data effectively.

5. Frontend Technologies: HTML, CSS, JavaScript

- **HTML** is used to structure the web pages, ensuring the content is displayed correctly across different devices.
- **CSS** provides styling to the web pages, ensuring a clean, user-friendly interface. It ensures the visual appeal and responsive design of the application across different screen sizes.
- **JavaScript** is used to add interactivity to the frontend, including handling dynamic content updates, form validation, and asynchronous requests. JavaScript enables real-time communication with the backend and provides a smooth user experience.

6. Backend Tools:

- **SQLAlchemy for ORM (Object-Relational Mapping)**:
  - **SQLAlchemy** is an ORM tool used to map Python objects to database tables in MySQL. It simplifies database operations such as querying, inserting, updating, and deleting records. SQLAlchemy abstracts the complexities of SQL syntax, making database interactions more intuitive.
  - With SQLAlchemy, the system defines models for each role (Admin, Manager, Super Distributor, Distributor, Kitchen) and their relationships, ensuring efficient data retrieval and updates.
  -
- **Flask-SocketIO for Real-Time Communication**:
  - **Flask-SocketIO** is used to implement real-time communication features within the **Hierarchy Management System**. It enables real-time updates and notifications for users, such as when orders are placed or when sales thresholds are reached for royalty calculations.
  - This tool leverages WebSockets to create a bi-directional communication channel between the server and the client. It ensures that any updates or changes in the backend are immediately reflected on the frontend without needing to refresh the page, enhancing user experience in a live environment.

These technologies together form a robust foundation for the **Hierarchy Management System**, ensuring scalability, real-time interactions, and a seamless user experience across various roles.

Libraries/Dependencies:

- **bcrypt**: Used for securely hashing passwords and comparing them during authentication, ensuring sensitive data is protected.

- **flask**: A web framework for Python that provides tools and libraries to build web applications, handling routing, templates, sessions, and more.

- **werkzeug.security**: Offers security utilities like password hashing and checking, often used in Flask applications for authentication.

- **flask_socketio, emit**: Enables real-time communication in Flask apps using WebSockets, allowing two-way messaging between client and server.

- **sqlalchemy.exc**: Contains exceptions raised by SQLAlchemy, such as errors during database operations, used for handling database-related errors.

- **base64**: Used for encoding and decoding binary data, often for handling file uploads or transmitting data over HTTP.

- **functools**: Provides higher-order functions like decorators for modifying or enhancing other functions, improving code efficiency.

- **datetime, timedelta, timezone**: Handles date and time operations, allowing you to manage time-related functionality and timezone-aware timestamps.

- **sqlalchemy.exc.IntegrityError**: Raised when a database operation violates integrity constraints, such as foreign key or unique constraint violations.

- **sqlalchemy.func**: Provides access to SQL functions like COUNT, SUM, AVG, etc., allowing you to perform database queries using these functions.

- **logging**: Used to log messages for debugging, error tracking, and monitoring application behavior.

- **base64.b64encode**: Specifically encodes binary data into base64 format, typically used for handling images or binary data in HTTP requests.

- **werkzeug.exceptions.NotFound**: Represents a "404 Not Found" error in Flask, used to handle situations where a requested resource does not exist.

- **os**: Provides functions for interacting with the operating system, like managing files, environment variables, and paths.

- **flask_bcrypt**: A Flask extension that integrates the bcrypt hashing algorithm, simplifying password hashing and comparison in Flask apps.

- **flask_migrate**: Provides database migration support for Flask applications using Alembic, making it easier to manage database schema changes.
- flask_migrate**Chart Libraries**: JavaScript-based chart libraries (e.g., Chart.js)

1. Sign-up and Login Functionality

Sign-up:

- **Role Selection**: The inclusion of role selection during the sign-up process ensures that users are assigned their respective roles (Admin, Manager, Super Distributor, Distributor, Kitchen) at the start. This is critical for managing permissions and access control.

- **User Input Fields**:
  - **Role Selection**: This ensures the correct role is assigned during registration.
  - **Name**: Essential for identifying users.
  - **Email**: Ensuring email is unique is key for user identification and login.
  - **Password (Securely hashed using bcrypt)**: Security is ensured by hashing passwords, which is a good practice for protecting user credentials.
  - **Phone Number**: Optional but could be important for user identification or communication.

- **Redirection**: After a successful registration, redirecting the user to the login page ensures a smooth flow for the next step in the process.

Login:

- **Login Flow**: Users can log in with their email and password, with bcrypt used for password verification. This enhances security and ensures that user credentials are handled safely.

- **Role-Based Redirection**: After login, the user is redirected to their respective dashboard based on their role. This provides a personalized experience:
  - **Admin → Admin Home**
  - **Manager → Manager Home**
  - **Super Distributor → Super Distributor Home**
  - **Distributor → Distributor Home**
  - **Kitchen → Kitchen Home**

  This is an effective way to handle different access levels based on roles.

- **Invalid Login Attempts**: Displaying appropriate error messages for invalid login attempts improves user experience by providing feedback when something goes wrong (e.g., "Incorrect email or password").

2. Session Management:

- **Secure Session Storage**: Using secure session storage ensures that the user's session remains intact throughout their interaction with the system. This is essential for managing user authentication across different pages. The use of **Flask-Session** or **Flask-Login** can be beneficial for handling sessions securely.

3. Logout:

- **Logout Functionality**: Allowing users to log out from their accounts ensures security and proper session management. Once logged out, the user session should be cleared, and the user should be redirected to the login page.

Key Features

1. **Role-Based Access Control:**
   - Admins have complete access to manage all roles.
   - Managers, Super Distributors, and Distributors have tiered access to data.
   - Kitchens can manage sales and report performance.

2. **Dynamic Role Management:**
   - Admins can add or remove roles.
   - On Admin removal, old data is reassigned to another Admin.

3. **Royalty Distribution:**
   - A royalty system distributes 20% of total sales equally among Admin, Manager, Super Distributor, and Distributor when a Kitchen achieves a total sale of 1000 rupees.
   - Admins can modify royalty amounts and percentages.

4. **Sales and Performance Tracking:**
   - Comprehensive tracking of sales for royalty calculations and operational analysis.

Roles and Features

1. Admin

The **Admin** role serves as the central authority in the hierarchical management system, with comprehensive access and responsibilities to oversee and manage all aspects of the system. Below is an expanded breakdown of the Admin's capabilities, permissions, and features:

Key Capabilities and Responsibilities

1. **Full Access:**
    - Admins have unrestricted access to view and control all roles and data within the system.
    - This includes hierarchical oversight and operational transparency across Managers, Super Distributors, Distributors, and Kitchens.

2. **Role Management:**
    - Admins can create, modify, or remove roles as needed.
    - Responsibilities include:
        - Defining privileges for each role.
        - Ensuring roles align with organizational requirements.
        - Adding or updating permissions for roles under their hierarchy.

3. **Data Reassignment:**
    - In case of Admin removal, all associated data (e.g., roles, reports, and performance metrics) is reassigned to another Admin.
    - This ensures operational continuity and prevents data loss.

4. **Royalty Rules Management:**
    - Admins exclusively control royalty distribution policies, including:
        - Adjusting royalty percentages or amounts allocated to various roles (e.g., Manager, Super Distributor, Distributor).
        - Setting new royalty eligibility thresholds (e.g., sales milestones for Kitchens).

Permissions

Admins possess comprehensive permissions for data management, role control, and system maintenance:

1. **CRUD Operations (Create, Read, Update, Delete):**
    - Perform CRUD operations on the following entities:
        - **Manager**
        - **Super Distributor**
        - **Distributor**
        - **Kitchen**

2. **Entity Management:**
    - **Lock/Unlock Entities:**
        - Temporarily disable or enable access for specific entities (e.g., lock a Kitchen for non-compliance).

    - **Dashboard Access:**
        - View statistical visualizations, including:
            - Bar charts, line charts, and pie charts representing sales, performance, and operational data.
        - Access an overview of key metrics, such as total entities and total sales, directly on the Home page.

3. **Cuisine Management:**
    - Add, delete, and manage details for cuisines, including:
        - **Cuisine Name**
        - **Description**

4. **Sales and Operational Data:**
    - View the total sales amount and detailed sales reports, presented on the Home page in an easy-to-understand format.

Features

1. **Profile Management:**
    - Admins can manage their own profiles by:
        - Viewing and updating profile details such as name, email, contact information, and profile picture.

2. **Dashboard Insights:**
    - Real-time access to:
        - The total count of all entities (e.g., Managers, Kitchens).
        - A summary of operational data to assist in decision-making.

3. **Sales Overview:**
    - View aggregated sales data, including:
        - Total sales achieved.
        - Breakdown of sales performance by entity (e.g., Kitchens, Distributors).

The Admin role serves as the backbone of the hierarchical management system, combining operational oversight with granular control. With features like role management, royalty rules customization, and comprehensive data access, Admins ensure the system operates efficiently while meeting organizational goals.

2. Manager

The **Manager** role is integral to overseeing Kitchens and maintaining efficient operations within their jurisdiction. Managers act as intermediaries between the Admin and lower-level entities, focusing on performance monitoring and sales management to ensure organizational goals are achieved. Below is an expanded breakdown of the Manager's responsibilities, permissions, and features:

Key Functionalities

1. **Sales Management:**
    - Managers have access to sales data of the Kitchens under their supervision.
    - They can:
        - Analyze performance trends to identify Kitchens meeting or exceeding targets.
        - Address underperformance by providing support or corrective measures.
        - Ensure accurate and up-to-date sales reporting to maintain transparency.

2. **Performance Monitoring:**
    - Regularly track performance metrics for assigned Kitchens.
    - Utilize data insights to:
        - Identify areas needing improvement.
        - Recognize and reward Kitchens achieving exceptional results.

Permissions

Managers are empowered with permissions to manage lower-level roles and their operations:

1. **CRUD Operations (Create, Read, Update, Delete):**
    - Perform CRUD operations for the following entities within their jurisdiction:
        - **Super Distributor**
        - **Distributor**
        - **Kitchen**
2. **Dashboard Access:**
    - View comprehensive statistics presented through:
        - Bar charts, line charts, and pie charts for operational insights.
    - Access data on the total count of entities under their supervision, such as the number of Distributors or Kitchens.
3. **Cuisine Management:**
    - Add, delete, and manage cuisines, including:
        - **Cuisine Name**
        - **Description**

Features

1. **Real-Time Data Insights:**
    - Access sales data and operational reports on the Home page, including:
        - Total sales amount for assigned entities.
        - A detailed sales table showcasing individual contributions.

2. **Profile Management:**
   - Managers can manage their own profiles by:
     - Viewing and updating personal details such as name, contact information, and credentials.

3. **Sales Overview and Entity Metrics:**
   - Monitor:
     - Total sales achieved by entities under their management.
     - Summary of operational performance metrics for subordinate roles (e.g., Kitchens, Distributors).

4. **Statistical Visualization:**
   - Visual representations of performance and operational data to enable better decision-making.

The Manager role plays a critical role in the hierarchical structure, bridging the gap between Admins and lower-level entities. With a strong focus on sales management, performance tracking, and operational oversight, Managers ensure that Kitchens operate efficiently, achieve targets, and align with the broader organizational objectives. Their ability to analyze data and address underperformance is essential for maintaining the system's overall effectiveness.

3. Super Distributor

The **Super Distributor** role holds a pivotal position in the hierarchy, serving as a direct link between the Manager and lower-level entities. Super Distributors are responsible for overseeing the activities of Distributors and Kitchens within their purview, ensuring compliance with organizational standards and efficient operations.

Key Functionalities

1. **Access to Lower Entities:**
   - Super Distributors have complete visibility into all subordinate entities, including:
     - **Distributors**
     - **Kitchens**
   - This enables them to monitor and guide operations effectively.

2. **Operational Oversight:**
   - Oversee and manage the performance and activities of lower entities.
   - Ensure that subordinate roles adhere to performance targets and operational guidelines.

Permissions

1. **CRUD Operations (Create, Read, Update, Delete):**
   - Super Distributors can perform CRUD operations for the following entities:
     - **Distributor:** Manage details and performance of Distributors under their jurisdiction.
     - **Kitchen:** Oversee and update operational data for Kitchens reporting to them.

2. **Data Visualization and Reporting:**
    - Access to dashboards with:
        - Bar charts, line charts, and pie charts for data representation.
        - Summarized statistics for subordinate entities.

3. **Profile Management:**
    - View and update their personal details, ensuring up-to-date records.

Features

1. **Real-Time Dashboards:**
    - Comprehensive dashboards showcasing key metrics, including:
        - Total number of subordinate entities (Distributors and Kitchens).
        - Total sales amount within their domain.

2. **Sales Tracking:**
    - Access a detailed sales table that provides insights into individual and collective performance of lower entities.

3. **Operational Insights:**
    - Monitor the count and activity of Distributors and Kitchens under their supervision.
    - Identify trends, outliers, and areas requiring intervention.

4. **Profile Management:**
    - Update profile information such as name, contact details, and credentials.

5. **Entity Management:**
    - Add, update, delete, or fetch details of subordinate entities, ensuring accurate and efficient data management.

The Super Distributor role is essential for maintaining operational harmony between Managers and lower-level entities. By combining robust permissions with advanced data visualization tools, Super Distributors can ensure that Distributors and Kitchens align with organizational goals, contributing to overall efficiency and success.

4. Distributor

The **Distributor** role serves as the primary connection between Super Distributors and Kitchens, ensuring smooth coordination and operations at the ground level. Distributors are responsible for managing Kitchens and their associated activities, facilitating operational efficiency, and supporting sales performance.

Key Functionalities

1. **Access to Lower Entities:**
   - Distributors have full visibility into the operations of all subordinate Kitchens within their assigned domain.
   - This includes:
     - Monitoring sales performance.
     - Tracking orders and operational metrics.

2. **Operational Oversight:**
   - Direct management and supervision of Kitchens under their purview.
   - Support Kitchens in achieving their sales and performance targets.

Permissions

1. **CRUD Operations (Create, Read, Update, Delete):**
   - Distributors can manage the following entities:
     - **Kitchen:**
       - Add new Kitchens and ensure their onboarding.
       - Update details such as name, location, and operational status.
       - Delete or deactivate Kitchens as required.
       - Fetch data for performance monitoring or reporting purposes.

2. **Data Visualization and Reporting:**
   - Access dashboards with:
     - Bar charts, line charts, and pie charts for sales and operational data.
     - Insights into the performance of subordinate Kitchens.

3. **Profile Management:**
   - View and update their personal profile to ensure accurate and up-to-date information.

Features

1. **Real-Time Dashboards:**
   - Dashboards provide critical insights, such as:
     - The total count of subordinate Kitchens.
     - Comprehensive sales data, including daily, weekly, and monthly performance.

2. **Sales Tracking and Reporting:**
   - Detailed sales table displaying:
     - Individual sales figures for each Kitchen.

        - Aggregated sales totals for all Kitchens under the Distributor's management.

3. **Order Management with Filters:**
    - Ability to view and manage Kitchen orders using advanced filtering options:
        - **By Kitchen:** Focus on a specific Kitchen's orders.
        - **By Time Period:** Analyze orders within a defined date range.
        - **By Status:** View orders based on their current status (e.g., pending, completed, or canceled).

4. **Profile Management:**
    - Update personal details such as name, contact information, and credentials.

The Distributor role plays a vital part in ensuring that Kitchens operate efficiently and contribute to the organization's overall goals. By leveraging permissions and advanced data tools, Distributors can provide valuable support and guidance to Kitchens, improving sales performance and operational consistency.

5. Kitchen

The **Kitchen** role plays a crucial part in the overall food delivery and sales process. This role is not only responsible for tracking sales and managing food items, but also for earning royalties based on the kitchen's performance. Here's a detailed breakdown of the responsibilities, functionalities, and permissions available to a Kitchen user in the system.

1. Sales Tracking

- **Daily Sales Reporting**: Kitchens track their daily sales within the system. This includes:
    - The total value of orders processed on a daily basis.
    - Accurate reporting ensures that royalty calculations are correct and timely.
    - Integration with the system enables real-time tracking and visibility of sales performance.

- **Royalty Eligibility**:
    - The system automatically monitors sales thresholds (e.g., total sales of 1000 rupees). When the threshold is met:
        - The Kitchen becomes eligible for a royalty distribution, which is calculated and distributed to other roles (Admin, Manager, Super Distributor, Distributor).
        - The royalty is based on a fixed percentage (e.g., 20%) of the total sales.

2. Royalty Calculation

- **Automatic Calculation**:
    - When the Kitchen meets the required sales target (e.g., 1000 rupees in total sales), the system triggers royalty distribution.
    - The royalty amount is calculated automatically, ensuring consistency and transparency.

- **Distribution**:
  - Once calculated, the royalty is distributed equally among Admin, Manager, Super Distributor, and Distributor roles.
  - Only the Admin has the ability to modify the royalty percentage, ensuring control over the royalty rules.

3. Permissions

a. Food Item Management:

- **Add New Food Items**: Kitchens can add new items to the menu, which includes:
  - **Item Name**: Name of the dish or food item.
  - **Price**: Cost of the item.
  - **Image**: Visual representation of the item.
  - **Description**: Detailed description of the item, including ingredients, preparation methods, or special features.

- **Update Existing Food Items**: Kitchens can also modify existing menu items by updating any of the above details.

- **View Food Items**: Kitchens have access to view the list of all available food items, including details such as prices and descriptions.

b. Order Management:

- **View Orders**: Kitchens can view all orders, and apply filters to sort by:
  - **Order Status** (e.g., Pending, Processing, Completed).
  - **Time Period** (e.g., Today, This Week, This Month).

- **Update Order Status**: Kitchens can modify the status of orders as they are processed. The available statuses are:
  - **Processing**: Order is being prepared.
  - **Pending**: Order has been placed but not yet started.
  - **Completed**: Order is ready for delivery or has been delivered.

4. Features

a. Dashboard and Analytics:

- **Sales Dashboards**: Kitchens have access to a visual representation of their sales data. This includes:
  - **Bar Charts**: Showing sales performance over different time periods (e.g., daily, weekly, monthly).
  - **Line Charts**: Trends in sales over time, helping to identify high-performing periods.
  - **Pie Charts**: Breakdown of sales by different categories (e.g., item type, order status).

b. Profile Management:

- **View Profile**: Kitchens can view their profile details, including:
  - Name, contact information, location, etc.
  - Sales data tied to their kitchen.

- **Update Profile**: Kitchens can modify their profile details as needed, ensuring the information stays up-to-date.

5. Role Access and Restrictions

- **Restricted Access to Other Roles**: Kitchens can only manage and access their own items, orders, and sales data. They do not have access to modify or view roles above them, like Admin or Manager roles.

- **Limited Control Over System Settings**: Kitchens can't alter system-wide settings or permissions but can request changes via Admin if needed.

- **Collaboration with Other Roles**: Kitchens work with the Admin, Manager, and other roles to ensure smooth operations. However, the Kitchen role does not have direct access to sensitive data about other roles.

6. Notifications and Alerts

- **Order Alerts**: Kitchens are notified when an order is placed, and when its status needs to be updated.

- **Sales Notifications**: If the Kitchen's sales approach the royalty threshold, they will receive alerts to track progress toward earning royalties.

- **Profile Update Alerts**: Kitchens receive notifications when their profile details are successfully updated or when an Admin modifies any royalty-related settings.

By structuring the Kitchen role in this way, it becomes an efficient and automated part of the food delivery system, ensuring the kitchen's sales are tracked, royalty calculations are accurate, and operations run smoothly with proper data visualization.

Models
Steps to Add Models

1. **Define Models in models.py:**

Models in the Project:

- **Admin**: Handles information related to Admin users, their roles, permissions, and management capabilities within the system.
- **Customer**: Stores data related to customers, including personal details and order history.
- **Distributor**: Represents the **Distributor** role, including distribution management and order assignments.
- **Kitchen**: Manages kitchen-related data, including food items, sales, order processing, and royalty calculations.
- **Manager**: Represents managers who oversee the operations of kitchens, distributors, and other roles within their domain.
- **SuperDistributor**: Manages the distribution of products at a higher level than regular distributors, overseeing multiple distributors and managing their operations.
- **Cuisine**: Contains details about various cuisines available in the kitchen, associated with food items.
- **Sales**: Tracks the sales data, including total sales amounts, sales per kitchen, and other metrics.
- **Order**: Represents the orders placed by customers, including details like order items, status, and timestamps.
- **FoodItem**: Stores information about food items available for sale, including name, price, description, and images.
- **OrderItem**: Represents individual items within an order, linking each item to its parent order.
- **ActivityLog**: Tracks activities within the system for auditing and monitoring purposes.
- **Notification**: Manages the notifications system, sending updates to users based on their role (e.g., order updates, royalty notifications).
- **RoyaltySettings**: Stores settings related to royalty calculations and the thresholds for distributing royalty.
- **RoyaltyWallet**: Represents the royalty balance for each role (Admin, Manager, Distributor, Kitchen, etc.) and tracks the distribution of royalties

2. **Define Relationships:**
   - Establish relationships between roles using foreign keys.

3. **Integrate Flask-Migrate:**
   - Use Flask-Migrate to manage database migrations.

Blueprints and Route Management in the Project

In Flask, **Blueprints** provide a way to organize the application into modular components, making it easier to manage different routes, views, and logic. The **create_app_routes** function in the project is responsible for registering various blueprints, which represent different parts of the application (e.g., Admin, Customer, Kitchen, etc.). Each blueprint corresponds to a specific section of the system and is prefixed with a URL path that helps organize the routes under different roles and functionalities.

1. **Create Route Files:**
    - Create separate route files for each role and functionality (e.g., admin.py, manager.py, kitchen.py).

2. **Define Role-Based Access:**
    - Use decorators or middleware to ensure role-based access.

3. **Register Blueprints:**
    - Use Flask blueprints to modularize routes.

The blueprints used in the project are as follows:

1. **admin_bp**:
    - **URL Prefix**: /admin
    - This blueprint manages routes related to **Admin** operations, such as adding or removing users, managing roles, and accessing administrative controls.

2. **customer_bp**:
    - **URL Prefix**: /customer
    - The **Customer** blueprint handles routes for customer actions, including profile management, viewing orders, and order tracking.

3. **distributor_bp**:
    - **URL Prefix**: /distributor
    - This blueprint includes routes for **Distributors**, managing their orders, shipments, and deliveries.

4. **kitchen_bp**:
    - **URL Prefix**: /kitchen
    - The **Kitchen** blueprint is responsible for managing kitchen operations, such as viewing food items, processing orders, and calculating royalty.

5. **manager_bp**:
    - **URL Prefix**: /manager
    - This blueprint is used for **Manager**-specific routes, including overseeing the kitchen and distributor activities.

6. **super_distributor_bp**:
   - **URL Prefix**: /super_distributor
   - The **Super Distributor** blueprint handles routes for users in the Super Distributor role, overseeing multiple distributors and handling higher-level distribution tasks.

7. **cuisine_bp**:
   - **URL Prefix**: /cuisine
   - This blueprint is used to manage the available **Cuisines** in the system, including adding and updating cuisines for kitchens.

8. **order_bp**:
   - **URL Prefix**: /order
   - The **Order** blueprint is responsible for managing customer orders, including placing, viewing, and updating order statuses.

9. **food_item_bp**:
   - **URL Prefix**: /fooditem
   - This blueprint handles routes related to food item management, including adding, updating, and viewing food items available in the system.

10. **sales_bp**:
    - **URL Prefix**: /admin/sales
    - **Sales** routes are managed here, allowing the Admin to track total sales, generate reports, and handle financial data.

11. **orders_bp**:
    - **URL Prefix**: /admin/orders
    - This blueprint handles **Admin**-specific order management routes, including viewing, updating, and managing customer orders at the administrative level.

12. **dashboard_bp**:
    - **URL Prefix**: /dashboard
    - The **Dashboard** blueprint handles route functionalities for viewing overall system statistics, user activity, and role-specific metrics, offering insights into operations for each user role.

13. **user_bp**:
    - **URL Prefix**: /user
    - This blueprint contains routes for user-related actions, such as managing user profiles, changing passwords, and updating contact details.

14. **wallet_bp**:
    - **URL Prefix**: /wallet
    - This blueprint manages routes related to **wallets**, including royalty distribution, balance tracking, and withdrawals.

Purpose and Usage:

- **Modularization**: Each blueprint corresponds to a specific module or functionality in the application, making it easier to maintain and extend the system. For example, all routes related to **orders** are handled by the **order_bp** blueprint, while **sales** are handled by **sales_bp**.

- **Role-Based Access**: The URL prefixes like /admin, /customer, and /kitchen ensure that routes are logically grouped based on the user's role, providing clear boundaries between functionalities. This is especially useful for access control and organizing permissions across different user levels.

- **Flexibility**: Registering blueprints allows for a more scalable and flexible application structure. If new features need to be added in the future, new blueprints can be easily created and registered without disrupting the existing structure.

How It Works:

- The **create_app_routes** function registers each blueprint with a specific URL prefix. This means that when a user accesses a URL like /admin, the routes defined in the **admin_bp** blueprint are served, ensuring that only the correct functionality is available for that role.

- This centralized routing system makes it easy to manage and extend the application while keeping different parts of the application isolated and organized.

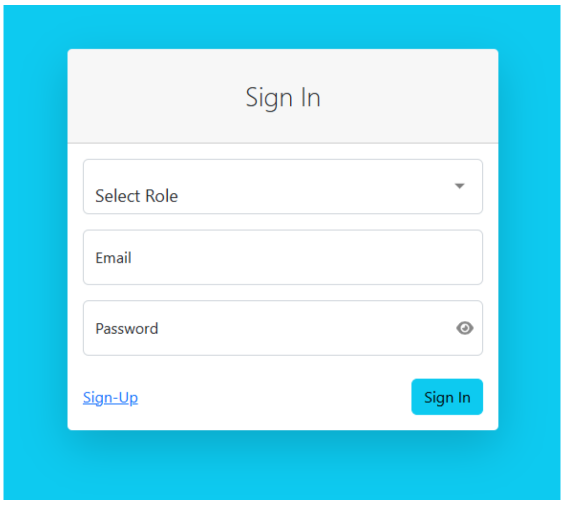
Project Folder Structure

```
FOOD_DELIVERY_HIERARCHY/
├── __pycache__
├── environment                      # Environment file
├── migrations                       # Migration file
├── models /                         # Define database models for all entities
│   ├── __init__.py
│   ├── activitylogs.py
│   ├── admin.py
│   ├── cuisine.py
│   ├── customer.py
│   ├── distributor.py
│   ├──food_item.py
│   ├── kitchen.py
│   ├── manager.py
│   ├── notificaton.py
│   ├── order.py
│   ├── royalty.py
│   ├── sales.py
│   └── super_distributor.py
├── routes/                          # Define routes files for all entities
│   ├── __init__.py
│   ├── activity_log_service.py
│   ├── admin.py
│   ├── cuisine.py
│   ├── customer.py
│   ├── dashboard.py
│   ├── distributor.py
│   ├── extensions.py
│   ├── food_item.py
│   ├── kitchen.py
│   ├── manager.py
│   ├── notification_route.py
│   ├── order.py
│   ├── royalty.py
│   ├── super_distributor.py
│   ├── user_routes.py
│   └──  wallet.py
├── static/                          # Fronted files
│   ├── assets/
│   ├── css/
│   ├── js/
│   ├── Images/
|   ├──app.js
|     └── style.css
├── templates/                      # Fronted files
│     ├── admin /
│     ├── customer /
│     ├── distributor /
│     ├── kitchen /
│     ├── manager /
│     ├── notifications /
│     ├── order/
│     ├── super_distributor /
│     ├── add_cuisine.html
│     ├── add_food_item.html
│     ├── edit_food_item.html
│     ├── food_item.html
│     ├── sd_footer.html
│     └── view_details.html
├── utils/                          # Utility files
│     ├── helpers.py
│     ├── notification_services.py
│     ├── services.py
│     ├── socketio.py
│     ├── wallet.py
├── app.py
├── extensions.py                   # Extensions of flask bcrypt and socketio
├── .gitignore
├── config.py                       # Configuration settings.
├── requirements.txt                # Python dependencies.
└── README.md  # Project description and setup instructions.
```

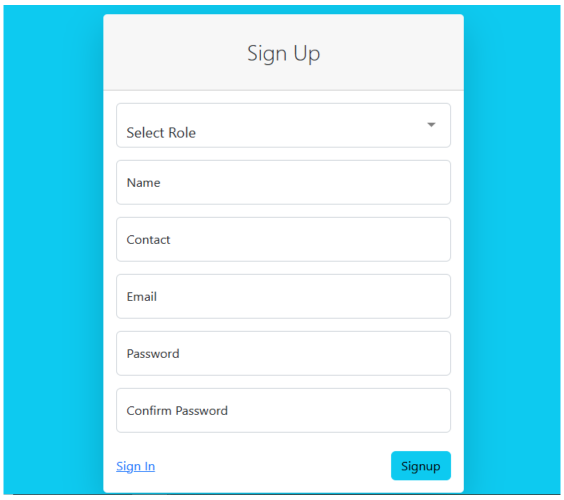
Application Screenshots

1. User Interface Screens

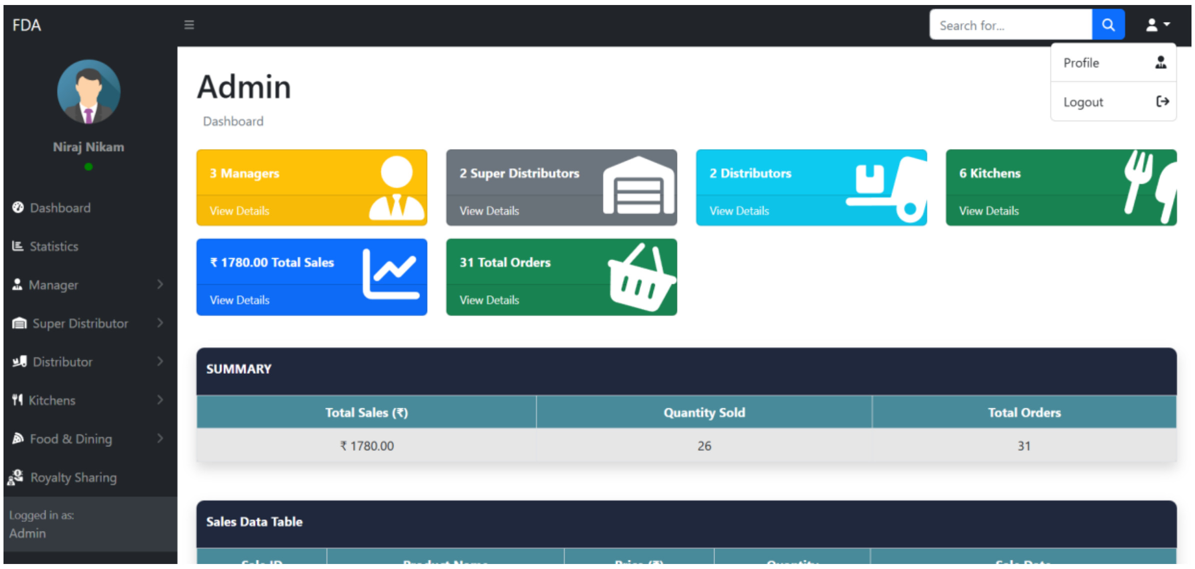
- **Sign In Page**:
  Screenshot of the sign in page, showing the design and functionality.
  Example:



- **Sign-Up Page**:
  Screenshot of the registration/sign-up page.



- **Admin Dashboard**:
  Overview of the Admin dashboard


FDA

Niraj Nikam

Admin
Dashboard

3 Managers — View Details
2 Super Distributors — View Details
2 Distributors — View Details
6 Kitchens — View Details
₹ 1780.00 Total Sales — View Details
31 Total Orders — View Details

SUMMARY

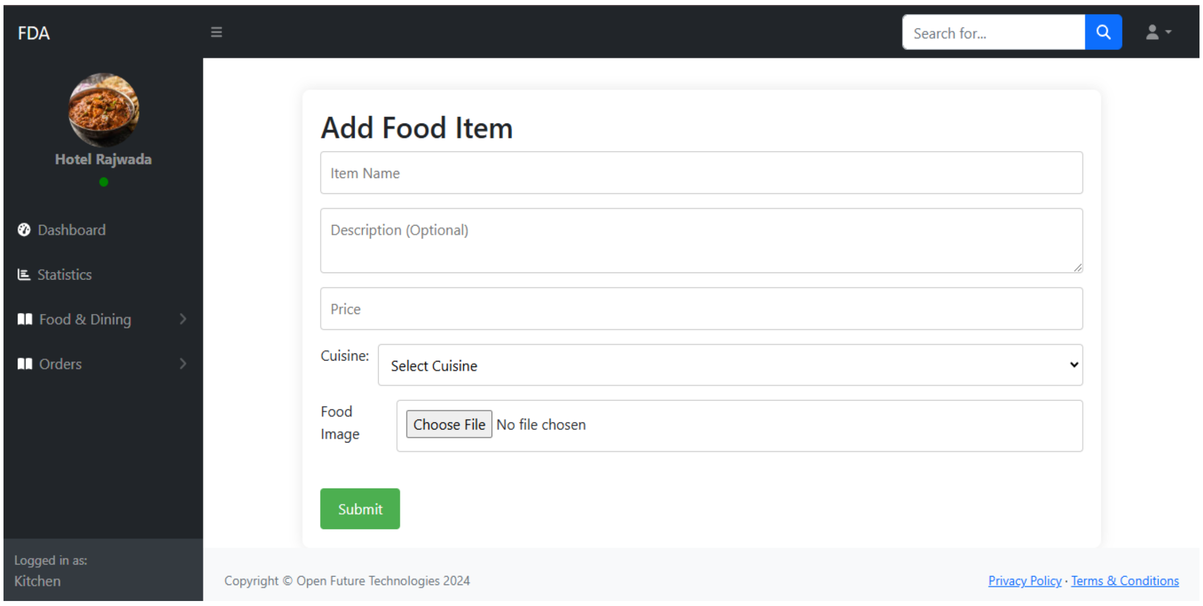
Total Sales (₹)	Quantity Sold	Total Orders
₹ 1780.00	26	31

Sales Data Table

Logged in as:
Admin

2. Functional Screens

- **Add Food Item**
  Screenshot of the form/interface to add a food item.


FDA

Hotel Rajwada

Add Food Item

Item Name

Description (Optional)

Price

Cuisine: Select Cuisine

Food Image: Choose File No file chosen

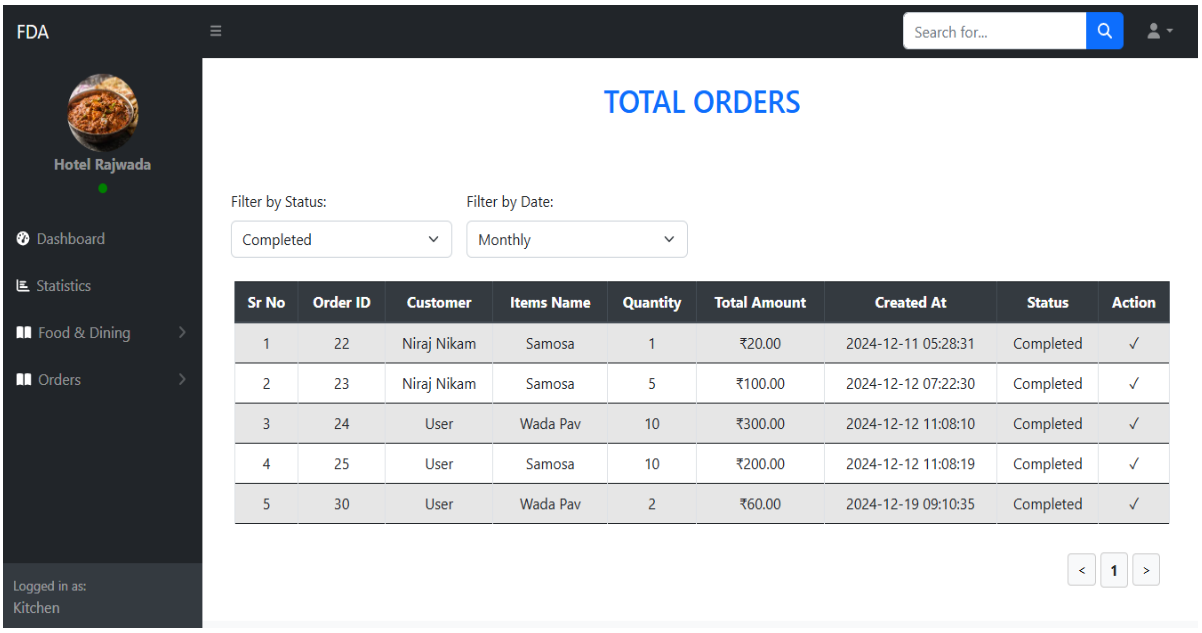
Submit

Logged in as:
Kitchen

Copyright © Open Future Technologies 2024

Privacy Policy · Terms & Conditions

- **Order Management**
  A view of the orders page for the including order details and update options.



TOTAL ORDERS

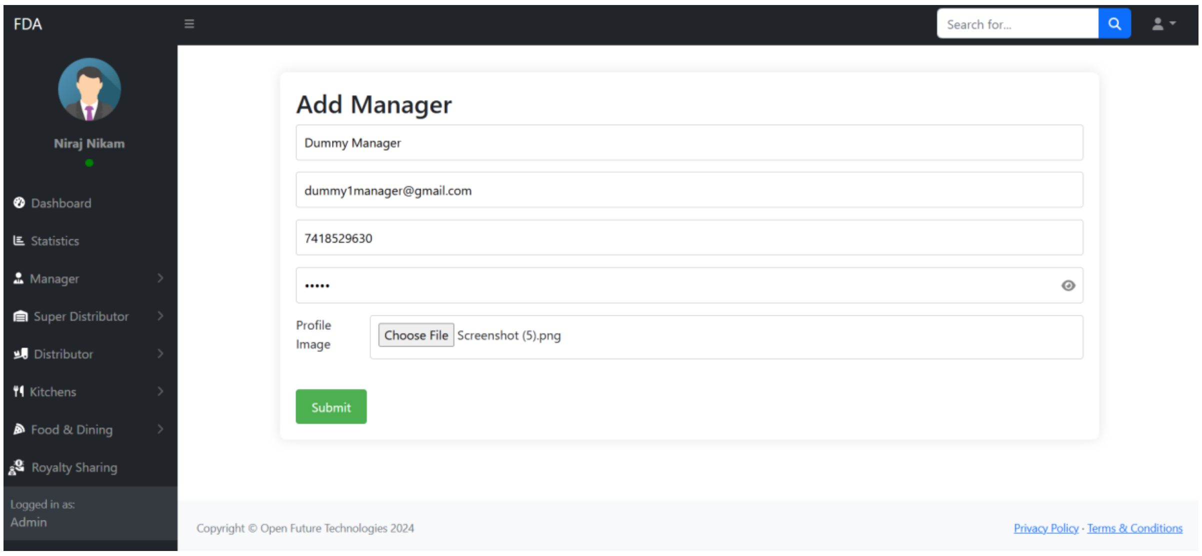
Filter by Status: Completed

Filter by Date: Monthly

Sr No	Order ID	Customer	Items Name	Quantity	Total Amount	Created At	Status	Action
1	22	Niraj Nikam	Samosa	1	₹20.00	2024-12-11 05:28:31	Completed	✓
2	23	Niraj Nikam	Samosa	5	₹100.00	2024-12-12 07:22:30	Completed	✓
3	24	User	Wada Pav	10	₹300.00	2024-12-12 11:08:10	Completed	✓
4	25	User	Samosa	10	₹200.00	2024-12-12 11:08:19	Completed	✓
5	30	User	Wada Pav	2	₹60.00	2024-12-19 09:10:35	Completed	✓

< 1 >

- **CRUD Operations for Roles**:
  Screenshots of CRUD operations for roles like adding, editing any entity.



Add Manager

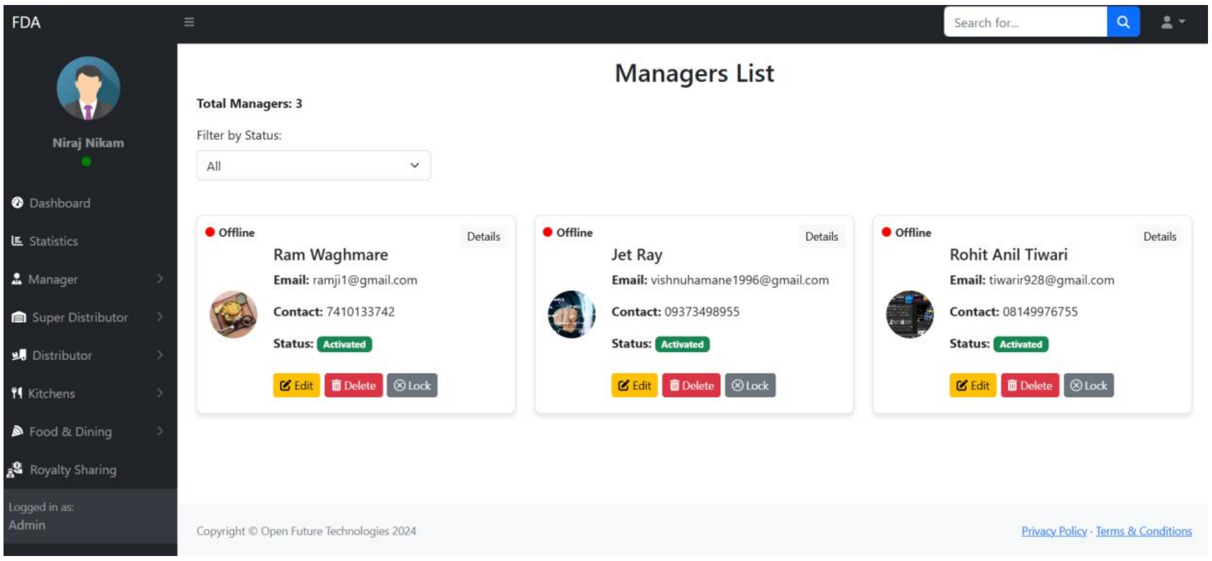
Dummy Manager

dummy1manager@gmail.com

7418529630

•••••

Profile Image: Screenshot (5).png



FDA

Niraj Nikam

Logged in as:
Admin

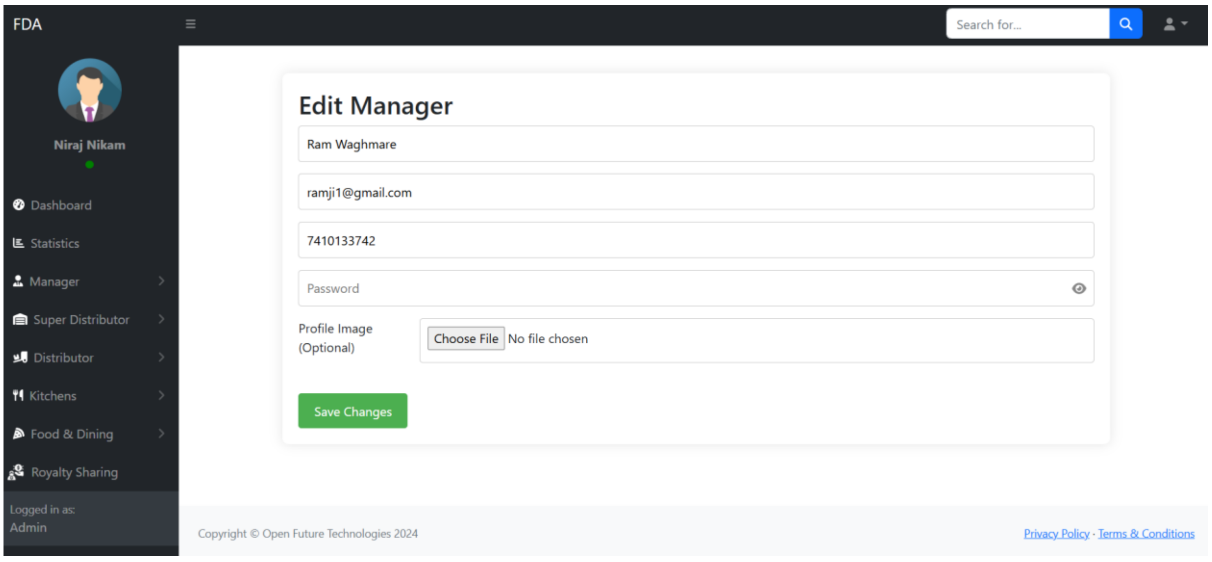
Managers List

Total Managers: 3

Filter by Status: All

● Offline · Details	● Offline · Details	● Offline · Details
Ram Waghmare	Jet Ray	Rohit Anil Tiwari
Email: ramji1@gmail.com	**Email:** vishnuhamane1996@gmail.com	**Email:** tiwarir928@gmail.com
Contact: 7410133742	**Contact:** 09373498955	**Contact:** 08149976755
Status: Activated	**Status:** Activated	**Status:** Activated
Edit · Delete · Lock	Edit · Delete · Lock	Edit · Delete · Lock

Copyright © Open Future Technologies 2024 — Privacy Policy · Terms & Conditions



FDA

Niraj Nikam

Logged in as:
Admin

Edit Manager

Ram Waghmare

ramji1@gmail.com

7410133742

Password

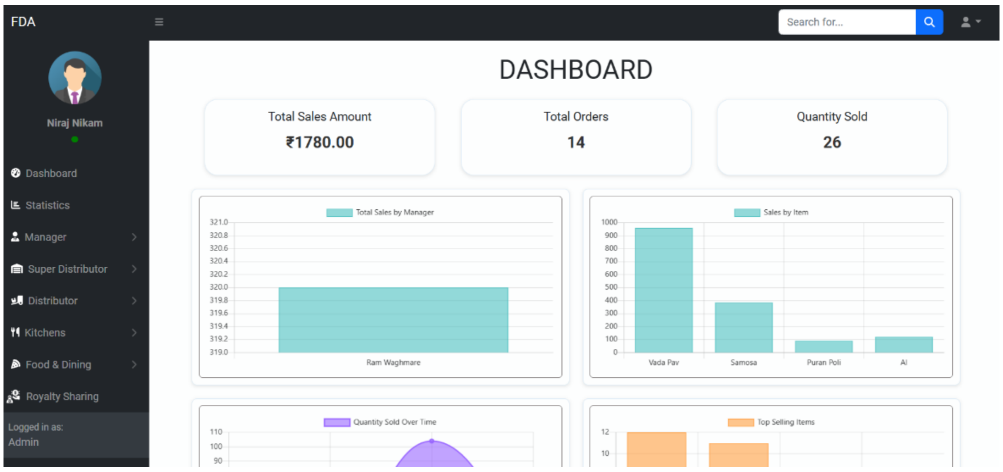
Profile Image (Optional): Choose File No file chosen

Save Changes

Copyright © Open Future Technologies 2024 — Privacy Policy · Terms & Conditions

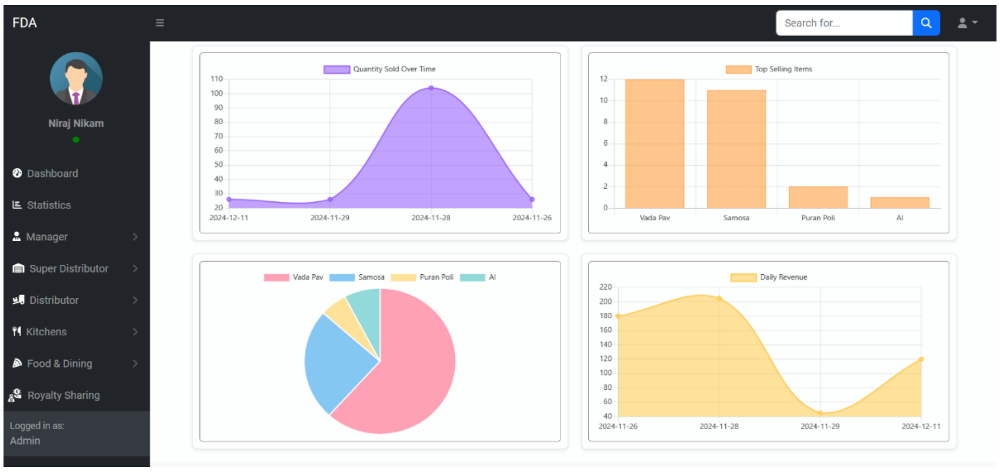
3. Charts and Reports

- **Chart Example**:



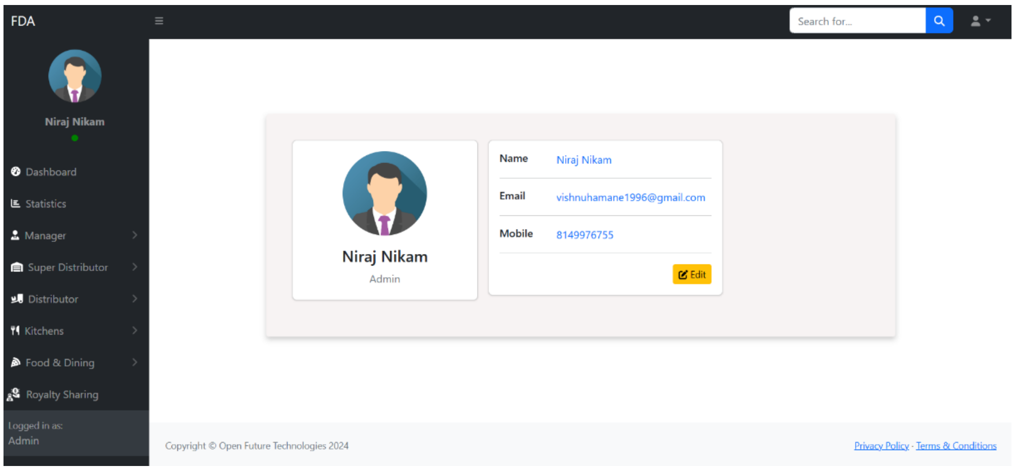
DASHBOARD

Total Sales Amount	Total Orders	Quantity Sold
₹1780.00	14	26



4. Profile Management

- **Profile Page**:
  Screenshot of the profile page showing user details and edit options.



Niraj Nikam
Admin

Name	Niraj Nikam
Email	vishnuhamane1996@gmail.com
Mobile	8149976755

Copyright © Open Future Technologies 2024

Steps to Run the Project

1. Clone the Repository

- Copy the repository URL from the provided link.
- Open your terminal or command prompt and run the following command to clone the repository:

```
git clone https://github.com/ramwaghmare2/food_delivery_hierarchy
```

2. Open the Project in Your IDE

- Launch your preferred Integrated Development Environment (IDE) (e.g., VS Code, PyCharm).
- Open the cloned repository folder in the IDE.

3. Set Up and Activate the Virtual Environment

- **Create a Virtual Environment**:

```
python -m venv env
```

- **Activate the Environment**:
    - **Windows**:

```
env\Scripts\activate
```

- Ensure that the virtual environment is activated by checking the prompt in your terminal.

4. Install Dependencies

- Use the requirements.txt file to install all required dependencies:

```
pip install -r requirements.txt
```

5. Configure the Database

- **Migrate the Database**: Run the migration commands to set up the database schema:

```
flask db init
flask db migrate
flask db upgrade
```

6. Run the Application

- Start the Flask application using the following command:

```
python app.py
```

- The application will start running on a default local server (e.g., http://127.0.0.1:5000/user/login).

7. Open the Application in the Browser

- Copy the URL displayed in the terminal (http://127.0.0.1:5000/user/login) and paste it into your browser.

8. Create a User Account

- Navigate to the **Sign-Up** page in the application.
- Fill in the required details, such as Name, Email, Password, and Role, and create your account.

9. Log In to the Application

- Use the email and password provided during the sign-up process to log in.
- After logging in, you will be redirected to the appropriate dashboard based on your role.

10. Test and Explore

- Verify that the features such as adding data, updating records, and other functionalities work as expected.

CONCLUSION

The **Role-Based Hierarchy Management System** successfully addresses the challenges associated with managing multi-level organizational roles. By implementing robust role-based access control, automated royalty calculations, and streamlined order management, the system enhances operational efficiency, reduces manual intervention, and ensures data security.

Key achievements of this project include:

- A scalable architecture that supports hierarchical role management.
- Seamless integration of user roles with custom dashboards and functionalities.
- Real-time data analytics for informed decision-making.
- Secure handling of sensitive user and transactional data using Flask and MySQL.

This system serves as a reliable platform for organizations looking to automate their hierarchical management processes while maintaining data integrity and security.

Acknowledgments
We extend our sincere gratitude to **Open Future Technology** and our Team Lead for providing us with this opportunity to develop the **Role-Based Hierarchy Management System**. The guidance, resources, and support were instrumental in completing this project.

We also thank our fellow team members for their collaboration and dedication to making this project a success.

Python Development Team
Open Future Technology