QuickFood: A Online Food Delivery Web App

Software Design Document (SDD)

1. Introduction....3
1.1 Purpose....3
1.2 Scope....3
1.3 Overview....5
1.3.1 Introduction:....5
1.3.2 System Description:....5
1.3.3 Design Overview:....5
1.3.4 Object Model:....5
1.3.5 Subsystem Decomposition:....5
1.3.6 Data Design:....5
1.3.7 User Requirement and Component Traceability Matrix:....6
1.4 References....6
1.5 Definition and Acronyms....6
2. System Description....7
3. Design Overview....9
3.1 Design Rationale....9
3.2 System Architecture....9
3.3 Constraint and Assumptions....10
3.3.1 List of Assumptions....10
3.3.2 List of Dependencies....11
4. Object Model....12
4.1 Object Description....12
4.1.1 Object 1: Menu....12
4.1.2 Object 2: Notification....13
4.1.3 Object 3: Restaurant....14
4.1.4 Object 4: User....15
4.1.5 Object 5: Order....16
4.1.6 Object 6: Review....17
4.1.7 Object 7: RiderStatus....18
4.1.8 Object 8: OrderDetails....19
4.2 Object Collaboration Diagram....20
5. Subsystem Decomposition:....21
5.1 Complete package diagram:....21
5.2 Subsystem Detail Description:....22
5.2.1 User Management Subsystem:....22
5.2.1.1 Module Description:....22
5.2.1.2 Class Diagram:....23
5.2.1.3 Subsystem interface:....24
5.2.2 Restaurant Management Subsystem:....24
5.2.2.1 Module Description:....24
5.2.2.2 Class Diagram:....25

5.2.2.3 Subsystem interface:..... 26
5.2.3 Order Management Subsystem:..... 26
5.2.3.1 Module Description:..... 26
5.2.3.2 Class Diagram:..... 27
5.2.3.3 Subsystem interface:..... 28
6. Data Design:..... 28
6.1 Data Description:..... 28
6.2 Data Dictionary:..... 29
6.3 Entity Relationship Diagram:..... 32
7. User Requirement and Component Traceability Matrix:..... 32

1. Introduction

1.1 Purpose

The purpose of this Software Design Document (SDD) is to provide a comprehensive blueprint for the development of QuickFood, a food delivery web application. It outlines the architecture, design decisions, and technical specifications necessary for the successful implementation of the software. The SDD serves as a guide for developers, designers, testers, and other stakeholders involved in the development process, ensuring a clear understanding of the system's structure and functionality.

Audience: The intended audience of this SDD includes:

1. **Development Team:** Software engineers, programmers, and developers responsible for implementing the QuickFood web application.
2. **Design Team:** User experience (UX) designers and user interface (UI) designers involved in crafting the visual design and user interactions of the application.
3. **Quality Assurance Team:** Testers and quality assurance analysts responsible for verifying the correctness, reliability, and performance of the software.
4. **Project Managers:** Individuals overseeing the development process, including scheduling, resource allocation, and project coordination.
5. **Stakeholders:** Clients, investors, and other parties with a vested interest in the success of the QuickFood project.

By addressing the needs of these key stakeholders, this SDD aims to facilitate effective communication, collaboration, and decision-making throughout the software development lifecycle.

1.2 Scope

QuickFood is a web-based food delivery application designed to streamline the process of ordering food from local restaurants and having it delivered to the customer's doorstep. The scope of the QuickFood project encompasses the development of both the customer-facing application and the administrative dashboard for restaurant owners and delivery drivers.

Goals, Objectives, and Benefits:

1. **Convenience**: The primary goal of QuickFood is to provide users with a convenient and efficient way to order food from a variety of restaurants without the need to visit them physically. By offering a user-friendly interface and seamless ordering process, QuickFood aims to enhance the overall dining experience for customers.
2. **Accessibility**: QuickFood seeks to make restaurant food accessible to a wider audience by enabling users to browse menus, place orders, and track deliveries from the comfort of their own homes or workplaces. This accessibility benefits busy individuals, families, and office workers who may not have the time or opportunity to dine out.
3. **Choice and Variety**: The application aims to offer users a diverse selection of cuisines and dining options from local restaurants, ensuring that customers can find something to suit their tastes and preferences. By partnering with a range of eateries, QuickFood provides users with access to a wide variety of dishes, from fast food to fine dining.
4. **Efficiency**: QuickFood aims to streamline the food ordering and delivery process for both customers and restaurant partners. By leveraging technology to optimize routing and logistics, the application seeks to minimize delivery times and ensure timely service. This efficiency benefits both customers, who receive their orders promptly, and restaurants, which can fulfill orders more effectively.
5. **Revenue Generation**: For restaurant owners, QuickFood offers an opportunity to increase revenue by reaching a broader customer base and expanding their delivery services without the need for significant investment in infrastructure or marketing. By joining the QuickFood platform, restaurants can attract new customers and boost sales while benefiting from the app's streamlined order management tools.

The QuickFood project aims to revolutionize the food delivery industry by leveraging technology to connect customers with local restaurants in a convenient, accessible, and efficient manner, thereby enhancing the dining experience for all stakeholders involved.

1.3 Overview

1.3.1 Introduction:

The introduction section of this Software Design Document (SDD) offers a concise overview of the document's purpose and its role in guiding the software development process. It provides a brief description of the software project, outlining its goals, objectives, and anticipated benefits.

1.3.2 System Description:

The system description section of this SDD delves into the detailed design of the software, encompassing its architectural design, software components, interfaces, and data management aspects. It also elucidates the development tools, technologies, and coding standards utilized in the software implementation. Furthermore, this section discusses the testing and maintenance strategies employed for the software.

1.3.3 Design Overview:

In this section, a high-level overview of the software design is presented, elucidating the system architecture and major components. It outlines the primary functionality of the software system and elucidates how these components synergize to achieve the project's goals and objectives.

1.3.4 Object Model:

This section delineates the object model for the software system, detailing the classes and their relationships. It provides a visual representation of the software design, elucidating how the various components interact with each other.

1.3.5 Subsystem Decomposition:

Here, the subsystems constituting the software system are described, including their respective functionalities and interrelationships. This section offers a comprehensive understanding of the distinct subsystems and their roles within the software system.

1.3.6 Data Design:

This section outlines the data design for the software system, comprising the database schema and data flow diagrams. It provides an in-depth description of the data structures utilized in the software system and elucidates how they are accessed and manipulated.

1.3.7 User Requirement and Component Traceability Matrix:

In this section, the user requirements for the software system are delineated, encompassing aspects such as user interface design and user experience. A detailed description of the various user requirements and their alignment with the software system is provided, ensuring comprehensive coverage and facilitating testing and verification processes.

1.4 References

https://nextjs.org/

https://spring.io/projects/spring-boot

https://dev.mysql.com/doc/

https://developer.mozilla.org/en-US/docs/Glossary/MVC

1.5 Definition and Acronyms

- UI: User Interface
- API: Application Programming Interface
- UX: User Experience
- JS: JavaScript
- CRUD: Create, Read, Update, Delete
- HTTPS: Hypertext Transfer Protocol Secure
- SQL: Structured Query Language
- JSON: JavaScript Object Notation

2. System Description

The QuickFood web application embodies a dynamic food delivery platform tailored to simplify the ordering process from local restaurants while equipping restaurant proprietors and delivery personnel with effective management tools. It operates within the contemporary landscape of dining preferences, where consumers increasingly favor the convenience of online food ordering.

Functionality Overview:

User Management: QuickFood prioritizes user convenience and security, offering a seamless user management experience. Users have the flexibility to create and maintain their profiles with ease, updating personal details and delivery addresses as needed. The platform ensures secure authentication and registration processes, safeguarding user information and accounts. Through a user-friendly interface, users can efficiently log in, register, and manage their accounts, enhancing their overall experience on the platform.

Restaurant Management: QuickFood empowers restaurant owners with a robust set of management tools, streamlining the process of menu and restaurant management. Administrators have access to a centralized dashboard where they can effortlessly update menus, add new items, and adjust prices to reflect changes in offerings. This centralized approach enhances efficiency and consistency across restaurant operations, allowing owners to maintain up-to-date menus and respond promptly to customer preferences and market trends.

Order Management: With QuickFood's order management system, users can seamlessly place orders while restaurant administrators efficiently handle incoming orders. The platform provides users with intuitive interfaces to specify delivery preferences and payment methods, ensuring a smooth ordering experience. Meanwhile, restaurant owners and customers can track delivery statuses and monitor performance metrics, optimizing order processing and delivery operations for enhanced customer satisfaction.

Rider Access: QuickFood ensures that riders have access to real-time notifications and relevant order details to facilitate efficient order fulfillment. Through the platform, riders receive notifications for new delivery assignments, enabling them to respond promptly and optimize delivery routes for timely service. With access to delivery addresses and order details, riders can navigate delivery routes effectively, ensuring timely and accurate order deliveries to customers.

Real-Time Notification: QuickFood leverages real-time notifications to keep users, restaurant administrators, and riders informed of important updates and events. Users receive timely

notifications regarding order confirmations, delivery statuses, and promotional offers, enhancing their overall experience on the platform. Similarly, restaurant administrators and riders receive notifications for new orders, ensuring prompt response and efficient order fulfillment. This real-time communication ensures transparency and fosters trust between all stakeholders within the QuickFood ecosystem.

Administrative Privilege: QuickFood grants administrative privilege to designated users, empowering them with enhanced access and control over platform functionalities. Administrator has the authority to manage user accounts, restaurant details, and system configurations, ensuring smooth operations and compliance with organizational policies. This administrative privilege enables the administrator to effectively oversee platform activities, troubleshoot issues, and enforce security measures to safeguard user data and system integrity.

Design:

The QuickFood system is structured using a three-tier model:

- **Presentation Layer**: This layer prioritizes delivering a user-friendly interface for customers to interact with the platform and for restaurant administrators to manage their operations. It ensures an intuitive and modern design that enhances user experience and simplifies navigation.
- **Application Layer**: At the core of the system lies the application layer, which handles the business logic and processes user requests. It orchestrates order management, authentication, and interaction between the presentation and data layers.
- **Data Layer**: Responsible for managing data storage and retrieval, this layer stores critical information such as user profiles, menu items, order history, and delivery details. It ensures data integrity, security, and efficient access to support the system's functionality.

Context:

The QuickFood system operates within the evolving landscape of the food delivery industry, addressing the growing demand for convenient and efficient dining solutions. In a market saturated with online ordering platforms, QuickFood distinguishes itself by offering a seamless user experience, a diverse selection of restaurants, and robust management capabilities for restaurant partners.

Overall, QuickFood aims to revolutionize the food delivery experience by providing a comprehensive solution that caters to the needs of both customers and restaurant owners. Its user-friendly interface, efficient order management, and scalable architecture position it as a leading player in the rapidly expanding online food delivery market.

3. Design Overview

3.1 Design Rationale

The chosen architecture for the QuickFood web application was meticulously selected after weighing various critical factors and trade-offs. Adopting a three-tier architecture, comprising the presentation layer, application layer, and database layer, was deemed the most suitable approach for several reasons.

The presentation layer encapsulates the user interface, ensuring a seamless and intuitive experience for both customers and restaurant administrators. Meanwhile, the application layer houses the core business logic and functionality of the system, facilitating efficient order processing, delivery management, and user authentication. Lastly, the database layer handles data storage and retrieval, ensuring data integrity and efficient access to support system operations.

The rationale behind selecting the three-tier architecture stems from its widespread usage and proven effectiveness in web-based applications. This architecture offers a clear separation of concerns among the layers, enhancing maintainability and facilitating modifications to the system. Furthermore, it allows for scalability and performance optimization, crucial for accommodating the expected volume of data and users in a food delivery platform.

Alternative architectures, including the two-tier architecture and client-server architecture, were considered but ultimately not chosen. The two-tier architecture was dismissed due to its limitations in scalability and performance optimization compared to the three-tier architecture. Similarly, the client-server architecture was deemed unsuitable as it would have necessitated additional resources for client-side installation and maintenance, introducing complexities and overheads.

In conclusion, the adoption of a three-tier architecture for the QuickFood web application offers a clear separation of concerns, scalability, and performance optimization, aligning with the project's objectives and requirements.

3.2 System Architecture

The system architecture of the QuickFood web application comprises several high-level subsystems that collaborate to achieve the desired functionality. These subsystems include:

- **User Interface Subsystem**: Responsible for providing the user interface to the system. It receives input from users and displays relevant output, facilitating user interactions.
- **Order Management Subsystem**: Manages the processing of incoming orders, including validation, order tracking, and status updates.
- **Restaurant Management Subsystem**:Responsible for registration, updation and maintenance of the restaurants.
- **Menu Management Subsystem**: Handles the management of restaurant menus, including adding new items, updating existing ones, and adjusting prices.
- **Delivery Management Subsystem**: Responsible for managing delivery logistics, including driver assignments, delivery routes, and delivery statuses.
- **Data Management Subsystem**: Manages data storage and retrieval for the system, including user profiles, menu items, order history, and delivery details.

These subsystems interact with each other to facilitate the flow of information and the execution of system functionalities. For example, the User Interface Subsystem interacts with the Order Management Subsystem to place orders and track their status. Similarly, the Menu Management Subsystem collaborates with the Data Management Subsystem to update menu items and prices in the database.

The subsystems are designed to work together seamlessly, with loosely coupled interactions to allow for flexibility and easy maintenance. This modular architecture ensures that each subsystem can be independently developed, tested, and modified, promoting scalability and adaptability as the QuickFood platform evolves.

3.3 Constraint and Assumptions

3.3.1 List of Assumptions

- **User Authentication**: The assumption is made that users will authenticate using email and password credentials. Besides, users can also login using gmail via OAUTH2 authentication provider. No additional authentication methods, such as social media login or two-factor authentication, are currently planned.
- **Restaurant Partners**: It is assumed that restaurants partnering with QuickFood will provide accurate and up-to-date menu information for inclusion in the platform. It is also assumed that restaurants will fulfill orders promptly and maintain quality standards for food preparation and delivery.
- **Delivery Logistics**: The assumption is made that delivery personnel will operate within designated service areas and adhere to specified delivery times. Additionally, it is

assumed that delivery vehicles will be properly maintained and comply with local regulations for food delivery.

- **Data Security**: It is assumed that appropriate measures will be implemented to ensure the security and privacy of user data, including encryption of sensitive information and adherence to data protection regulations.
- **Scalability**: The assumption is made that the QuickFood platform will be designed with scalability in mind, allowing for increased user traffic and data volume as the platform grows. This includes provisions for scaling server infrastructure, database capacity, and system performance.
- **Customer Feedback**: It is assumed that customers will provide accurate and constructive feedback on their dining experiences through the QuickFood platform. Feedback will be used to improve service quality and enhance the user experience over time.
- **Payment Processing**: The assumption is made that payment processing will be handled securely and efficiently through integrated payment gateways. It is assumed that customers will provide valid payment information and complete transactions promptly.
- **Regulatory Compliance**: It is assumed that QuickFood will comply with relevant regulations and licensing requirements governing food delivery services in the regions where it operates. This includes adherence to food safety standards, labor laws, and taxation regulations.
- **Technical Infrastructure**: The assumption is made that QuickFood will have access to reliable technical infrastructure, including servers, databases, and network resources, to support the operation of the platform without significant downtime or performance issues.

3.3.2 List of Dependencies

- **External APIs**: The system will be dependent on external APIs for services such as geolocation, payment processing, and restaurant menu data retrieval. Changes or disruptions to these APIs could impact the functionality of the QuickFood platform.
- **OAUTH2 Authentication**: The system will depend on user authentication and authorization services provided by the OAuth2 provider in case of g-mail authentication.
- **Internet Connectivity**: The system relies on Internet connectivity to facilitate user interactions, order processing, and communication with external services. Any interruptions to internet connectivity could affect the availability and performance of the QuickFood platform.
- **Third-Party Services**: QuickFood may depend on third-party services for tasks such as email delivery, SMS notifications, and cloud storage. The availability and reliability of these services may impact the overall functionality and user experience of the platform.
- **Hardware Infrastructure**: The system's performance will be dependent on the underlying hardware infrastructure, including servers, network equipment, and mobile

devices used by customers and delivery personnel. Upgrades or maintenance of hardware components could affect system performance and availability.

4. Object Model

4.1 Object Description

The System Object Model section of the QuickFood web application enables the description of the subsystems in operation. This functionality allows for an overarching portrayal of the system, illustrating the grouping of various components into their respective systems. In the context of the QuickFood web app, a single system is employed, with no specified subsystems.

4.1.1 Object 1: Menu

Class Name	Menu
Brief Description	The `Menu` class models individual menu items within the QuickFood application. It stores essential details such as item name, price, category, associated restaurant, image, and quantity available. This class facilitates the organization and management of menu items, supporting user browsing, selection, and ordering.
Attributes	**Description**
`id`	int - Unique identifier for the menu item
`name`	String - Name of the menu item
`restaurant`	Restaurant - The restaurant to which the menu item belongs
`price`	double - Price of the menu item

`category`	Category - Category of the menu item (e.g., appetizer, main course, etc.)
`image`	byte[] - Image representing the menu item (stored as a byte array)
`quantity`	int - Quantity of the menu item available in stock
Methods	**Description**
`List<Menu> findByRestaurantId(String restaurantId)`	Retrieves a list of menu items belonging to the specified restaurant identified by its unique ID.
`Page<Menu> findByRestaurantId(String restaurantId, Pageable pageable)`	Retrieves a paginated list of menu items belonging to the specified restaurant identified by its unique ID. The results are returned in a paginated format, according to the provided `Pageable` configuration.
`void addMenu(Menu menu)`	This method is used to add a new menu item.
`void updateMenu(Menu menu)`	This method is used to update an existing menu
`List<Menu> getFilteredMenu(Filter filters)`	This method is used to get a menu after applying filters. Users can apply filters by name of a menu, price range, rating category etc.

4.1.2 Object 2: Notification

Class Name	**Notification**
Description	The `Notification` class represents a notification entity in a quick food delivery system. It contains information about the notification's id, associated

	user, description, timestamp, and whether it has been seen by the user.
Attribute	**Description**
id	Unique identifier for the notification.
user	User associated with the notification.
description	Description or content of the notification.
timestamp	Timestamp indicating when the notification was created.
isSeen	Boolean flag indicating whether the notification has been seen by the user.
Method	**Description**
`List<Notification> findByUserId(String userId)`	Retrieves a list of notifications associated with the specified user ID.
`void makeSeen(int id)`	Mark a notification as seen.
`void deleteByUserId(String userId)`	Deletes all notifications associated with the specified user ID.

4.1.3 Object 3: Restaurant

Class Name	**Restaurant**
Description	The `Restaurant` class represents a restaurant entity in a quick food delivery system. It contains information about the restaurant's id, name, owner, address, mobile number, registration date, and an image.
Attribute	**Description**
id	Unique identifier for the restaurant.
name	Name of the restaurant.
owner	User who owns the restaurant.

address	Address of the restaurant.
mobile	Mobile number of the restaurant.
regDate	Timestamp indicating when the restaurant was registered.
image	Image representing the restaurant (stored as bytes).
Method Signature	**Description**
`Optional<Restaurant> findById(String resId)`	Retrieves a restaurant by its unique identifier (`resId`). Returns an empty `Optional` if no restaurant is found.
`List<Restaurant> findByOwnerId(String owner)`	Retrieves a list of restaurants owned by the specified owner.
`void addRestaurant(Restaurant restaurant)`	This method is used to add a new restaurant.
`void updateRestaurant(Restaurant restaurant)`	This method is used to update a restaurant.

4.1.4 Object 4: User

Class Name	**User**
Description	The `User` class represents a user entity in a quick food delivery system. It implements the `UserDetails` interface, providing authentication and authorization details. It contains information about the user's id, name, password, role, address, mobile number, registration date, and profile picture.
Attribute	**Description**
id	Unique identifier for the user.
name	Name of the user.

password	Password for user authentication.
role	Role of the user (e.g., admin, customer, etc.).
address	Address of the user.
mobile	Mobile number of the user.
regDate	Timestamp indicating when the user was registered.
profilePic	Profile picture of the user (stored as bytes).
Method Signature	**Description**
`Optional<User> findById(String id)`	Retrieves a user by their unique identifier (`id`). Returns an empty `Optional` if no user is found.
`void addUser(User user)`	This method is used to add a new user.
`void updateProfile(ProfileDto profileDto)`	This method is used to update the profile of a user.

4.1.5 Object 5: Order

Class Name	**Order**
Description	The Order object encapsulates essential details of a user's purchase request within the QuickFood system. It includes information about the user and restaurant involved, the total price of the order, delivery specifics like address and time, the chosen payment method, and timestamps indicating key order milestones such as placement, delivery initiation, user notification, and completion. Additionally, it tracks whether the order has been prepared by the restaurant and provides a field for users to submit any complaints or feedback regarding the order.
Attribute	**Description**
id	Unique identifier for the order.
user	User who placed the order.

restaurant	Restaurant from where the order is placed.
rider	Rider who will deliver the order.
deliveryAddress	Delivery address of the order.
deliveryTime	Time required to deliver the order.
price	Total food cost.
deliveryFee	Delivery charge for the order.
paymentMethod	Payment method (online, cash on delivery)
orderPlaced	When the order is placed.
deliveryTaken	When the rider took the order from restaurant
deliveryCompleted	When the order delivery is done.
complain	User's complain about the order or delivery.
Method Signature	**Description**
`List<Integer> findPendingOrderOfRestaurant(@Param("restaurantId") String restaurantId)`	Retrieves pending orders of a restaurant.
`List<Integer> findPendingOrderOfUser(@Param("userId") String userId)`	Retrieves pending orders of a user.
`Integer getDeliveryOfRider(@Param("riderId") String riderId)`	Retrieves the order a rider needs to deliver.
`void placeOrder(Order order)`	Add a new order.

4.1.6 Object 6: Review

Class Name	Review

Description	The `Review` class represents a review entity in a quick food delivery system. It contains information about the review's id, associated order, associated menu item, rating, and timestamp.
Attribute	**Description**
id	Unique identifier for the review.
order	Order associated with the review.
menu	Menu item associated with the review.
rating	Rating given to the menu item in the review (e.g., on a scale of 1 to 5).
timestamp	Timestamp indicating when the review was created.
Method	**Description**
`Double getRestaurantRating(@Param("restaurantId") String restaurantId)`	Retrieves the average rating of a restaurant based on the reviews associated with its menu items. Requires the restaurant's ID (`restaurantId`).
`Double getMenuRating(@Param("menuId") int menuId)`	Retrieves the average rating of a specific menu item based on its reviews. Requires the menu item's ID (`menuId`).
`void submitReview(Review review)`	This method gives a rating to a menu item.

4.1.7 Object 7: RiderStatus

Class Name	RiderStatus
Description	The `RiderStatus` class represents the status of a rider in a quick food delivery system. It contains information about the rider's id and availability status.

Attribute	Description
id	Unique identifier for the rider status.
isAvailable	Boolean flag indicating whether the rider is available for delivery.
Method	**Description**
`String findAvailableRider()`	Finds and returns the ID of an available rider for delivery. Uses a native SQL query to retrieve an available rider ID.

4.1.8 Object 8: OrderDetails

Class Name	OrderDetails
Description	The `OrderDetails` class represents the details of an order in a quick food delivery system. It contains information about the order detail's id, associated order, associated menu item, and quantity.
Attribute	**Description**
id	Unique identifier for the order detail.
order	Order associated with the order detail.
menu	Menu item associated with the order detail.
quantity	Quantity of the menu item ordered in the order detail.
Method	**Description**
`List<OrderDetails> findByOrderId(int orderId)`	Retrieves a list of order details associated with a specific order ID.

4.2 Object Collaboration Diagram

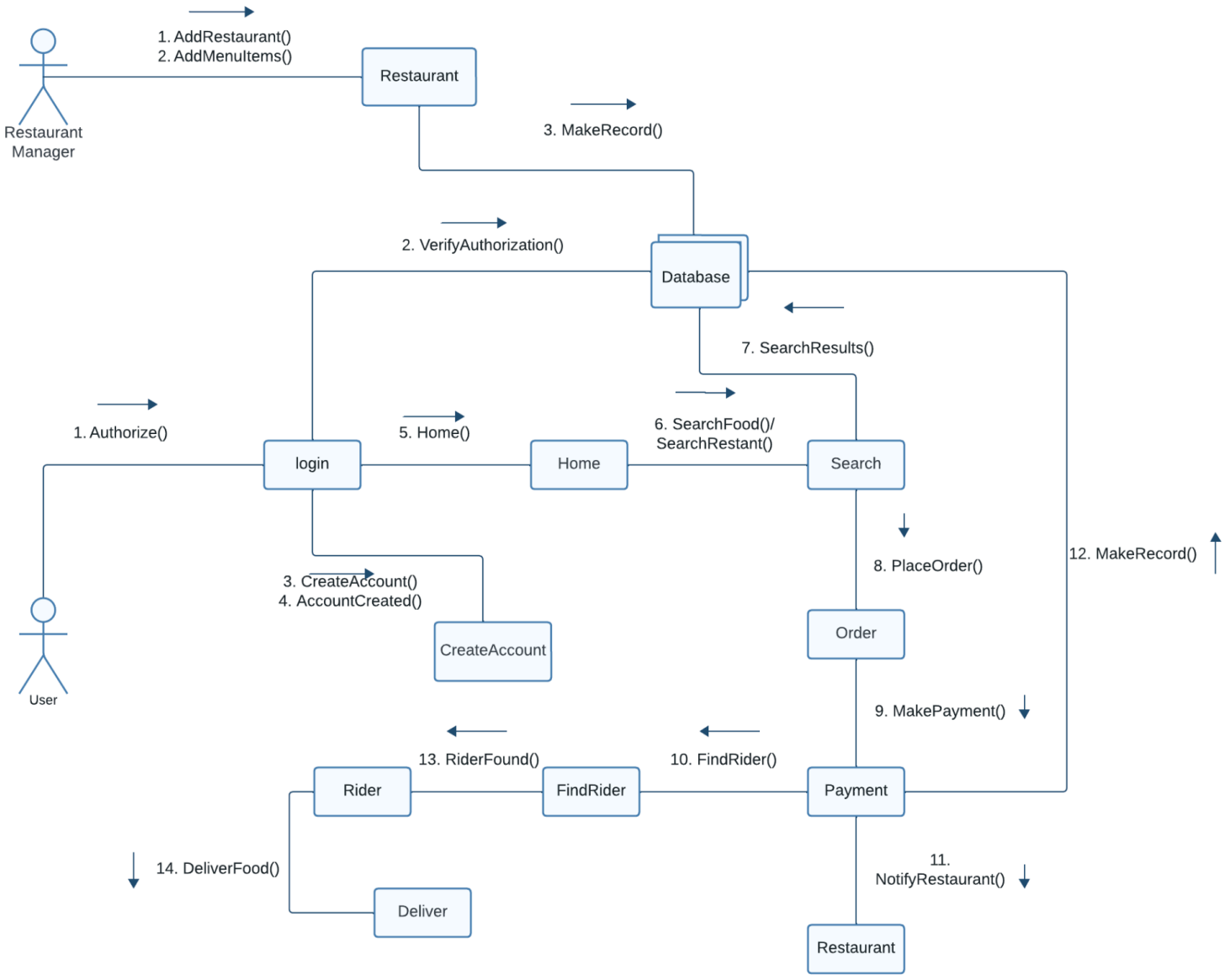

Figure: Collaboration Diagram of QuickFood web app

5. Subsystem Decomposition:

5.1 Complete package diagram:

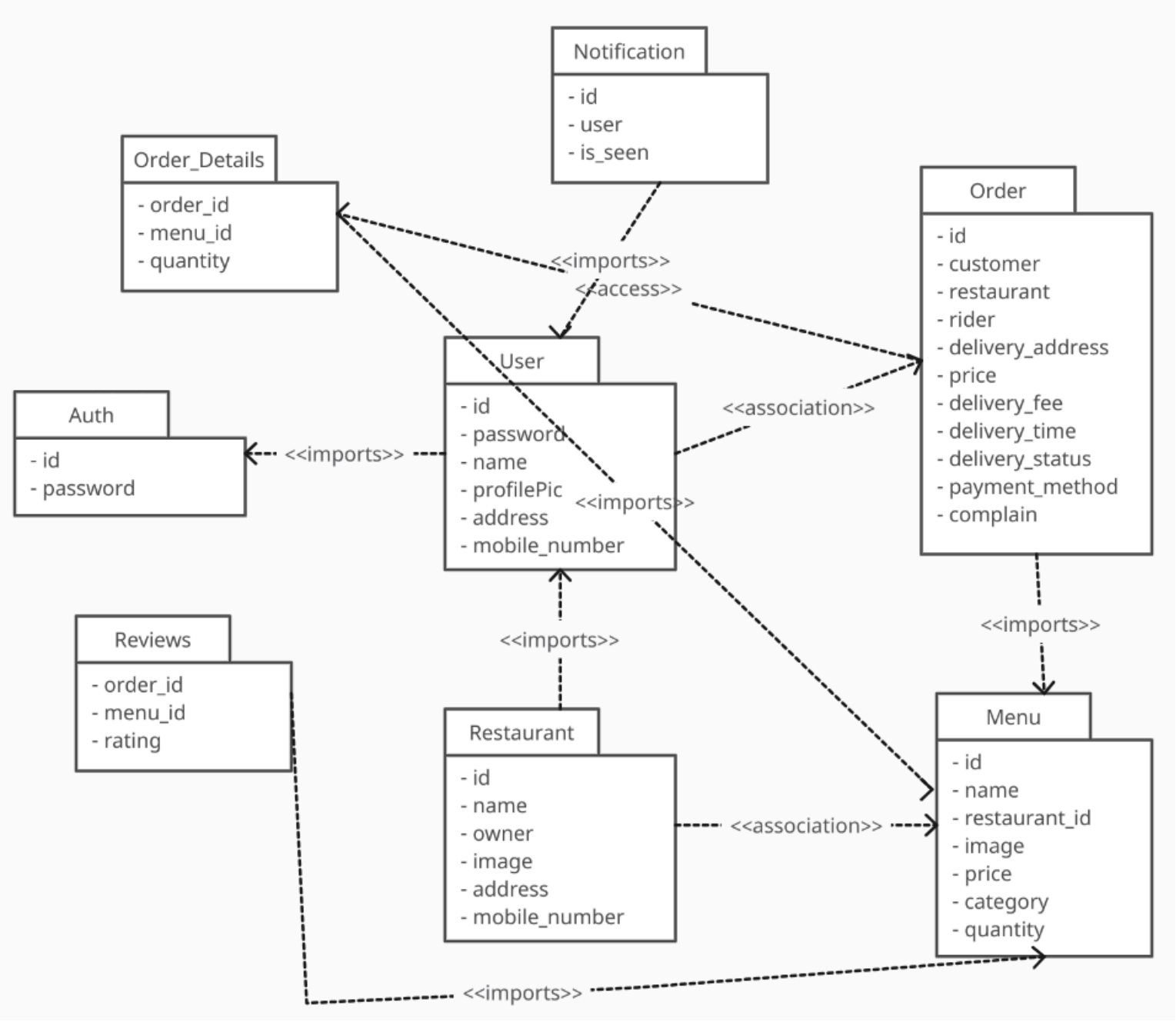

Figure : Complete Package Diagram

5.2 Subsystem Detail Description:

5.2.1 User Management Subsystem:

5.2.1.1 Module Description:

In the QuickFood web app, the user management subsystem encompasses the User class, which serves as the foundational entity for user-related functionalities. The User class is further specialized into General_User and Employee classes, representing different user roles within the system. General_User is subclassed into Customer and Restaurant_Owner classes, catering to end-users and restaurant owners, respectively. On the other hand, Employee is subclassed into Rider and Admin classes, addressing the roles of delivery riders and system administrators. The Authentication class and Authorization class are tightly associated with the User class, providing authentication and authorization mechanisms to ensure secure access and permissions management within the system.

5.2.1.2 Class Diagram:

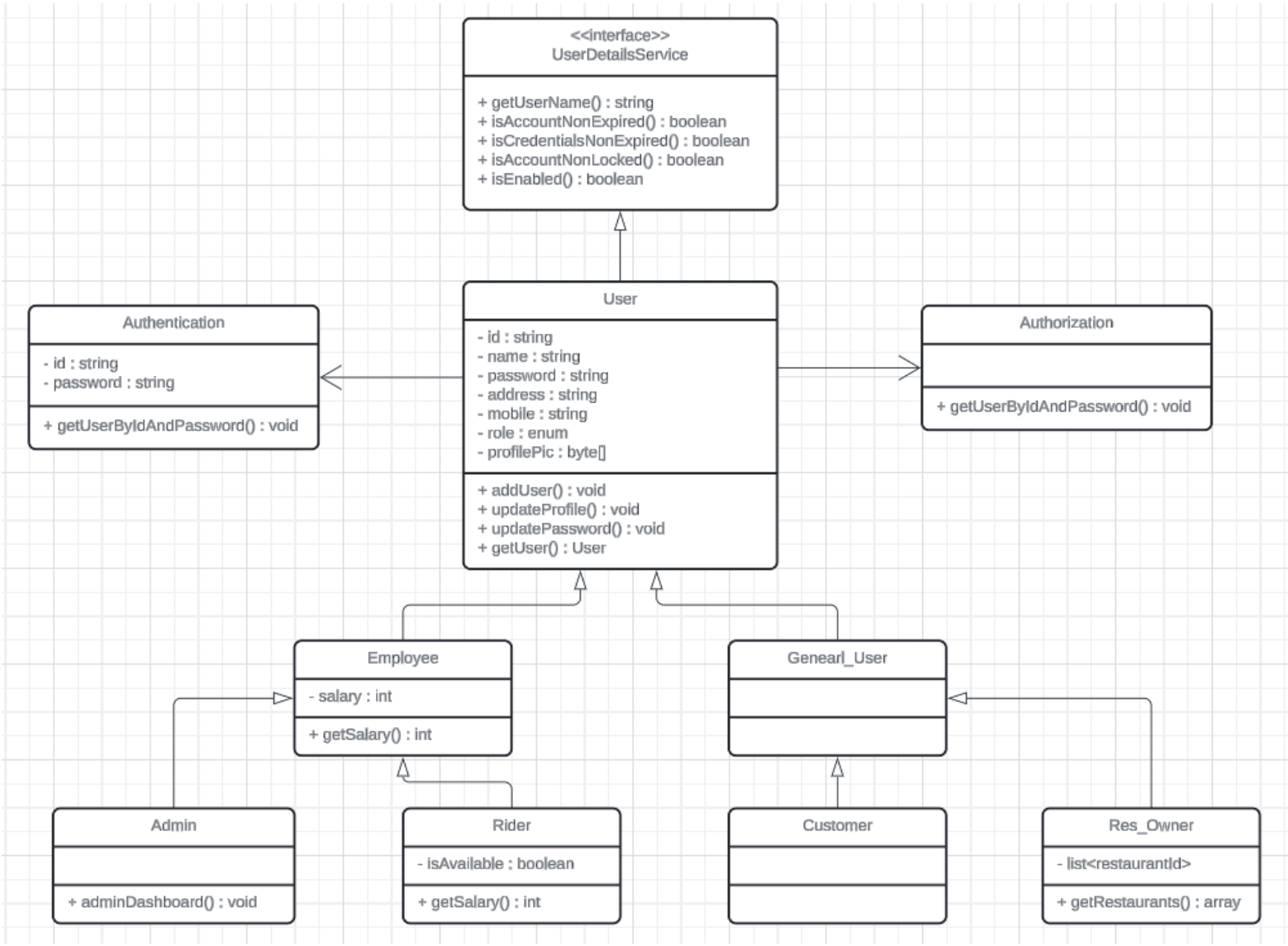

Figure : User Management Subsystem Class Diagram

5.2.1.3 Subsystem interface:

The subsystem interface for user management in the QuickFood web app encompasses profile handling and account management functionalities, providing users with comprehensive control over their accounts and profiles. Profile handling features enable users to view, update, and manage their personal information, such as contact details, preferences, and delivery addresses. Account management functionalities facilitate tasks such as account creation, password management, and email verification, empowering users to maintain the security and integrity of their accounts. Additionally, the interface offers seamless integration with the User class hierarchy, Authentication class, and Authorization class, ensuring consistent user experiences and robust security measures across all user-related operations. By encapsulating profile handling and account management functionalities, the subsystem interface enhances user satisfaction and system usability, fostering a seamless and intuitive user experience within the QuickFood web app.

5.2.2 Restaurant Management Subsystem:

5.2.2.1 Module Description:

The restaurant management subsystem in the QuickFood web app comprises the Restaurant class, Menu class, RestaurantSearch class, and MenuSearch class, each serving distinct functionalities to facilitate efficient restaurant management. The Restaurant class encapsulates essential attributes and operations related to restaurants, including add and update functionalities for managing restaurant details. Similarly, the Menu class enables the creation and modification of menu items associated with restaurants, ensuring comprehensive menu management. The RestaurantSearch class facilitates restaurant discovery and retrieval based on name queries, enhancing user navigation and exploration within the system. Conversely, the MenuSearch class empowers users to search for menu items using various criteria such as name, rating, price, and category, facilitating targeted menu exploration and selection. Notably, each menu item is associated with a specific restaurant, establishing a hierarchical relationship between restaurants and menus. Additionally, restaurants are linked to users, with owners designated as proprietors of individual establishments. Furthermore, the admin role is correlated with restaurants in performance analytics, enabling administrators to monitor and analyze restaurant performance metrics for optimization and strategic decision-making.

5.2.2.2 Class Diagram:

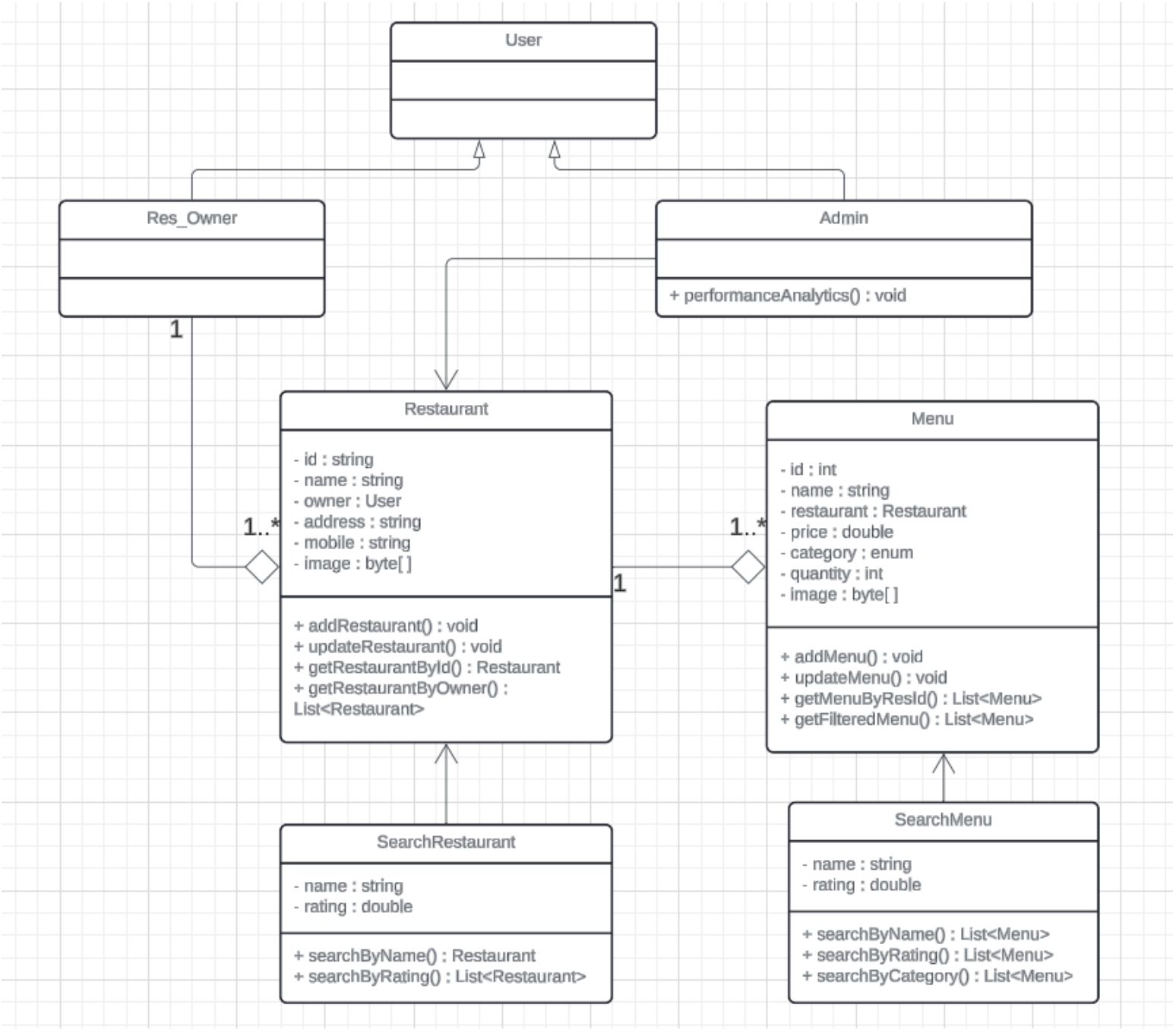

Figure : Restaurant Management Subsystem Class Diagram

5.2.2.3 Subsystem interface:

The subsystem interface for restaurant management provides users with intuitive access to functionalities for restaurant and menu administration, search, and analysis. Users can seamlessly add and update restaurant details and menu items using the Restaurant and Menu classes, respectively. RestaurantSearch and MenuSearch classes offer efficient search capabilities, allowing users to find restaurants and menu items based on specific criteria. Through the association between restaurants and users, owners can manage their establishments, while administrators leverage performance analytics to evaluate restaurant performance and make informed decisions. This cohesive interface streamlines restaurant management tasks, enhancing user productivity and facilitating effective system utilization within the QuickFood web app.

5.2.3 Order Management Subsystem:

5.2.3.1 Module Description:

The Order Management subsystem in QuickFood orchestrates the seamless processing and fulfillment of orders, ensuring a smooth journey from placement to delivery. At its core are the Order, OrderStatus, OrderDetails, Cart, and Payment classes, tightly integrated to manage every aspect of the ordering process. Users initiate orders through the Cart class, specifying their preferences and confirming purchases. These orders are then represented by instances of the Order class, which track essential details such as order ID, user, restaurant, and delivery address. The OrderStatus class facilitates real-time updates on order progress, while the OrderDetails class maintains a comprehensive record of items included in each order. Additionally, the Payment class handles secure transactions, ensuring a seamless and trustworthy payment experience. As orders progress, notifications are managed by the Notification class, providing users, restaurants, and riders with real-time updates on order statuses.

5.2.3.2 Class Diagram:

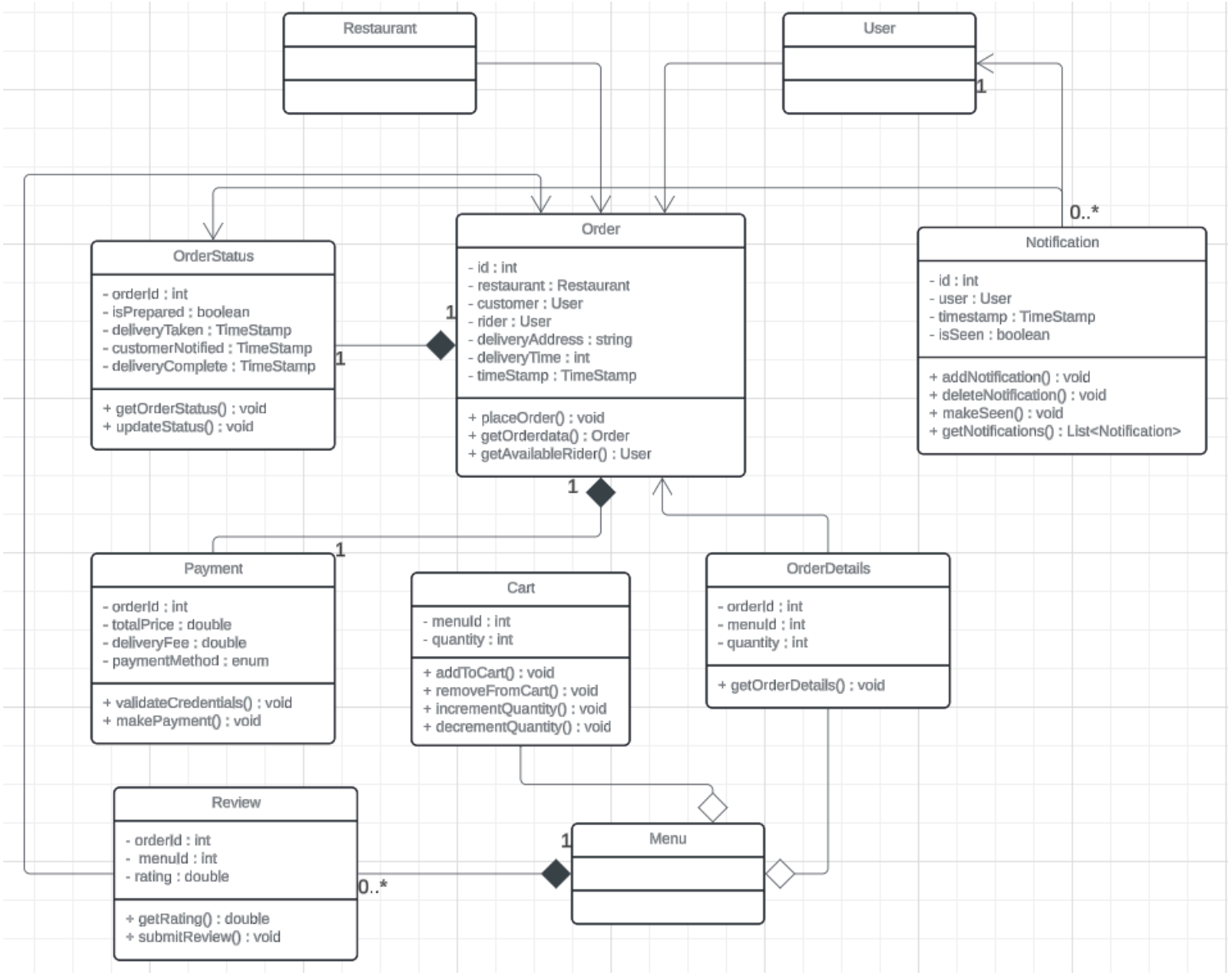

Figure : Order Management Subsystem Class Diagram

5.2.3.3 Subsystem interface:

The Order Management subsystem interface offers a comprehensive suite of functionalities to facilitate efficient order processing and delivery. Users interact with the Cart class to browse menus, add items to their cart, and proceed to checkout. Once orders are confirmed, instances of the Order class are created, capturing essential order details and initiating the fulfillment process. Real-time updates on order statuses are provided through the OrderStatus class, ensuring transparency and visibility throughout the delivery journey. The OrderDetails class enables users to view comprehensive information about their orders, including item details and quantities. Secure payment transactions are facilitated by the Payment class, ensuring a seamless and trustworthy payment experience for users. Meanwhile, the Notification class handles real-time notifications, keeping users, restaurants, and riders informed of order progress and updates. This cohesive interface streamlines order management operations, enhancing user satisfaction and facilitating efficient order processing within the QuickFood ecosystem.

6. Data Design:

6.1 Data Description:

In the QuickFood system, essential data entities such as users, restaurants, menus, orders, and order details are managed and organized using a MySQL relational database management system (RDBMS). These data structures facilitate the storage, processing, and retrieval of information related to food ordering and delivery services. QuickFood relies on MySQL as its primary database for efficient management and organization of critical system data.

The User entity stores information about registered users, including their credentials and contact details, facilitating user authentication and communication. Restaurant entities capture details about food establishments, such as their names and locations, enabling users to browse and select dining options. Menu entities contain food items offered by restaurants, including their names, prices, and availability status, providing users with a comprehensive selection of menu options. Order entity facilitates transactions between users and restaurants by recording order details, such as order IDs, items purchased, and delivery information, ensuring smooth order processing and delivery management. OrderStatus entity tracks the status of orders, including delivery and payment status, allowing users and administrators to monitor order progress. OrderDetails entity provides a breakdown of items included in each order, assisting in accurate order fulfillment and inventory management. Review entity stores feedback provided by users on their orders, facilitating user feedback analysis and restaurant rating systems. Finally, the Notification entity records notifications sent to users, helping to keep them informed about important updates and events within the system.

6.2 Data Dictionary:

Data	Type	Max Size	Description
id	VARCHAR	255	id of a user
name	VARCHAR	255	name of a user
password	VARCHAR	255	password of a user
role	ENUM	_	type of a user(customer, restaurant_manager, rider, admin)
address	VARCHAR	255	address of a user
mobile	VARCHAR	255	contact number of a user
reg_date	DATETIME	_	registration date of the user
profile_pic	BLOB	_	profile picture of a user
is_available	BOOLEAN	_	is a rider available or not
id	VARCHAR	255	id of a restaurant
name	VARCHAR	255	name of a restaurant
owner_id	VARCHAR	255	owner of a restaurant
address	VARCHAR	255	address of a restaurant
mobile	VARCHAR	255	contact number of a restaurant
image	BLOB	_	image of a restaurant
reg_date	DATETIME	_	registration date of the restaurant
reg_date	DATETIME	_	registration date of the user
profile_pic	BLOB	_	profile picture of a user
is_available	BOOLEAN	_	is a rider available or not
id	VARCHAR	255	id of a restaurant
name	VARCHAR	255	name of a restaurant
owner_id	VARCHAR	255	owner of a restaurant

address	VARCHAR	255	address of a restaurant
mobile	VARCHAR	255	contact number of a restaurant
image	BLOB	_	image of a restaurant
reg_date	DATETIME	_	registration date of the restaurant
id	INT	_	id of a menu item
restaurant_id	VARCHAR	255	id of the restaurant the menu belongs to
name	VARCHAR	255	name of a menu
quantity	INT	_	available quantity of a menu item
price	DOUBLE	_	price of a menu item
category	ENUM	_	category of the menu item (veg, non-veg, vegan, drink)

image	BLOB	_	image of a menu item
id	INT	_	id of an order
restaurant_id	VARCHAR	255	restaurant id from where the order is placed
customer_id	VARCHAR	255	customer who placed the order
rider_id	VARCHAR	255	rider who will deliver the order
price	DOUBLE	_	total price of food items without delivery charge
delivery_fee	DOUBLE	_	delivery charge of an order
delivery_time	INT	_	required time to deliver the order
delivery_address	VARCHAR	255	delivery address of an order
is_prepared	BOOLEAN	_	has the restaurant prepared the order
order_placed	DATETIME	_	timestamp of the order
delivery_taken	DATETIME	_	when rider took the delivery from the restaurant
user_notified	DATETIME	_	when rider notified the customer for delivery

delivery_completed	DATETIME	_	when rider delivered the order to the customer
payment_type	ENUM	_	payment type(COD, online)
complain	VARCHAR	1024	customer has any complain for the order
quantity	INT	_	quantity of a menu item in an order
rating	DOUBLE	_	rating of a menu item in an order
timestamp	DATETIME	_	timestamp of submitting the rating
notification_id	INT	_	id of a notification
user_id	VARCHAR	255	id of the user associated with the notification
timestamp	DATETIME	_	timestamp of the notification
description	VARCHAR	512	description of the notification
is_seen	BOOLEAN	_	has the user seen the notification

6.3 Entity Relationship Diagram:

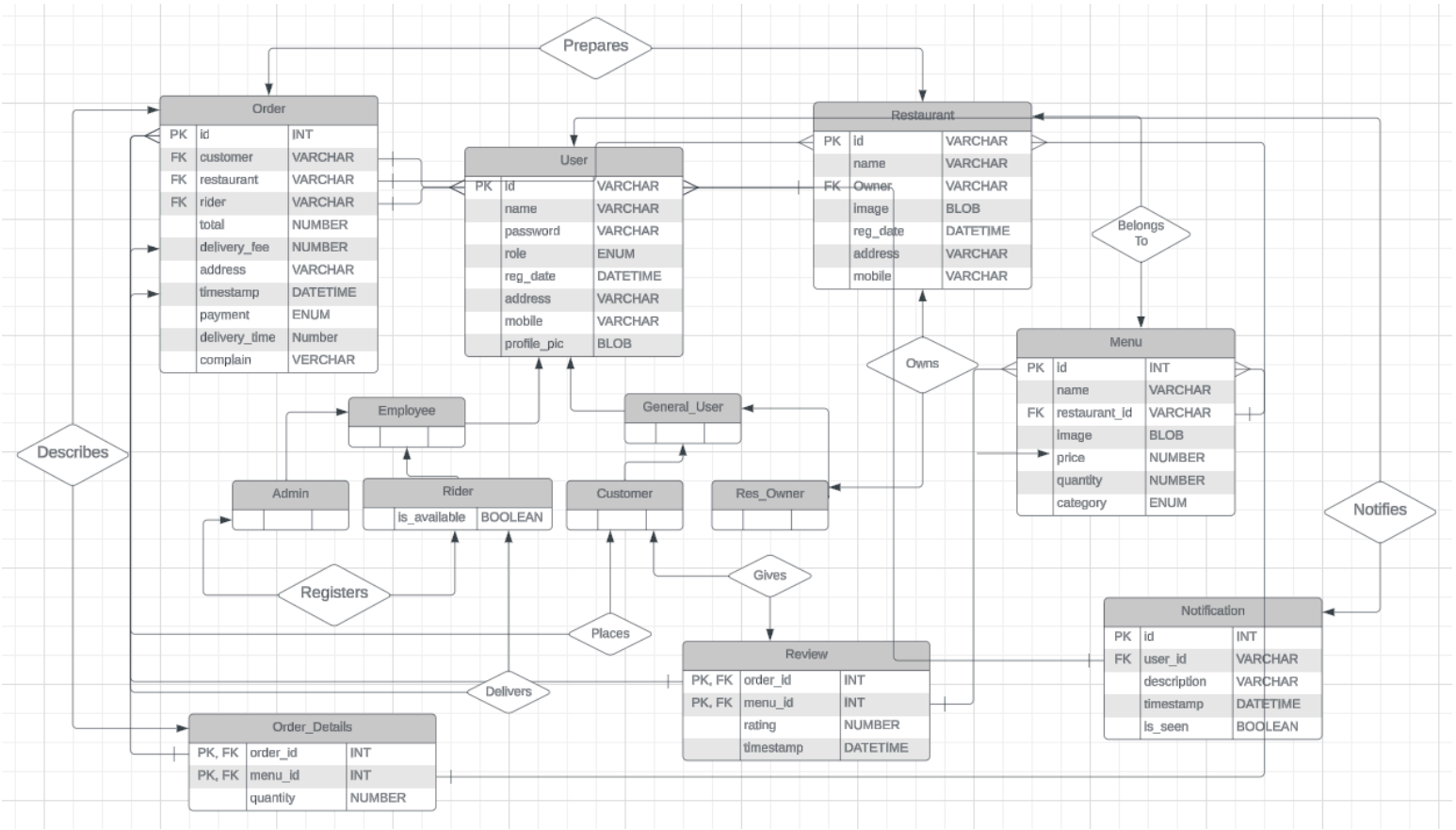

Fig : Entity Relationship Diagram

7. User Requirement and Component Traceability Matrix:

	Customer	Res_Owner	Rider	Admin
Registration	_	_	_	_
Authentication/Login	_	_	_	_
Update Profile	_	_	_	_
View food items	_	_	x	x
Place Order	_	_	x	x
Payment	_	_	x	x
Give Rating	_	_	x	x
Add New Rider	x	x	x	_
Register and update restaurants	x	_	x	x
Manage Menu	x	_	x	x
Rider interface	x	x	_	x
Administrative privilege	x	x	x	_