GitGrub

Software Design Document

Submitted By:

Meherun Farzana — Roll No : 05

Mehrajul Abadin Miraj — Roll No : 20

Aniket Joarder — Roll No : 48

Submitted On :

April 16, 2024

Submitted To :

Dr. Saifuddin Md. Tareeq, Professor

Redwan Ahmed Rizvee, Lecturer

Department of Computer Science and Engineering
University of Dhaka

Contents

1 Introduction

1.1. Purpose

This Software Design Document (SDD) aims to describe the design and architecture of the GitGrub software. The intended audience for this document includes developers, designers, project managers, and any other stakeholders involved in the development and implementation of the GitGrub system. The document provides an overview of the design and implementation approach, architecture, system components, and interfaces between the components. The purpose of this SDD is to ensure a common understanding of the GitGrub software design and to provide a reference for future development and maintenance.

1.2. Scope

The GitGrub software is a web-based application designed to order food online as well as offline from a canteen. This software enables users to order food online instead of waiting in a long queue. In addition, the waiter system of an offline canteen can be eliminated through this software. The sellers will also be able to manage their inventory, sales, and employees and track their orders.

The scope of this project includes designing and implementing the software, testing its functionality, and providing user documentation. The goals of the project are to improve the efficiency of the canteen and also make the food ordering process easier.

The objectives of the project are to develop a user-friendly and intuitive interface for both users and owners.

1.3. Overview

GitGrub will be built as a modern web-based system that uses micro-service architecture to provide scalable and efficient functionality. The front end will be designed using Next.Js, allowing users to order food. The back-end will be built using Node.js and Express.js and will be responsible for handling all the application logic and data management. It will interact with external services, authentication services, and a database management system like MongoDB. The micro-service architecture allows the application to provide a seamless food ordering experience for users, while also ensuring that the system is scalable and efficient.

1.4. Reference Material

The references used to build the application include:

- Microservice Introduction:How to build a microservices architecture with Node.Js to achieve scale?
- Kubernetes: Kubernetes
- Docker : Understanding Docker as if it were a Game Boy
- Express: Express.js
- MongoDB functions: MongoDB

1.5. Definition and Acronyms

Here are some definitions of terms, acronyms, and abbreviations that may be relevant to the travel agency micro-service project: Terms:

Micro-service: A small, independent component of a larger software application that can be developed, deployed, and scaled independently of the rest of the application.

Authentication: The process of verifying the identity of a user or system, typically through the use of a username and password or other credentials.

Invoice: A document that provides a summary of a transaction, typically including the cost of goods or services purchased and payment details.

Sales report: A document that provides an overview of sales data, typically including revenue, number of sales, and other relevant metrics.

SDD: Software Design Document - a document that describes the architecture and system design of the software.

REST: Representational State Transfer, a set of architectural principles for building web services.

2 System Description

Functionalities

- User can create an account
- User can search for preferred menu items of a canteen from the app
- User can order food from the canteen
- User can manage their cart from the app
- User can choose delivery and dine-in option
- The Canteen owner can manage the menu
- The Canteen owner can see the sales records and manage the orders
- The Canteen owner can manage the employees associated with the canteen

Context

- GitGrub is a web-based application
- The application is built using a micro-services architecture
- The application consists of several components, including a food ordering service, a dine-in canteen service, an authentication service, and a front-end

Design

- The architecture and system design ensure that the application is scalable and flexible for future enhancements and updates
- The microservices architecture enables components to be developed, deployed, and scaled separately, providing greater flexibility and scalability
- The front-end is built using Next.Js, providing an intuitive and user-friendly interface for customers
- The design focuses on providing a reliable and user-friendly food ordering experience for customers
- The security and privacy of customer data and transactions are key aspects of the design

3 Design Overview

GitGrub uses a micro-services architecture with Node.js back-end and Next.Js frontend. The design ensures a hassle-free and efficient food ordering experience, personalized to customer preference. The micro-services architecture allows for easy scaling and updating of individual components, providing greater flexibility and scalability for future enhancements and updates. The goal is to provide a reliable, secure, and user-friendly service that meets customer needs while enabling the company to improve and grow continuously.

3.1. Design Rationale

GitGrub is designed with the primary goal of providing users with a streamlined and intuitive food ordering experience. The interface is designed to be user-friendly, allowing users to effortlessly search for restaurants and food items, manage their cart, and choose between delivery or dine-in options.

Ensuring the security and privacy of user data and transactions is paramount. GitGrub employs robust security measures to safeguard sensitive information, such as personal details and payment data, throughout the ordering process. Encryption protocols and secure authentication mechanisms are implemented to protect user privacy.

GitGrub's architecture is built upon a micro-services approach, enabling greater scalability and flexibility for future enhancements and updates. Each micro-service is designed to follow the Model-View-Controller (MVC) pattern, facilitating the separation of concerns and improving maintainability. This architecture allows components to be developed, deployed, and scaled independently, providing agility and scalability as the application evolves.

GitGrub leverages modern technologies to deliver optimal performance and user experience. Node.js is chosen for the back-end due to its popularity, efficiency, and scalability. On the front-end, Next.Js is employed to create a responsive and interactive interface, enhancing user engagement and satisfaction.

In addition to catering to end-users, GitGrub offers comprehensive tools for canteen owners to manage their menus, track sales, and process orders efficiently. Owners have access to a dashboard where they can easily update menu items, view analytics, and monitor incoming orders in real time, empowering them to optimize their operations and provide exceptional service.

GitGrub is designed to be accessible across various devices, including desktops, laptops, tablets, and smartphones. This multi-platform compatibility ensures that users can access the application anytime, anywhere, enhancing convenience and accessibility.

The design philosophy of GitGrub emphasizes continuous improvement and innovation. User feedback is actively solicited and incorporated into iterative updates to enhance usability and address emerging needs. Data analytics are leveraged to gain insights into user behavior and preferences, enabling data-driven decision-making to drive the evolution of the platform.

In conclusion, GitGrub is designed to offer a high-quality and efficient online food ordering service that prioritizes user satisfaction, security, scalability, and innovation. By combining user-centric design principles with robust technology solutions, GitGrub aims to revolutionize the food ordering experience while empowering canteen owners to thrive in the digital landscape.

3.2. System Architecture

The system architecture overview for GitGrub is given:

- The system architecture for GitGrub is a microservices architecture that follows the MVC design pattern.
- Four MongoDB databases are used to store data for the different microservices.
- The front-end is built using Next.Js and the back-end is built using Node.js.
- There are seven microservices in the system: menu search, menu management, placing an order, employee management, inventory management, authentication, and writing a review.

3.3. Constraint and Assumptions

3.3.1. List of Assumptions

User Interface

- Users will access the application through a web browser that supports Next.Js.
- The application will have a responsive design to accommodate different screen sizes.
- The user interface will be designed with the user experience (UX) best practices in mind.

Network

- Users will have access to a stable and reliable internet connection.
- The micro-services will be deployed as Docker containers.

Security

- The application will implement basic security measures such as encryption of sensitive data and authentication/authorization for user access.
- The micro-services will be secured with APIs and RESTful web services.
- The databases will be hosted on a MongoDB database server.

Testing

- The application will be tested thoroughly before deployment, with automated tests for the different micro-services.
- The testing will cover functionality, performance, and security aspects.

Scalability

- The application will be scalable to handle increasing user traffic and data volume.
- The microservices will be designed to scale independently of each other.
- The application will use load balancing and auto-scaling techniques to handle varying loads.

Device Compatibility

- The application will be compatible with multiple devices, including desktops, laptops, tablets, and smartphones.
- The user interface will be optimized for different devices and screen sizes.
- The application will support different operating systems and web browsers.

⟨⟨ example: Security architecture is the design artifacts that describe how the security controls (security countermeasures) are positioned and how they relate to the overall systems architecture.

3.3.2. List of Dependencies

MongoDB: The application will rely on MongoDB databases to store data for the different micro-services.

Express: The backend will use the Express framework to handle API requests and responses.

APIs and RESTful web services: The micro-services will communicate with each other using APIs and RESTful web services.

Git: The application code will be managed using Git for version control.

4 Object Model

4.1. Object Description

4.1.1. User ⟨⟨entity⟩⟩

Class Name	User
Brief Description	Represents a user of the app.
Attribute name (String)	**Attribute Descriptions**
userID (String)	Unique identifier for the user.
email (String)	Email address of the user.
password (String) image (String)	Password of the user. Image path of the user
Methods	**Methods Descriptions**
createUserSchema	Creates the table or document in the database

Table 1: User

4.1.2. UserInfo ⟨⟨entity⟩⟩

Class Name	UserInfo
Brief Description	Represents all the information of a user for placing an order
Attribute	**Attribute Descriptions**
role (Role)	Role of the user (e.g. customer, admin, employee).
email (String)	Email address of the user.
streetAddress (String)	Street Address for online delivery
postalCode (String)	Postal code for online delivery
city (String)	City the user living in
country (String)	Country the user living in
phoneNumber (String)	Local phone number of the user
Methods	**Methods Descriptions**
createUserInfoSchema	Creates the table or document in the database

Table 2: UserInfo

4.1.3. Order ⟨⟨entity⟩⟩

Class Name	Order
Brief Description	Represents an order placed by a user.
Attribute	**Attribute Descriptions**
orderId (String)	Unique identifier for the order.
userId (String) userEmail (String)	Identifier of the user who placed the order. Email of the user who placed the order
items (List ⟨MenuItem ⟩)	List of menu items included in the order.
status (OrderStatus)	Current status of the order (e.g., pending, completed).
totalAmount (double)	Total amount of the order.
orderDate (Date)	Date and time when the order was placed.
Methods	**Methods Descriptions**
createOrderSchema	Creates the table or document in the database

Table 3: Order

4.1.4. MenuItem ⟨⟨entity⟩⟩

Class Name	MenuItem
Brief Description	Represents a menu item available for ordering.
Attribute	**Attribute Descriptions**
itemId (String)	Unique identifier for the menu item.
name (String)	Name of the menu item.
category (String) size (String)	Category of the menu item (e.g., breakfast, lunch). Size of the item
basePrice (double) extraIngradientPrice (double)	Price of the menu item. Price of the additional ingradients
description (String) image (String)	Item description on display Image path of the item
Methods	**Methods Descriptions**
createMenuItemSchema	Creates the table or document in the database

Table 4: MenuItem

4.1.5. Employee ⟨⟨entity⟩⟩

Class Name	Employee
Brief Description	Represents an employee working in the canteen.
Attribute	**Attribute Descriptions**
employeeId (String)	Unique identifier for the employee.
employeeName (String)	Name of the employee.
email (String)	Email address of the employee.
role (Role)	Role of the employee (e.g., chef, waiter).
hireDate (Date)	Date when the employee was hired.
Methods	**Methods Descriptions**
createEmployeeSchema	Creates the table or document in the database

Table 5: Employee

4.1.6. Review ⟨⟨entity⟩⟩

Class Name	Review
Brief Description	Represents a review submitted by a user for a menu item.
Attribute	**Attribute Descriptions**
reviewId (String)	Unique identifier for the review.
userId (String)	Identifier of the user who submitted the review.
menuItemId (String)	Identifier of the menu item being reviewed.
rating (int)	Rating given by the user for the menu item.
comment (String)	Comment provided by the user.
reviewDate (Date)	Date when the review was submitted.
Methods	**Methods Descriptions**
createReviewSchema	Creates the table or document in the database

Table 6: Review

4.1.7. InventoryItem ⟨⟨entity⟩⟩

Class Name	InventoryItem
Brief Description	Represents an item in the canteen's inventory.
Attribute	**Attribute Descriptions**
itemId (String)	Unique identifier for the inventory item.
name (String)	Name of the inventory item.
quantity (int)	Quantity of the inventory item available.
Methods	**Methods Descriptions**
createInventoryItemSchema	Creates the table or document in the database

Table 7: InventoryItem

4.1.8. UserInterface ⟨⟨boundary⟩⟩

Class Name	UserInterface
Brief Description	Represents the user interface of the web application.
Attribute	**Attribute Descriptions**
None	
Methods	**Methods Descriptions**
displayLoginScreen(): void	Displays the login screen for users to sign in.
displaySignupScreen(): void	Displays the signup screen for users to create new accounts.
displayOrderHistory(): void	Displays the order history for logged-in users.
displayMenu(): void	Displays the menu for users to view and order items.
displayFeedbackForm(): void	Displays the feedback form for users to submit reviews.
displayInventory(): void	Displays the inventory management interface for employees.

Table 8: UserInterface

4.1.9. NotificationInterface ⟨⟨boundary⟩⟩

Class Name	NotificationSystem
Brief Description	Represents the notification system for sending notifications to users.
Attribute	**Attribute Descriptions**
None	
Methods	**Methods Descriptions**
sendOrderConfirmation(): void	Sends a confirmation notification to users after placing an order.
sendPasswordResetLink(): void	Sends a password reset link to users who forgot their passwords.
sendOrderReadyNotification(): void	Sends a notification to users when their order is ready for pickup.

Table 9: NotificationInterface

4.1.10. FeedbackInterface ⟨⟨boundary⟩⟩

Class Name	FeedbackInterface
Brief Description	Represents the feedback form for submitting reviews.
Attribute	**Attribute Descriptions**
None	
Methods	**Methods Descriptions**
collectFeedback(): void	Collects feedback from users in the form of ratings and comments.

Table 10: FeedbackInterface

4.1.11. MenuManagementInterface ⟨⟨boundary⟩⟩

Class Name	MenuManagementInterface
Brief Description	Represents the menu management system for managing menu items.
Attribute	**Attribute Descriptions**
None	
Methods	**Methods Descriptions**
displayMenu(): void	Displays the menu management interface for shop owners.
updateMenuItem(): void	Updates menu items based on shop owners' requirements.

Table 11: MenuManagementInterface

4.1.12. MenuSearchInterface ⟨⟨boundary⟩⟩

Class Name	MenuSearchInterface
Brief Description	Searches menu items for the user
Attribute	**Attribute Descriptions**
searchQuery (String):	Holds the user's search query entered in the menu search interface.
Methods	**Methods Descriptions**
displayMenu(): void	Displays the menu management interface for shop owners.
getSearchQuery(): String	Retrieves the user's search query from the menu search interface.
performSearch(query: String)	List ⟨MenuItem ⟩: Executes a search based on the provided query and returns a list of menu items matching the search criteria.

Table 12: MenuSearchInterface

4.1.13. AuthenticationController ⟨⟨controller⟩⟩

Class Name	AuthenticationController
Brief Description	Handles user authentication-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
signUpUser(POST Request): boolean	Facilitates user signup process.
signInUser(POST Request): boolean	Facilitates user sign in process.
forgotPassword(POST Request): boolean	Initiates the password reset process for users.
resetPassword(PUT Request): boolean	Resets user password.

Table 13: AuthenticationController

4.1.14. OrderController ⟨⟨controller⟩⟩

Class Name	OrderController
Brief Description	Handles order-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
placeOrder(POST Request): Order	Places a new order for users.
updateOrderStatus(PUT Request): boolean	Updates the status of orders.
getOrderHistory(GET Request): List ⟨Order ⟩	Retrieves order history for users.
getOrderById(GET Request): Order	Retrieves order by ID.

Table 14: OrderController

4.1.15. MenuController ⟨⟨controller⟩⟩

Class Name	MenuController
Brief Description	Handles menu-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
addMenuItem (POST Request): boolean	Adds new menu items.
updateMenuItem (PUT Request): boolean	Updates menu items.
removeMenuItem (DELETE Request): boolean	Removes menu items.
getMenu (GET Request): List ⟨MenuItem⟩	Retrieves the menu.

Table 15: MenuController

4.1.16. MenuSearchController ⟨⟨controller⟩⟩

Class Name	MenuSearchController
Brief Description	Handles menu-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
searchMenu(query: String): List¡MenuItem¿	Executes a search in the menu based on the provided query and returns a list of menu items matching the search criteria.
displaySearchResults (results: List ⟨MenuItem ⟩): void	Displays the search results to the user.

Table 16: MenuSearchController

4.1.17. EmployeeController ⟨⟨controller⟩⟩

Class Name	EmployeeController
Brief Description	Handles employee-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
addEmployee(POST Request): boolean	Adds new employees.
updateEmployeeRole(PUT Request): boolean	Updates employee roles.
getEmployee(GET Request): Employee	Retrieves employee status.

Table 17: EmployeeController

4.1.18. ReviewController ⟨⟨controller⟩⟩

Class Name	ReviewController
Brief Description	Handles review-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
addReview(POST Request): boolean	Adds new reviews.
getReviewsByMenuItem (GET Request): List ⟨Review⟩	Retrieves reviews by menu item.

Table 18: ReviewController

4.1.19. InventoryController ⟨⟨controller⟩⟩

Class Name	InventoryController
Brief Description	Handles inventory-related operations.
Attribute	**Attribute Descriptions**
req.body.data(Object)	The data sent from the frontend
Methods	**Methods Descriptions**
updateInventory(PUT Request): boolean	Updates inventory items.
getInventory(GET Request): List ⟨InventoryItem ⟩	Retrieves the inventory.

Table 19: InventoryController

4.2. Object Collaboration Diagram

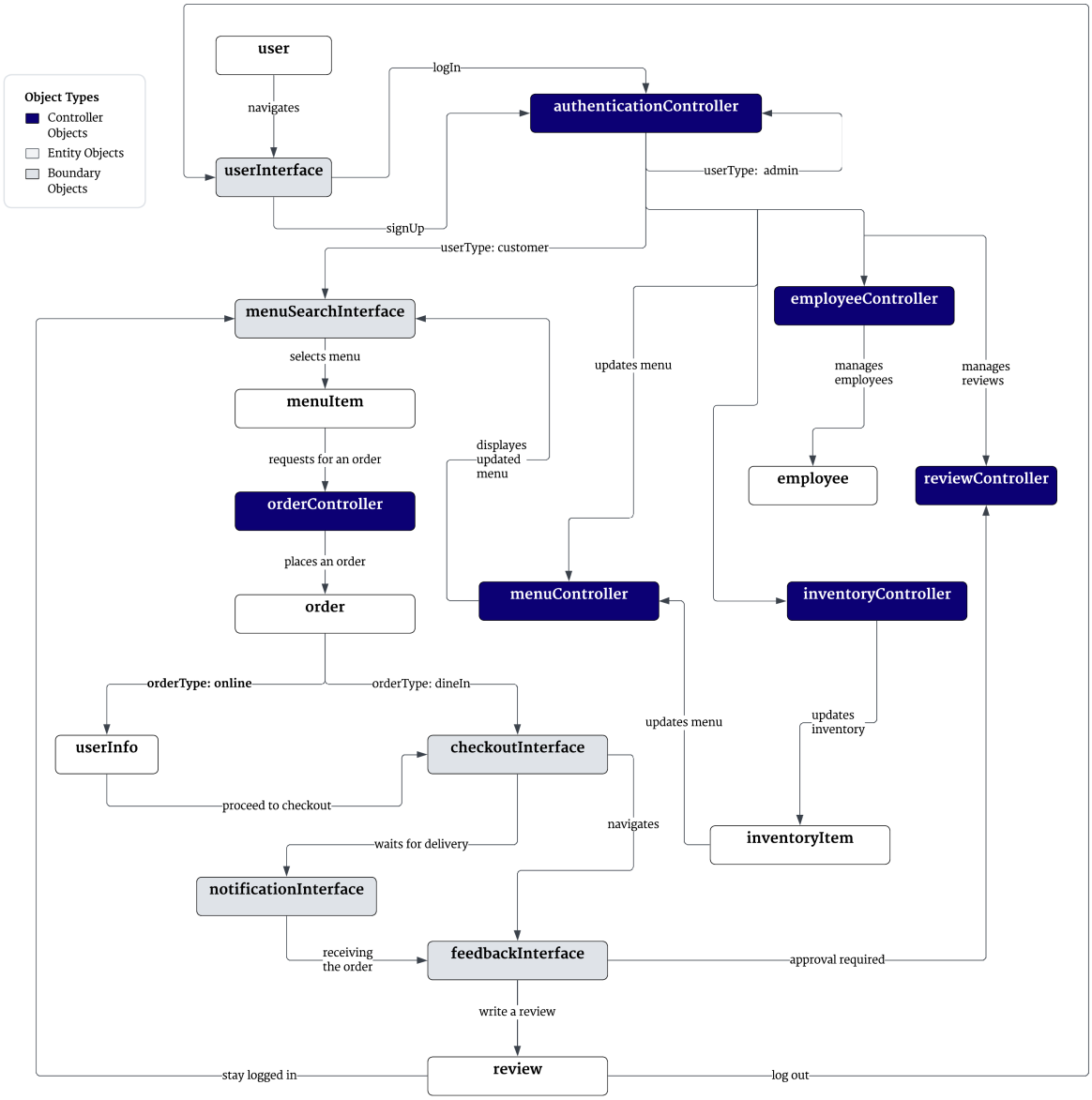

Figure 1: Object Collaboration Diagram

5 Subsystem Decomposition

5.1. Complete Package Diagram

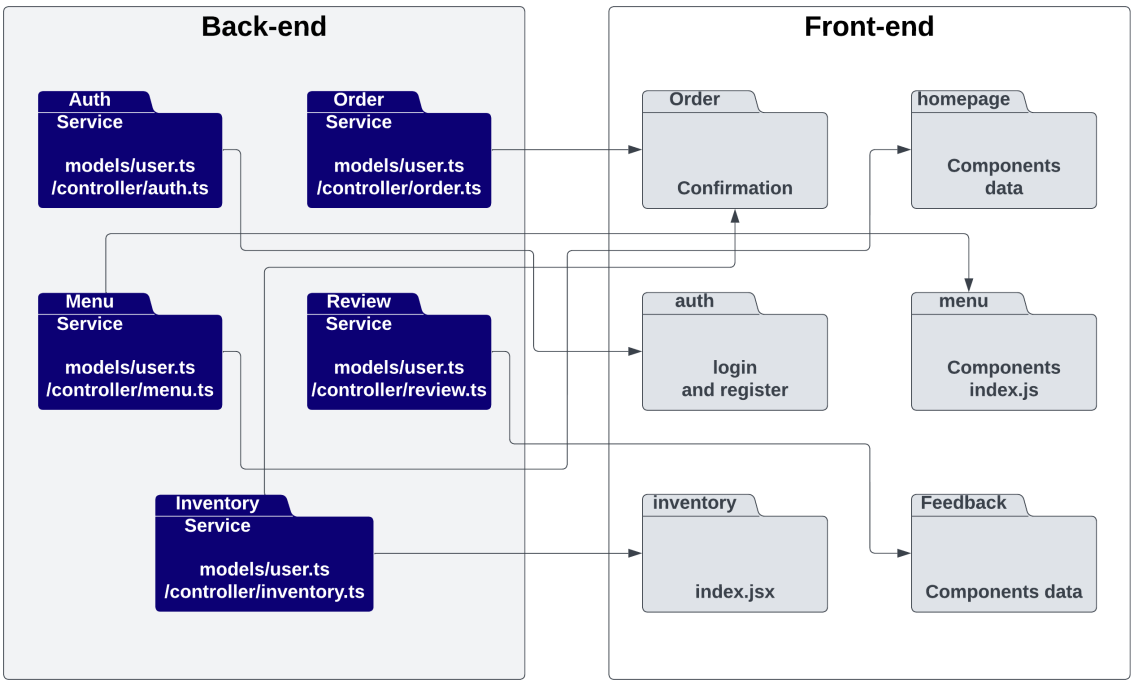

Figure 2: Complete Package Diagram

5.2. System Detail Description

5.2.1. Menu Search

The menu search service is dedicated to a user for searching his/her preferred menu items available in the canteen.

- **menuItem:** This entity represents features of the selected menu, with attributes like itemID, itemName, category, base price, extra ingredient price, description, and an image.
- **menuSearchController:** This controller manages the process of menu searching. It has methods to search menu items from the search bar and display them to the user.

The Menu Search module provides a well-defined interface menuSearchInterface for interacting with the search functionality. Other modules can use these interfaces to perform searches without needing to know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

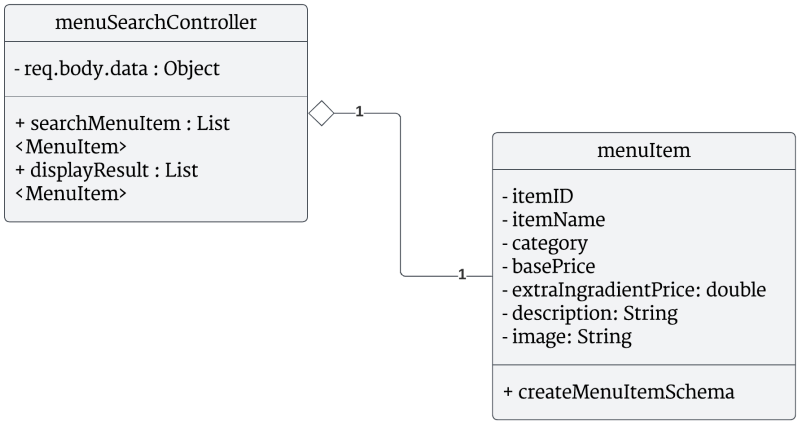

Figure 3: Subsystem: Menu Search

5.2.2. Placing an Order

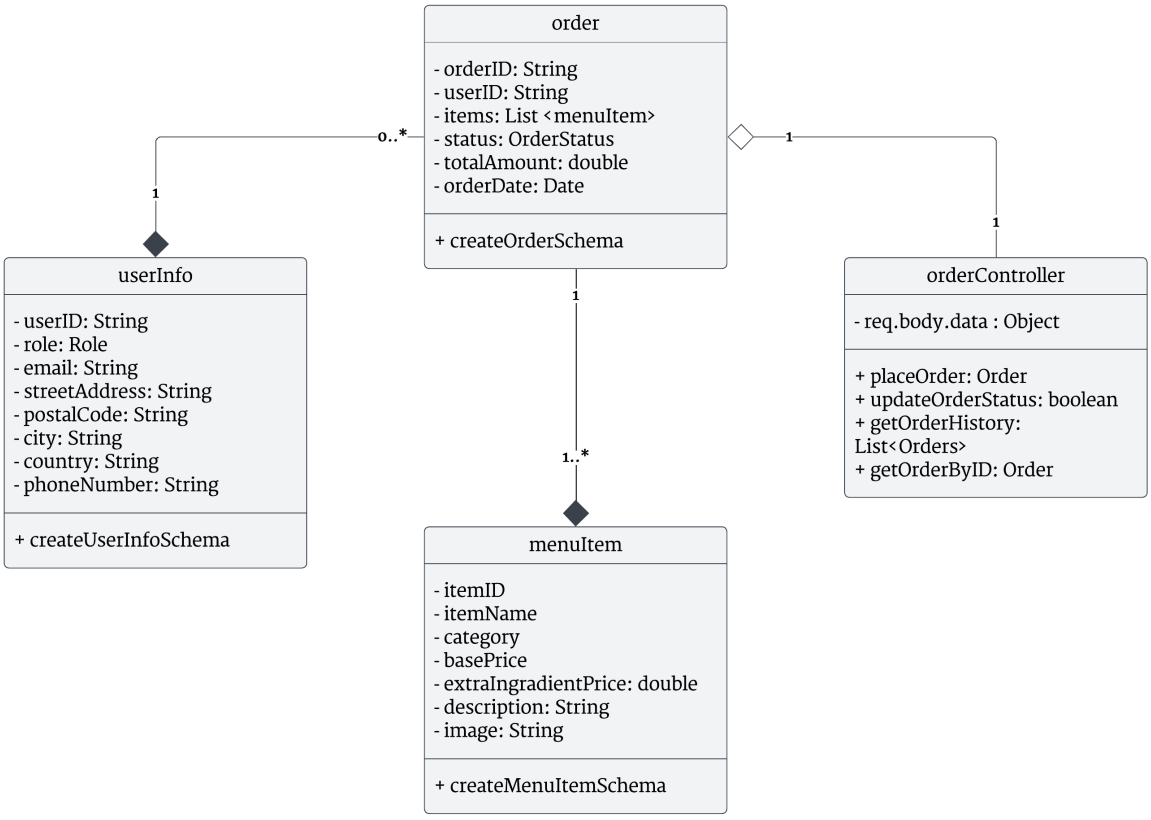

Figure 4: Subsystem: Placing an Order

The diagram illustrates a subsystem for placing an order. It consists of three main entities and a controller:

- **userInfo:** This entity is designed to manage user information within the system.
- **order:** This entity captures all the necessary details of an order.
- **menuItem:** This entity represents features of the selected menu, with attributes like itemID, itemName, category, base price, extra ingredient price, description, and an image.
- **orderController:** This controller manages the order process. It has methods to place an order, update the order status, retrieve order history, and get details of an order by its ID.

The order placing module provides a well-defined interface checkOutInterface for placing the order. Other modules can use these interfaces to perform checkout without needing to know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

5.2.3. Writing a Review

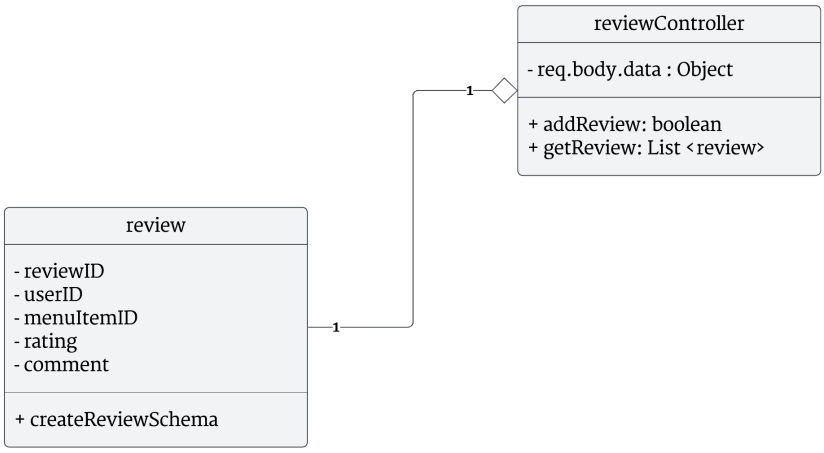

Figure 5: Subsystem: Writing a Review

The review service is responsible for a user writing a review with proper approval.

- **review:** This entity represents a review with all its attributes to identify it uniquely.
- **reviewController:** This controller manages the verification and approval of a review writing process.

The review service module provides a well-defined interface feedbackInterface for placing the order. Other modules can use these interfaces to write and read reviews without needing to know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

5.2.4. Menu Management

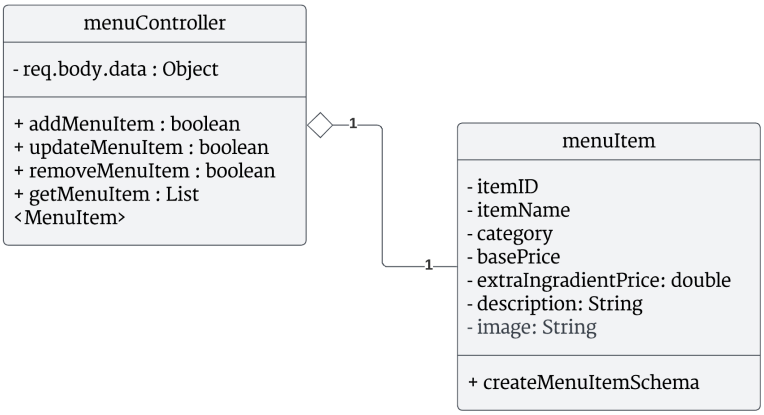

Figure 6: Subsystem: Menu Management

The menu management service allows the admin to manage and update all the menu items coherently.

- **menuItem:** This entity represents a menu with all its attributes to identify it uniquely.
- **menuController:** Through this controller, the admin can handle the management of all the menu items of the canteen.

The menu management service module provides a well-defined interface menuManagementInterface for managing the menu items. Other modules can use these interfaces to write and read reviews without needing to know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

5.2.5. Inventory Management

The inventory management service allows the admin to update the quantities of the menu items daily so that the chefs and customers are aware of the available menu items and ingredients.

- **inventoryItem:** This entity is similar to the menuItem. All the instances here consist of the same itemID from the menuItem; however, here each item is associated with its available quantity.
- **inventoryController:** Through this controller, the admin can update the quantity of all the existing items and add new items as well.

The review service module provides a well-defined interface inventoryInterface for managing the inventory. Other modules can use these interfaces to write and read reviews without needing to

know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

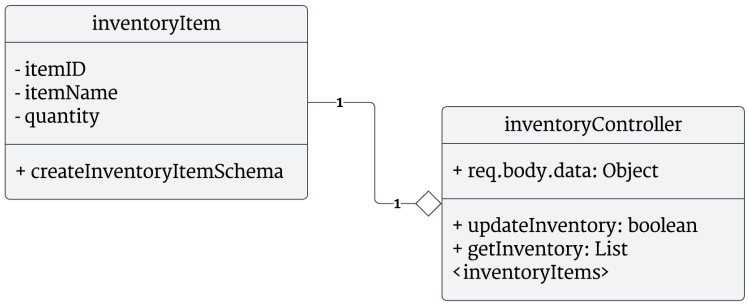

Figure 7: Subsystem: Inventory Management

5.2.6. Employee Management

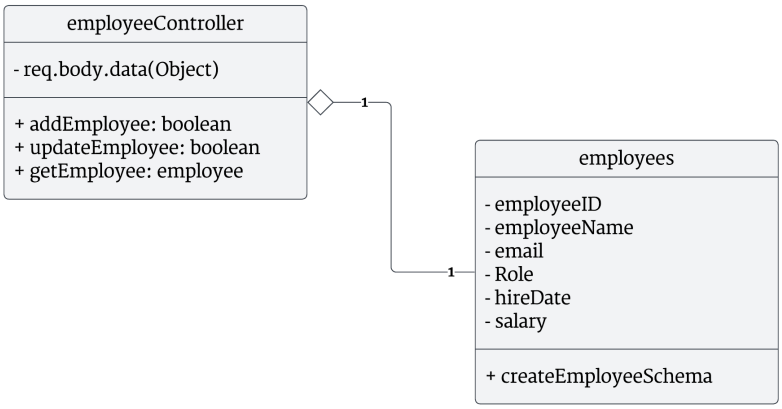

Figure 8: Subsystem: Employee Management

The employee management service allows the admin to manage the information of all the employees.

- **employees:** Every instance of this entity represents an employee associated with the canteen.
- **employeeController:** Through this controller, the admin can manage, update, and add employees related to the canteen.

The employee managing service module provides a well-defined interface employeeManagementInterface for managing the employees. Other modules can use these interfaces to write and read reviews without needing to know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

5.2.7. Authentication

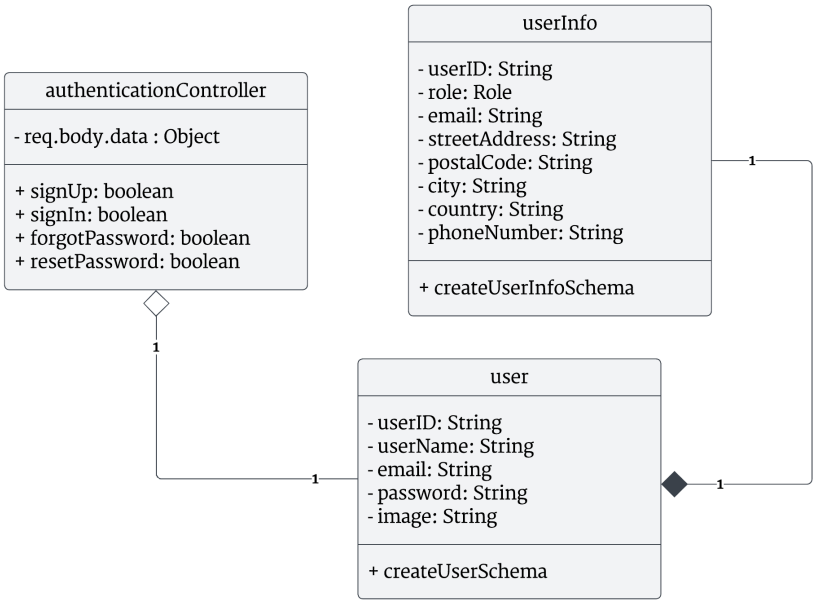

Figure 9: Subsystem: Authentication

The authentication service allows all the users and employees to be authenticated every time they log in and place an order. It uses the userInfo entity to perform this service.

- **user:** Every user is assigned a username and a password with this entity after a signup. Using these, they can log in.
- **userInfo:** During the sign up phase, a user has to give all his/her necessary info whether they are customers or employees, their address, and phone number for placing and delivering an order. Using this info, the authentication service authenticates the users during login and placing an order.
- **authenticationController:** Through this controller, the admin performs the whole process of authentication.

The authentication service module provides a well-defined interface authenticationInterface for verifying the users. Other modules can use these interfaces to write and read reviews without needing to know the implementation details. It does not have any dependencies on other subsystems for its core functionality. There exists a low coupling between this service and other ones.

6 Data Design

6.1. Data Description

The information domain of the Gitgrub system is transformed into data structures through the use of databases and data models. Each microservice has its data model that defines the structure of the data it stores and processes.

For example, in the authentication service has a User data model that stores user information including email and hashed passwords.

Similarly, The ordering Service has an Order data model that defines the structure of the data related to ordering, including fields such as Order id, User id, Ordered Item, and payment information.

The MenuItem has a menuItem data model that defines the structure of the data related to the menu items, Including menuID, Size, base price, extra ingredient price, etc.

The userInfo has a UserInfo data model that defines the structure of the data related to the userInfo, like role, email, joining date, Each of these microservices uses a MongoDB database to store its data. MongoDB was chosen because it is a flexible and scalable NoSQL database that can handle the varied data structures required by each microservice

6.2. Data Dictionary

Entity	**Type**	**Description**
Base Price	Number	The base price of a item
Employee	Object	A registered Employee of the GitGrub Canteen
Extra Ingredient Price	Number	The amount a customer should pay for the add-ons
Hire Date	Date	The date when an employee joins the job
Inventory Item	Object	The items stored in the inventory
Items	List(MenuItem)	The items presented in a single order
ItemId	String	An unique number for identifying the item
MenuItem	Object	A food item available in the canteen.
Order	Object	An order made by a customer.
Order Date	Date	In which date a customer placed an order
OrderId	String	An unique identifier for each of the order
Rating	Number	The point a customer gives to a customer out of a fixed number

Entity	Type	Description
Review	Object	The feedback a customer gives to the canteen
Role	String	It tells whether the user is a customer or an employee
Sales Report	Object	A report detailing the sales figures for a given time period
Status	OrderStatus	The status of the order: prepared, delivered, pending or canceled
totalAmount	Number	The total Amount a customer has to pay for the order
User	Object	A registered customer of the GitGrub Canteen
User Id	String	The unique identifier for a user or employee
UserInfo	Object	The information of a registered user or employee

Table 20: Data Dictionary

6.3. Entity Relationship Diagram

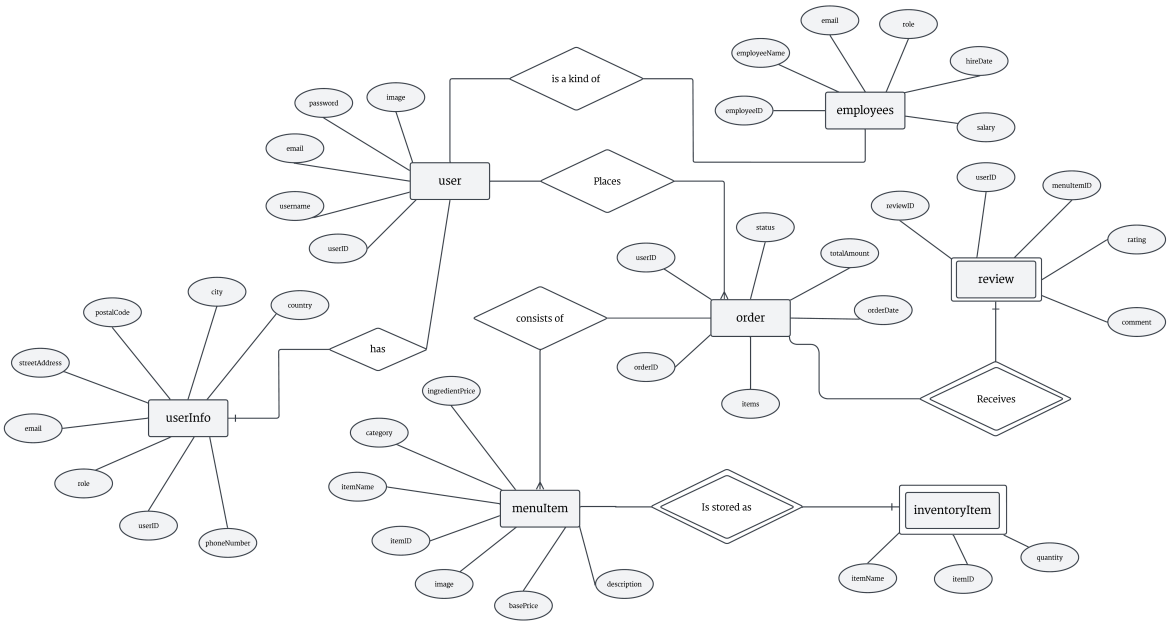

Figure 10: Entity Relationship Diagram

7 User Requirement and Component Traceability Matrix

	customer	admin	Order	Review	Inventory
UR1: User Management	X		X	X	X
UR2: Employee Management	X			X	X
UR3: Placing Order				X	
UR4: Order management	X			X	
UR5: Delivering Order				X	X
UR6: Menu management	X		X	X	
UR7: Inventory management	X		X	X	
UR8: Transaction management	X			X	X
UR9: Feedback System			X		X

Table 21: User Requirement and Component Traceability Matrix