Gisma University of Applied Sciences

Assessment Submission Form

Student Number (If this is group work, please include the student numbers of all group participants)	1. Pavan Kakadiya GH1039356 2. Nikhil Ratilalbhai Sojitra GH1039510
Assessment Title	Restaurant Management System
Module Code	M604
Module Title	Advanced Programming
Module Tutor	Dr. Mazhar Hammed
Date Submitted	27 May 2025

Declaration of Authorship

I declare that all material in this assessment is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

I fully understand that the unacknowledged inclusion of another person's writings or ideas or works in this work may be considered plagiarism and that, should a formal investigation process confirms the allegation, I would be subject to the penalties associated with plagiarism, as per GISMA Business School, University of Applied Sciences' regulations for academic misconduct.

Signed…Pavan Kakadiya, Nikhil Ratilalbhai Sojitra…. Date …27 May 2025……

GitHub Link : https://github.com/pavankakadiya/Restaurant_Management

VideoLink :https://drive.google.com/file/d/12Oe8fvhpiKuYAcp55QZEHZxRlo0qtsVv/view?usp=sharing

INDEX

Abstract

The Restaurant Management System is a web-based application designed to streamline restaurant operations, enhance service efficiency, and improve customer experience. The system enables waiters to log in, select tables, take customer orders, and generate bills with ease. Once a table is occupied, it remains locked until the bill is generated, ensuring proper tracking of table usage and preventing order conflicts. The system securely stores key data such as wWaiter ID, Table number, Order details, Bill amount for accurate record-keeping and reporting.

Developed using Spring Boot (Java 21), Maven 3.4.3, and an SQL database, the project follows a RESTful API architecture to enable seamless data handling and integration with potential frontend interfaces. The system not only minimizes manual errors in order processing and billing but also optimizes workflow, ensuring quick and accurate service. By automating restaurant management tasks, this system enhances efficiency, reduces operational costs, and provides a seamless dining experience for customers.

Introduction

Managing a restaurant involves a variety of tasks, including taking customer orders, assigning tables, preparing food, generating bills, and ensuring smooth service. In many restaurants, these processes are still handled manually, leading to delays, errors, and inefficiencies. Waiters often have to keep track of multiple tables and orders, increasing the chances of mistakes, such as missing or incorrect orders. Similarly, billing errors can result in disputes, causing dissatisfaction among customers. To address these challenges, an automated Restaurant Management System is essential for improving service efficiency and customer experience.

This system is designed to replace traditional manual processes with a digital and structured approach that simplifies restaurant operations. By providing a platform where waiters can log in, select tables, take orders, and generate bills seamlessly, the system reduces the reliance on paper-based records and minimizes human errors. One of its key features is table management, which ensures that once a table is assigned to a waiter, it remains locked until the bill is finalized. This prevents confusion and ensures that multiple waiters do not take orders for the same table.

Additionally, the system stores essential details such as waiter ID, table number, order history, billing time, and transaction records, which helps in tracking sales and analyzing restaurant performance. The automation of billing and order processing allows restaurant staff to focus more on customer service rather than manual paperwork. The system is built using Spring Boot (Java 21), Maven 3.4.3, and an SQL database, making it a robust and scalable solution for restaurant businesses of all sizes.

Objectives

The Restaurant Management System aims to achieve the following:

- Increase Efficiency – Reduce waiting time by enabling waiters to take and process orders faster.
- Improve Order Accuracy – Minimize errors in order-taking and food preparation by ensuring that all orders are digitally recorded.
- Automate Table Allocation – Prevent table conflicts and improve restaurant seating management.

- Enhance Billing Process – Ensure accurate bill calculations and seamless payment processing.
- Better Data Management – Keep detailed transaction records for future reference, sales tracking, and analysis.
- Scalability & Future Expansion – Allow easy integration of additional features like online reservations, mobile ordering, or digital payments.
- Improve Customer Experience – Ensure faster service, reducing long wait times and enhancing overall satisfaction.
- Minimize Paperwork – Shift to a digital system, reducing reliance on paper-based order slips and receipts.
- Support Business Growth – Help restaurant owners manage operations more effectively, leading to higher profitability.

With this system in place, restaurants can ensure smooth operations, better customer service, and a more organized workflow, ultimately leading to greater customer satisfaction and business success.

Literature Review

The restaurant industry has seen significant technological advancements over the years, with many businesses adopting digital solutions to streamline operations. Traditional restaurant management relied heavily on manual order-taking, paper-based billing, and human coordination, which often led to inefficiencies, order mix-ups, and longer wait times for customers. To address these issues, various restaurant management systems have been developed, each with unique features and functionalities.

Existing Systems

1. Point of Sale (POS) Systems
   a) POS systems are widely used in restaurants for billing and payment processing.
   b) Some advanced POS solutions integrate inventory management, customer relationship management (CRM), and employee scheduling.
   c) Examples: Square POS, Toast POS, and Lightspeed POS.

2. Online Food Ordering Systems
   a) Platforms like UberEats, DoorDash, and Zomato allow customers to place orders online.
   b) Many restaurants integrate with these services to expand their reach and improve sales.
   c) Limitations: These systems focus on online orders but do not optimize in-house restaurant operations like table management and real-time order tracking for waiters.

3. Standalone Restaurant Management Applications
   a) Some restaurants use custom-built applications for internal management, offering features like order tracking, table reservations, and waiter assignments.
   b) Examples: OpenTable for reservations, ResDiary for table management, and Toast for restaurant operations.
   c) Limitations: Many of these solutions require high customization, and small restaurants may find them costly or complex to implement.

- **How Our System Stands Out:**

Compared to existing systems, our Restaurant Management System offers:

- ✓ A unified solution for in-house restaurant operations, combining table management, waiter assignment, order processing, and billing.
- ✓ A structured database for tracking orders and billing.
- ✓ Customizable and scalable architecture, allowing future expansion to integrate online orders, digital payments, and customer feedback management.
- ✓ A user-friendly interface for easy adoption, ensuring that restaurant staff can quickly learn and utilize the system without extensive training.

By addressing the limitations of traditional and existing systems, our Restaurant Management System provides a cost-effective, efficient, and scalable solution tailored for modern restaurants looking to improve operational workflow and customer service.

Competitive Analysis

The restaurant management industry has several digital solutions, including POS systems, online ordering platforms, and full-scale restaurant management systems. Here's a brief comparison of these solutions against our Restaurant Management System:

- **POS Systems** (e.g., Toast, Square): These systems manage orders and payments but lack table tracking and real-time table selection. They are typically costly, with expensive monthly fees.
- **Online Ordering Systems** (e.g., UberEats, Zomato): While excellent for online orders, these platforms do not support in-house dining operations, table management, or waiter logins. They often charge high commission fees.
- **Full Restaurant Management Systems** (e.g., OpenTable, Lightspeed): These provide comprehensive features like reservations and table management but come with high setup costs and are typically not customizable for smaller restaurants.

• **Our System**: Our system combines the best features of the above solutions at an affordable price. It provides real-time table management, waiter logins, order handling, and detailed billing, all while being customizable and scalable for growing restaurants.

Methodology

The Restaurant Management System follows a structured methodology to ensure efficiency, scalability, and reliability. The system is built using Spring Boot (Java 21), Maven 3.4.3, and an SQL database, implementing a Restful API architecture. The development process includes requirement analysis, system design, implementation, testing, and deployment.

System Architecture

The system is based on a two-tier architecture, which consists of:

1. Application Layer (Backend – Spring Boot APIs)
    - Handles user authentication, table management, order processing, and billing.
    - Implements RESTful APIs for communication between the frontend and database.

2. Database Layer (SQL Database)

- Stores waiter details, table statuses, order records, and billing transactions.
- Ensures data integrity, preventing order mix-ups and tracking restaurant performance.

System Modules & Implementation

The Restaurant Management System consists of several core modules:

1. User Authentication & Role Management
   - Waiters log in using username and password.
   - The system verifies credentials before granting access.

2. Table Management
   - The system has 15 tables, and each table is locked once assigned to a waiter.
   - A table is unlocked only after the bill is generated to prevent duplicate assignments.

3. Order Management
   - Waiters can add items to a table's order from the restaurant's menu.
   - Orders are stored in the database and remain active until billing.

4. Billing System
   - Calculates the total bill amount based on the order details.
   - Generates a unique ID for each completed bill.
   - Once the bill is finalized, the table becomes available for new customers.

5. Data Storage & Logging
   - Stores waiter ID, table number, total amount, and bill ID.
   - Ensures data consistency and security.

Development Tools & Technologies

- Backend: Spring Boot (Java 21), Maven 3.4.3
- Database: MySQL (Username: root, Password: Gisma@123)
- API Framework: RESTful Web Services
- Security: User authentication with password encryption
- Version Control: GitHub for code management

System Workflow

- Waiter logs into the system → Access granted based on credentials.
- Waiter selects a table → The system locks the table to prevent duplicate orders.
- Waiter takes an order → Items are added to the table's bill.
- Customer requests a bill → The system calculates the total amount and generates a unique bill ID.
- Payment is completed → The table is unlocked and available for new customers.

Testing & Debugging

- Unit Testing: Each module is tested individually to ensure it functions correctly.
- Integration Testing: The API endpoints are tested to verify smooth communication between components.

- Performance Testing: The system is checked under multiple waiter and order transactions to ensure stability.

Deployment & Future Enhancements

- The system can be deployed on a local server or cloud platform (AWS, Heroku, or Google Cloud).
- Future enhancements may include:
  - Mobile app integration for easier order-taking.
  - Digital payment gateway for seamless transactions.
  - Online table reservation system for customers.

By following this structured methodology, the Restaurant Management System ensures smooth operations, minimizes human errors, and provides a scalable solution for restaurant management.

Project Structure

Explanation of Core Components

1. controller/ (API Endpoints)
   - Handles all HTTP requests and responses.
   - Example controllers in your project:
     - DishController.java – Manages menu items (dishes).
     - LoginController.java – Handles user authentication.
     - OrderController.java – Processes order-related operations.
     - TableController.java – Manages restaurant table assignments.
     - SignUp.java – Handles new waiter or user registrations.

2. entity/ (Database Models)
   - Defines database tables using JPA annotations.
   - Example entity classes in your project:
     - BillResponse.java – Stores billing transaction details.
     - Dish.java – Represents dishes available in the restaurant.
     - Order.java – Defines order details (items, quantity, price).

3. repository/ (Data Access Layer)
   - Interfaces for database operations using Spring Data JPA.
   - Example:
     - DishRepository.java – Fetches and updates dishes in the database.
     - OrderRepository.java – Manages order storage.

4. service/ (Business Logic Layer)
   - Implements the logic for handling business operations.
   - Example:
     - OrderService.java – Handles order processing and validation.
     - BillingService.java – Manages payment and bill generation.

5. dto/ (Data Transfer Objects)
   - Used to transfer data between different layers without exposing entity details.
   - Example: OrderDTO.java, BillDTO.java.

6. config/ (Security & Configurations)
- Stores Spring Security settings and configurations for authentication.

7. utils/ (Utility Classes)
- Contains helper methods such as formatting dates, logging, and validation functions.

Configuration & Properties

The application.properties file configures database connectivity and other application settings.
properties
CopyEdit
# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/restaurant_db
spring.datasource.username=root
spring.datasource.password=Gisma@123
spring.jpa.hibernate.ddl-auto=update

# Server Port
server.port=8080

Execution Workflow

1. Run the Application
- Execute RestaurantManagementApplication.java to start the Spring Boot server.
- The server runs at http://localhost:8080/.

2. Access API Endpoints
- GET /api/orders→ To see all orders
- GET /api/orders/1 → To see Particular orders by their ID
- GET /api/ordersiitems→ To see all orders Dishes
- GET /api/dishes→ To see Menu of Items
- GET /api/tables→ To see all table data

- POST /api/SignUp → User authentication, User already exist
- POST /api/login → User authentication, Add new user
- POST /api/booktabel?tablenumber= 1 → To book a table manually
- POST /api/freetabel?tablenumber= 1 → To free a table manually

- DELETE/api/deteleuser → Enter parameters and delete a user from user table

- UPDATE/api/updateuser?username= " "&password=" "&newpassword=" "

3. Database Operations
- Uses MySQL for data storage.
- Tables for waiters, orders, tables, and bills are automatically managed by JPA.

Future Enhancements

- Frontend Integration – Develop a React.js UI for better user experience.
- Mobile App Compatibility – Create an Android/iOS app for waiters.
- Online Payment Gateway – Integrate digital payment options like PayPal.

Technology Stack

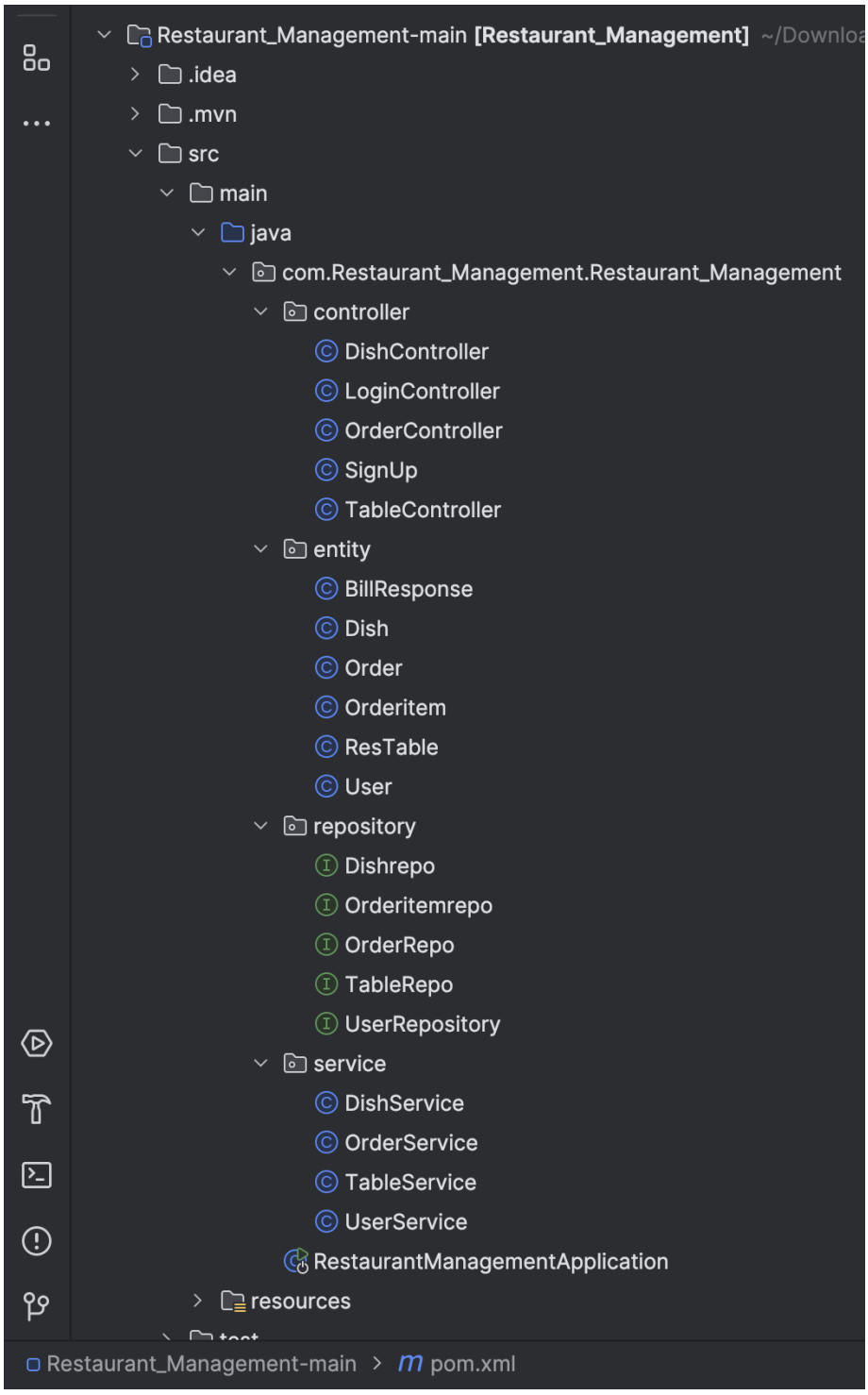
Explanation of Core Components

1. controller/ (API Endpoints)
   - Handles all HTTP requests and responses.
   - Example controllers in your project:
     - DishController.java – Manages menu items (dishes).
     - LoginController.java – Handles user authentication.
     - OrderController.java – Processes order-related operations.
     - TableController.java – Manages restaurant table assignments.
     - SignUp.java – Handles new waiter or user registrations.

2. entity/ (Database Models)
   - Defines database tables using JPA annotations.
   - Example entity classes in your project:
     - BillResponse.java – Stores billing transaction details.
     - Dish.java – Represents dishes available in the restaurant.
     - Order.java – Defines order details (items, quantity, price).
     - OrderItem,.java
     - ResTable.java
     - User.java

3. repository/ (Data Access Layer)
   - Interfaces for database operations using Spring Data JPA.
   - Example:
     - DishRepository.java – Fetches and updates dishes in the database.
     - OrderRepository.java – Manages order storage.
     - Orderitemsrepo.java
     - TableRepo.java
     - UserRepo.java

4. service/ (Business Logic Layer)
   - Implements the logic for handling business operations.
   - Example:
     - OrderService.java – Handles order processing and validation.
     - TableService.java – Manages payment and bill generation.
     - UserService.java
     - DishService.java



5. dto/ (Data Transfer Objects)
   - Used to transfer data between different layers without exposing entity details.
   - Example: OrderDTO.java, BillDTO.java.

6. config/ (Security & Configurations)
   - Stores Spring Security settings and configurations for authentication.

7. utils/ (Utility Classes)
   - Contains helper methods such as formatting dates, logging, and validation functions.

Database Design

The **Restaurant Management System** stores data in a relational SQL database that ensures efficient and reliable data storage, management, and retrieval. The system is designed to handle key aspects such as orders, waiters, tables, and dishes in a restaurant setting.

Database Overview:

The system employs the MySQL database management system to store the data and maintain relationships between entities like waiters, tables, orders, and dishes. Below is an outline of the core tables and their relationships:

1. **Table: waiters**
   - **Purpose**: Stores information about the waiters, including their login credentials and metadata.
   - **Key Fields**:
     - id: Primary key, uniquely identifies each waiter.
     - username, email, password: Used for waiter authentication.
     - created_at, updated_at: Timestamp for tracking record creation and updates.
2. **Table: tables**
   - **Purpose**: Represents the tables available in the restaurant.
   - **Key Fields**:
     - id: Primary key, uniquely identifies each table.
     - tablenumber: The physical table number (e.g., Table 1, Table 2, etc.).
     - capacity: The number of customers the table can seat.
     - status: Indicates whether the table is available (0) or occupied (1).
3. **Table: dish**
   - **Purpose**: Stores information about the menu dishes.
   - **Key Fields**:
     - dish_id: Primary key, uniquely identifies each dish.
     - dish_name: The name of the dish (e.g., Pizza, Burger).
     - price: The price of the dish.
4. **Table: orders**
   - **Purpose**: Stores order details linked to specific waiters and tables.
   - **Key Fields**:
     - order_id: Primary key, uniquely identifies each order.
     - waiter_id: Foreign key referencing the waiter who took the order.
     - table_id: Foreign key referencing the table where the order was placed.
     - order_start_time, bill_payment_time: Timestamps for tracking the order's lifecycle.
5. **Table: order_items**
   - **Purpose**: Stores the individual items ordered at a table.
   - **Key Fields**:
     - item_id: Primary key, uniquely identifies each order item.

- order_id: Foreign key referencing the order.
- dish_id: Foreign key referencing the dish ordered.
- quantity: The number of servings for each dish.
- price: The price of the dish at the time of the order.

Relationships Between Tables:

1. **Waiter to Orders**:
   - A waiter can have multiple orders. This is a **one-to-many** relationship between waiters and orders.
2. **Table to Orders**:
   - A table can have multiple orders throughout the day. This is a **one-to-many** relationship between tables and orders.
3. **Order to Order Items**:
   - An order can contain multiple items. This is a **one-to-many** relationship between orders and order_items.
4. **Dish to Order Items**:

- A dish can appear in multiple order items. This is a **one-to-many** relationship between dish and order_items.

Data Integrity and Constraints:

- **Foreign Keys** ensure consistency and integrity between related tables. For example, each order item must correspond to a valid dish and a valid order.
- **Unique Constraints** ensure that waiters' emails are unique, preventing duplication.
- **Default Values** such as status in the tables table help to automate operations, ensuring smooth workflow.
- **Timestamp Fields** (created_at, updated_at, order_start_time, bill_payment_time) provide historical tracking of the operations.

Indexing for Performance:

- Primary keys are automatically indexed.
- Foreign key columns (waiter_id, table_id, dish_id) can be indexed for faster lookups.

Future Enhancements in Database Design:

1. **Add Online Reservation System:**
   - Add a reservations table to store customer reservation data.
2. **Integration of Payment Systems:**
   - Add a payment table to store payment transactions associated with orders.
3. **Advanced Reporting:**
   - Add a reports table or views to generate daily, weekly, or monthly performance reports, including total sales, table occupancy, etc.

Conclusion

A vital tool for optimizing restaurant operations, increasing productivity, and improving the patron experience is the “Restaurant Management System”. By providing a centralized platform for servers to track tables, handle orders, and create invoices, the project is intended to satisfy the demands of contemporary dining establishments. To guarantee a seamless operation and peak performance, the system incorporates a number of features, including order management, table availability, billing, and data storage.

Key Achievements:
Centralized Management: By centralizing all required resources and data, the system streamlines processes and lowers the possibility of mistakes.

Efficient Billing: The system guarantees that the restaurant can handle orders and payments more efficiently by automating bill generation and offering real-time updates on table availability.

Database Design: Scalability, consistency, and integrity are guaranteed by the relational database design. The system can manage higher data volumes while maintaining performance by using indexing, foreign keys, and unique constraints.

User Experience: Waiters are given an easy-to-use platform to complete their tasks by the straightforward yet efficient user interface, which eventually improves customer satisfaction.

Challenges Faced::
Complexity of Integration: A lot of work may be needed to ensure smooth communication between the client and the server because the project entails integrating frontend interfaces with backend APIs.

Data Accuracy: It is essential to preserve data accuracy when there is concurrent access (for example, when several waiters are interacting with the same table). This can be addressed in part by implementing transaction management and locking mechanisms in the database.

User authentication and authorization: Using secure authentication techniques is necessary to guarantee adequate security for waiter login and data management, which introduces another level of complexity.

Conclusion:
In conclusion, the Restaurant Management System is a dependable and expandable solution that addresses the main operational issues that restaurants encounter. It greatly enhances workflow and lowers human error by automating order management, table tracking, and billing. The system can develop into a complete platform that can support a variety of restaurant operations with the addition of new features and enhancements.

To ensure this project's long-term success, it will be crucial to keep an eye on user feedback, system performance, and future scalability as it develops. The system can develop into a useful tool for contemporary restaurant management with continued improvements to the current design, which offers a solid foundation.