**Kingston University, BSc (Hons) (top-up)**

**Coursework Coversheet**

**Draft Coursework – Subject to Moderation**

**Part 1 - To Remain with the Assignment after Marking**

|  |  |
| --- | --- |
| **Student ID: E174226** | **Student Name: J A Shasheen Kaushika** |
| **Module Code:** | **Module Name: PROGRAMMING III - PATTERNS AND ALGORITHMS** |
| **Assignment number:** | **ESoft Module Leader:** |
| **Date set:** | **Date due: 2024/12/28** |

**Guidelines for the Submission of Coursework**

1. Print this coversheet and securely attach both pages to your assignment. You can help us ensure work is marked more quickly by submitting at the specified location for your module. You are advised to keep a copy of every assignment.

2. Coursework deadlines are strictly enforced by the University.

3. You should not leave the handing in of work until the last minute. Once an assignment has been submitted it cannot be submitted again.

**Academic Misconduct**: **Plagiarism** and/or **collusion** constitute **academic misconduct** under the University's Academic Regulations. Examples of academic misconduct in coursework: making available your work to other students; presenting work produced in collaboration with other students as your own (unless an explicit assessment requirement); submitting work, taken from sources that are not properly referenced, as your own. By printing and submitting this coversheet with your coursework you are confirming that the work is your own.

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | ESoft Office Use Only:  Date stamp: work received | | |  | | --- | |  | |

**Kingston University, BSc (Hons) (top-up)**

**Coursework Coversheet**

**Part 2 – Student Feedback**

|  |  |
| --- | --- |
| **Student ID: E174226** | **Student Name: J A Shasheen Kaushika** |
| **Module Code:** | **Module Name: PROGRAMMING III - PATTERNS AND ALGORITHMS** |
| **Assignment number:** | **ESoft Module Leader:** |
| **Date set:** | **Date due:** |

|  |
| --- |
| Strengths (areas with well-developed answers) |

|  |
| --- |
| Weaknesses (areas with room for improvement) |

|  |
| --- |
| Additional Comments |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **ESoft Module Lecturer:** | **Provisional mark as %:** |  |
| **ESoft Module Marker:** | **Date marked:** |

|  |  |
| --- | --- |
| CI6115 Programming III - Patterns and Algorithms | |
| Weighting | 50% |
| Description | The aim of this coursework is to develop and document a software application using any Object-Oriented Programming (OOP) language and compatible tools of any version. The application can be either console-based or GUI-based and should address the specified case study. |
| Expected Deliverables | The submission must include the following three parts in a single document, along with all project files uploaded as a separate compressed folder:  **Class Diagram and Discussion**:   * A comprehensive class diagram for the proposed system. * Accompanying discussion that highlights the key design decisions and rationale behind them.   **Source Code**:   * Complete source code of the proposed system, demonstrating the use of:   1. Classes and objects   2. Object-Oriented Programming concepts   3. Data structures   4. Relevant algorithms and design patterns   **Test Cases:**   * A full set of test cases used to validate the system |

**Case Study**

You have been assigned the task of designing a Food Ordering System for a Pizza Shop. The pizza shop offers a variety of crusts, sauces, toppings, and customization options. Customers should be able to place orders for pickup or delivery. The system should enhance the overall user experience, allowing customers to create unique pizza combinations and track their orders.

**Requirements:**

1. Pizza Customization:
   * Allow customers to choose from a variety of crusts, sauces, toppings, and cheese options.
   * Implement a system that enables customers to create custom pizza combinations and name them.
2. Ordering Process:
   * Design an intuitive ordering process that guides customers through crust selection, sauces, toppings, quantity, and order review.
   * Support both pickup and delivery options. For delivery, integrate with a mapping service to provide accurate delivery estimates.
3. User Profiles and Favorites:
   * Implement user profiles where customers can save their favorite pizza combinations.
   * Allow users to reorder their favorite combinations with a single click.
4. Real-Time Order Tracking:
   * Integrate a real-time order tracking system that updates customers on the status of their pizza orders, including preparation and delivery stages.
   * Provide notifications for significant updates, such as the pizza being prepared or out for delivery.
5. Payment and Loyalty Program:
   * Design a payment processing system that supports credit cards, digital wallets, and possibly a loyalty program for repeat customers (No need to integrate an actual payment service. Just mock the procedure for this requirement.).
   * Implement a loyalty program where customers earn points for each purchase, leading to discounts or free items.
6. Seasonal Specials and Promotions:
   * Allow the shop to easily introduce and manage seasonal specials and promotions, such as discounts on certain toppings or pizza sizes during specific times of the year.
7. Feedback and Ratings:
   * Enable customers to provide feedback and ratings for each pizza order.
   * Use the feedback to improve service and showcase highly-rated combinations to other customers.

**Design Patterns to Apply:**

* Builder Pattern:

Apply the Builder pattern to construct complex orders with various customizations, ensuring a flexible and readable order creation process.

* Observer Pattern:

Use the Observer pattern to notify users about order status changes and provide real-time updates on the progress of their orders.

* Strategy Pattern:

Apply the Strategy pattern for payment processing, allowing the system to easily integrate new payment methods and promotions.

* Chain of Responsibility Pattern:

Implement the Chain of Responsibility pattern for processing order customization requests, allowing different handlers to manage specific customizations.

* State Pattern:

Utilize the State pattern to represent the different states of an order (placed, in preparation, out for delivery), making it easy to manage transitions and updates.

* Command Pattern:

Apply the Command pattern to represent user actions, such as placing an order or providing feedback, as objects that can be queued, undone, or logged.

* Decorator Pattern:

Use the Decorator pattern to enhance orders with additional features, such as extra toppings or special packaging.

Justify your choice of design patterns based on the specific requirements and how they contribute to the overall robustness and maintainability of the Pizza Ordering System. Marks will only be awarded if the above criteria comply.

**Further information and reminders:**

* **Class Diagram**: Use any standard UML tool to draw the class diagram. Ensure that the images are clear. Watermarks are acceptable if they are not obtrusive. **Each class name in the diagram must be prefixed with your Kingston University student ID, otherwise, marks will be deducted**.
* **Programming Language**: You may select any object-oriented programming language, but Java is preferable.
* **Frameworks**: Use frameworks with careful evaluation. If a framework generates all the classes for you, marks will be deducted.
* **User Interfaces**: User interfaces can be either Command Line Interfaces (CLI) or Graphical User Interfaces (GUI). Evaluate the user experience in terms of UI/UX for the chosen interface type. The choice of interface type will not impact grading; the focus should be on the quality of implementation.
* **Screenshots**: Include clear screenshots of the code and user interfaces, following the detailed instructions provided.

**Marking Scheme**

|  |  |  |
| --- | --- | --- |
| **Item** | | **Marks Allocation** |
| **Class Diagram (30)** | | |
| 1 | Identification of classes & objects | 15 |
| 2 | Use of associations effectively | 10 |
| 3 | Sensible naming of programmer-defined variables, classes, properties, and methods | 05 |
| **Software Implementation (70)** | | |
| 1 | Completion of functional modules | 25 |
| 2 | Effective use of object-oriented programming concepts | 05 |
| 3 | Effective use of data structures | 05 |
| 4 | Effective use of design patterns | 20 |
| 5 | Justification of the application and design pattern | 10 |
| 6 | Use of Test Cases | 05 |
| **Total** | | **100** |

**Academic Integrity:**

Academic integrity means demonstrating honest, moral behaviours when producing academic work. This involves acknowledging the work of others, giving appropriate credit to others where their ideas are presented as part of your work and the importance of producing work in your own voice. Contributions by artificial intelligence (AI) tools must be properly acknowledged. As part of a learning community students share ideas and develop new ones - you need to be able to interpret and present other people's ideas and combine these with your own when producing work.

**Plagiarism (including copying, self-plagiarism and collusion)**

The act of presenting the work of another person (or people) and/or content generated by artificial intelligence (AI) tools as your own without proper acknowledgement. This includes copying the work of another student or other students.

The University expects students to take responsibility for the security of their work (i.e. with written work, to ensure that other students do not get access to electronic or hard copy of the work). Failure to keep work secure may allow others to cheat and could result in an allegation of academic misconduct for students whose work have been copied, particularly if the origin of the work is in doubt.

**Self-plagiarism**

The act of presenting part or all of your work that has been previously submitted to meet the requirements of a different assessment, except where the nature of the assessment makes this permissible.

**Collusion**

The act, by two or more students of presenting a piece of work jointly without acknowledging the collaboration. This could include permitting or assisting another to present work that has been copied or paraphrased from your own work.

The University also defines collusion as the act of one student presenting a piece of work as their own independent work when the work was undertaken by a group. With group work, where individual members submit parts of the total assignment, each member of a group must take responsibility for checking the legitimacy of the work submitted in his/her name. If even part of the work is found to contain academic misconduct, penalties will normally be imposed on all group members equally.

**Purchasing or Commissioning**

The act of attempting to purchase or purchasing work for an assessment including, for example from the internet, or attempting to commission, or commissioning someone else to complete an assessment on your behalf.

The procedures for investigating suspected cases of academic misconduct are set out in Academic Regulations 6 Academic Integrity - Taught Courses 2023/24

**Acknowledging Generative AI in coursework**

Where generative AI has contributed to an assignment the following information should be included in the submission:

A statement on the use of generative AI as part of the assessment, including the extent of use, and how it was used as part of all stages in creating the final submission, e.g., including planning, and generating ideas. This should normally be provided at the end of a written assignment with the heading ‘Acknowledgement of AI Contribution’. For other assignment types, module staff will advise on how this should be done.

**You must meet all deadlines set. Failure to do so will result in a penalty.**

Work submitted late but within a week of the deadline will be capped at 40% and receive a grade of LP (Late Pass) unless it is not of a passing standard in which case it will receive a grade of LF (Late Fail). Work submitted beyond a week of the deadline without approval will get 0% with a grade of F0. If, however, you have a serious problem, which prevents you from, meeting the deadline you may be able to negotiate an extension in advance. In the first instance you should contact the module team for advice. However, any extension will need to be formally agreed by the Faculty via the Mitigating Circumstances process, your work will then be marked without penalty.



**Programming Patterns and Algorithms**

**E174226 – J A Shasheen Kaushika**

# Table of Content

# Table of figures

# Class Diagram

The class diagram represents a database schema for a pizza ordering system. It’s designed to handle key functionalities, such as managing products, customer orders, payments, and feedback. Each class (table) stores specific data, and relationships between classes are implemented using **foreign keys** to maintain data integrity.

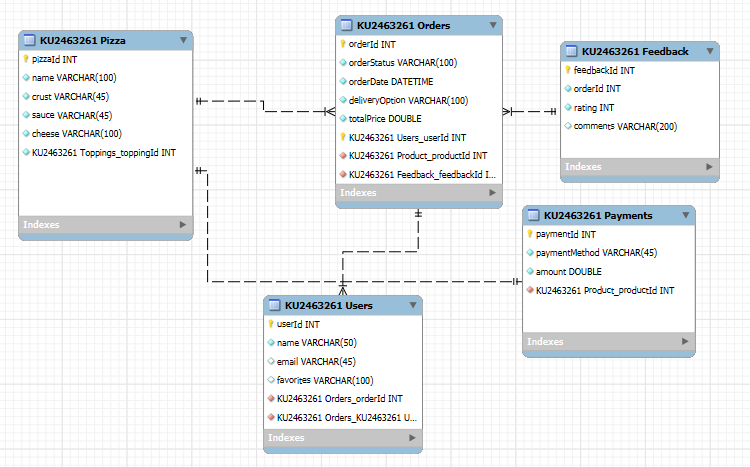


Figure 1 Class diagram

## Classes and Attributes

**A. KU2463261\_pizza**

* **Attributes:**

pizzaId: A unique number identifying a product, like an ID tag on a pizza.

name: The product's name, e.g., "Pepperoni Pizza" or "Veggie Delight."

crust: The type of crust (e.g., Thin Crust, Pan Crust).

sauce: The type of sauce (e.g., Tomato Sauce, Alfredo Sauce).

cheese: The type of cheese (e.g., Mozzarella, Parmesan).

Toppings\_toppingId: Links the product to a specific set of toppings (many-to-one relationship).

* **Purpose:**

Represents items available for customers to order.

Acts as a menu for the pizza shop.

**B. KU2463261\_Orders**

* **Attributes:**

orderId: Unique identifier for each order, like a receipt number.

orderStatus: Status of the order, such as "Pending," "Completed," or "Cancelled."

totalPrice: The total cost of the order, calculated from the product price .

Users\_userId: Links the order to a specific user (foreign key).

pizza\_pizzaId: Links the order to the product being purchased (foreign key).

Feedback\_feedbackId: Links to feedback provided for this order.

* **Purpose:**

Tracks customer orders.

Records all necessary details about the order, such as its status, customer information, and associated product.

**C. KU2463261\_Users**

* **Attributes:**

userId: Unique identifier for a user, like a customer ID.

name: Name of the customer.

email: Email address for communication and login.

* **Purpose:**

Represents users/customers of the system.

Stores personal details and preferences.

**D. KU2463261\_Payment**

* **Attributes:**

paymentId: Unique identifier for each payment transaction.

paymentMethod: How the payment was made (e.g., Credit Card, Cash, or Digital Wallet).

amount: The total amount paid for the order.

order\_orderId: Links the payment to the product being purchased (foreign key).

* **Purpose:**

Records payment transactions for each order.

Ensures every order is tracked with a successful payment.

**E. KU2463261\_Feedback**

* **Attributes:**

feedbackId: Unique identifier for feedback entries.

orderId: Links the feedback to a specific order (foreign key).

comments: Text feedback from the customer.

* **Purpose:**

Captures customer feedback about orders or products.

Helps improve services based on customer insights.

## Relationships Between Classes

**KU2463261\_Orders ↔ KU2463261\_Pizza**

* Type: Many-to-One
* Explanation: Multiple orders can include the same product (e.g., "Pepperoni Pizza").
* Real-World Example: Two customers can order the same pizza type.

**KU2463261\_Users ↔ KU2463261\_Orders**

* Type: One-to-Many
* Explanation: A single customer can place multiple orders over time.
* Real-World Example: John orders pizza today and another tomorrow.

**KU2463261\_Orders ↔ KU2463261\_Payment**

* Type: One-to-One
* Explanation: Every order corresponds to exactly one payment.
* Real-World Example: Once you place an order, you make a single payment for it.

**KU2463261\_Orders ↔ KU2463261\_Feedback**

* Type: One-to-One
* Explanation: Every order can have only one feedback entry.
* Real-World Example: After completing an order, a customer leaves a review or rating.

**KU2463261\_Payment ↔ KU2463261\_Pizza**

* Type: Many-to-One
* Explanation: A payment is associated with one product, but the product can appear in multiple payments.
* Real-World Example: If "Pepperoni Pizza" is ordered several times, it will be referenced in multiple payment records.

The class diagram represents a robust and scalable backend system tailored for PizzaCraft. It connects the shop's operations, ensuring smooth workflows, excellent customer experiences, and accurate tracking of all business activities. By automating these processes, PizzaCraft can focus on what they do best making delicious pizzas.

# Test cases

1. **Test Case for Order Placement**

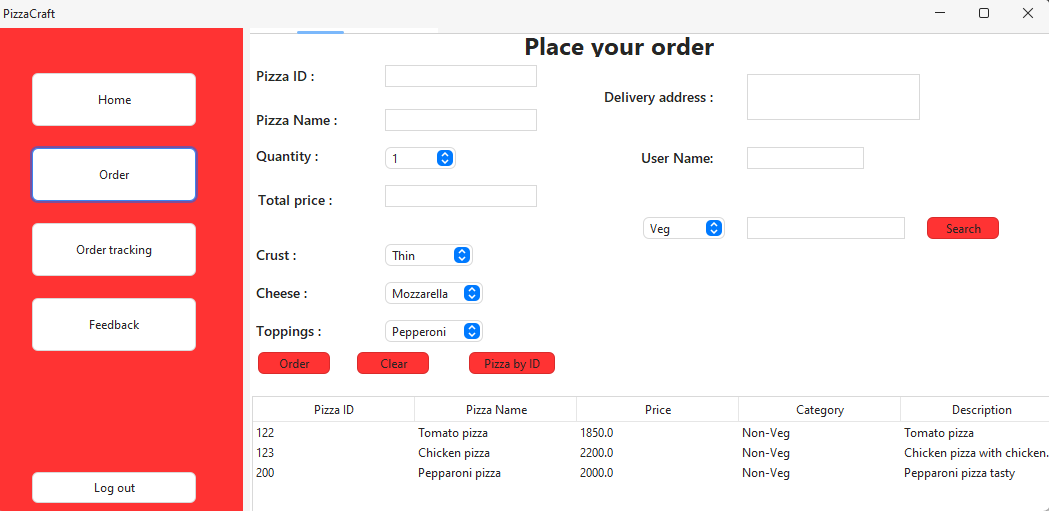
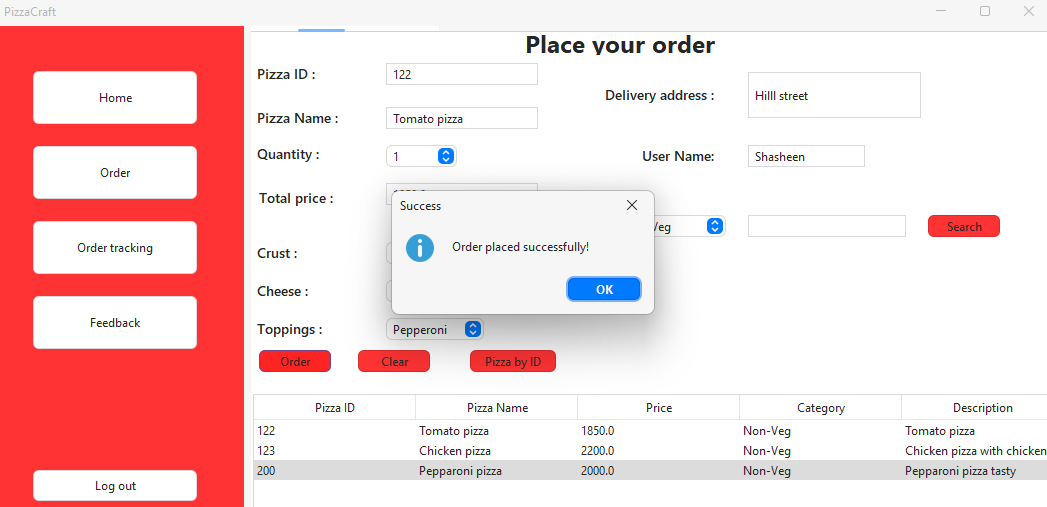
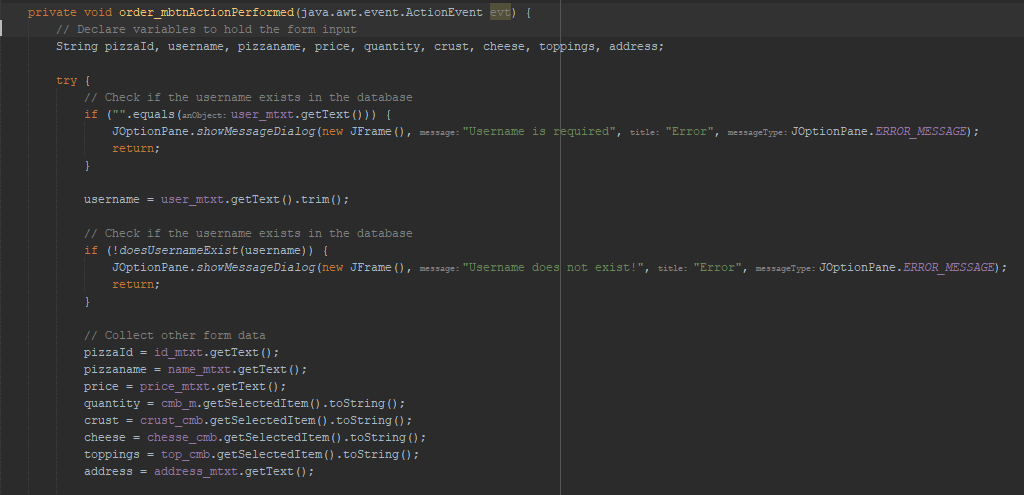


Figure 2 Order placing UI





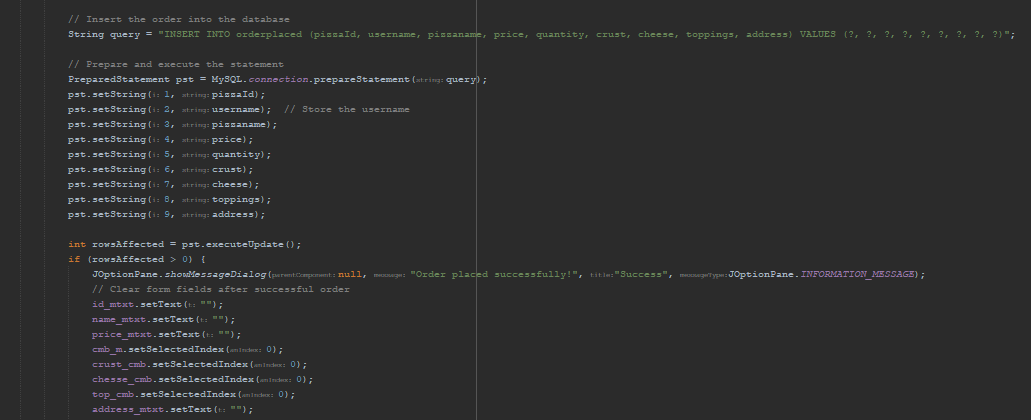
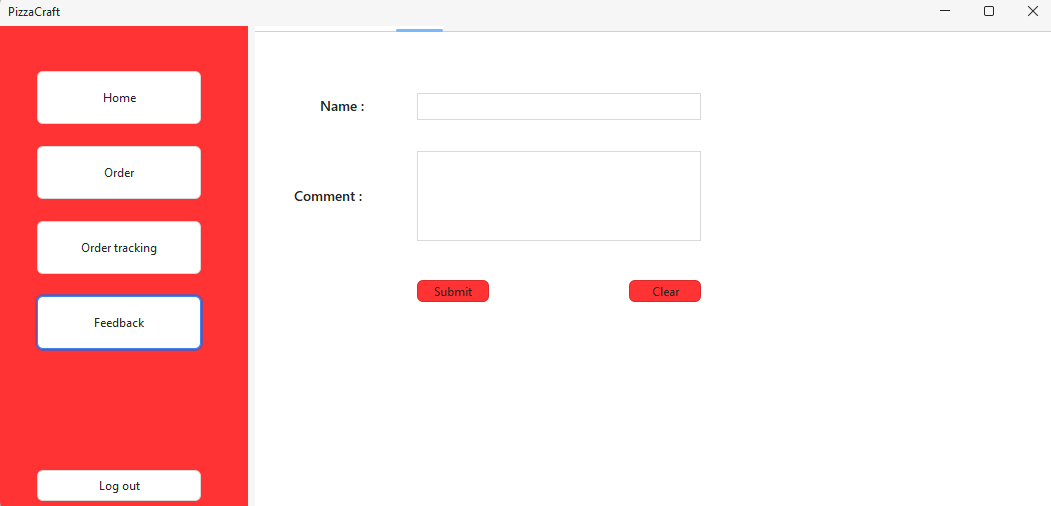


Figure 3 Order button

* **Test Case ID**: TC001
* **Test Case Name**: Validate order placement functionality
* **Module**: Orders
* **Preconditions**:
  1. User must be logged in.
  2. Product (pizza) must exist in the database.
* **Test Steps**:
  1. Log in with a valid username and password.
  2. Navigate to the menu and select a pizza.
  3. Click "Add to Cart."
  4. Proceed to the checkout and confirm the order.
* **Expected Results**:
  1. A new order entry is created in the KU2463261\_Orders table.
  2. The order status is set to "Pending."

1. **Test Case for Feedback Submission**

****

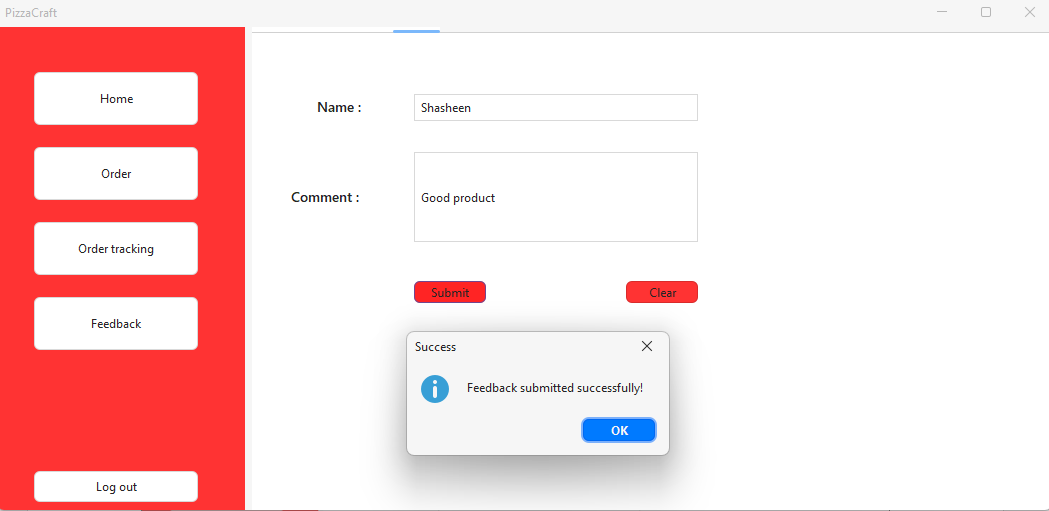
****

Figure 4 Feedback UI

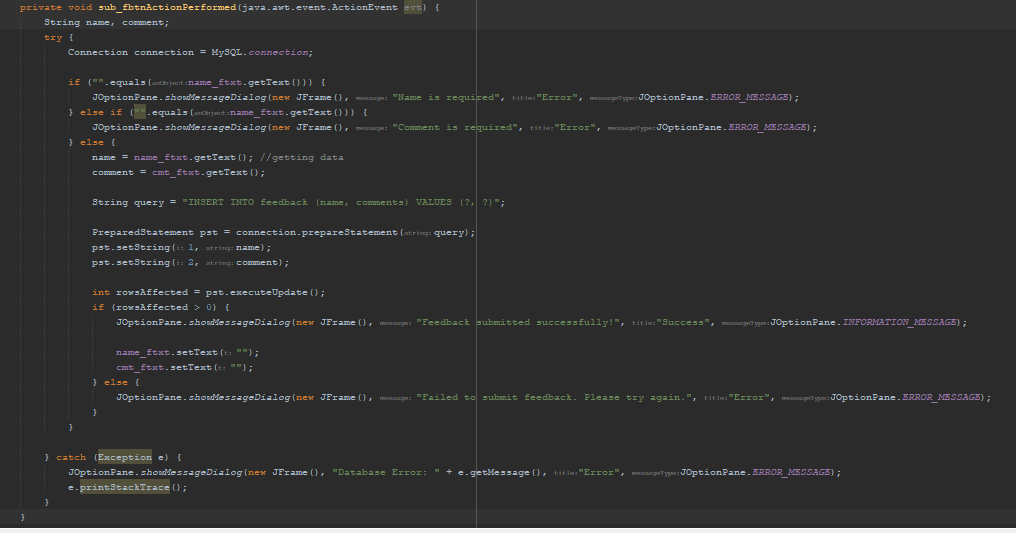


Figure 5 Submit button

* **Test Case ID**: TC002
* **Test Case Name**: Validate feedback submission functionality
* **Module**: Feedback
* **Preconditions**:

Order must exist and be marked as "Completed."

* **Test Steps**:

Navigate to the feedback page.

Select a completed order.

Enter feedback details (rating and comments).

Submit the feedback.

* **Expected Results**:

A new feedback entry is added to the KU2463261\_Feedback table.

The system confirms "Feedback Submitted Successfully."

1. **Test Case for User Login**

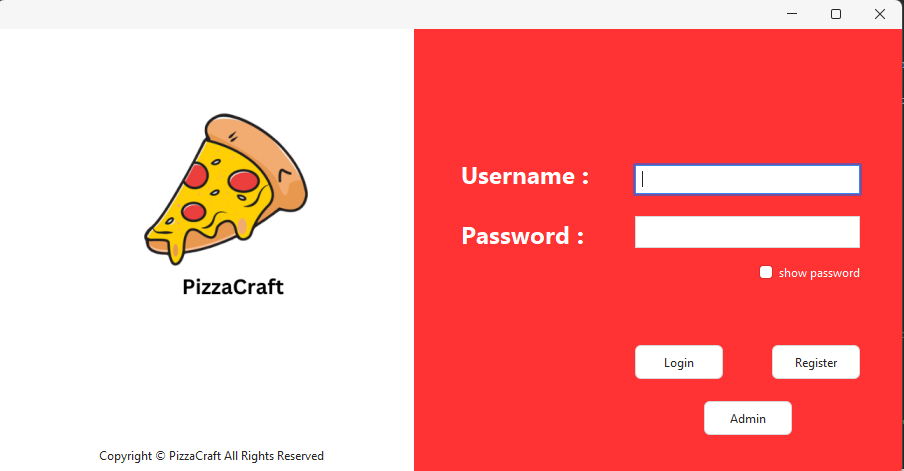
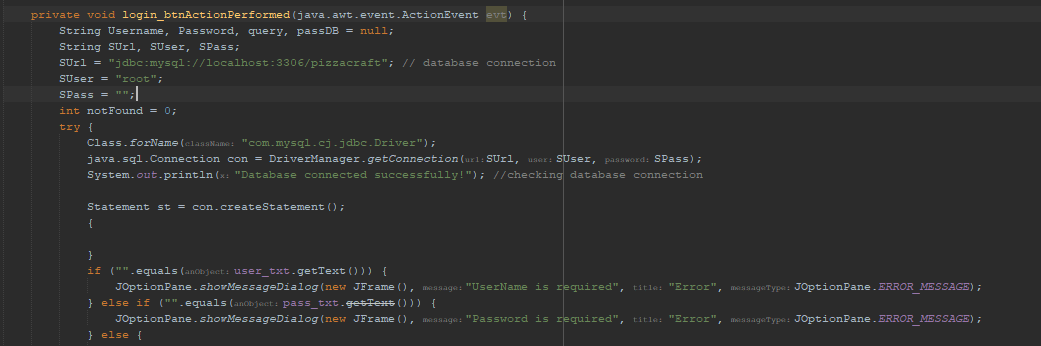


Figure 6 Login UI



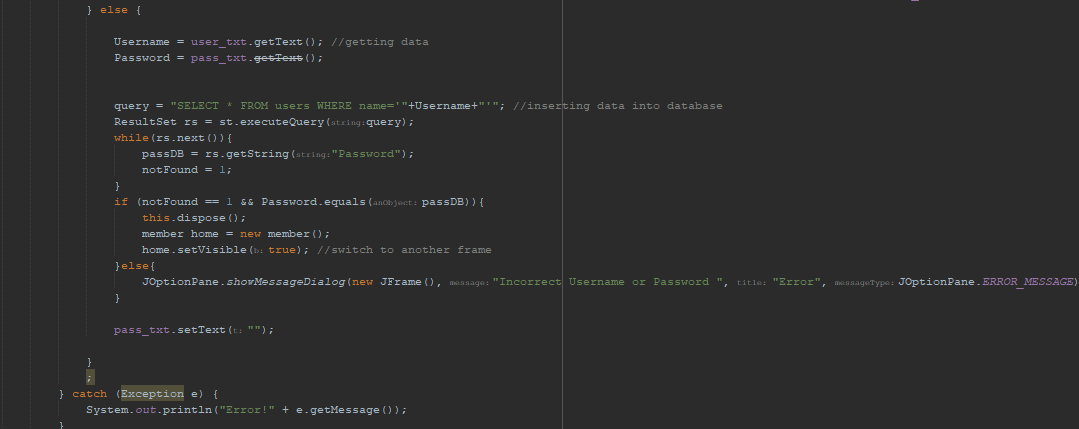


Figure 7 Login button

* **Test Case ID**: TC003
* **Test Case Name**: Validate user login functionality
* **Module**: User Authentication
* **Preconditions**:

1. User credentials must exist in the KU2463261\_Users table.

* **Test Steps**:

1. Open the application.
2. Enter a valid username and password.
3. Click "Login."

* **Expected Results**:

1. The user is redirected to the member page.

## Types of Testing

Include different types of testing in your assignment:

**A. Unit Testing**

* Tests individual components (e.g., database queries for fetching orders).  
  **Example:** Ensure the SQL query for fetching the latest order (ORDER BY order\_id DESC LIMIT 1) works as expected.

**B. Integration Testing**

* Tests the interaction between components (e.g., Orders and Payments).  
  **Example:** Validate that a payment updates the order status to "Paid."

**C. Functional Testing**

* Tests specific functionality (e.g., the payment form processes payments correctly).  
  **Example:** Ensure selecting a payment method updates the payment table accurately.

**D. User Acceptance Testing (UAT)**

* Verifies that the system meets user requirements.  
  **Example:** Ensure that users can place an order, make a payment, and leave feedback without errors.

# Implementation

## Pizza Customization

Pizza customization allows users to build their own pizzas according to their preferences. Users can select:

* **Base options**: Crust type, size, and sauce.
* **Toppings**: A list of available toppings stored in the, which connects to KU2463261\_Pizza through a foreign key.
* **Cheese options**: Users can select the type and amount of cheese.

**How it works**:

1. The user selects a pizza base.
2. The system fetches available toppings from the database.
3. Upon selection, the system calculates the price dynamically based on the selected toppings.

## Ordering Process

The ordering process is handled through the KU2463261\_Orders table:

* Users select a pizza or customized order and confirm the quantity.
* Delivery or pickup options are chosen by the user.
* The system generates an entry in the Orders table, assigning a unique orderId.
* The totalPrice field is calculated and stored in the database.

**Flow**:

1. User places an order.
2. Order details are saved, and the status is set to "Pending."
3. The user is redirected to the payment page for further action.

## Real-Time Order Tracking

Real-time order tracking keeps users informed about their order's status, leveraging the orderStatus field in the KU2463261\_Orders table.

* Status values: "Pending," "In Progress," "Out for Delivery," "Delivered."
* Updates occur as the kitchen or delivery staff modify the order's progress.

**How it works**:

1. The system periodically updates the order's status.
2. The user views the status on the tracking page, which fetches live data from the Orders table.

# Payment

Payment processing and loyalty rewards are managed in the KU2463261\_Payment table.

* **Payment Options**: Users can choose between Credit/Debit Cards, or Digital wallet.

**Flow**:

1. After confirming the order, the user selects a payment method.
2. Payment details are processed, and a record is saved in the Payment table.

## Feedback and Ratings

The feedback and ratings module allows users to review their experience and rate completed orders.

* Data is stored in the KU2463261\_Feedback table, linked to the Orders table.
* Users can provide:

Written comments.

**Flow**:

1. Feedback is saved in the Feedback table, linked by orderId.

**Benefits**:

* Admins can view user feedback to improve services.
* High-rated items can be promoted.

# Design Patterns

## Builder Pattern

The Builder Pattern is used for **pizza customization**. This allows users to create a complex pizza object step by step with various options like crust, sauce, cheese, and toppings.

* A Orderbutton class handles the construction of the pizza object.
* Users specify their preferences (e.g., crust type, toppings), and the builder assembles the pizza accordingly.

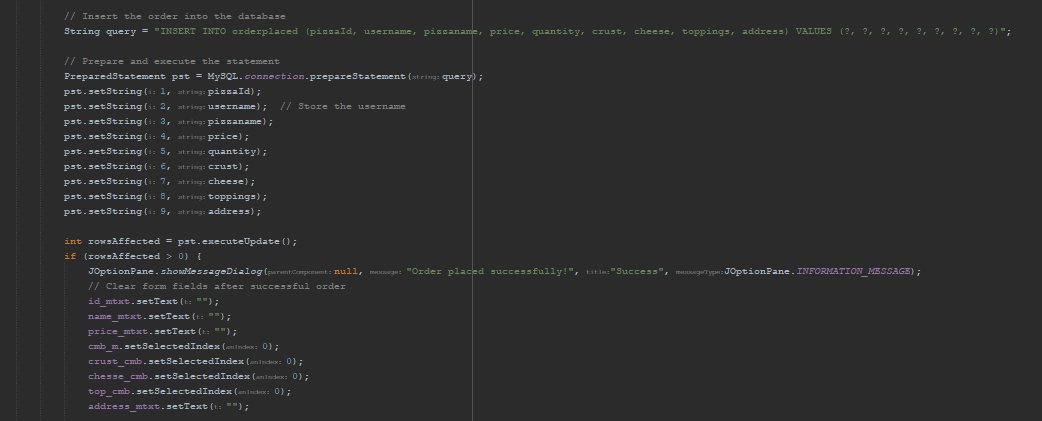


Figure 8 order pizza

## Chain of Responsibility Pattern

The Chain of Responsibility Pattern is applied to the **order validation process**. Each step in the validation chain (e.g., checking inventory, verifying user details) is handled by a separate handler.

**How it works**:

* Each handler processes a specific step in the validation and passes the request to the next handler if necessary.
* Allows flexible reordering or addition of validation steps.
* Enhances code readability and maintainability.

# Justification of Application