DOCUMENTATION

ASSIGNMENT 3

STUDENT NAME: CORNOC ROLAND NICHOLAS

GROUP: 30423

# CONTENTS

[CONTENTS 2](#_Toc166798691)

[1. Assignment Objective 3](#_Toc166798692)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 4](#_Toc166798693)

[3. Design 8](#_Toc166798694)

[4. Implementation 10](#_Toc166798695)

[5. Conclusions 13](#_Toc166798696)

[6. Bibliography 14](#_Toc166798697)

# Assignment Objective

**Objective**:

Design and implement an efficient and user-friendly application for managing client orders in a warehouse environment.

**Sub-Objectives**:

1. **Problem Analysis and Requirements Identification:**

* Analyze the current manual process of managing products, clients, and orders.
* Identify the shortcomings and challenges faced by warehouse staff.
* Determine the specific requirements of the application to address these challenges effectively.

1. **Designing the Orders Management Application:**

* Develop a comprehensive plan for the application's architecture and functionality.
* Define the user interface elements for ease of use and accessibility.
* Create a scalable and modular design to accommodate future updates and expansions.

1. **Implementation of the Orders Management Application:**

* Translate the design into functional code using appropriate programming languages and frameworks.
* Implement features such as adding new clients, adding new products, and processing orders.
* Ensure the application's reliability, security, and performance during the development phase.

1. **Testing the Orders Management Application:**

* Conduct rigorous testing to validate the functionality and usability of the application.
* Perform unit tests to verify individual components' correctness.
* Conduct integration tests to ensure seamless interaction between different modules.
* Gather feedback from warehouse staff and stakeholders to identify any potential issues or improvements.
* By achieving these sub-objectives, the main objective of developing an efficient and user-friendly order management application can be realized. Each sub-objective plays a crucial role in ensuring the successful implementation and adoption of the application within the warehouse environment.

# Problem Analysis, Modeling, Scenarios, Use Cases

**Functional Requirements:**

Functional requirements outline what the system or software should do. In the context of the order management application for a warehouse, functional requirements specify the actions and capabilities the application must provide to its users. Here are some examples of functional requirements for this application:

1. **Adding New Clients**: The application should allow authorized employees to add new clients to the system. This includes entering client details such as name, contact information, and billing address.
2. **Adding New Products**: Authorized employees should be able to add new products to the system. This involves entering product information such as name, description, price, and current stock level.
3. **Managing Orders**: The application should facilitate the management of orders placed by clients. This includes functionalities such as:
4. **Creating new orders**.
5. **Tracking current stock levels of products**.

* Automatically updating stock levels when new orders are processed or when new products are added.
* Generating alerts or notifications when stock levels are low.
* Client Communication: The application should support communication with clients regarding their orders.

1. **Sending order confirmation emails to clients upon order placement.**

* Sending status updates or tracking information to clients as orders are processed and shipped.
* Providing a platform for clients to inquire about order status or request assistance.
* Reporting and Analytics: The application should provide reporting and analytics features to help warehouse managers make informed decisions. This includes functionalities such as:

1. **Generating reports on sales performance, order fulfillment rates, and inventory turnover.**

* Analyzing trends in client orders to forecast demand and plan inventory levels.
* Providing data visualization tools to present key metrics in an easy-to-understand format.
* These functional requirements define the core features and capabilities that the order management application should deliver to meet the needs of warehouse operations effectively. Each requirement contributes to streamlining processes, improving efficiency, and enhancing the
* overall management of products, clients, and orders within the warehouse environment.

**Non-functional Requirements**:

* **Usability**: The application should be intuitive and easy to use for warehouse staff with varying levels of technical expertise. It should feature a user-friendly interface with clear navigation and instructions.
* **Performance**: The application should respond promptly to user interactions and provide real-time updates on order status and inventory levels. It should be capable of handling multiple concurrent users without significant slowdowns.
* **Scalability**: The application should be scalable to accommodate growing data volumes and increasing numbers of users. It should be able to handle additional clients, products, and orders without degradation in performance.
* **Reliability**: The application should be reliable and available for use during warehouse operating hours. It should minimize downtime and system failures, ensuring uninterrupted access to critical functionalities.
* **Security**: The application should enforce robust security measures to protect sensitive data, such as client information and financial transactions. This includes implementing authentication mechanisms, role-based access control, encryption of data in transit and at rest, and regular security audits.
* **Compatibility**: The application should be compatible with various devices and operating systems commonly used by warehouse staff, including desktop computers, tablets, and mobile devices. It should support modern web browsers and mobile platforms.
* **Maintainability**: The application should be designed and implemented in a modular and well-structured manner to facilitate ease of maintenance and future updates. This includes documenting code, adhering to coding standards, and minimizing dependencies.
* **Data Integrity**: The application should ensure the integrity and consistency of data stored in the system. It should implement mechanisms to prevent data loss, corruption, or unauthorized modifications.

**Use Case Diagram**

The use case diagram provides a visual representation of the various interactions between users and the orders management application. It illustrates the different actions and functionalities accessible to users. This diagram serves as a blueprint for understanding the system's behavior and guiding the development process to ensure alignment with user requirements and expectations.

***O imagine care conține diagramă, text, cerc, linie

Descriere generată automat***

**Figure 1. Use Case Diagram**

**Use Case Scenarios:**

1. **Add/Edit/Delete Product:**

**Primary Actor**: Warehouse Staff (Employee)

**Description**: This use case involves adding, editing, or deleting a product from the system.

**Preconditions**: The employee must have appropriate permissions to perform product management tasks.

**Main Success Scenario:**

* The employee selects the option to manage products.
* For adding a new product:
* The employee enters the details of the new product, such as name, description, price, and initial stock level.
* The system validates the input data and adds the new product to the inventory.
* For editing an existing product:
* The employee selects the product to be edited from the list of available products.
* The system displays the product details and allows the employee to make changes.
* The employee saves the changes, and the system updates the product information.
* For deleting a product:
* The employee selects the product to be deleted from the list of available products.
* The system prompts for confirmation before deleting the product.
* The employee confirms the deletion, and the system removes the product from the inventory.

**Alternative Scenario:**

* If the input data is invalid or incomplete, the system displays error messages and prompts the employee to correct the information.

1. **Add/Edit/Delete Customer:**

**Primary Actor**: Warehouse Staff (Employee)

**Description**: This use case involves adding, editing, or deleting a customer (client) from the system.

**Preconditions**: The employee must have appropriate permissions to perform customer management tasks.

**Main Success Scenario**:

* The employee selects the option to manage customers.
* For adding a new customer:
* The employee enters the details of the new customer, such as name, contact information, and billing address.
* The system validates the input data and adds the new customer to the database.
* For editing an existing customer:
* The employee selects the customer to be edited from the list of existing customers.
* The system displays the customer details and allows the employee to make changes.
* The employee saves the changes, and the system updates the customer information.
* For deleting a customer:
* The employee selects the customer to be deleted from the list of existing customers.
* The system prompts for confirmation before deleting the customer.
* The employee confirms the deletion, and the system removes the customer from the database.

**Alternative Scenario:** If the input data is invalid or incomplete, the system displays error messages and prompts the employee to correct the information.

1. **Place Order:**

**Primary Actor**: Warehouse Staff (Employee) or Customer (Client)

**Description**: This use case involves placing an order for products.

**Preconditions**: The employee must have appropriate permissions to place orders, or the customer must be authenticated and authorized to place orders.

**Main Success Scenario:**

* The actor selects the option to place a new order.
* The actor adds products to the order by selecting them from the available inventory.
* The actor specifies the quantity of each product to be ordered.
* The actor provides additional order details, such as delivery address, shipping method, and payment information.
* The actor confirms the order.
* The system validates the order details and processes the order.
* The system updates the inventory to reflect the products' reserved quantities for the order.

**Alternative Scenario:** If there are insufficient quantities of certain products in the inventory, the system notifies the actor and allows them to adjust the order accordingly.

# Design

At the top level of the design structure, the system is organized into distinct packages, each representing a critical aspect of functionality. The Graphical User Interface (GUI) package serves as the focal point for user interaction, containing classes responsible for orchestrating the interface seamlessly. Within the Business Logic domain, one encounters the intellectual powerhouses, where classes employ their expertise in managing simulation configurations envisioned by the user. Meanwhile, the Data Models domain houses the core essence of the application, encapsulating classes that translate abstract concepts into tangible entities within the digital realm. Through this hierarchical arrangement, the design evolves into a cohesive symphony of skill and precision, with each package fulfilling its unique role in the intricate landscape of software engineering excellence.

O imagine care conține text, diagramă, linie, captură de ecran

Descriere generată automat

**Figure 2. UML package diagram**

O imagine care conține text, captură de ecran, proiectare

Descriere generată automat

**Figure 3. UML Class Diagram**

# Implementation

**AbstractDAO Class**

**Fields**:

* LOGGER: A logger instance for logging messages and exc­­­­eptions.
* type: A Class object representing the generic type parameter T of the DAO.

**Methods:**

* getAttributeNamesWithoutId(): Returns a list of attribute names of the generic type T, excluding the "id" attribute.
* createSelectQuery(String field): Constructs and returns a SELECT query string for retrieving records from the database table associated with the generic type T, based on a specific field value.
* createSelectQuery(): Constructs and returns a SELECT query string for retrieving all records from the database table associated with the generic type T.
* createInsertQuery(): Constructs and returns an INSERT query string for inserting records into the database table associated with the generic type T.
* createUpdateQuery(): Constructs and returns an UPDATE query string for updating records in the database table associated with the generic type T.
* createDeleteQuery(): Constructs and returns a DELETE query string for deleting records from the database table associated with the generic type T.
* findAll(): Retrieves all records from the database table associated with the generic type T and returns a list of corresponding objects.
* findById(int id): Retrieves a record from the database table associated with the generic type T based on the specified ID and returns the corresponding object.
* createObjects(ResultSet resultSet): Converts the ResultSet obtained from a database query into a list of objects of the generic type T.
* insert(T t): Inserts a new record represented by the object t into the database table associated with the generic type T and returns the inserted object.
* update(T t): Updates an existing record represented by the object t in the database table associated with the generic type T and returns the updated object.
* delete(T t): Deletes the record represented by the object t from the database table associated with the generic type T and returns a boolean indicating success or failure.

**CustomerBLL**

**Methods:**

* getCustomerAttributes(): Retrieves a list of attribute names of the Customer class (excluding the ID attribute) using the CustomerDAO.
* findCustomerById(int id): Finds a customer by their ID using the CustomerDAO.
* findAllCustomers(): Retrieves all customers using the CustomerDAO.
* insertCustomer(Customer customer): Inserts a new customer using the CustomerDAO after validating customer data with Validator methods.
* updateCustomer(Customer customer): Updates an existing customer using the CustomerDAO after validating customer data with Validator methods.
* deleteCustomer(Customer customer): Deletes a customer using the CustomerDAO.

**ProductBLL:**

**Methods:**

* getProductAttributes(): Retrieves a list of attribute names of the Product class (excluding the ID attribute) using the ProductDAO.
* findProductById(int id): Finds a product by its ID using the ProductDAO.
* findAllProducts(): Retrieves all products using the ProductDAO.
* insertProduct(Product product): Inserts a new product using the ProductDAO after validating product data with Validator methods.
* updateProduct(Product product): Updates an existing product using the ProductDAO after validating product data with Validator methods.
* deleteProduct(Product product): Deletes a product using the ProductDAO.

**OrdersBLL:**

**Methods:**

* findOrderById(int id): Finds an order by its ID using the OrdersDAO.
* findAllOrders(): Retrieves all orders using the OrdersDAO.
* insertOrder(Orders order): Inserts a new order using the OrdersDAO after validating order data with Validator methods.
* updateOrder(Orders order): Updates an existing order using the OrdersDAO after validating order data with Validator methods.
* deleteOrder(Orders order): Deletes an order using the OrdersDAO.

Each class serves as a layer of abstraction between the data access layer (DAO) and the user interface or client code. They encapsulate business logic and provide methods for interacting with the underlying data model while ensuring data integrity through validation.

# Conclusions

In conclusion, the development of the order management application has been a significant endeavor, yielding valuable insights and outcomes. Our main objective to design and implement an efficient system for managing client orders in a warehouse has been successfully achieved, albeit with some challenges along the way.

Throughout the project, we focused on implementing core functionalities such as product and customer management, order placement, and database interaction. These features were designed with scalability and usability in mind, ensuring that the application can accommodate future growth and evolving user needs.

We utilized a robust technology stack and development tools to build the application, leveraging best practices in software engineering and design patterns. The architecture of the system was carefully planned to promote modularity and maintainability, allowing for easy extension and adaptation in the future.

Testing played a crucial role in ensuring the reliability and robustness of the application. We employed various testing methodologies to identify and address defects, ultimately delivering a high-quality product to our users.

In terms of user experience, we prioritized intuitive interface design and smooth interaction flow. Feedback from user testing sessions helped us refine the UI/UX design and optimize the overall user experience.

Performance and scalability were key considerations throughout the development process. We conducted thorough performance testing to assess the application's responsiveness under different load conditions and implemented optimizations to improve efficiency.

Despite facing challenges during development, such as database connectivity issues and integration complexities, our team persevered and successfully delivered a functional and reliable order management solution.

Looking ahead, there are opportunities for further enhancement and refinement of the application. Future iterations could focus on incorporating additional features such as reporting and analytics, as well as enhancing security measures to protect sensitive data.

In conclusion, the order management application represents a significant achievement for our team, delivering tangible value to our organization and end-users alike. We are proud of the work we have accomplished and look forward to future opportunities for innovation and growth.

# Bibliography

<https://www.geeksforgeeks.org/what-is-object-relational-mapping-orm-in-dbms/>

https://www.geeksforgeeks.org/reflection-in-java/

Lecture Notes – Programming Techniques