ORDERS MANAGEMENT APPLICATION

STUDENT NAME: Flanja Tudor

GROUP: 30424

# CONTENTS

[1. Assignment Objective](#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases](#_Toc128043140)

[3. Design](#_Toc128043141)

[4. Implementation](#_Toc128043142)

[5. Conclusions](#_Toc128043143)

# Assignment Objective

The primary goal of the "Orders Management" software suite is to offer an effective mechanism for handling customer order operations within a warehouse setting. This cutting-edge application is designed to leverage the power and reliability of relational database systems to safely store and manage a variety of data. This includes critical details about the range of products available, comprehensive client information, and extensive order history.

A keen emphasis is placed on following a multi-layered architectural model to uphold a robust level of separation of responsibilities, ensuring a smoother functioning of individual components and ultimately bolstering system maintainability. This strategy not only ensures flexibility in modifying individual layers without impacting the whole system but also improves the manageability and scalability of the software.

This advanced application comprises several key components, each serving a unique purpose within the system. Model classes, which function as the cornerstone of the software's structure, serve to create a representation of the data in the system. Business logic classes act as the nerve center of the application, implementing the rules and procedures that drive the operations of the warehouse. The presentation classes cater to the interface requirements, providing the users with a clear, intuitive, and interactive graphical user interface. Finally, the data access classes streamline the communication between the system and the databases, managing all aspects of data storage, retrieval, and manipulation.

# Problem analysis

The challenge we face is the design and development of a software application with the capability to efficiently manage client orders within a warehouse environment. This task involves a meticulous analysis of the requirements, carefully identifying the vital data models, the underlying business logic, and the key components of the user interface.

The system is fundamentally built around three primary entities: the products, the clients, and the orders. Each of these entities plays a unique and critical role within the application and is associated with specific details that are integral to the system's operations.

The product entity, for instance, is associated with critical details such as the product's name, a comprehensive description that gives the user a clear understanding of the product, the quantity currently available in the warehouse, and the retail price. These details are key to ensuring the effective tracking, marketing, and sale of the products in the warehouse.

In the case of the client entity, it's important to store and manage information such as the client's name, their contact details (including phone numbers, email addresses, and other relevant contact data), and their physical address. This information is vital for maintaining robust client relationships and for the successful fulfillment and delivery of orders.

The order entity is another major component of the system and is typically associated with details such as the client who placed the order, the products involved in the order, the quantity of each product, and the current status of the order (for example, whether it's being processed, has been shipped, or has been delivered). These details are essential for tracking the order's progress through the system, ensuring that the client's needs are met and that the warehouse operates smoothly and efficiently.

# Design

To effectively tackle the aforementioned problem, a set of strategic design decisions has been made to ensure the efficient development and functionality of the application:

* Layered Architecture: The software will adhere to a layered architecture pattern, enabling modularity, promoting a clear separation of concerns, and facilitating easier maintenance in the long run. The architecture will comprise the following distinct layers: - Presentation Layer: This layer will be home to classes related to the graphical user interface (GUI), enabling seamless user interaction with the software. - Business Logic Layer: This layer will contain classes that encapsulate the core operations and logic of the application. - Data Access Layer: This layer will host classes that manage interactions with the database, handling all necessary database operations. - Model Layer: This layer will include classes that represent the data models within the application, mirroring the structure of the entities in the system.
* Model Classes: The application will feature model classes to mirror the entities within the system, including 'Product', 'Client', and 'Order'. Each of these classes will boast properties that correspond directly to the key attributes associated with each entity.
* Business Logic Classes: These classes will be the heart of the application's operations, encapsulating the business logic. They will manage vital operations such as processing orders, managing the inventory of products, and overseeing all aspects of client management.
* Presentation Classes: The presentation classes will be responsible for managing the user interface components. They provide a user-friendly GUI for users to interact with the system, coordinating with the business logic layer to perform operations and display the results in an intuitive manner.
* Data Access Classes: These classes form the bridge between the application and the underlying relational database. They provide methods that allow the application to interact with the database, managing all necessary operations such as retrieving data, inserting new entries, updating existing data, and deleting records as required.

# Implementation

The centerpiece of this application is the 'Generic' class from the Data Access (DA) Layer. This class serves as the primary conduit for establishing a connection with the database, utilizing the flexibility of generic arguments. It is designed to be adaptable and compatible with all the classes it interfaces with.

Upon instantiation, the 'Generic' class accepts an object of the type with which it was initiated. It extracts the constructor from the provided argument, proceeds to create a new object of that type, and subsequently establishes a connection with the database. Following these steps, it performs the desired operation, thereby facilitating an efficient interaction with the database.

Meanwhile, the Business Logic Layer (BLL) Package is responsible for managing the distinct operations associated with each class. It does so by defining and employing appropriate methods derived from the generic ones, customizing the operations based on the requirements of each class.

On the other hand, the Presentation Layer houses all the classes related to the graphical user interface (GUI). It encompasses four main windows, each dedicated to a specific model class. User interactions occur exclusively within this layer. Here, the GUI captures user input, executes the chosen operation, and displays relevant information from the database back to the user. In this way, the application ensures a smooth and intuitive user experience.

# Conclusions

# Gaining insight into a new approach for structuring projects has been an invaluable experience. This approach provides a clearer, more organized method for designing and implementing software applications, which can lead to improved efficiency and easier maintenance in the long term.

In addition, the knowledge of crafting generic functions has been particularly enlightening. These functions, by their very nature, add a layer of flexibility to the codebase, allowing functions to operate on data of various types while still maintaining type safety. The benefits of this are manifold, from encouraging code reuse to reducing the complexity of the code. This, in turn, contributes to the overall readability and maintainability of the application.

Overall, these learnings contribute to a deeper understanding of efficient software design and the importance of flexibility and modularity in code. With these skills, tackling diverse and complex programming challenges in the future becomes a more manageable endeavor.

# Bibliography

* <https://dsrl.eu/courses/pt/materials/PT2023_A3_S1.pdf>
* <https://dsrl.eu/courses/pt/materials/PT2023_A3_S2.pdf>