

# DOCUMENTATION

## ASSIGNMENT *Order Management System*

STUDENT NAME: Gozman-Pop Maria-Eliza  
GROUP: 30424

# CONTENTS

1. Assignment Objective .....	3
2. Problem Analysis, Modeling, Scenarios, Use Cases.....	3
3. Design .....	5
4. Implementation .....	9
5. Conclusions .....	15
6. Bibliography .....	15

# 1. Assignment Objective

**The main objective** is to manage orders in a system where clients can place orders for products. The system allows for creating, reading, updating, and deleting (CRUD) operations on clients, products, and orders. Additionally, it generates bills and tracks all operations.

## **Sub-objectives:**

- **Problem Analysis and Requirements Identification:** Analyze requirements for managing clients, products, and orders and define input formats, error handling, and GUI specifications.
- **Design Phase:** Create detailed application architecture, GUI layout, parsing algorithms, and error handling mechanisms and design a layered architecture with data access, business logic, and presentation layers.
- **Implementation:** Translate design specifications into code, develop the data access layer with DAOs, business logic for processing data, and the GUI using Java Swing and implement CRUD operations, bill generation, and logging.
- **Testing:** Execute comprehensive testing, including unit tests, integration tests, and user acceptance testing.

# 2. Problem Analysis, Modeling, Scenarios, Use Cases

## 2.1.

### **Functional Requirements:**

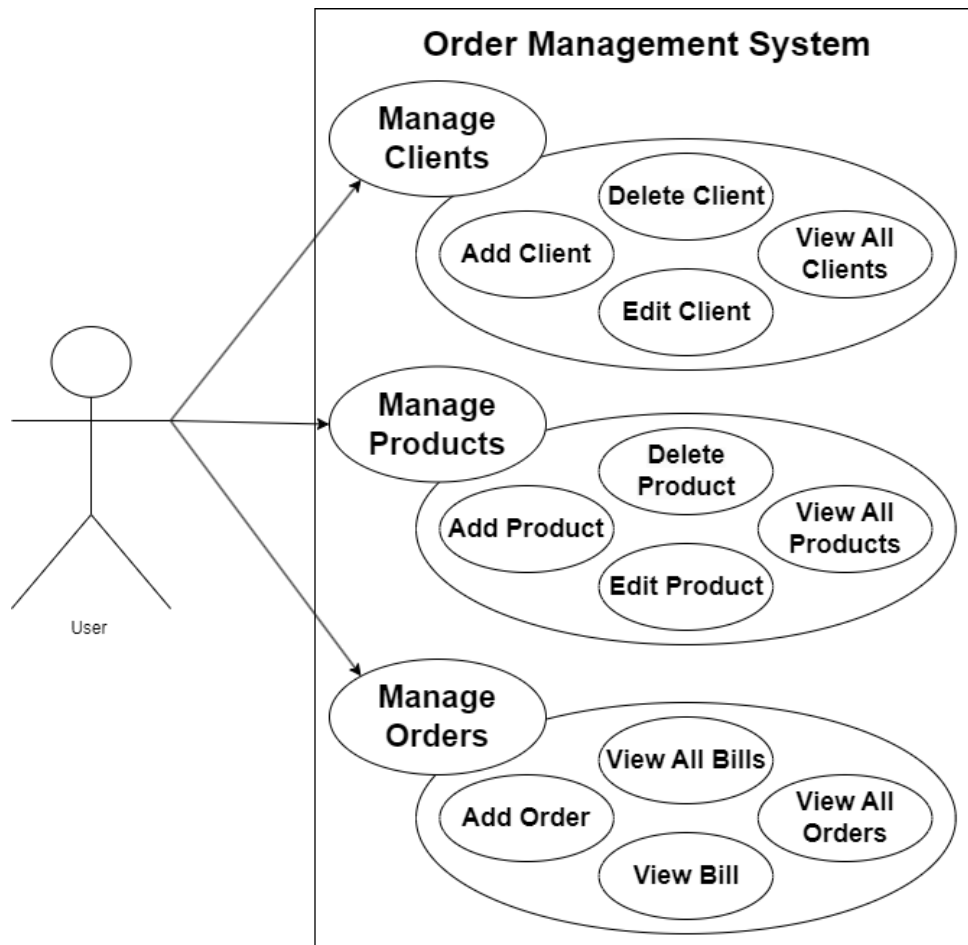
- Client Management:
  - the system should allow users to insert, update, view, and delete client records.
  - each client record should include details such as name, address, and contact information.
- Product Management:
  - the system should allow users to insert, update, view, and delete product records.
  - each product record should include details such as name, quantity, price, and description.
- Order Management:
  - the system should allow users to create new orders by selecting clients and products.
  - the system should allow users to update, view, and delete existing orders.
  - each order should include details such as client information, product details, quantity, total price, and order date.
- Billing:
  - the system should generate a bill for each order, detailing the client information, products purchased, quantities, individual prices, and the total amount.
- Logging:
  - the system should log all operations, including insertions, updates, deletions, and order creations.
- Reports:

- the system should calculate and display the average waiting time, average service time, and peak hour for order processing.
- Error Handling:
- the system should detect and handle errors, providing informative messages to users for invalid inputs or system errors.

### **Non-functional Requirements:**

- User Interface:
  - the GUI should be intuitive and easy to use, with clear labels, input fields, and navigation.
  - the system should provide real-time feedback and confirmation messages for user actions.
- Performance:
  - the system should perform operations efficiently, even with a large number of clients, products, and orders.
  - the system should ensure a smooth user experience with minimal latency.
- Reliability:
  - the system should handle errors gracefully, preventing crashes and providing clear error messages to users.
  - the system should ensure data integrity and consistency across all operations.
- Scalability:
  - the system should be designed in a modular and extensible manner, allowing for easy integration of new features or updates without significant rework.
  - the system should accommodate future enhancements or expansions in terms of functionality and performance.

## **2.2. Use Cases:**



- User to Manage Clients: The user interacts with the system to manage client information by selecting options to add, update, delete, or view client details.
- User to Manage Products: The user interacts with the system to manage product information by selecting options to add, update, delete, or view product details.
- User to Manage Orders: The user interacts with the system to manage orders by selecting options to create, update, delete, or view order details and also see the billing information.

## 3. Design

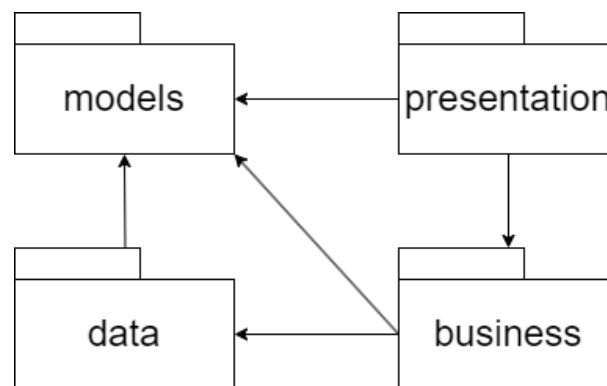
### 3.1.

The Orders Management System demonstrates several key principles of object-oriented programming (OOP), facilitating a robust and flexible design:

- **Abstraction:** Achieved through the use of classes, interfaces, and methods, hiding implementation details from users and focusing on essential characteristics. Classes such as Client, Product, Order, and Bill, abstract real-world concepts related to order management, providing clear interfaces for interaction.

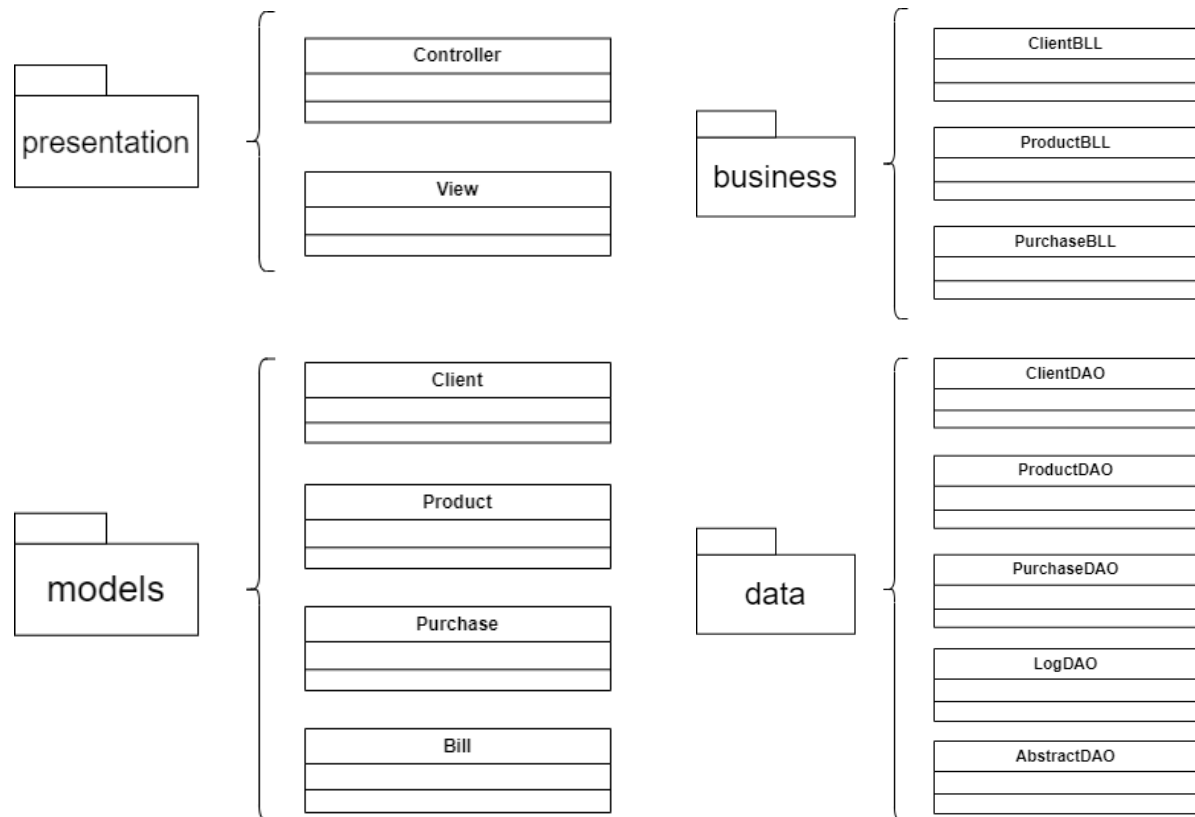
- **Encapsulation:** Employed to bundle data (e.g., client details, product information, order specifics) and methods (e.g., CRUD operations) into cohesive units, ensuring data integrity and promoting modular design. Each class encapsulates its state and behavior, exposing only necessary interfaces to interact with external components.
- **Polymorphism:** Leveraged through method overriding and interface implementation, allowing objects of different types to be treated uniformly. Interface-based design enables polymorphic behavior, allowing classes to adhere to common contracts while providing specialized implementations for data access and business logic.
- **Modularity:** The application is organized into modular packages, each containing related classes and functionalities. For instance, the data, models, business, and presentation packages encapsulate related components and minimize dependencies. This modular design promotes code reuse, maintainability, and scalability by encapsulating related components and minimizing dependencies between modules.
- **Reflection:** Utilized in the AbstractDAO class to dynamically interact with class fields and methods at runtime. This allows for generic data access operations without hardcoding specific class details, enhancing the flexibility and maintainability of the data access layer, needed in ClientDAO, ProductDAO and PurchaseDAO.
- **Separation of Concerns:** The system is divided into distinct layers (data access, business logic, and presentation), each responsible for specific aspects of the application. This separation enhances maintainability, as changes in one layer have minimal impact on others.
- **Error Handling:** The system includes mechanisms for error detection and handling, ensuring robust and reliable operation. For example, invalid input data or database errors are managed gracefully, with appropriate error messages provided to users and logs maintained for troubleshooting.

### 3.2. Division into packages



- The **models package** is central, as it is used by all other packages.
- The **data package** uses the models package to define the structure of the data it interacts with.
- The **business package** uses both the data and models packages to perform business operations.
- The **presentation package** uses the business and models packages to present data and interact with users.

### 3.3. Division into classes



- **presentation Package:**

- **View**: this class represents the user interface of the Orders Management System, handling the presentation layer, user inputs, and displaying data. It interacts with the user and triggers actions in the business layer.
- **Controller**: this class manages the interaction between the user interface and the business logic, processing user commands, and updating the view based on the results of business operations.

- **business Package:**

- **ClientBLL**: This class contains the business logic for handling client-related operations, such as adding, updating, and retrieving client information. It encapsulates the core functionalities related to client management.
- **ProductBLL**: This class handles the business logic for product-related operations, managing tasks like adding products, updating product details, and retrieving product information.
- **PurchaseBLL**: This class manages the business logic for order processing, including creating orders, calculating totals, and handling order-specific business rules and validations.

- **data Package:**

- **AbstractDAO**: This class provides generic data access methods using Java reflection, serving as a base class for specific DAOs. It encapsulates common database operations like insert, update, delete, and find.

- ClientDAO: This class handles the data access operations for client data, such as retrieving client records from the database, inserting new clients, and updating client information.
- ProductDAO: This class manages the data access operations for product data, dealing with database interactions for retrieving, adding, and updating product records.
- PurchaseDAO: This class is responsible for accessing order-related data from the database, performing operations such as inserting new orders, retrieving order details, and updating order information.
- LogDAO: This class manages the logging of various operations and actions performed within the system, maintaining a record of activities for audit and debugging purposes.

- **models Package:**

- Client: This class represents a client within the system, encapsulating client-specific attributes such as name, address, and contact details. It provides a data structure to store and manage client information.
- Product: This class represents a product, containing attributes like product name, price, and stock quantity. It encapsulates the properties and behaviors of a product within the inventory system.
- Bill: This class represents a billing document generated for an order, including order ID, total amount, and timestamp. It encapsulates the billing details for each order.
- Purchase: This class represents a purchase transaction, containing purchase details such as product ID, client ID, and quantity purchased. It encapsulates the data related to individual purchase transactions.

### **3.4. Division into routines**



View		
m	View()	
f	selectedProduct	Product
f	selectedClient	Client
m	displayAddClientWindow(ActionListener)	void
m	createButton(String, ActionListener, Color, Color)	JButton
m	displayProductsWindow(ActionListener, ActionListener, ActionListene	
m	displayClientsWindow(ActionListener, ActionListener, ActionListene	
m	createClientsFrame()	JFrame
m	displayEditClientWindow(Client, ActionListener)	void
m	createViewAllButtonProduct(ActionListener)	JButton
m	displayPurchaseWindow(ActionListener, ActionListener, ActionListe	
m	populateProductTable(List<Object>)	void
m	getClientFromTable(DefaultTableModel, int)	Client
m	populateTableData(List<Object>)	void
m	createProductsFrame()	JFrame
m	populateClientTable(List<Object>)	void
m	displayEditProductWindow(Product, ActionListener)	void
m	updateTableData(DefaultTableModel, List<T>)	void
m	show()	void
m	createViewAllButtonClients(ActionListener)	JButton
m	getProductFromTable(DefaultTableModel, int)	Product
m	displayOptionsWindow(ActionListener, ActionListener, ActionListene	
m	createButtonPanelClient(ActionListener, ActionListener, ActionListe	
m	createButtonPanelProduct(ActionListener, ActionListener, ActionListe	
m	displayAddProductWindow(ActionListener)	void
m	populateTable(DefaultTableModel, List<Object>)	void
m	displayAll(List<Object>)	void
p	selectedClient	Client
p	productStock	String
p	selectedProduct	Product
p	clientEmail	String
p	clientPhoneNumber	String
p	clientName	String
p	productName	String
p	productPrice	String

Controller		
m	Controller(View, Connection)	
m	populateClientTable()	void
m	editProduct(Product)	void
m	main(String[])	void
m	editClient(Client)	void
m	deleteProduct(Product)	void
m	populateProductTable()	void
m	deleteClient(Client)	void

LogDAO		
m	LogDAO(Connection)	
m	addBill(Bill)	void
p	allBills	List<Bill>

AbstractDAO<T>		
m	AbstractDAO(Connection)	
m	addObject(T)	int
m	deleteObject(int)	void
m	findObject(int, Class<T>)	T
m	editObject(int, T)	void
m	createObjectFromResultSet(ResultSet)	T
p	primaryKeyName	String
p	allObjects	List<T>
p	objectType	Class<T>
p	tableName	String

PurchaseDAO		
m	PurchaseDAO(Connection)	
p	primaryKeyName	String
p	tableName	String
p	objectType	Class<Purchase>

ProductDAO		
m	ProductDAO(Connection)	
p	primaryKeyName	String
p	tableName	String
p	objectType	Class<Product>

ClientDAO		
m	ClientDAO(Connection)	
p	primaryKeyName	String
p	tableName	String
p	objectType	Class<Client>

ProductBLL		
m	ProductBLL(Connection)	
m	deleteProduct(int)	int
m	addProduct(Product)	int
m	isValidPrice(double)	boolean
m	updateProduct(Product)	int
p	allProducts	List<Product>

ClientBLL		
m	ClientBLL(Connection)	
m	updateClient(Client)	int
m	isValidEmail(String)	boolean
m	deleteClient(int)	int
m	addClient(Client)	int
m	isValidPhoneNumber(String)	boolean
p	allClients	List<Client>

PurchaseBLL		
m	PurchaseBLL(Connection)	
f	bill	Bill
m	isStockValid(Product, int)	boolean
m	generateBill(LogDAO, Purchase, double)	void
m	createPurchase(Client, Product, int)	int
p	allPurchases	List<Purchase>
p	bill	Bill
p	allBills	List<Bill>

Client		
m	Client(int, String, String, String)	
m	Client()	
f	name	String
f	email	String
m	toString()	String
p	name	String
p	email	String
p	id	int
p	phoneNumber	String

Product		
m	Product(int, String, double, int)	
m	Product()	
f	name	String
f	price	double
f	stock	int
m	toString()	String
p	name	String
p	price	double
p	stock	int
p	id	int

Purchase		
m	Purchase(int, int, int, Timestamp)	
m	Purchase()	
m	toString()	String
p	id	int
p	orderDate	Timestamp

Bill		
m	Bill(int, double, Timestamp)	
m	timestamp()	Timestamp
m	orderid()	int
m	totalAmount()	double

## 4. Implementation

### 4.1. Class Description

- **View:**

Description: The View class is responsible for the graphical user interface (GUI) of the Orders Management System. It creates and manages the various visual components that users interact with, such as buttons, text fields, and tables. This class captures user input and displays data returned from the business layer.

Key Responsibilities:

- Render the GUI layout using Swing components.
- Capture user actions, such as button clicks and text input.

- Display data (e.g., client lists, product information, order details) in a user-friendly format.
- Communicate user commands to the Controller.

- **Controller:**

Description: The Controller class acts as an intermediary between the View and the business logic classes. It processes user commands from the View, calls the appropriate business logic methods, and updates the View with the results.

Key Responsibilities:

- Listen to events triggered by the View.
- Invoke business logic methods in response to user actions.
- Update the View with data returned from the business logic layer.
- Handle navigation between different views or panels within the GUI.

- **ClientBLL:**

Description: The ClientBLL (Business Logic Layer) class contains the core logic for managing clients. It performs operations such as adding, updating, retrieving, and validating client data.

Key Responsibilities:

- Add new clients to the system.
- Update existing client information.
- Retrieve client data based on various criteria.
- Validate client data to ensure it meets business rules.

- **ProductBLL:**

Description: The ProductBLL class manages the business logic related to products. It handles operations such as adding new products, updating product details, retrieving product information, and performing validations.

Key Responsibilities:

- Add new products to the inventory.
- Update product information, such as price and stock levels.
- Retrieve product details for display or processing.
- Validate product data to ensure consistency and correctness.

- **PurchaseBLL:**

Description: The PurchaseBLL class oversees the business logic for order processing. It includes functionality for creating new orders, calculating order totals, managing order status, and ensuring that order data adheres to business rules.

Key Responsibilities:

- Create and process new orders.
- Calculate the total amount for an order, including any discounts or taxes.
- Retrieve and update order information.
- Validate order details to ensure they comply with business requirements.

- **AbstractDAO:**

Description: The AbstractDAO class provides a generic template for data access operations using Java reflection. It includes common methods for interacting with the database, such as insert, update, delete, and find operations.

Key Responsibilities:

- Perform generic CRUD (Create, Read, Update, Delete) operations.
- Use reflection to dynamically handle different types of entities.
- Serve as a base class for more specific DAO implementations.

- **LogDAO:**

Description: The LogDAO class manages logging operations within the system. It records various actions and events for auditing and debugging purposes, ensuring that a comprehensive log is maintained.

Key Responsibilities:

- Insert log entries into the database.
- Retrieve log records for review and analysis.
- Manage the lifecycle of log entries, including deletion of old logs.

- **ClientDAO:**

Description: The ClientDAO class handles the data access operations specific to client data. It interacts with the database to retrieve, insert, update, and delete client records.

Key Responsibilities:

- Retrieve client records from the database.
- Insert new client records.
- Update existing client data.
- Delete client records as needed.

- **ProductDAO:**

Description: The ProductDAO class manages data access for product-related operations. It handles interactions with the database for tasks such as retrieving product information, adding new products, and updating existing product records.

Key Responsibilities:

- Retrieve product details from the database.
- Insert new products into the database.
- Update product information.
- Delete product records if necessary.

- **PurchaseDAO:**

Description: The PurchaseDAO class is responsible for data access operations related to orders. It performs database interactions to manage order data, including retrieving order details, adding new orders, and updating order status.

Key Responsibilities:

- Retrieve order information from the database.
- Insert new orders into the database.
- Update order records.
- Delete orders when required.

- **Client:**

Description: The Client class represents a client within the system. It includes attributes such as name, address, and contact details, encapsulating the data and behavior associated with a client.

Key Responsibilities:

- Store and manage client information.
- Provide methods for accessing and modifying client attributes.

- **Product:**

Description: The Product class represents a product in the inventory. It contains attributes like product name, price, and stock quantity, encapsulating the data and behavior associated with a product.

Key Responsibilities:

- Store product details.
- Provide methods for accessing and modifying product attributes.

- **Purchase:**

Description: The Purchase class represents an order placed by a client. It includes details such as order ID, client information, product quantities, and order status.

Key Responsibilities:

- Store and manage order details.
- Provide methods for accessing and modifying order attributes.

- **Bill:**

Description: The Bill class represents a billing document generated for an order. It includes the order ID, total amount, and timestamp, encapsulating the data related to the billing process.

Key Responsibilities:

- Store billing information.
- Provide methods for accessing billing details.

(\*more information about classes and methods in the Java doc files\*)

## 4.2. Graphical User Interface

The main window displays three buttons that offer quick access to essential actions:

- **Manage Clients:** Click this button to access client management functionalities, including adding, editing, and deleting clients, as well as viewing all clients in a table.
- **Manage Products:** Use this button to access product management functionalities, such as adding, editing, and deleting products, as well as viewing all products in a table.
- **Manage Purchases:** Click this button to create new product orders, allowing you to select existing clients and products, insert desired quantities, and finalize orders efficiently.



- **Client Management:**

The Order Management System allows you to seamlessly manage your clients' information. Here are the key functionalities available for client management:

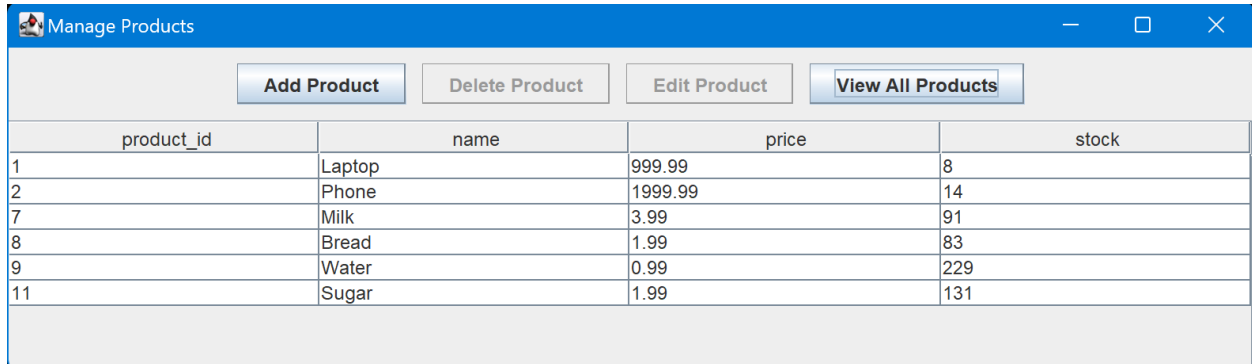
A screenshot of a web application window titled "Manage Clients". At the top, there are four buttons: "Add Client", "Delete Client", "Edit Client", and "View All Clients". Below these buttons is a table with four columns: "client\_id", "name", "email", and "phone". The table contains 10 rows of client data. The row for "Janny Smith" (client\_id 8) is highlighted in blue.

client_id	name	email	phone
1	John Doe	john@example.com	1234567890
2	Maria Gozman-Pop	maria@example.com	0757245574
7	Richard Smith	rsmith@example.com	6789012345
8	Janny Smith	jsmith@example.com	6789012344
9	Anne Miller	annie@example.com	8769012345
11	Bob Milley	bob@example.com	8769012343
28	Calina Borzan	calinuta@example.com	0754675234
30	Francesca Bellina	fran@example.com	1231234560

- **Add New Client:** To add a new client, navigate to the client management section and click on the "Add New Client" option. Enter the relevant details for the new client, such as name, contact information, and address, and click "Save" to add them to your client database.
- **Edit Client:** If you need to update the information of an existing client, simply select the client from the list, make the necessary changes, and click "Save" to update their details.
- **Delete Client:** To remove a client from your database, select the client you wish to delete and click on the "Delete Client" option. Confirm the deletion, and the client will be permanently removed from your records.
- **View All Clients:** The system provides a convenient table view where you can browse through all your clients' details, making it easy to review and manage your client database efficiently.

- **Product Management:**

Efficiently manage your product inventory with the following features available in the Order Management System:



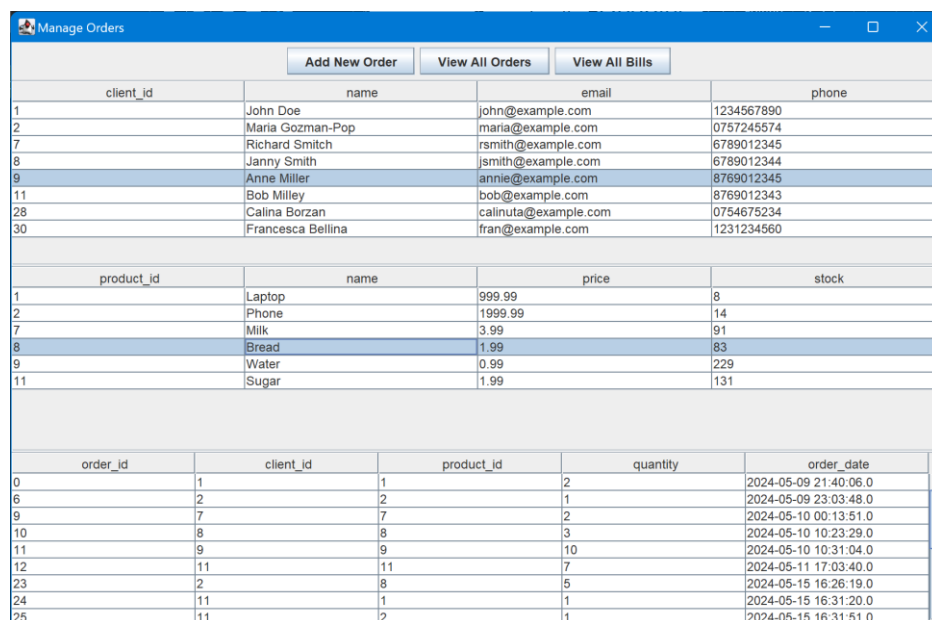
The screenshot shows a window titled "Manage Products" with a blue header bar. Below the header, there are four buttons: "Add Product", "Delete Product", "Edit Product", and "View All Products". Below the buttons is a table with four columns: "product\_id", "name", "price", and "stock". The table contains six rows of data.

product_id	name	price	stock
1	Laptop	999.99	8
2	Phone	1999.99	14
7	Milk	3.99	91
8	Bread	1.99	83
9	Water	0.99	229
11	Sugar	1.99	131

- **Add New Product:** To add a new product to your inventory, navigate to the product management section and click on the "Add New Product" option. Enter the product details, such as name, description, and price, and click "Save" to add the product to your catalog.
- **Edit Product:** If you need to update the details of an existing product, select the product from the list, make the necessary modifications, and click "Save" to update the product information.
- **Delete Product:** To remove a product from your inventory, select the product you wish to delete and click on the "Delete Product" option. Confirm the deletion, and the product will be removed from your catalog.
- **View All Products:** The system provides a comprehensive table view where you can view all your products' details, allowing you to easily monitor and manage your inventory.

- **Order Creation:**

Creating orders is a streamlined process in the Order Management System. Follow these steps to create new orders:



The screenshot shows a window titled "Manage Orders" with a blue header bar. Below the header, there are three buttons: "Add New Order", "View All Orders", and "View All Bills". Below the buttons is a table with four columns: "client\_id", "name", "email", and "phone". The table contains six rows of data. Below this table is another table with four columns: "product\_id", "name", "price", and "stock". The table contains six rows of data. Below this table is a third table with five columns: "order\_id", "client\_id", "product\_id", "quantity", and "order\_date". The table contains ten rows of data.

client_id	name	email	phone
1	John Doe	john@example.com	1234567890
2	Maria Gozman-Pop	maria@example.com	0757245574
7	Richard Smith	rsmith@example.com	6789012345
8	Janny Smith	jsmith@example.com	6789012344
9	Anne Miller	annie@example.com	8769012345
11	Bob Milley	bob@example.com	8769012343
28	Calina Borzan	calinuta@example.com	0754675234
30	Francesca Bellina	fran@example.com	1231234560

product_id	name	price	stock
1	Laptop	999.99	8
2	Phone	1999.99	14
7	Milk	3.99	91
8	Bread	1.99	83
9	Water	0.99	229
11	Sugar	1.99	131

order_id	client_id	product_id	quantity	order_date
0	1	1	2	2024-05-09 21:40:06.0
6	2	2	1	2024-05-09 23:03:48.0
9	7	7	2	2024-05-10 00:13:51.0
10	8	8	3	2024-05-10 10:23:29.0
11	9	9	10	2024-05-10 10:31:04.0
12	11	11	7	2024-05-11 17:03:40.0
23	2	8	5	2024-05-15 16:26:19.0
24	11	1	1	2024-05-15 16:31:20.0
25	11	2	1	2024-05-15 16:31:51.0

- **Select Product and Client:** Begin by selecting the desired product from your inventory and the client placing the order.
- **Specify Quantity:** Enter the quantity of the selected product that the client wishes to purchase.
- **Finalize Order:** Once the order details are confirmed, click on the "Finalize Order" button to complete the order process.
- **Under-Stock Notification:** If the requested quantity exceeds the available stock for the selected product, the system will display an under-stock message, alerting you to adjust the order accordingly.
- **Automated Stock Update:** Upon finalizing the order, the system automatically updates the product stock to reflect the quantity sold, ensuring accurate inventory management.

## 5. Conclusions

In conclusion, the Order Management System serves as a robust platform for efficiently handling client, product, and purchase data, facilitating seamless order creation and inventory management processes. Throughout the assignment, several points have been achieved:

- Core functionalities such as client and product management, as well as order creation, have been successfully implemented. Users can easily manage clients and products, view them in organized tables, and create valid orders with minimal effort.
- The graphical user interface (GUI) offers an intuitive and straightforward experience, enabling users to navigate through various functionalities effortlessly. The clear layout and user-centric design contribute to a positive overall user experience.
- The system enables users to create orders efficiently, ensuring accuracy in processing and updating product stocks. Additionally, users receive prompts in case of under-stock situations, enhancing overall order management efficiency.

Looking ahead, potential areas for future development and enhancement include advanced reporting features, integration with external systems, enhanced security measures, and mobile compatibility.

In summary, while the current version of the Order Management System meets immediate requirements, ongoing development efforts are essential to address evolving business needs and technological advancements. By leveraging insights gained from this assignment and incorporating user feedback, the system can continue to evolve, providing increased functionality, usability, and value.

## 6. Bibliography

1. [https://dsrl.eu/courses/pt/materials/PT2024\\_A3.pdf](https://dsrl.eu/courses/pt/materials/PT2024_A3.pdf)
2. [https://dsrl.eu/courses/pt/materials/PT2024\\_A3\\_S1.pdf](https://dsrl.eu/courses/pt/materials/PT2024_A3_S1.pdf)
3. [https://dsrl.eu/courses/pt/materials/PT2024\\_A3\\_S2.pdf](https://dsrl.eu/courses/pt/materials/PT2024_A3_S2.pdf)
4. <https://docs.oracle.com/en/java/>
5. <https://java-design-patterns.com/patterns/layers/>
6. <https://dev.mysql.com/doc/mysql-getting-started/en/>