BrewLink Project

Final report

Veronica Elisa Martinez Contreras

german camilo cardenas bejarano

Natalia herrera espinosa

Jose antonio bellorin JULIO

Javier alcantara perez

2023

# INTRODUCTION

"Welcome to Brewed Link, an innovative project that delivers a one-of-a-kind experience in the realm of beverages and pastry! Our program seamlessly blends the best offerings from four distinctive coffee shops with the efficient services of three delivery companies, all under one roof.

Envision having a plethora of options at your fingertips: from the inviting ambiance of your cherished café to the ease of receiving orders directly at your home or office. This meticulously curated program empowers you to choose from a diverse selection of exquisite beverages, ranging from expertly crafted coffees to artisanal teas, complemented by delectable desserts designed to tantalize even the most discerning palates.

Through collaboration with four renowned coffee shops, each renowned for their commitment to quality, flavor, and customer experience, alongside three delivery companies dedicated to punctuality and customer satisfaction, our program represents the pinnacle of convenience fused with culinary excellence.

Prepare to embark on a journey exploring a myriad of flavors, uncovering novel pleasures for your taste buds, and relishing the convenience of having your orders delivered at your preferred time and place. Welcome to an unparalleled experience celebrating the diverse tapestry of choices in the world of beverages and pastry

# INTERFACES

## **INTERFACE** (ClientsWaiter)

This interface provides methods to interact with the database concerning the "Clients" entity. It performs queries to retrieve, display, insert, and obtain specific information about clients in the database.

GetOneByID: Retrieves a specific client from the database based on their ID. It constructs and executes an SQL query to search for a client in the database and returns an instance of the Clients class with the corresponding information.

GetAll: Fetches all clients stored in the database. It executes an SQL query to retrieve all clients and stores them in a list of Clients objects.

DisplayClient and DisplayClientById: These methods print the client's information to the console obtained using the GetAll and GetOneByID methods, respectively. It displays details such as ID, email, names, last names, and phone number.

InsertClient: Allows the insertion of a new client into the database. It creates an SQL query to insert a new client record using the information provided in the Clients object.

getUserPaswd and getUserIdFromClient: These methods retrieve the password and ID of a client, respectively, using their email address as an identifier.

## **INTERFACE** (Products Waiter)

This interface encapsulates various methods to manage products within a coffee shop's database, providing functionalities for retrieving, adding, updating, and deleting product information based on different criteria.

Get Products: This method retrieves products based on specified criteria, such as by store ID or product name. It constructs SQL queries dynamically based on the provided parameters (type, Store\_ID, prodName) to fetch products meeting the specified conditions.

Get All Product: Fetches all products available in the database by executing a simple SQL query without specific filtering criteria.

Get One Product: Retrieves a single product from the database based on the provided parameter (presumably a product ID).

Get All Product(String parameter): Retrieves a list of products matching a partial product name provided as a parameter.

Add Factory Builder Product: Adds a new product to the database using the details provided, including category, store ID, product name, price, commission, and activation status.

Update Product: Updates an existing product's details in the database based on the information provided in the Product object.

Delete Product: Deletes a specific product from the database based on the provided product ID.

Throughout the code, the class utilizes the Data Waiter singleton to interact with the database by executing SQL queries and handling the retrieval, insertion, updating, and deletion of product-related data.

# IMPLEMENTATIONS

## Abstract Factory

This is the abstract class defining the interface for creating different products associated with various coffee shops.

CODE

public abstract class AbstractFactory {

private static AbstractFactory drinksFactory = new DrinksFactory();

private static AbstractFactory bakeryFactory = new BakeryFactory();

public static AbstractFactory factory(FactoryType type){

AbstractFactory factory = null;

switch (type)

{

case Drinks:

factory = drinksFactory;

break;

case Bakery:

factory = bakeryFactory;

break;

}

return factory;

}

public class BakeryFactory extends AbstractFactory {

@Override

public ProductCoffeeCulture createCoffeeCulture(String prodName, BigDecimal price, int commission) {

IBakeryCoffeeCulture iBakery = new IBakeryCoffeeCulture();

BuilderCoffeeCultureProduct builder = new BuilderCoffeeCultureProduct(iBakery);

builder.createProductShop(prodName,price,commission);

ProductCoffeeCulture product = builder.getProduct();

ProductsWaiter productsWaiter = new ProductsWaiter();

if (productsWaiter.addFactoryBuilderProduct(product.getCategory\_Id(), product.getStore\_ID(),product.getProdName(),product.getPrice(),product.getComission(), product.getActive() ))

System.out.println("Product added successfully to the database");

return product;

}

public abstract ProductCoffeeCulture createCoffeeCulture(String prodName, BigDecimal price, int commission);

public abstract ProductSecondCup createSecondCup(String prodName, BigDecimal price, int commission);

public abstract ProductStarbucks createStarbucks(String prodName, BigDecimal price, int commission);

public abstract ProductTimbertrain createTimbertrain(String prodName, BigDecimal price, int commission);

public abstract ProductTimHortons createTimHortons(String prodName, BigDecimal price, int commission);

## Drinks Factory and Bakery Factory

These are concrete factory classes implementing the abstract factory. They provide specific implementations for creating products associated with drinks or bakery items.

public class DrinksFactory extends AbstractFactory{

@Override

public ProductCoffeeCulture createCoffeeCulture(String prodName, BigDecimal price, int commission) {

IDrinksCoffeeCulture iDrinks = new IDrinksCoffeeCulture();

BuilderCoffeeCultureProduct builder = new BuilderCoffeeCultureProduct(iDrinks);

builder.createProductShop(prodName,price,commission);

ProductCoffeeCulture product = builder.getProduct();

ProductsWaiter productsWaiter = new ProductsWaiter();

if (productsWaiter.addFactoryBuilderProduct(product.getCategory\_Id(), product.getStore\_ID(),product.getProdName(),product.getPrice(),product.getComission(), product.getActive() ))

System.out.println("Product added successfully to the database");

return product;

}

### Usage:

Using the abstract factory pattern, you can create specific product instances without directly specifying their concrete classes, allowing for flexibility and encapsulation.

This pattern allows for creating families of related products (associated with different coffee shops in this case) without the client code needing to know the exact concrete classes of the products, promoting flexibility and easier maintenance when adding new products or modifying existing ones.

## Builder Coffee Culture Product

This class acts as a director in the Builder pattern. It coordinates the construction steps of a ProductCoffeeCulture object using a ProductBuilderCoffeeCulture, allowing it to work with a specific builder.

public class BuilderCoffeeCultureProduct {

private ProductBuilderCoffeeCulture productShopBuilder;

public BuilderCoffeeCultureProduct(ProductBuilderCoffeeCulture productShopBuilder)

{

this.productShopBuilder = productShopBuilder;

}

public void createProductShop(String prodName, BigDecimal price, int commission)

{

this.productShopBuilder.BuildCategory\_Id();

this.productShopBuilder.BuildStore\_ID();

this.productShopBuilder.BuildProduct\_Id();

this.productShopBuilder.BuildProdName(prodName);

this.productShopBuilder.BuildPrice(price);

this.productShopBuilder.BuildCommission(commission);

this.productShopBuilder.BuildActive();

}

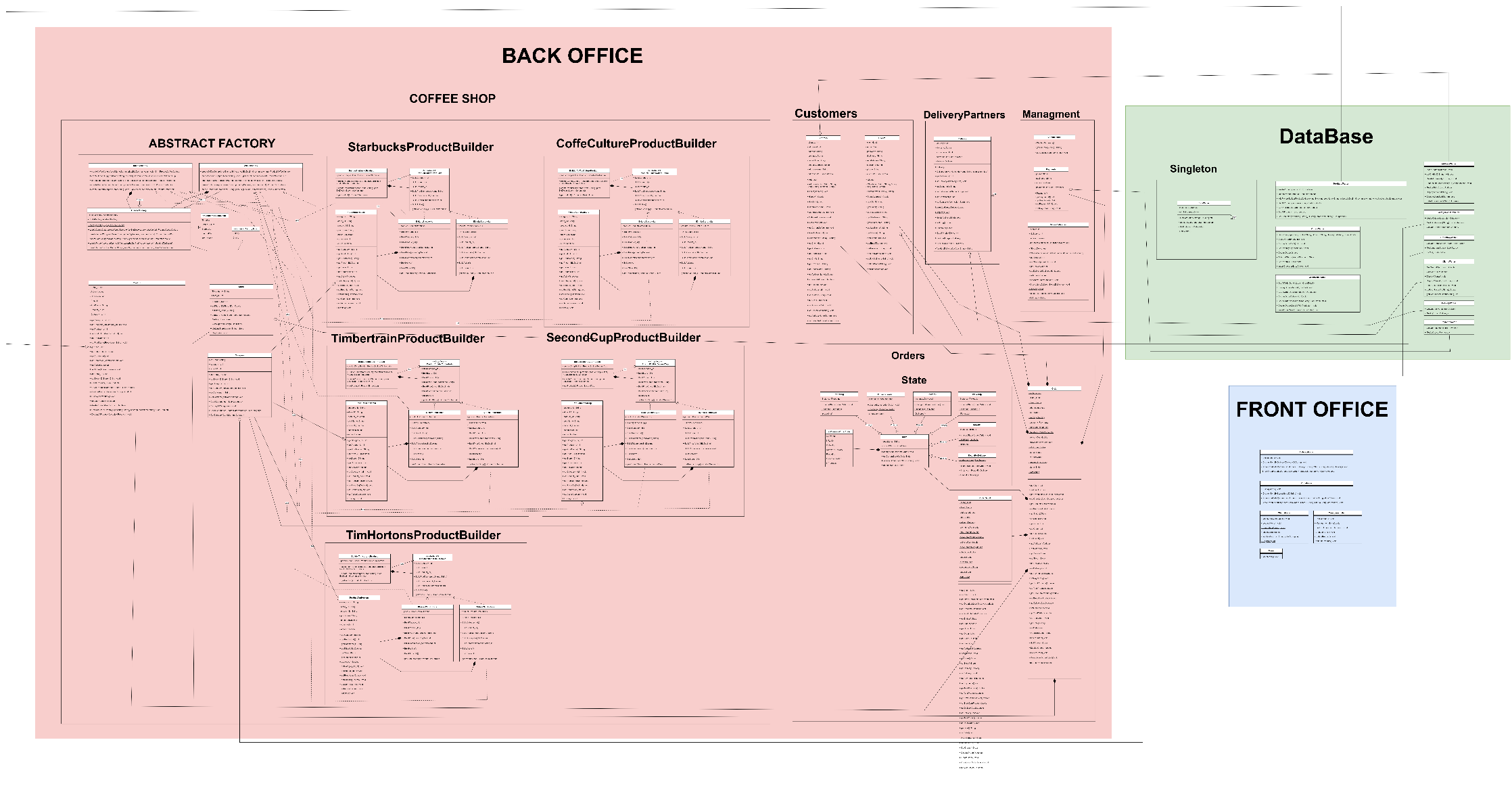
public Product Coffee Culture Get Product() { return this.productShopBuilder.getProductShop(); }

}

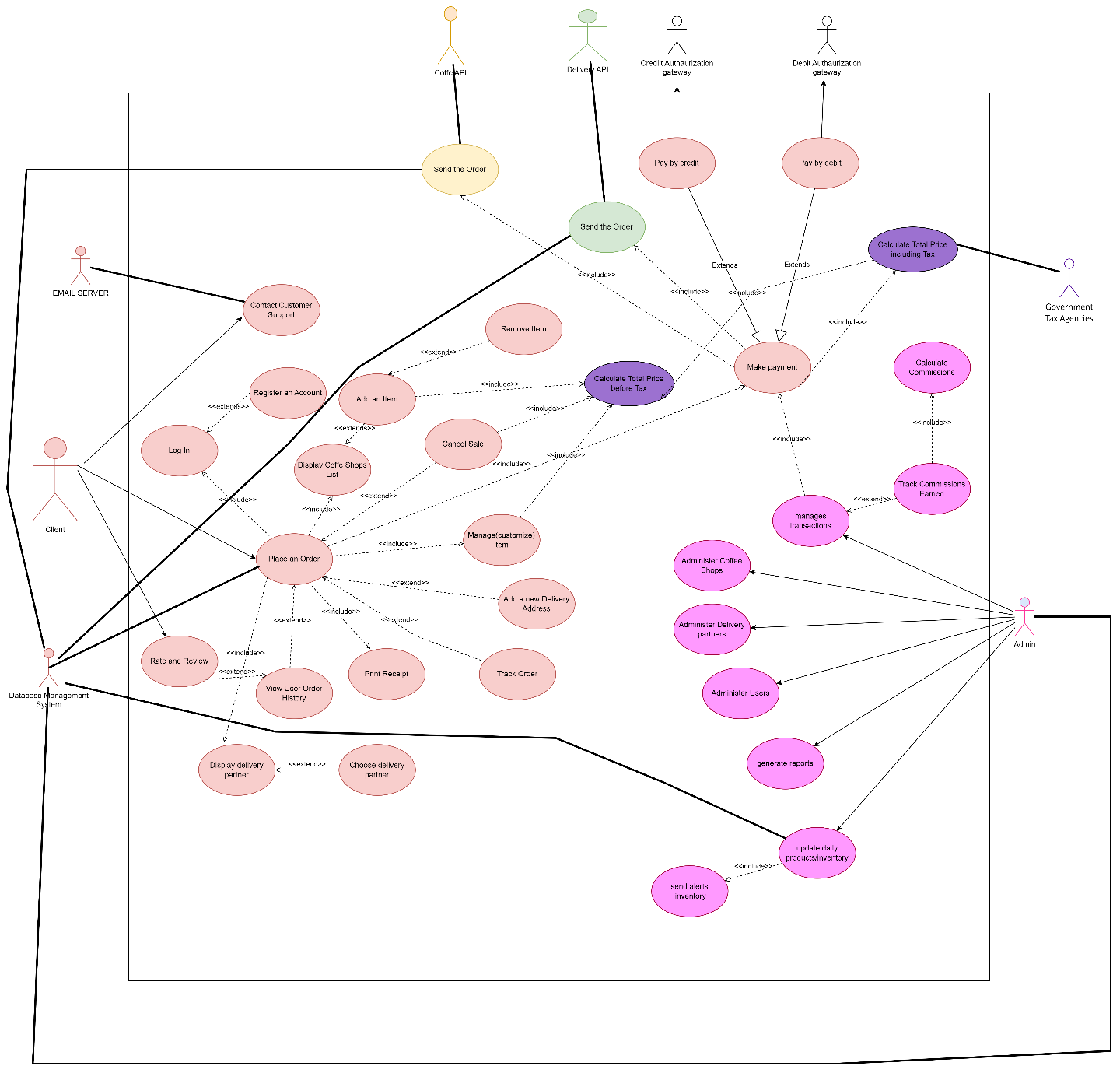
Create Product Shop: This method orchestrates the step-by-step creation of a Product Coffee Culture object. It calls various build methods (Build Category\_Id, Build Store\_ID, etc.) of the given Product Builder Coffee Culture to build the product incrementally.

Get Product: Once the product is constructed, this method retrieves the final Product Coffee Culture object from the builder.

# Class Diagram



# Use Case Diagram



# User Cases

### USE CASE: Verónica Elisa Martínez Contreras

**USE CASE**: Client Account Access and Management

**ACTORS** : Customer                                                    **Mode:** Online

**OBJECTIVES** : To allow users to log in or register for a personal user area within the application.

**RULES OF INITIATION**:

1. Users must possess a registered account or create one to access the application.
2. The application must be installed and have an active internet connection.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1.Open the application. |  |
| 2. For Login:   * Select the "Login" option. * Enter email and password. * Submits the credentials. | * Verifies and authenticates the user's entered details. * Grants access to the user area upon successful authentication. |
| 3. For Register:   * Choose the "Register" option. * Fills out the registration form with personal and contact information. * Submits the registration details. | * Verifies the uniqueness of the provided email. * Stores the user's information upon successful validation. |

**RULES OF TERMINATION**:

* Successful Login: The customer successfully logs into the application and gains access to their account.
* Failed Login Attempts: If the customer enters invalid credentials, there may be a rule for a limited number of failed login attempts before temporarily locking the account.
* Account Lockout: If the customer exceeds the allowed number of failed login attempts, the account may be temporarily locked for security purposes.
* Password Reset: In case the customer forgets their password, there may be a rule for initiating a password reset process.

**ALTERNATIVE CASES**

* If the user forgets their password, they can initiate a password reset by selecting "Forgot my password."
* For user assistance, options like "Help" or "Support" are available within the application.

**ADDITIONAL REQUIREMENTS:**

1. The system must maintain secure encryption for storing user credentials.
2. User data must be stored securely to prevent unauthorized access.
3. User interface design should ensure ease of navigation and user-friendly interactions.
4. The application should send confirmation messages upon successful registration or password reset.

**MESSAGES:**

1. "Invalid credentials. Please try again."
2. "This email is already registered."
3. "Error connecting to the database."

### USE CASE: Verónica Elisa Martínez Contreras

**|USE CASE**: Navigation Menu Display

**ACTORS** : User                                                 **Mode:** Online

**OBJECTIVES** :Display the main navigation menu options to enable users to navigate through the program.

**RULES OF INITIATION**:

1. Front Office User opens the application.
2. The system presents the main navigation menu upon application launch.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. Front Office User is presented with the main navigation menu. | Displays the main menu options:     - Option 1: Clients     - Option 2: Coffee Shop     - Option 3: Deliveries     - Option 4: Management     - Option 0: Exit |
| 1. Front Office User selects an option from the menu. | Executes the corresponding action based on the selected option:     - Option 1: Redirects to the Clients Menu.     - Option 2: Redirects to the Coffee Shop Menu.     - Option 3: Redirects to the Deliveries Menu.     - Option 4: Redirects to the Management Menu.     - Option 0: Exit the application. |

**RULES OF TERMINATION**:

* User exits the application by selecting option 0.
* Users complete their intended actions within the selected menu option.

**ALTERNATIVE CASES**

* If the user inputs an invalid choice, the system displays an error message and prompts for a valid input.

**ADDITIONAL REQUIREMENTS:**

* Ensure an intuitive and user-friendly menu interface.
* Implement robust error handling for incorrect user inputs.
* Provide appropriate navigation and redirection based on user selections.

**MESSAGES:**

* "Invalid choice. Please select a valid option."
* "Error: Unable to process the request. Try again later."
* "Access Denied: Permission required to access this section."
* "Invalid input detected. Please enter a number between 0 and 4."

### USE CASE: German Camilo Cardenas Bejarano

**|USE CASE**: Manage Order Details

**ACTORS** : System, Administrators                                                   **Mode:** Online

**OBJECTIVES** :Enable administrators to manage order details such as creation, modification, deletion, and retrieval, and maintain accurate order information.

**RULES OF INITIATION**:

1. The administrator logs into their management account.
2. The administrator has access to order management functionalities including create, modify, delete, and retrieve order details.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. The administrator navigates to the order details section. | The system presents a comprehensive interface for order detail management with options to create, modify, delete, or retrieve order details. |
| 1. The administrator selects an action: create, modify, delete, or retrieve order details. | The system presents a comprehensive interface for order detail management with options to create, modify, delete, or retrieve order details.   1. For "Create Order Detail":    * The system receives a list of products and a store ID to create new order details.    * It verifies the validity of provided information and ensures no duplicate entries. 2. For "Modify Order Detail":    * The system allows the administrator to edit specific order details such as product quantity or commission details. 3. For "Delete Order Detail":    * The system enables the deletion of specific order details upon confirmation from the administrator. 4. For "Retrieve Order Detail":    * The system provides a detailed summary of order details based on specified criteria like store ID or product. |

**RULES OF TERMINATION**:

* The administrator successfully creates, modifies, deletes, or retrieves order details.
* The administrator exits the order detail management section or logs out.

**ALTERNATIVE CASES**

* If the system encounters errors during order detail creation or modification, it displays an error message and prompts the administrator to take corrective action.

**ADDITIONAL REQUIREMENTS**

* The system should maintain a log of order detail modifications for auditing purposes.
* Implement security measures to ensure authorized access to order detail management functionalities.
* Provide efficient search capabilities within the order detail retrieval process

**MESSAGES:**

* "Please select an action: create, modify, delete, or retrieve order details."
* "Order detail successfully created/modified/deleted."
* "Error encountered during order detail creation/modification. Please try again."
* "Order detail retrieval successful/no details found. Please refine your search criteria."

### USE CASE: German Camilo Cardenas Bejarano

**|USE CASE**: Show Order Historical

**ACTORS** : User, System                                                 **Mode:** Online

**OBJECTIVES** :To track historical orders for a selected delivery.

**RULES OF INITIATION**:

1. The system waits for the user to input a valid delivery ID.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| The user, interacting with the FrontOffice Delivery Menu system, intends to access and analyze historical order details linked to specific deliveries. | The system offers functionality allowing users to access and view historical orders concerning the specified delivery ID.   * Input Valid Delivery ID:   + The user initiates the process by entering a valid delivery ID into the system to gain access to historical order information. * Retrieve and Display Historical Orders:   + Upon receiving the valid delivery ID, the system searches and retrieves historical order records associated with the specified delivery ID.   + The system then presents these historical orders in an organized and structured manner for the user to review. * View Order Details:   + The user is provided with the capability to view detailed information for each historical order presented.   + The displayed details may include but are not limited to order IDs, statuses, timestamps, and relevant order specifics. * Navigation and Interaction:   + The user can navigate through the displayed historical orders and interact with the system to access detailed information for each order as desired. |

**RULES OF TERMINATION**:

* The user concludes the interaction with historical orders by entering '0' when prompted or upon completing the review of historical order details.

**ALTERNATIVE CASES**

* If the user enters an invalid or non-existent delivery ID, the system detects this and prompts the user to provide a valid delivery ID to proceed with accessing historical orders.

**ADDITIONAL REQUIREMENTS**

* Users must supply a valid delivery ID to access and inspect historical orders related to the specified delivery within the system.

**MESSAGES:**

* Success message: "Historical orders displayed successfully."
* Error message: "Invalid delivery ID entered. Please input a valid delivery ID."

### USE CASE: German Camilo Cardenas Bejarano

**USE CASE**: Display Menu By Delivery

**ACTORS** : User, System                                                 **Mode:** Online

**OBJECTIVES** :To manage delivery orders based on the selected delivery ID.

**RULES OF INITIATION**:

1. The system waits for the user to input a valid delivery ID.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. The user, operating within the FrontOffice Delivery Menu system, interacts with the system to manage delivery orders. | The system presents a menu-driven interface where users interact to manage delivery orders for specific delivery IDs.   1. Input Delivery ID:    * The user inputs a specific delivery ID into the system to access delivery-related options. 2. Display Options:    * The system, upon receiving a valid delivery ID, displays a set of actions or options related to managing orders for the selected delivery. 3. Select Action:    * The user selects an option from the displayed set of actions to perform a specific task related to the selected delivery orders. 4. System Response:    * The system responds according to the user's selected action, executing the chosen operation on the delivery orders associated with the provided delivery ID. |

**RULES OF TERMINATION**:

* The user exits the menu interface by entering '0' when prompted.

**ALTERNATIVE CASES**

* If an invalid or non-existent delivery ID is entered by the user, the system notifies the user of the invalid entry and prompts them to re-enter a valid delivery ID to proceed.

**ADDITIONAL REQUIREMENTS**

* Users need a valid delivery ID to access and perform actions related to delivery orders within the system.

**MESSAGES:**

* Success message: "Action performed successfully."
* Error message: "Invalid delivery ID entered. Please input a valid delivery ID."

### USE CASE: Natalia Herrera

**Use Case:** A customer order and pay an order

**Actors:** Customer/ System **Mode:** App Online

**Objective**: Allow customers to order coffee and other products and complete payment.

**Rules of Initiation:**

1. Customer logs in to their account with user ID and password.
2. Customer accesses the menu to view available items.
3. Customer adds items to their order, with the system displaying updated prices and items.
4. Customer proceeds to payment for the order.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. A customer requests access with their user ID and password | 1.The system validates the access code; it displays the available transactions. |
| 2. . A customer Selects 'Store Menu' to view available products. | 2. The system Displays product availability for the selected store. |
| 3. Customer selects products from the menu | 3. Calculates total price and displays it to the customer. |
| 4. Customer confirms order and enters payment details. | 4. The system Processes payment, registers the order, and initiates preparation, then confirms the order. |

**Rules of Termination:**

* Order is successfully placed and paid for.
* System generates a receipt.
* If payment is declined, the order is not processed.

**Alternative Cases:**

|  |  |
| --- | --- |
| 1. The access code or password is invalid; | 1. The system displays the message |
| 4. Payment method is declined | 4. The system displays to try a different payment method |

**Additional Cases:**

* Customer adjusts (adds/removes items) order before final confirmation.
* Customer views past orders and has an option to reorder.

**Additional Requirements:**

Transaction processing time must not exceed 5 seconds.

**Messages:**

* "Payment successful. Your order number is #[Order Number]."
* "Login failed. Please check your user ID and password and try again."
* "Payment declined. Please try a different payment method."

### USE CASE: Natalia Herrera

**Use Case:** Manager updates products and reviews commissions

**Actors:** Manager / System **Mode:** Online / App

**Objective:** Enable the manager to add, update, delete products, and review commissions.

**Rules of Initiation:**

1. Manager logs into their management account.
2. Manager has access to the management menu with product management and commission review functionalities.
3. Manager can perform operations like adding, updating, and deleting products.

**Description of the Main Scenario:**

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. Manager Logs in with management credentials. | 1.The system validates the access code. |
| 2. . Manager Chooses an operation (add a new product, update an existing product, delete a product, display commissions). | 2. Executes the selected operation:  For product addition: System guides the manager through the product creation process.  For product update: System enables editing of product details.  For product deletion: System removes the product from the listing.  For commission review: System displays commission summaries. |
| 3. Completes the operation and receives confirmation from the system. | 3. confirms message. |

**RULES OF TERMINATION**:

* Manager completes the desired operation successfully.
* System updates the product database or displays commission details.
* Manager exits the management menu or logs out.

**ALTERNATIVE CASE**

|  |  |
| --- | --- |
| 1. The access code or password is invalid; | 1. The system displays the message |
|  | 2. System error during product update: |
|  | 3 System displays an error message and prompts retry. |

**ADDITIONAL CASES**:

* Manager requires to review specific product details.
* Manager needs to generate reports of product updates or commission earnings.

**ADDITIONAL REQUIREMENTS:**

* Management operations must be processed within a reasonable time frame (e.g., under 5 seconds).
* System should handle exceptions gracefully and provide informative error messages.

**MESSAGES:**

"Product successfully added/updated/deleted."

"Login failed. Please check your credentials and try again."

"Error encountered during operation. Please try again."

### USE CASE: Jose Antonio Bellorin Julio

**USE CASE**: Manage Stores

**ACTORS** : Administrator                                                   **Mode:** Online

**OBJECTIVES** :To enable the administrator to view and review detailed information about stores within the system.

**RULES OF INITIATION**:

1. The administrator must have authorized access to the system.
2. The system must have an active and stable connection to access store data from the database.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. Logs into the system using valid credentials. | Displays a complete and accurate list of stores upon request. |
| 1. Navigates to the "Store Management" section within the administrator dashboard. | Presents store information that is                             current and up-to-date. |
| 1. Select the option to view all stores. | Retrieves and presents a complete and accurate list of stores from the database. |
| 1. The system retrieves and displays a comprehensive list of stores, including their IDs, names, addresses, contact details, operational status, and last update timestamp. | Ensures that the displayed information is up-to-date and consistent with the latest data stored. |
| 1. Reviews the detailed list of stores, assesses the information, and identifies any inconsistencies or outdated data. | Offers options for the administrator to sort, filter, or search the store list for specific details. |
| 1. Selects an individual store from the list to access more specific details, such as store performance metrics, recent transactions, or inventory status. | Allows access to additional detailed information upon the selection of a specific store. |

**RULES OF TERMINATION**:

* Automatic Timeout: If the administrator remains inactive within the "Store Management" section for a specified period, the system automatically logs them out to ensure security and data integrity.
* Exit Confirmation: Upon choosing to exit the "Store Management" view, the system prompts the administrator for confirmation to prevent accidental exits and data loss.
* Access Restrictions: If the administrator's access privileges change or are revoked, the system restricts further access to the "Store Management" section to maintain data confidentiality and security.
* System Maintenance: During scheduled system maintenance or updates, the "Store Management" section may temporarily become inaccessible. The system notifies the administrator beforehand and provides an estimated time for resuming access.

**ALTERNATIVE CASES**

* If the system encounters difficulties displaying the store details, it should guide the administrator on potential troubleshooting steps or inform about temporary connectivity issues.

**ADDITIONAL REQUIREMENTS:**

1. Error Handling: Implement robust error handling mechanisms to gracefully manage unexpected system errors encountered during the display of store information. Ensure clear and informative error messages that guide administrators on resolving issues or seeking further assistance.
2. Real-time Updates: Enable the system to provide real-time updates when new store data is added or existing information is modified. Ensure that any changes made to the stores are immediately reflected in the displayed information.
3. Intuitive User Interface: Develop an intuitive and user-friendly interface for the "Store Management" section, ensuring ease of navigation, clear organization of store details, and intuitive controls for sorting, filtering, and accessing store-specific information.

**MESSAGES:**

"Failed to retrieve store information. Please try again later."

### USE CASE: Jose Antonio Bellorin Julio

**USE CASE**: Manage Delivery Information

**ACTORS** : Administrator                                                   **Mode:** Online

**OBJECTIVES** :To enable the administrator to manage delivery information efficiently.

**RULES OF INITIATION**:

1. Administrator logs into the BackOffice system.
2. Administrator accesses the delivery management section.
3. Administrator performs operations such as adding, updating, or deleting delivery information.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. BackOffice Administrator logs into the system with their credentials. | Validates the administrator's credentials to grant access. |
| 1. Administrator accesses the delivery management section from the system's menu. | Presents the delivery management          section for the administrator. |
| 1. Administrator chooses an operation to perform: (a) Add a new delivery; (b) Update existing delivery details; (c) Delete a delivery. | * Guides the administrator through the process to add a new delivery. * Allows the administrator to edit existing delivery details. * Removes the selected delivery from the system upon deletion request. |
| 1. Administrator completes the chosen operation and receives a confirmation message. | Confirms the successful completion of the operation. |

**RULES OF TERMINATION**:

* Administrator successfully completes the desired operation.
* System updates the delivery information accordingly.
* Administrator logs out or exits the delivery management section.

**ALTERNATIVE CASES**

* Invalid credentials during login attempt.
* System error occurs during delivery information update or deletion.
* System displays appropriate error messages and suggests corrective actions.

**ADDITIONAL REQUIREMENTS:**

* Ensure secure access and authentication mechanisms.
* Deliveries' records should be stored efficiently for quick retrieval.
* Error handling must provide clear and actionable messages for effective troubleshooting.

**MESSAGES:**

* "Delivery information successfully updated."
* "Login failed. Please verify your credentials."
* "Error encountered during operation. Please try again."

### USE CASE: Javier de Jesús Alcántara Pérez

**|USE CASE**: Coffe Shop manage orders

**ACTORS** : User (client) , Coffee Shop                                              **Mode:** Online

**OBJECTIVES** :Receive new orders, prepare orders and set orders ready for delivery

**RULES OF INITIATION**:

1. The order must contain products and a delivery partners selected
2. The order must be valid, it means, that the payment process was completed successfully

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1.- Initiation: The user completes an order selecting their products and complete the payment process | The system performs the validation verifying that contains products, these are assigned to the same store, has a delivery partner selected, and the payment process is completed |
| 2.- Once the order is completed, this order arrives at the store selected (status = new order). | The store will receive a notification that a new order has arrived. |
| 3.- The user store can accept or not the order, if the order is accepted the status of the order change to (order in progress) | The user store can select the order and request to move it to the status preparing (order in progress) |
| 4.- When the order is completed the user store must modify the status of the store to ready for delivery, it means, that now the Delivery assigned to the order can pick up the order | The user stores once the order is totally prepared access to the coffee shop menu, search the order and change the status to ready for delivery. |
| 5.- When the order is ready for Delivery, the coffee shop user and the delivery can modify the status to picked up by delivery, It was created in order to avoid waste time. | In the application the Coffee shop user and the delivery selected can modify the status to “picked up by delivery” |

**RULES OF TERMINATION**:

* Success Message: In each change of the status the coffee shop user will see a message that the order changes to the new status and will see on the screen the order details with the new status as a summary.
* Error Handling: If any issues occur during the status change, the user receives a message indicating the issue type and the order maintains the current status.

**ALTERNATIVE CASES**

* Invalid order number: If the coffee shop user looks for an invalid order number, the user will receive a message that the order that tries to modify it is not valid.
* Order ready for delivery: if the coffee shop user finishes to prepare the order and is marked as completed the delivery can go to the store and pick up the order even the coffee shop user did not update the order to ready for delivery, this was created in order to reduce the waste of time when an order is completed (It is the inly case where both actors can modify the same status) .
* Database Connection Issues: In case of any issues with the database connection or if the update on the status process fails due to unforeseen errors, the system alerts the user about the problem and advises them to check the connection or retry the operation.

**ADDITIONAL REQUIREMENTS**

* The coffee shop user must ensure to map correctly the order status in the system against the reality in the coffee shop to avoid inconsistencies.
* Implementing robust error handling to manage various error scenarios during the update process.
* Prioritizing secure and safe data entry into the database, only valid order numbers are accepted and validated.

**MESSAGES:**

* Success Message: "New Order received"
* Success Message: "Order In progress"
* Success Message: "Order ready for delivery"
* Success Message: "Order pick up by delivery"
* Error Message: "Order number is not correct please ensure that you are selecting a valid number.
* Error Message: “Order does not belong to the store, please selecta valid order”

### USE CASE: Javier de Jesús Alcántara Pérez

**|USE CASE**: Add New Client

**ACTORS** : User, System                                              **Mode:** Online

**OBJECTIVES** :To add a new client to the database.

**RULES OF INITIATION**:

1. The system expects valid client details from the user before proceeding with the addition.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. Initiation: The user starts the process by accessing the system interface designed for adding a new client. | The system presents a comprehensive interface for order detail management with options to create, modify, delete, or retrieve order details. |
| 1. Client Details Input: The user provides comprehensive and accurate details of the new client, including email, password, first name, last name, and phone number. | Validation and Insertion: The system  validates the provided client details for completeness and accuracy. |
| 3.   Confirmation: After entering the client details, the user signals to the system that all information has been provided correctly and is ready for insertion. | Insertion into Database: Upon successful validation, the system securely inserts the new client's information into the database. |

**RULES OF TERMINATION**:

* Success Message: The system confirms the successful addition of the new client into the database and displays a success message to the user.
* Error Handling: If any issues occur during validation or insertion (e.g., incomplete details, database connection problems, or insertion failures), the system prompts the user with an appropriate error message and guidance on resolving the issue.

**ALTERNATIVE CASES**

* Invalid or Incomplete Details: If the user provides incomplete or invalid client details, the system prompts the user to re-enter the missing or correct information before proceeding with the insertion.
* Database Connection Issues: In case of any issues with the database connection or if the insertion process fails due to unforeseen errors, the system alerts the user about the problem and advises them to check the connection or retry the operation.

**ADDITIONAL REQUIREMENTS**

* Ensuring the user provides complete and accurate client details for successful insertion.
* Implementing robust error handling to manage various error scenarios during the insertion process.
* Prioritizing secure and safe data entry into the database.

**MESSAGES:**

* Success Message: "New client added successfully."
* Error Message: "Failed to add the new client. Please ensure valid and complete client details are provided and try again."

### USE CASE: Javier de Jesús Alcántara Pérez

**|USE CASE**: Delete Client by ID

**ACTORS** : User, System                                              **Mode:** Online

**OBJECTIVES** :To remove a client from the database based on their provided ID.

**RULES OF INITIATION**:

1. The system anticipates receiving a valid client ID input from the user.

**DESCRIPTION OF THE MAIN SCENARIO**:

|  |  |
| --- | --- |
| **Actor** | **System** |
| 1. Initiates the deletion process by providing a client ID to delete a specific client. | Receives the client ID for deletion from the user. |
| 1. Confirms the deletion request of the identified client. | * Executes a SQL delete command to remove the client associated with the given ID from the database. * Verifies the successful deletion of the client entry. |

**RULES OF TERMINATION**:

* Successful Deletion:

Displays a confirmation message indicating successful client deletion.

* Error Handling:

If the system encounters an issue during deletion (e.g., invalid ID, database connection error), an error message is displayed.

**ALTERNATIVE CASES**

Invalid Client ID:

* If the provided client ID is invalid or not found in the database, the system prompts the user to enter a valid client ID for deletion.

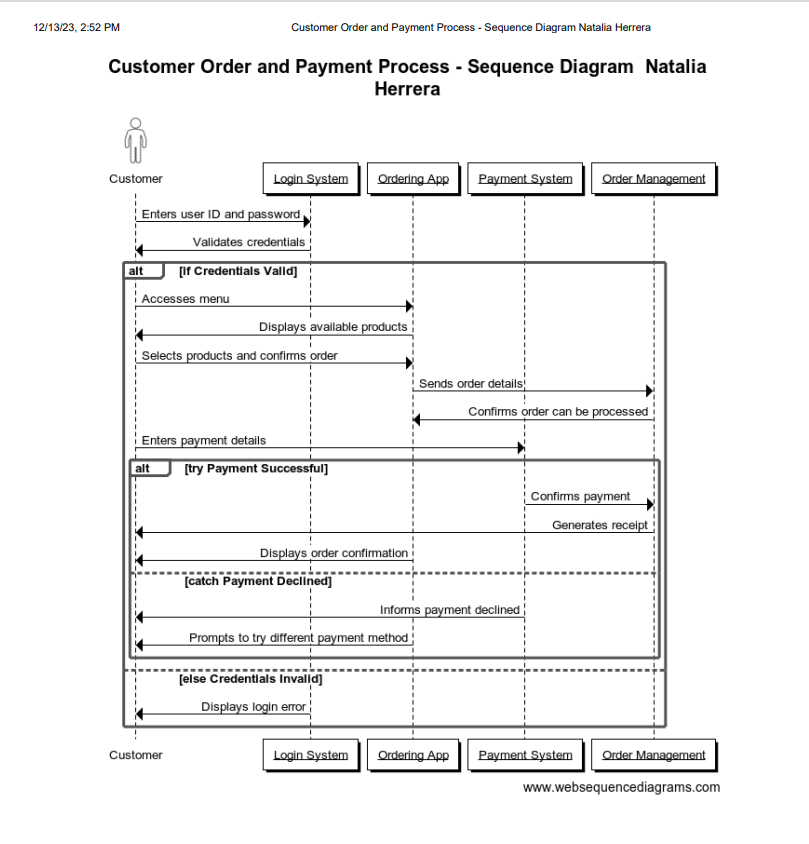
**ADDITIONAL REQUIREMENTS**

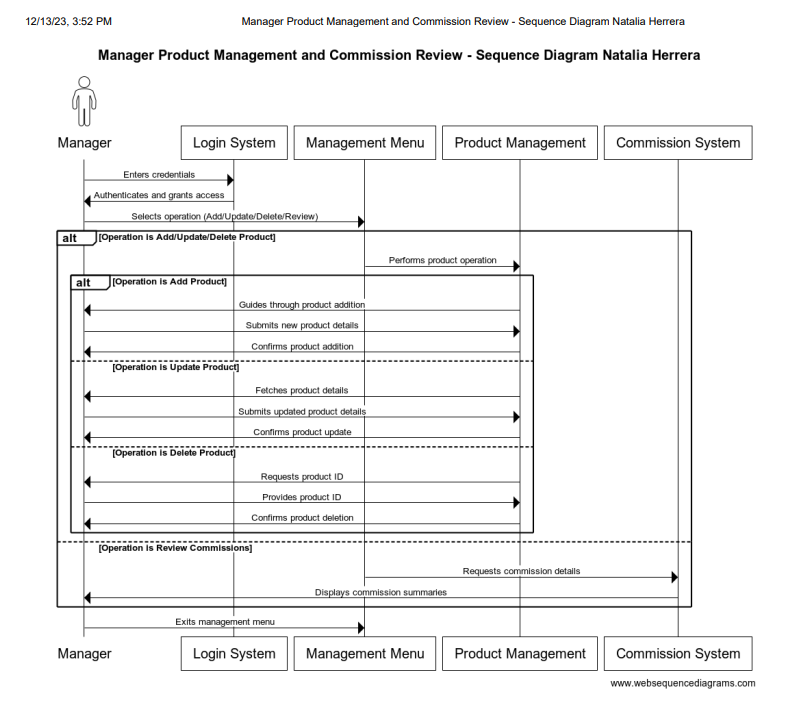
* The system should enforce validation to prevent deletion without a valid client ID.
* User permissions and access rights must be verified before allowing the deletion process to proceed.

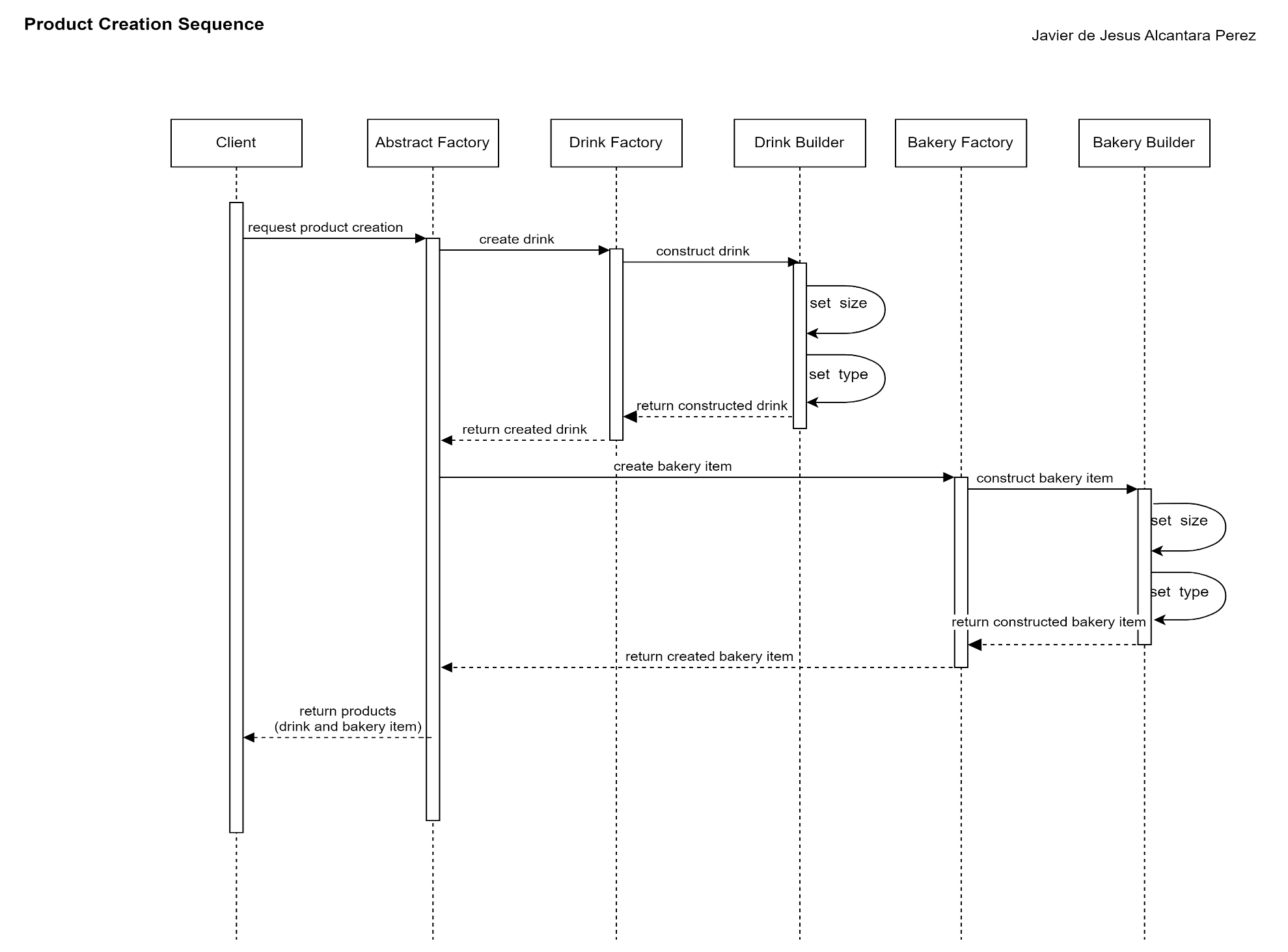
**MESSAGES:**

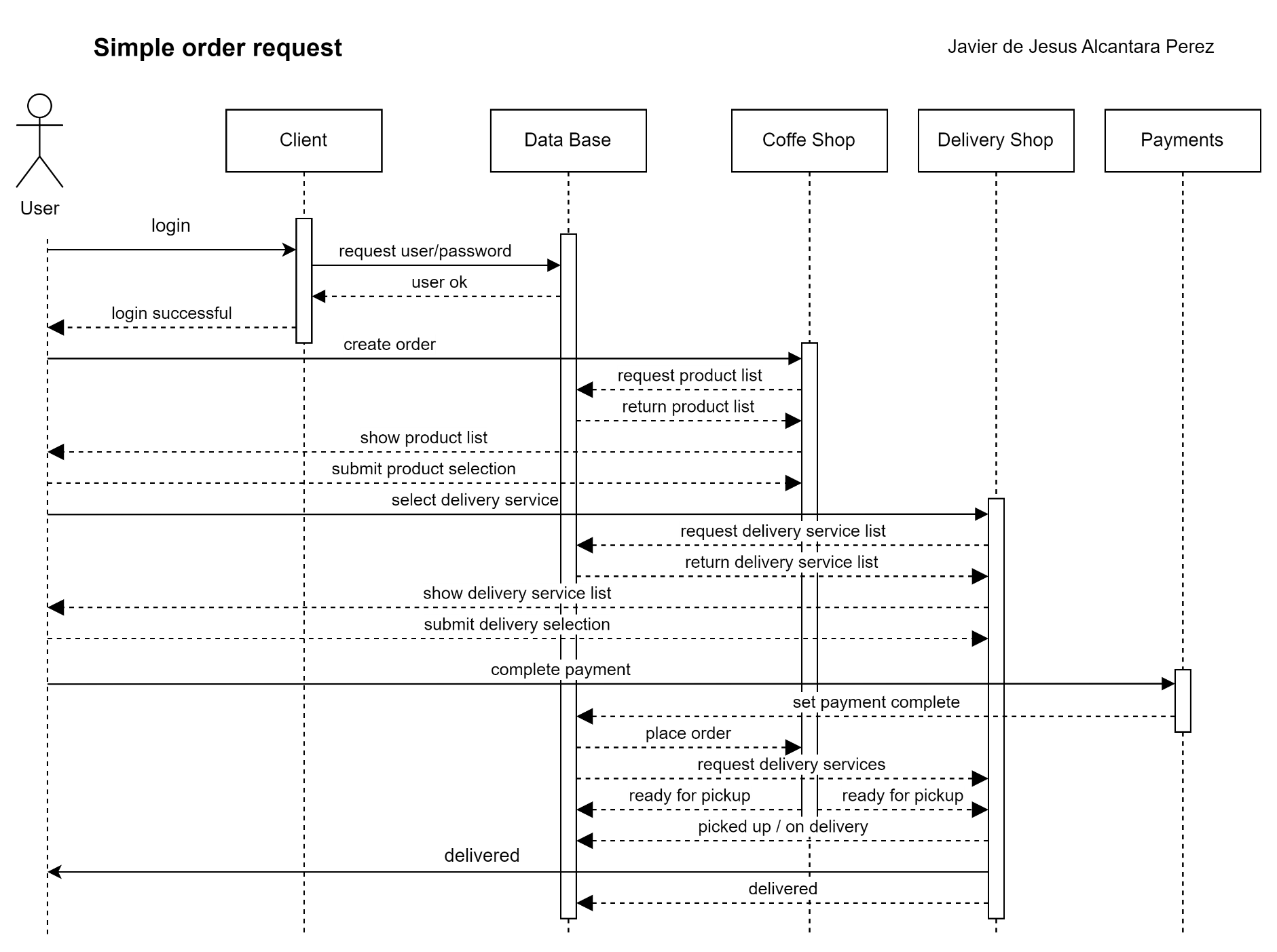
* Success: "Client with ID [Client\_ID] has been successfully deleted."
* Error: "Failed to delete the client. Please ensure the provided Client ID is correct and try again."

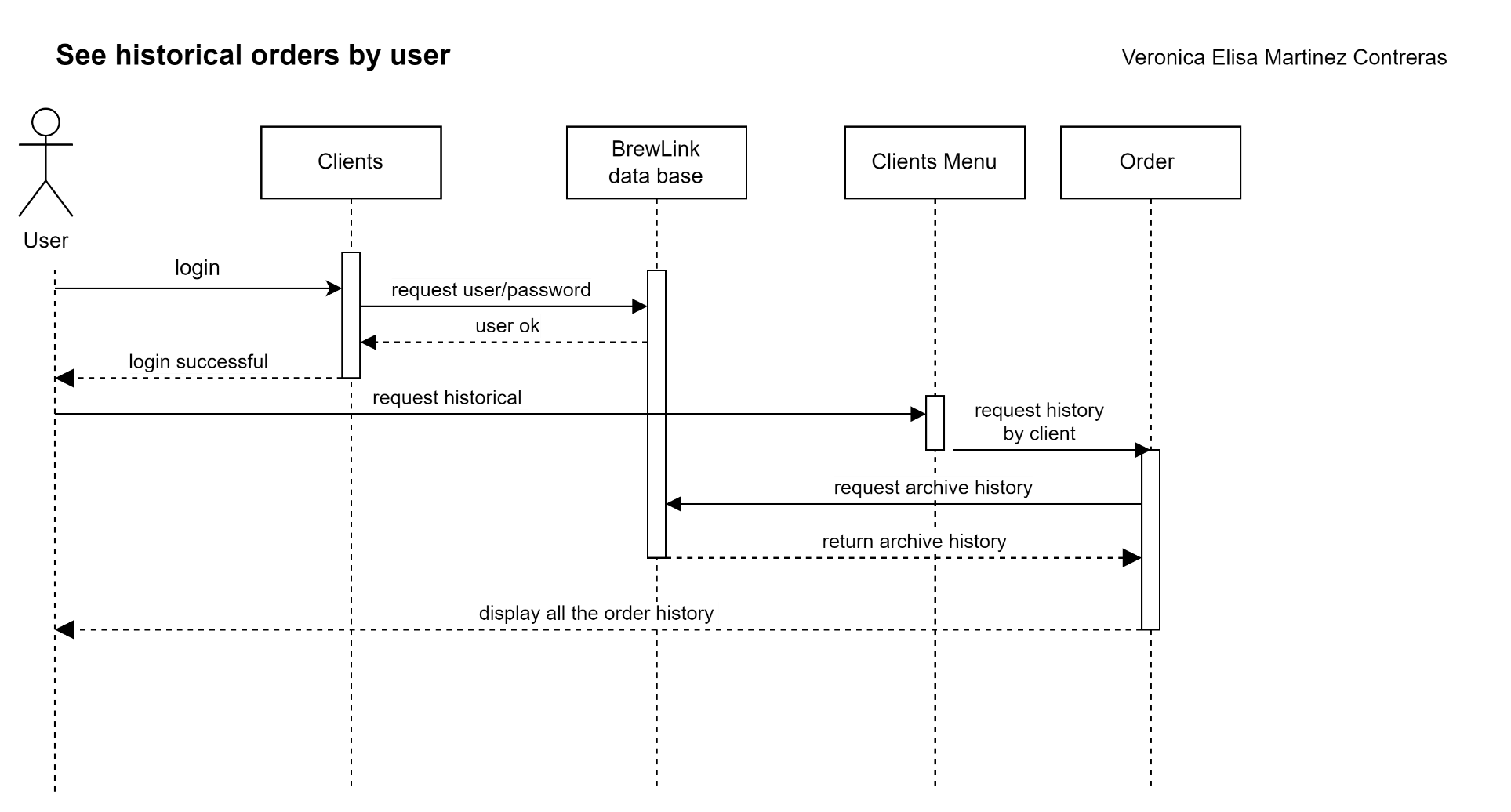
# Sequence Diagrams

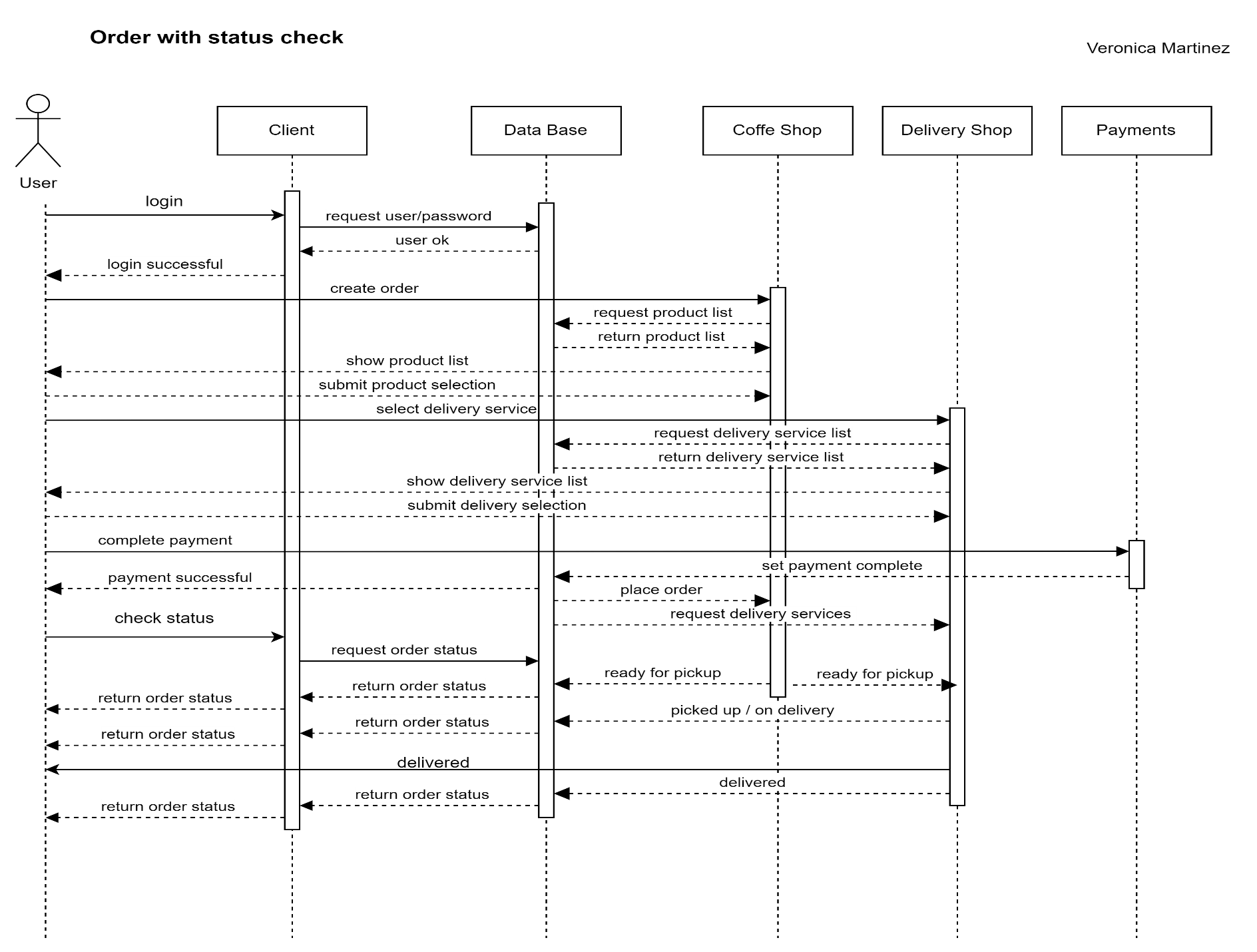






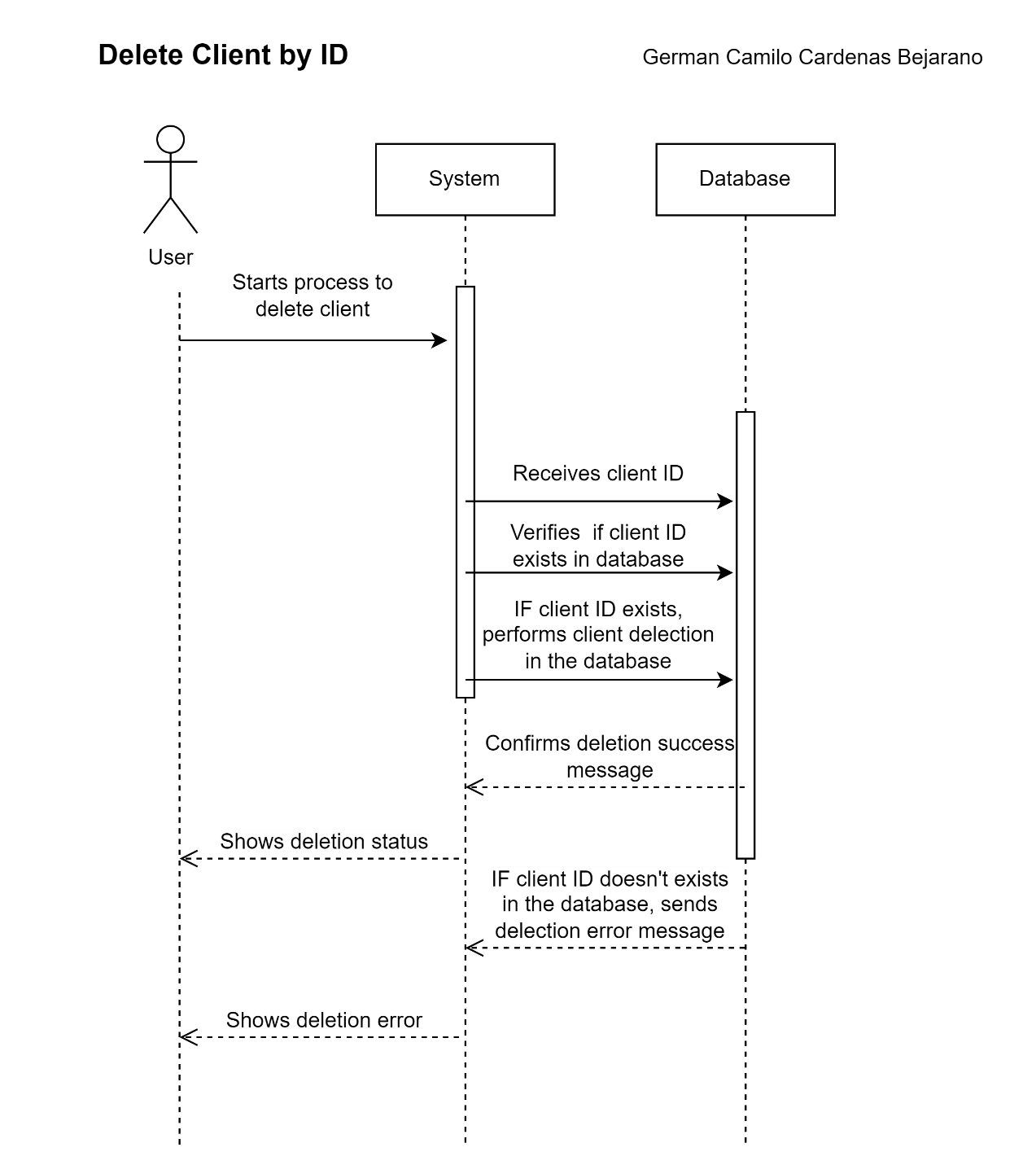


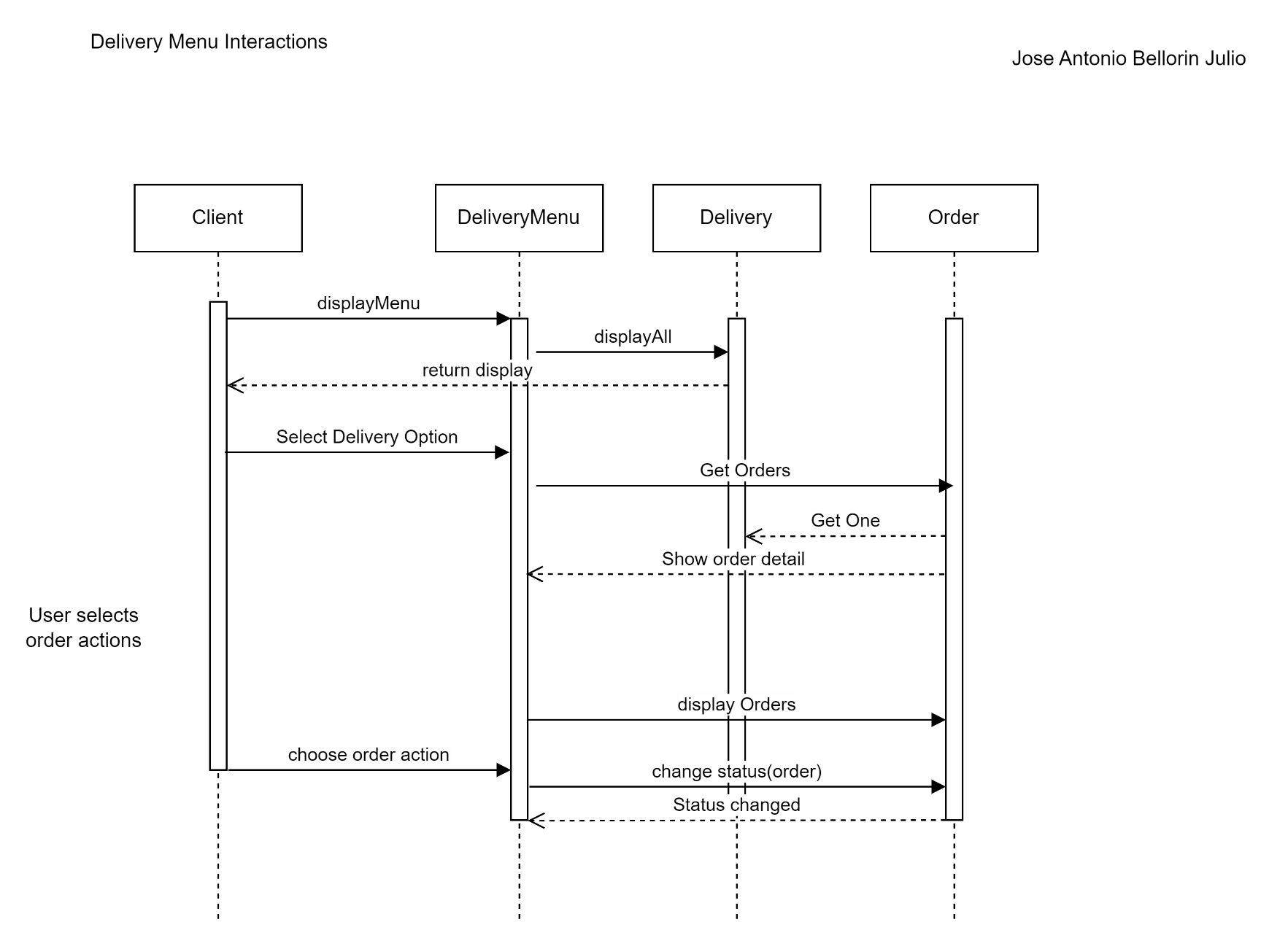




A black screen with white text

Description automatically generated





# Challenges

In the conceptualization of the solution and the step-by-step execution prior to the code implementation, we encountered significant challenges in delineating the scope of the idea, considering our level of knowledge and expertise in project formulation.

Graphically representing the functionalities expected to be implemented in the project, using use case, sequence, and class diagrams, posed initial difficulties in comprehending the usage of each diagram and the concepts they ought to represent. Eventually, through teamwork, we managed to conclude the task by applying the knowledge acquired in class and during the project execution.

Regarding databases, we faced several challenges to resolve in establishing the connection and meeting each of the application's requirements, such as creating a new client, deleting an already created client, and ensuring that information was managed both from the program and the database. In this regard, SQL Lite facilitated the task as being a compact database, allowing practical data access with favorable execution times.

Concerning different design patterns, we initially struggled to relate them according to the project's needs. However, we succeeded in identifying common points based on the created classes, data integrity, and objects.

Wewanted to organize the code in the best possible way, for that reason we investigated the MVC model and applied it, also once that we abstracted all the functionalities of the projects we identified how to implement Status design pattern, Factory design pattern, builder design pattern and singleton as part ot the retro that we had in our presentation.

# Patterns

## Singleton Pattern

The Singleton pattern is used to restrict the instantiation of a class to a single object. In other words, its main goal is to ensure that a class has only one instance and provide a global access point to that instance. We apply with the data base connection.

Private Constructor:

private Data Waiter () {}

Unique Instance:

private static Data Waiter instance = new Data Waiter();

Global Access:

public static Data Waiter Get Instance() {

return instance;

}

## State Pattern

OrdersState (Enum):

Enumerates the different possible states of an order, such as order generation, preparation, delivery, and more, providing a clear and defined representation of these states.

State (Abstract Class):

Defines a common structure and base methods to represent order states.

Contains an attribute current State, likely of type Orders State, maintaining the current state of the order.

Provides methods to get and set the description of the state (get Desc Status() and set Desc Status()).

Declares an abstract method change Status(Order order) responsible for changing the order's state. This method will be implemented by concrete classes representing specific states.

Concrete state classes (Delivered, In Route, Order Generated, Order State, Picked Up, Preparing, Ready For Delivery):

These classes extend the State class and provide specific implementations for the change Status() method, defining the behavior that occurs when an order is in that specific state. For instance, the Delivered class might handle the behavior when an order is marked as delivered.

Orders State enumerates potential order states, while State offers a common and abstract structure to represent and manage these states. Concrete classes inherit from State and define specific behaviors for each individual order state, allowing for a more structured and modular management of the order lifecycle within the order management system.

## Abstract Factory and Builder

The Abstract Factory pattern is used to create families of related objects without specifying their concrete classes. In our case, we have two concrete factories: DrinksFactory and BakeryFactory, each responsible for creating specific products for beverages and bakery items, respectively. Both factories are subclasses of AbstractFactory, which declares the methods for creating the products.

On the other hand, the Builder pattern is used to build an object step by step. In our code, BuilderCoffeeCultureProduct, BuilderSecondCupProduct, etc., are Builder classes that create specific products for each type of beverage or bakery item.

The concrete factories (DrinksFactory and BakeryFactory) implement the abstract methods of the AbstractFactory class to create specific products of each type. Each method in the concrete factories creates a corresponding Builder object (BuilderCoffeeCultureProduct, BuilderSecondCupProduct, etc.). The Builder objects are responsible for the step-by-step construction of the products, setting their attributes such as product name, price, commission, etc. Once the product is built, it is added to the database using the ProductsWaiter class.

In summary, the Abstract Factory organizes the creation of families of related products, while the Builder is responsible for the step-by-step construction of these products, separating their construction from their final representation.

# Conclusions

The "Brewed Link" project embodies an innovative integration among four distinct coffee shops and three delivery companies. This collaboration offers a unique experience by blending the variety and quality of beverages and pastries from multiple cafes with the convenience of direct orders for home or office delivery.

The ClientsWaiter and Products Waiter interfaces provide efficient methods to interact with the database regarding client and product management. This capability allows for easy retrieval, display, insertion, and retrieval of specific information, providing an abstraction layer between operations and the database logic.

The project implements design patterns like Abstract Factory and Builder to create products and manage object construction flexibly and systematically. The use of these patterns offers a more maintainable and scalable way to handle product creation and manipulation.

The defined use cases in the project encompass a wide range of interactions between users and the system, from order management to menu navigation and customer account administration. These use cases are well-detailed, offer clear messages, and handle error situations appropriately.

There's a clear focus on user experience, particularly in use cases related to clients and order management. The application provides clear messages, manages errors informatively, and allows intuitive interactions to facilitate use and understanding for the end user.

# Bibliography

* Refactoring.Guru. (2014-2023). Design Patterns. All rights reserved.
* OpenAI. (2022). ChatGPT.