



## **Assignment1**

### **Food Delivery Application**

Dragan Iulia-Andreea, 30433/1

## Table of Contents

1.	Requirements Analysis.....	3
1.1.	Assignment Specification .....	3
1.2.	Functional Requirements .....	3
1.3.	Non-Functional Requirements .....	4
2.	Use-Case Model .....	4
3.	System Architectural Design .....	6
4.	UML Sequence Diagrams .....	6
5.	Class Design.....	11
6.	Data Model .....	13
7.	System Testing .....	14
8.	Bibliography .....	18

# 1. Requirements Analysis

## 1.1. Assignment Specification

The purpose of this assignment is to become more familiar with the most used and common architectural patterns by designing and implementing an application which highlights the features of those architectures.

The main objective is to develop a food delivery application in Java, which allows for the existence of two types of users, i.e. the administrator and the customer, who have to provide a username and a password upon logging into their account.

The application will allow the user (administrator or customer) to perform a series of Create/Read/Update/Delete (CRUD) operations on a certain database, when giving a correctly defined input. Further details will be presented below.

## 1.2. Functional Requirements

The database of the system simulates in a brief manner the behavior of a food delivery application, containing a list of customers, products, and orders / carts which have been placed or are currently pending. The administrator is offered an overview of the entire system and can modify the structure and the composition of a certain entry in the table, be it a customer or a product.

The goal of the program is to perform CRUD operations on a given dataset as well as some additional operations. Therefore, the two existing users have different abilities which can generate a multitude of outputs. It is essential to note that all inputs must be validated before inserting them into the database.

	Registration	CRUD	Other
<b>Customer</b>	Add / Create information through logging in and signing up  Use the obtained information to gain access to the database	CRUD on personal account  CRUD on personal shopping cart / order	Choose a payment method  Visualize historical data, i.e. past orders
<b>Administrator</b>	User the preexisting data in the database to gain information about the rest of the tables by logging into the account  Cannot change account information	CRUD on customer information  CRUD on products and menu information	Generate activity reports given a certain period of time  Set customer as loyal to offer them a discount

### 1.3. Non-Functional Requirements

In terms of non-functional requirements, the constraints are minimal. It was essential to use a programming language which allows for object-oriented programming to efficiently model the database. Therefore, Java was the obvious choice, since it is possible to enhance its database communication features by using the Hibernate framework.

Other constraints include the existence of a Layered architectural pattern to organize the packages and the classes of the application, therefore a clear distinction between the packages had to be made. This will be further discussed in the System Architectural Design chapter.

In terms of Maintainability and Reusability, while the requirements do not state a necessity for them, the layered architectural pattern does offer a more flexible usage and helps changes to propagate from one class to another.

The response time is minimal, with a longer waiting time for the database setup. Most of the resources are consumed when performing intensive computational and database operations.

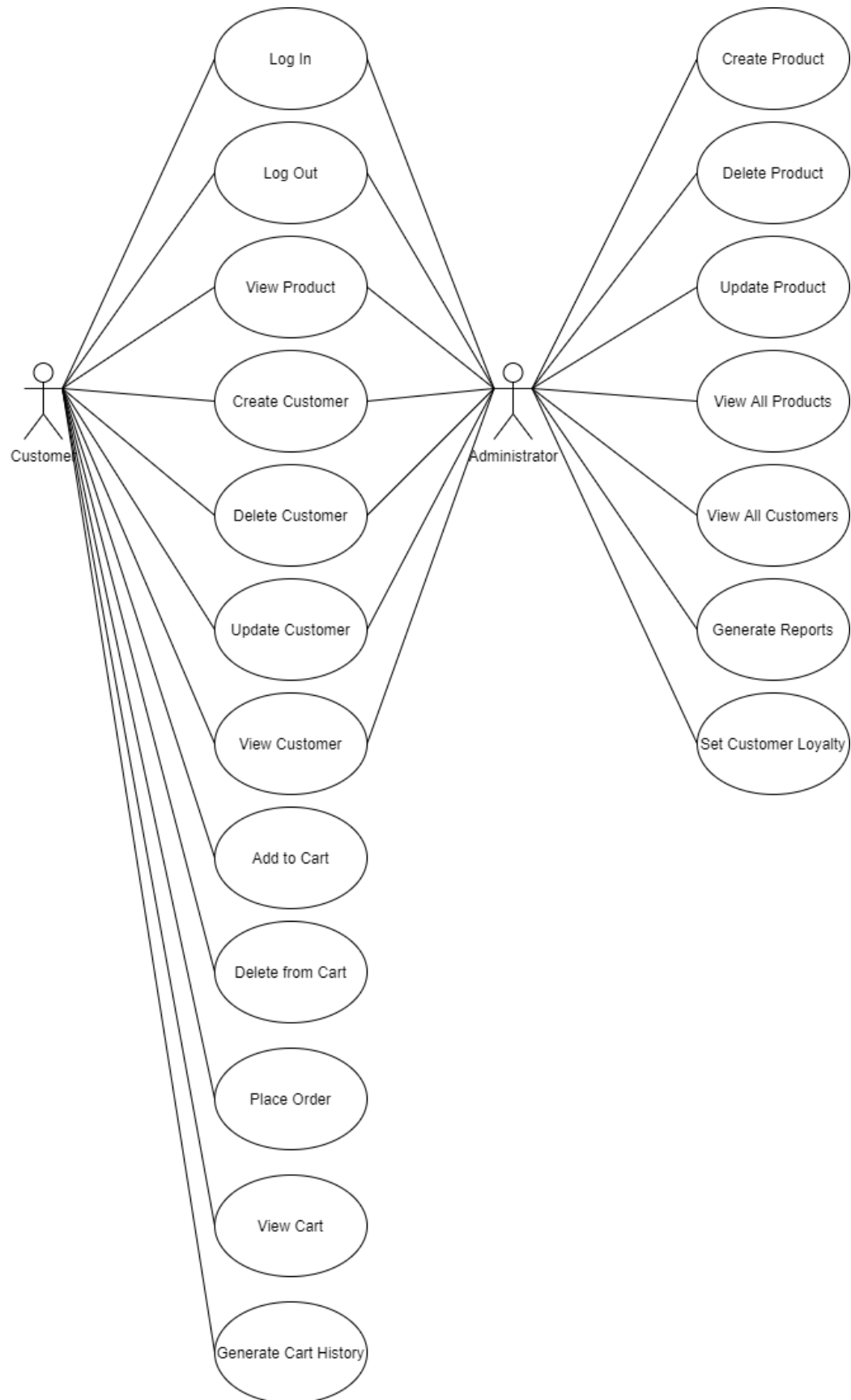
## 2. Use-Case Model

Since the application contains only two types of actors, the use-case model is quite simple. As mentioned above, both users can perform registration, CRUD and additional operations based on the information available on the database. In order to be more specific, we can assign the following abilities to the users.

Administrator: log into account, perform CRUD operations on customer data, perform CRUD operations on menu data, set a customer as loyal (offer them a 5% discount), generate reports for a particular period of time.

Customer: create account, log into account, perform CRUD operations on personal data, perform CRUD operations on the cart and the order, choose a payment method and visualize historical data (past orders).

The user interface is minimal and it is simply a way to exemplify the correctness and the functionality of the program to a third-party user. It is intuitive and it allows the user to experiment and to explore the application and its features.



### 3. System Architectural Design

In the case of this application, one of the requirements was to use a layered architectural pattern in order to design and implement the program. In order to obtain this, the most logical choice was to use a data source hybrid pattern (in our case, table module) and a data source pure pattern (table data gateway) to organize the structure of the application.

The layers of the application correspond to an architecture containing the following: presentation layers, for the user interface, business layers for the logic of the program and persistence layers for the communication with the database.

**Database:** The database layer, which contains the tables storing the information about the users and the additional objects, such as the carts and the products.

**Model:** Each class in this layer corresponds to a certain table in the database and each model is a perfect representation of the existing information from the database, with all its available fields. This is a clear example of the usage of the patterns mentioned above.

**Repository:** This layer contains all the possible operations which can be performed on the database, by performing the queries necessary to store and retrieve information from into and from the database.

**Service:** The classes from this layer correspond to the business logic of the application, further enhancing the classes from the repository and allowing for input validation and for a clear separation between the controller / presentation layers and the database layer.

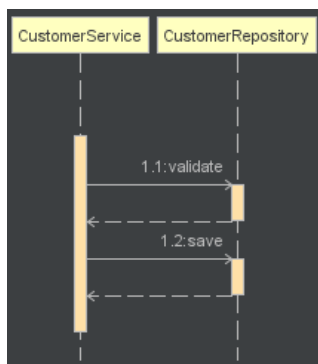
**Controller:** This layer controls the activity of the presentation layer, by obtaining inputs written in by the users and calling the methods necessary to perform the desired operation.

**Presentation:** This layer contains the user interface, which displays the possible outputs and allows the users to input their data. This layer does receive input directly from the Service layer, but it cannot perform any changes.

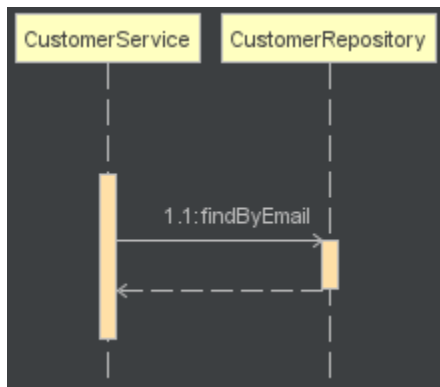
### 4. UML Sequence Diagrams

Below are presented a collection of sequence diagrams which help to exemplify the execution flow of the application.

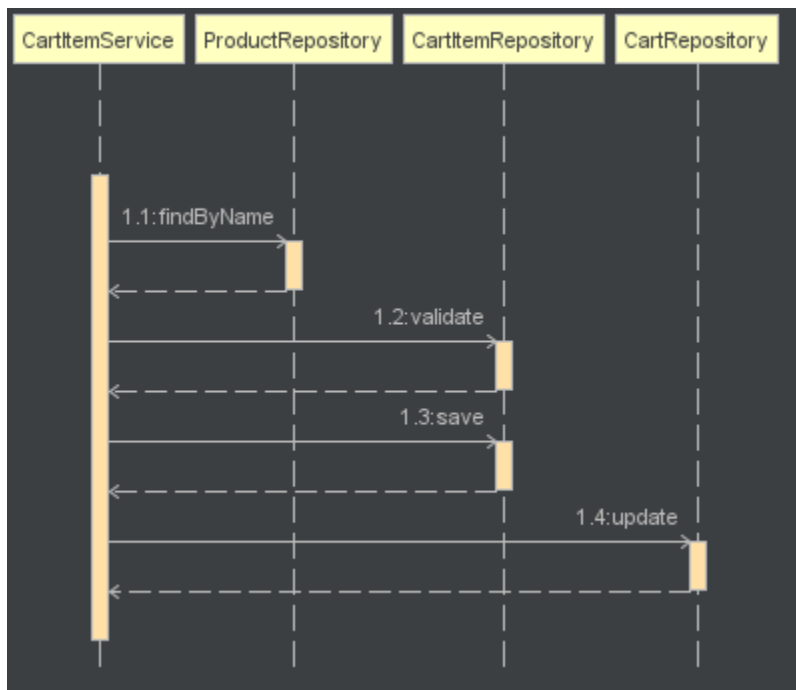
//customer registration



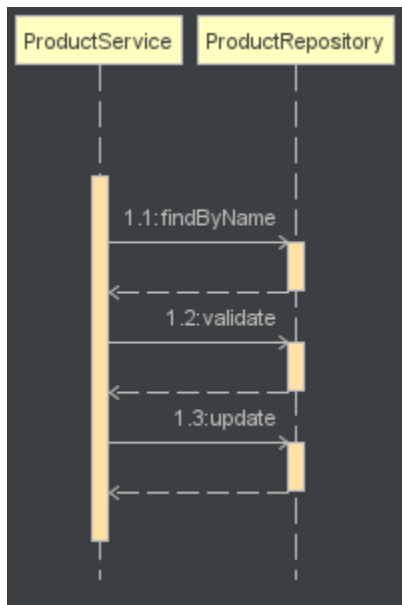
//customer log in



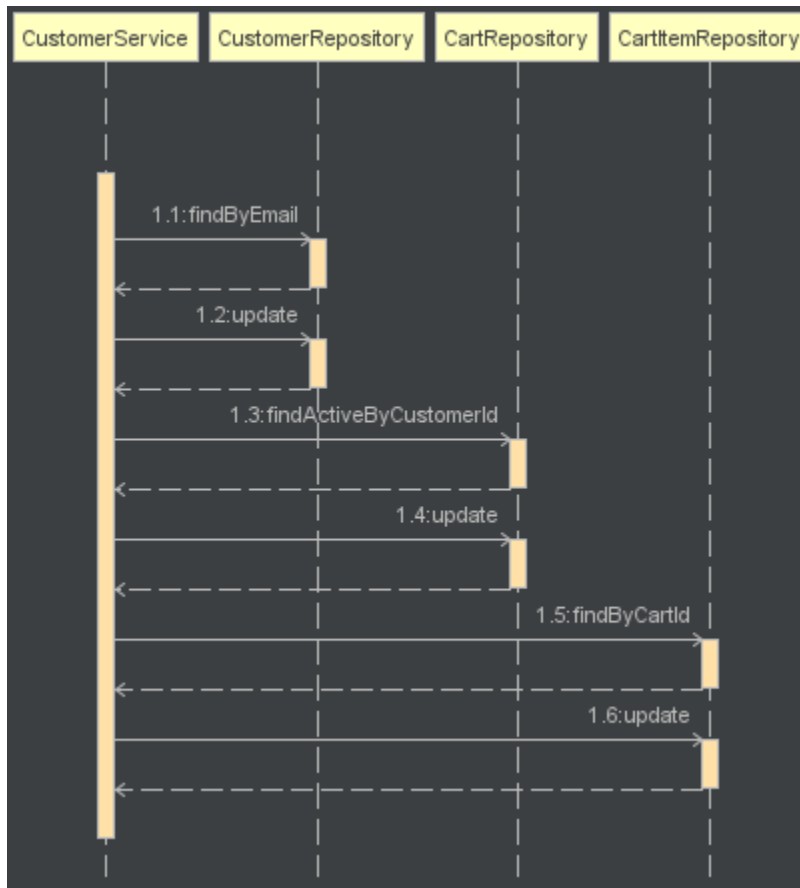
//add to cart



//update product

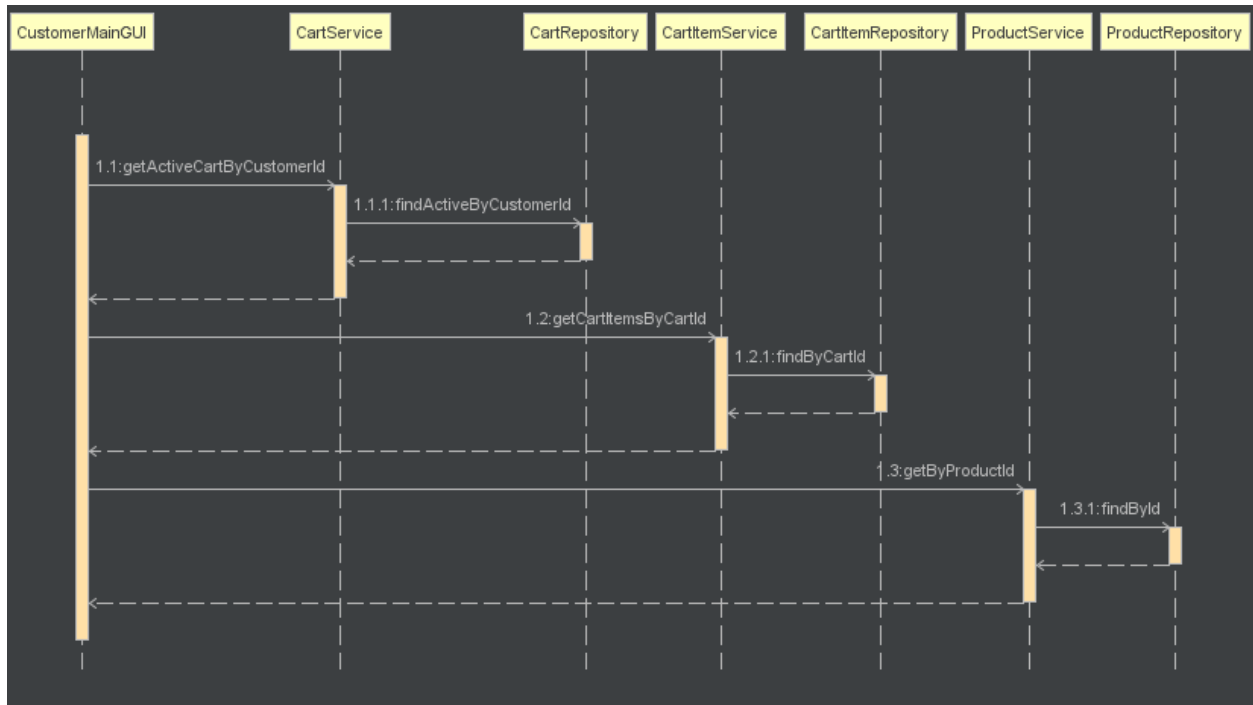


//delete customer

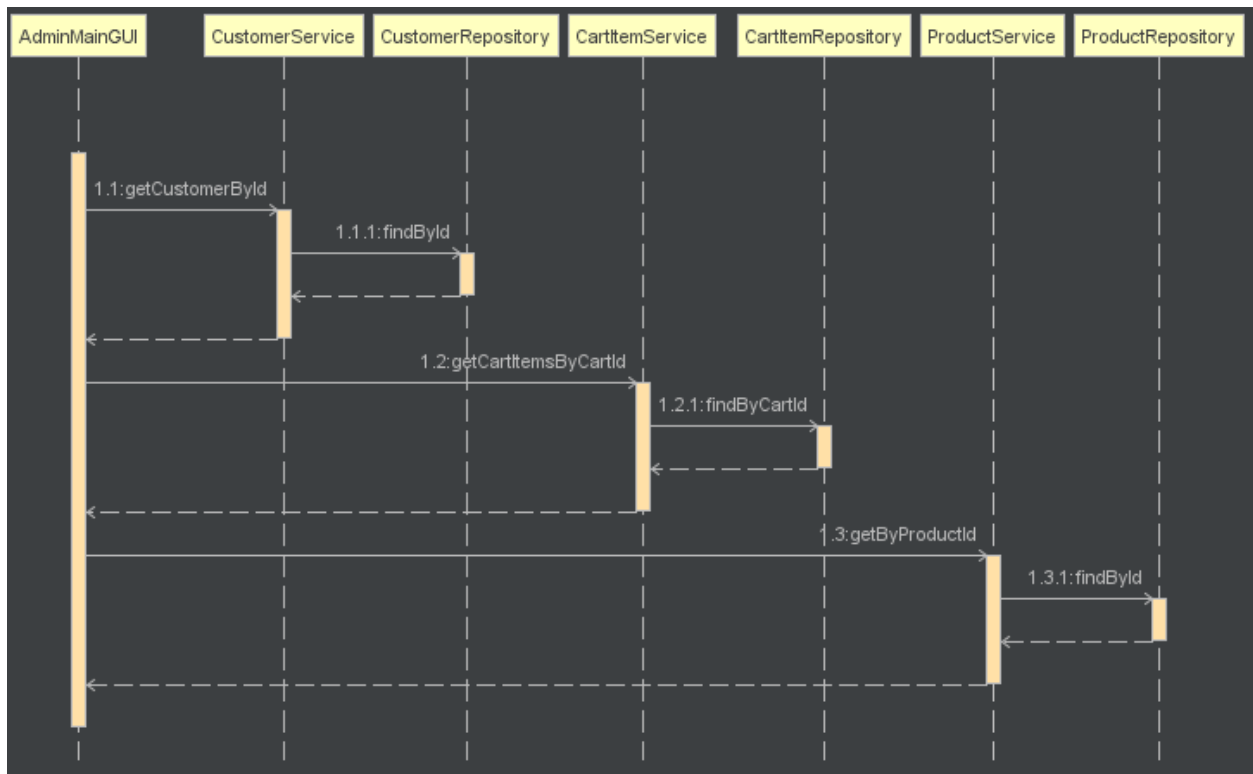




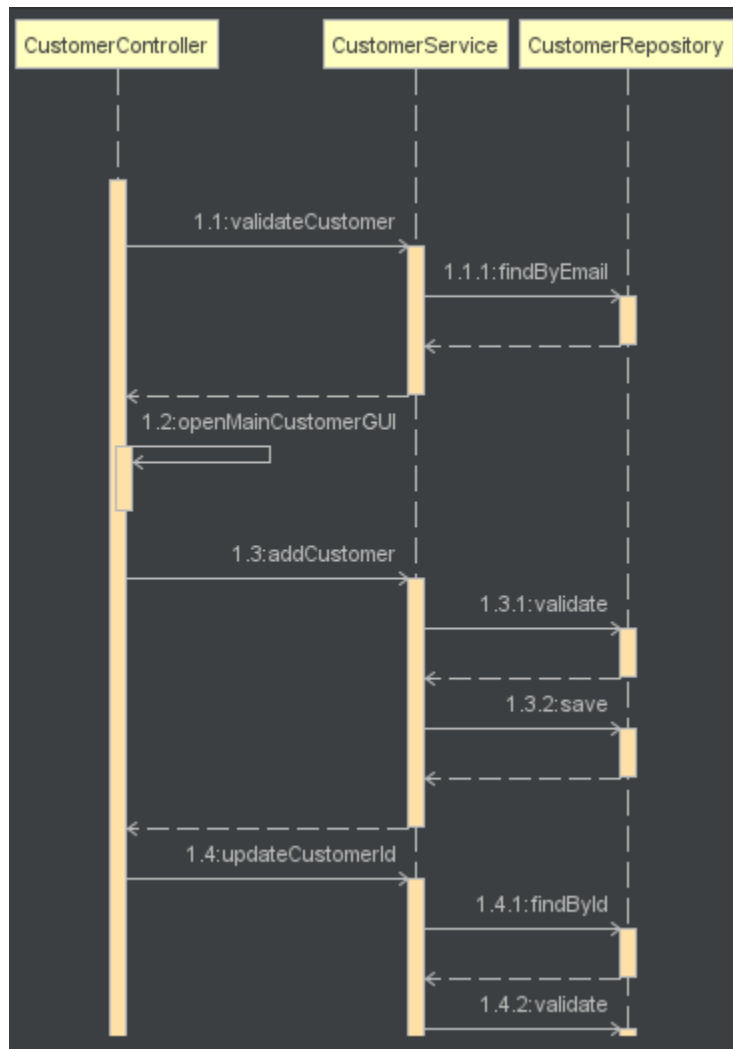
//view cart



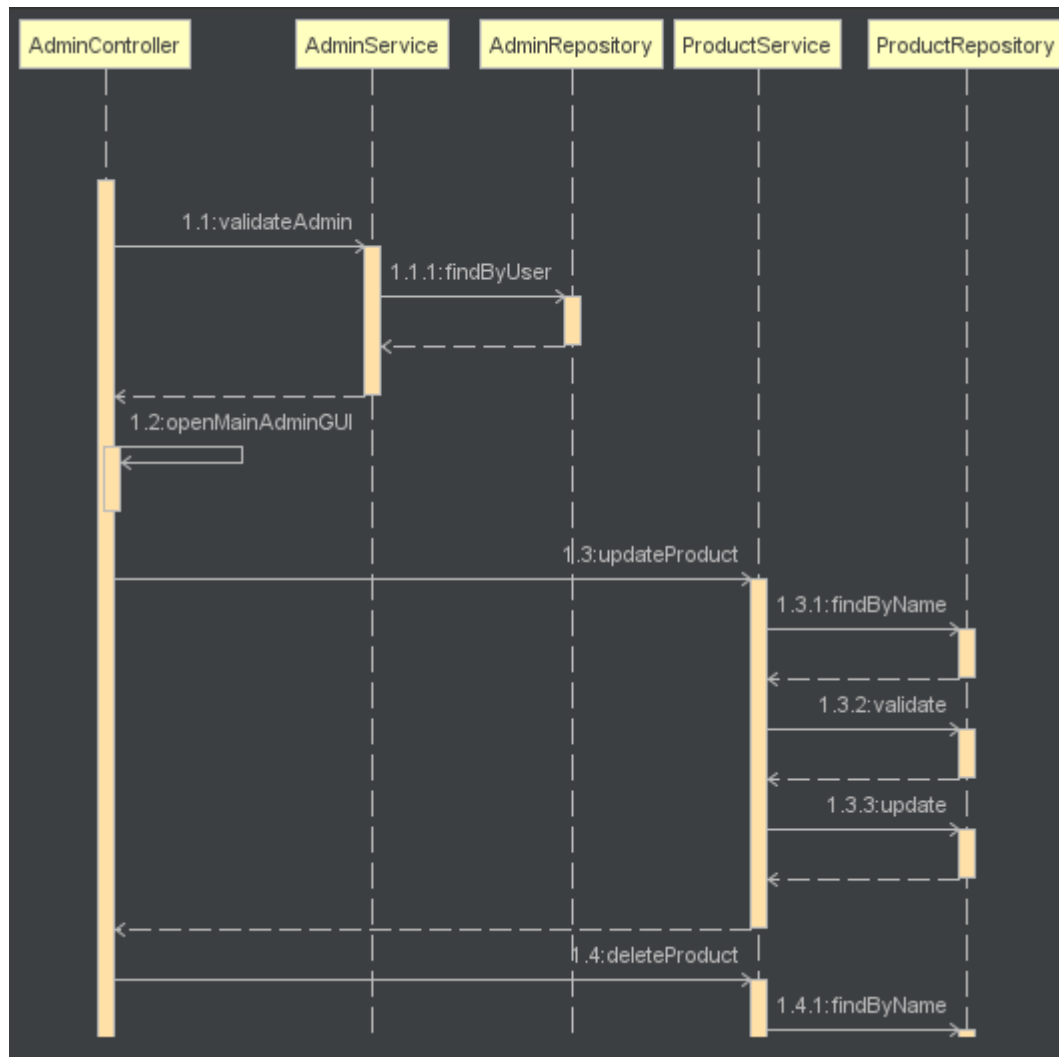
//reports



//customer controller



//admin controller



## 5. Class Design

This section provides a more in-depth view of the packages and the classes of the program by offering more information on the functionality of each class and particular design patterns, where it is the case.

The application contains a total of six packages, each with a well-defined role and structure.

**App:** Contains two classes, HibernateService and Main. The first one ensures the connection to the database through the use of the Singleton design pattern. The second one contains the main method of the program, which initializes the two registration windows for the customer and the administrator.

**Presentation:** Contains four classes, one for each of the available frames, i.e. the two registration frames and the two post-registration frames, which allow the user to perform the desired operations on the database as well as data visualization.

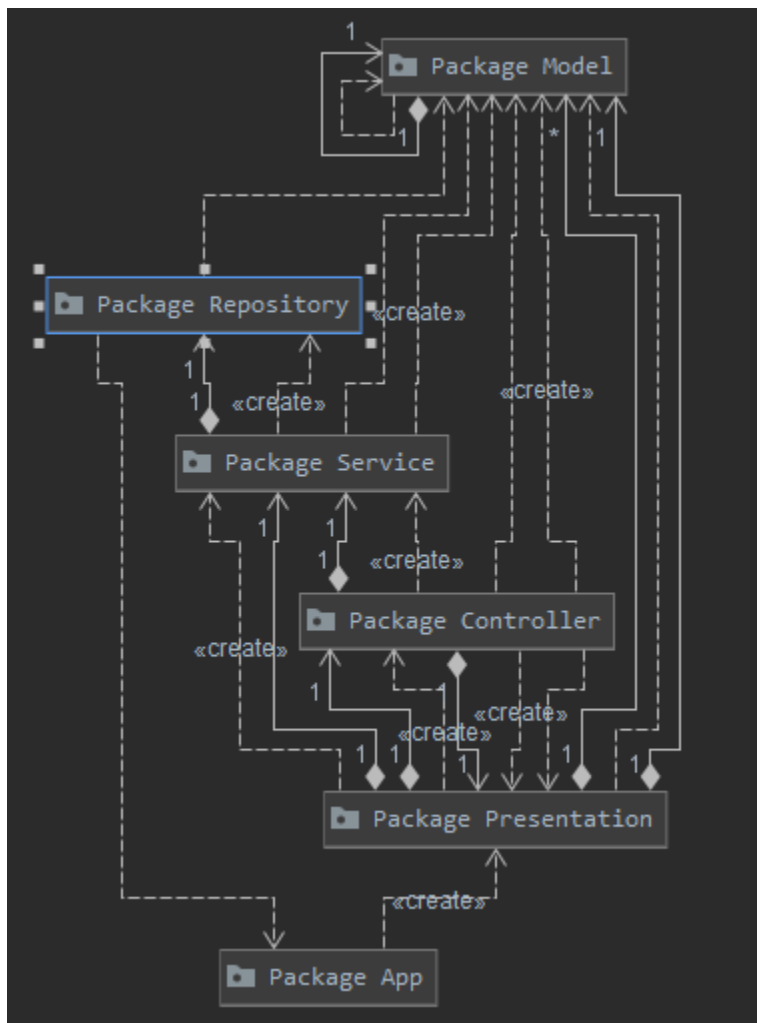
**Controller:** Implements the ActionListeners for the buttons in the previous package and calls the necessary function from the Service package to connect to the database and perform them.

**Service:** It holds the logic of the program, i.e. addition validations to raw database operations, in order to ensure a correct input and legal changes. It also holds the methods responsible for the user validation when logging into their account, by matching the username / email and password given as an input to the one available in the database.

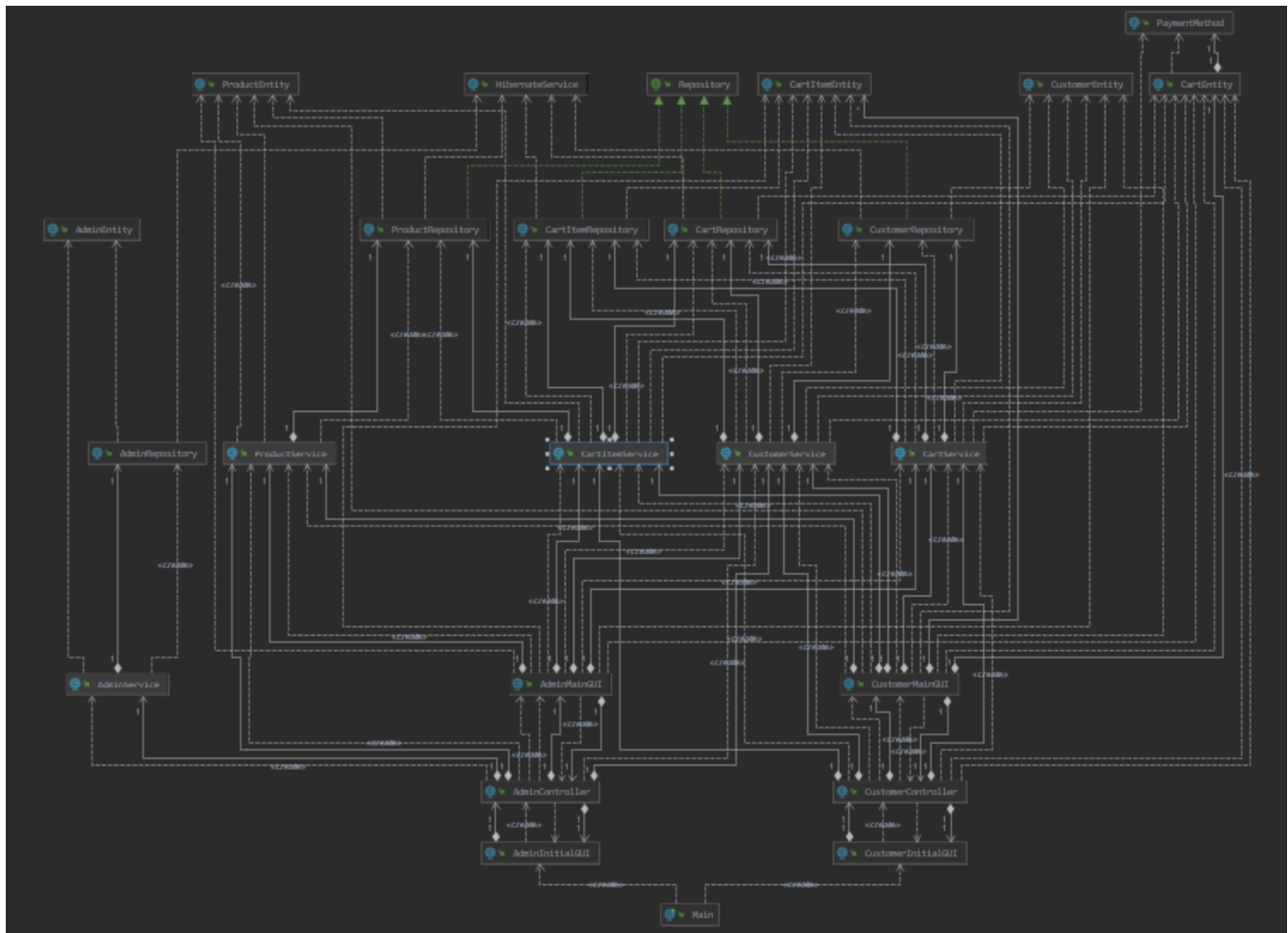
**Repository:** Contains all the classes which directly execute queries on the database. All classes implement a Repository interface, in order to ensure the same basic functionality. In addition to those functionalities, each repository class contains additional CRUD operations based on fields specific to that particular model / table.

**Model:** This package stores all the classes which are directly mapped to tables in the database, i.e. Admin, Cart, CartItem, Customer, PaymentMethod, Product. It contains all the corresponding fields from the database, as well as getter and setter methods and validation annotations for each field.

An overview of the packages and the relationship between them is presented below:



A more in-depth view of the class design of the application and the links between each of the classes:



## 6. Data Model

In order to implement all the required operations and have a clear distinction between each of the objects as well as the information in the database, there are five tables which store the data. The entities of the program are as follows:

**Admin:** contains a single entry, which cannot be edited from the user interface containing the administrator information. To be more specific, it is composed of four fields, one of which is irrelevant in our case: id, username, password and deleted. The deleted field is not used in our case, but it will be explained when elaborating on the other tables. The username and the password fields are the two required to log into the administrator window.

**Customer:** contains customer objects and customer-related information, i.e. id, name, email, password, address, loyalty and the deleted flag. The two fields required to enter the customer window are the email and the password, as the former one is a unique field and there cannot be two customers with the same email. The loyalty flag is by default set to false, and it can only be changed by the administrator.

**Cart:** contains all the existing carts with the necessary information, as well as a method through which to verify if the order has been placed or if the cart is still incomplete or pending. Therefore, it contains the following fields: id, idclient (to make the connection to the customer table), total price, date, payment method, completed, deleted flag. The second to last one is the one which states if the order was placed.

**CartItem:** contains a list of all the items which are currently stored in cart or which were part of a past order. It has the following fields: id, idcart (since it is in a relationship of one to many with the cart model, which can contain multiple cartitem objects), idproduct (a direct link to the product table), quantity, intermediate price (price of the product \* quantity), and the deleted flag.

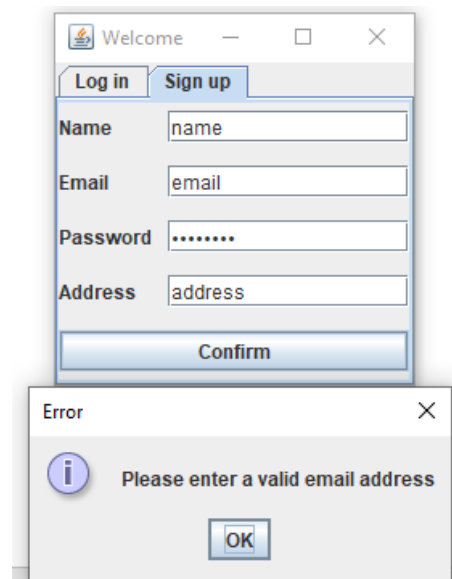
**Product:** contains all the products which are currently available in stock. It is made of the following fields: id, name, price, description, deleted flag. Only the administrator can modify the data available in this table.

In terms of the **deleted flag**, it is a method to safely delete an entry from the database. Instead of fully erasing it and possibly generating linking errors between the tables, it is much easier to maintain a flag such as this one. When set to true, the respective field is considered to be deleted from the database and may not be taken into consideration in further operations. This implementation not only helps maintain a history of transactions of the database, but it also makes it possible to cascade the deletion through tables without risking information loss.

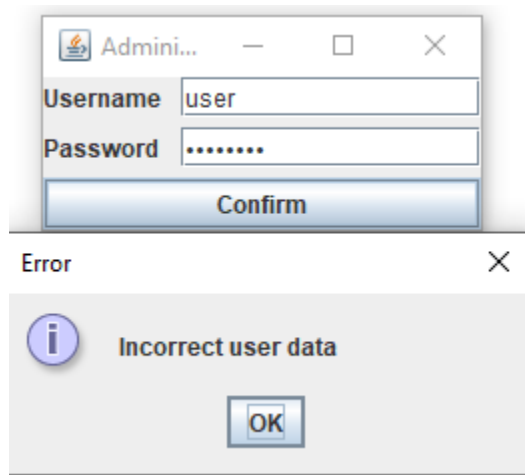
## 7. System Testing

The application works as expected when performing all the required operations: registration, CRUD and the mentioned additional functionalities. System testing was mostly performed through the use of the interface. Below are presented a series of examples:

//sign up

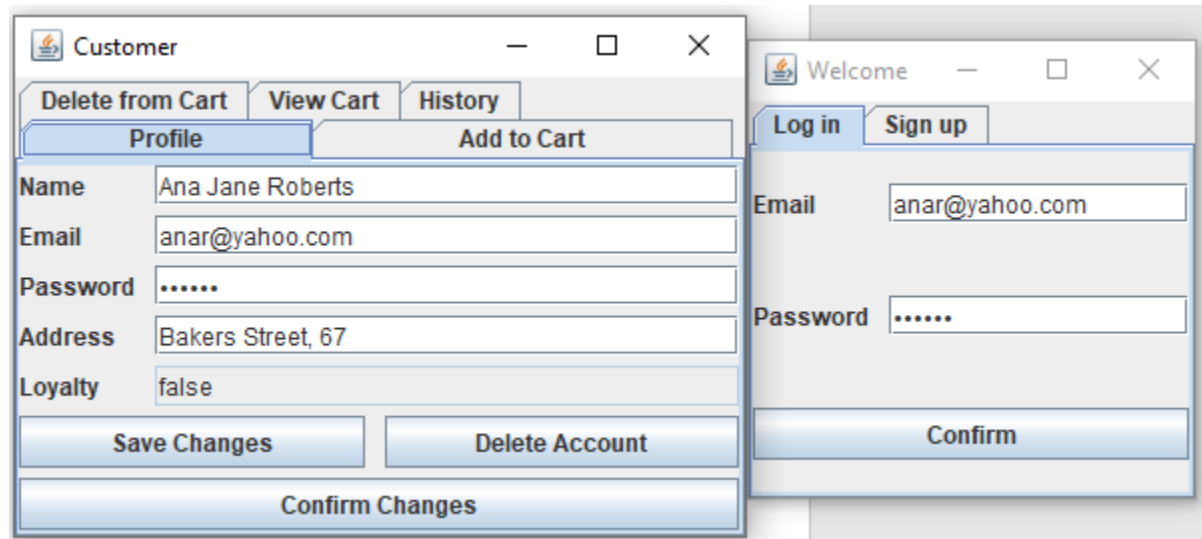


//incorrect log in

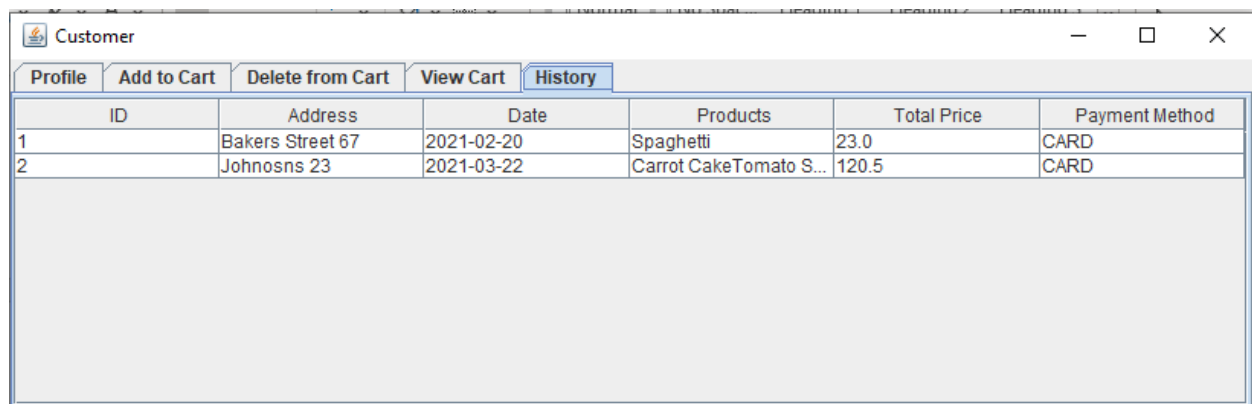


The image shows an 'Admin...' login window with fields for 'Username' (containing 'user') and 'Password' (containing seven dots). A 'Confirm' button is at the bottom. Below it is an 'Error' dialog box with an information icon and the text 'Incorrect user data', with an 'OK' button.

//customer panels



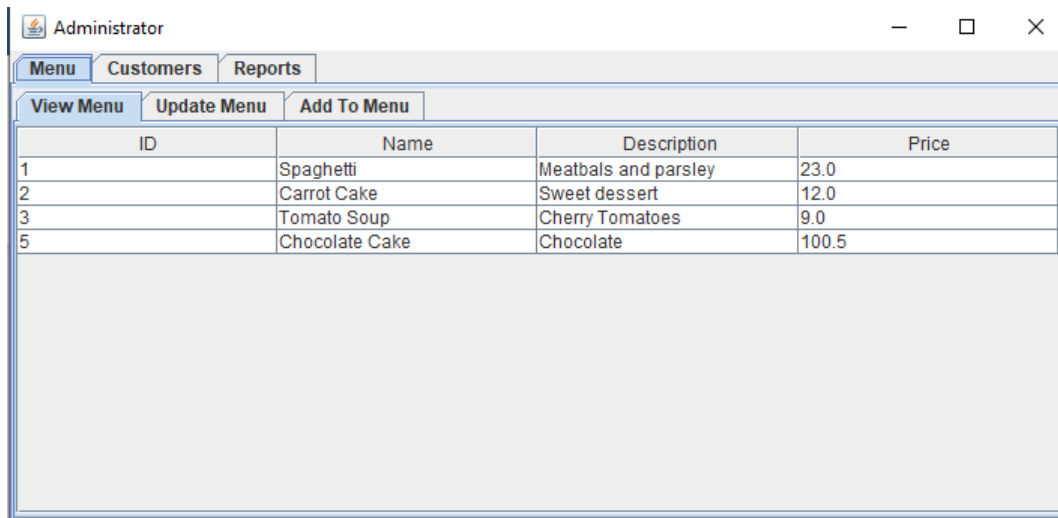
The image shows two overlapping windows. The 'Customer' window has tabs for 'Delete from Cart', 'View Cart', and 'History'. The 'Profile' tab is active, showing fields for 'Name' (Ana Jane Roberts), 'Email' (anar@yahoo.com), 'Password' (seven dots), 'Address' (Bakers Street, 67), and 'Loyalty' (false). There are 'Save Changes', 'Delete Account', and 'Confirm Changes' buttons. The 'Welcome' window has 'Log in' and 'Sign up' tabs. The 'Log in' tab is active, showing fields for 'Email' (anar@yahoo.com) and 'Password' (seven dots), with a 'Confirm' button.



The image shows the 'History' tab of the 'Customer' window. It contains a table with 6 columns: ID, Address, Date, Products, Total Price, and Payment Method. There are two rows of data.

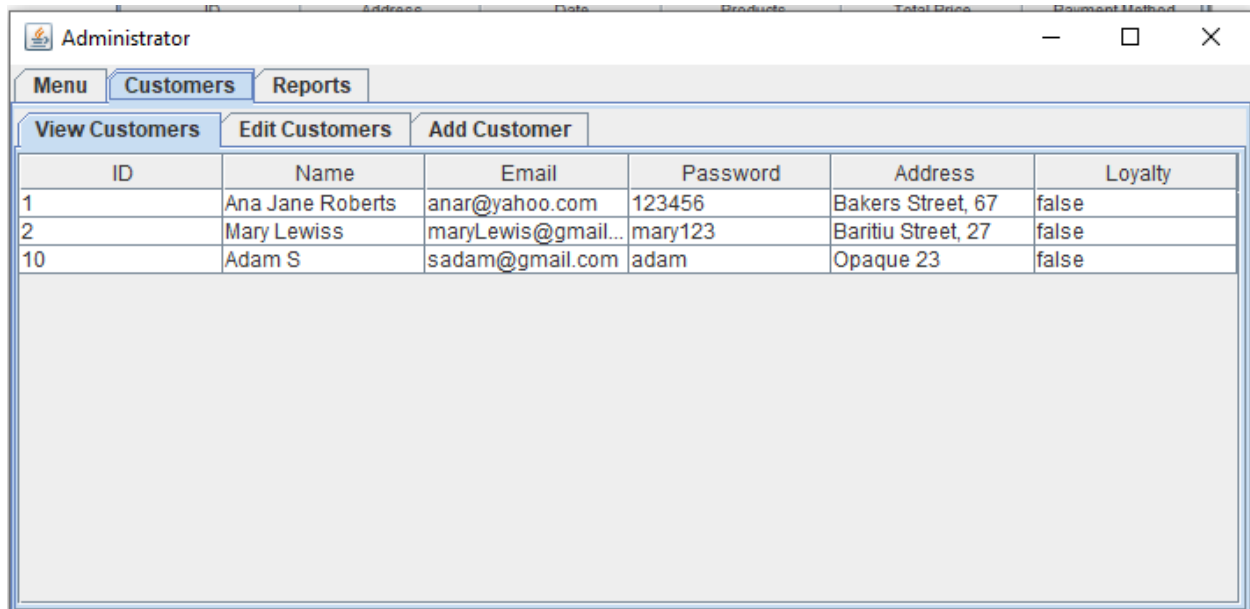
ID	Address	Date	Products	Total Price	Payment Method
1	Bakers Street 67	2021-02-20	Spaghetti	23.0	CARD
2	Johnosns 23	2021-03-22	Carrot CakeTomato S...	120.5	CARD

//administrator panels



The image shows a web browser window titled "Administrator". It has three tabs: "Menu", "Customers", and "Reports". The "Menu" tab is active. Below the tabs are three sub-tabs: "View Menu", "Update Menu", and "Add To Menu". The "View Menu" sub-tab is active, displaying a table with four columns: "ID", "Name", "Description", and "Price". The table contains four rows of menu items. Below the table is a large, empty rectangular area.

ID	Name	Description	Price
1	Spaghetti	Meatbals and parsley	23.0
2	Carrot Cake	Sweet dessert	12.0
3	Tomato Soup	Cherry Tomatoes	9.0
5	Chocolate Cake	Chocolate	100.5



The image shows a web browser window titled "Administrator". It has three tabs: "Menu", "Customers", and "Reports". The "Customers" tab is active. Below the tabs are three sub-tabs: "View Customers", "Edit Customers", and "Add Customer". The "View Customers" sub-tab is active, displaying a table with six columns: "ID", "Name", "Email", "Password", "Address", and "Loyalty". The table contains three rows of customer data. Below the table is a large, empty rectangular area.

ID	Name	Email	Password	Address	Loyalty
1	Ana Jane Roberts	anar@yahoo.com	123456	Bakers Street, 67	false
2	Mary Lewiss	maryLewis@gmail...	mary123	Baritiu Street, 27	false
10	Adam S	sadam@gmail.com	adam	Opaque 23	false



//input wrong data when inserting

Administrator

Menu Customers Reports

View Customers Edit Customers Add Customer

Name: a

Email: joanne@yahoo.com

Password: password

Address: JoanneStreet 23

Loyalty: ☒

Add customer

Confirm Changes

Error

Please enter a valid name

OK

//generate report

Administrator

Menu Customers Reports

Start Date:

End Date:

Show By Dates Show All

ID	Email	DeliveryAddress	Products	Payment Method	Total Price
1	anar@yahoo.com	Bakers Street 67	Spaghetti	CARD	23.0
2	anar@yahoo.com	Johnosns 23	Carrot Cake Tomat...	CARD	120.5
11	marysue@yahoo.c...	Bakers Street 22	Tomato Soup Carr...	null	20.0
13	sadam@gmail.com	One Street, 1	Carrot Cake	null	12.0

Administrator

Menu Customers Reports

Start Date 20.01.2021

End Date 20.03.2021

Show By Dates Show All

ID	Email	DeliveryAddress	Products	Payment Method	Total Price
1	anar@yahoo.com	Bakers Street 67	Spaghetti	CARD	23.0

## 8. Bibliography

<https://priyalwalpita.medium.com/software-architecture-patterns-layered-architecture-a3b89b71a057>

[https://docs.jboss.org/hibernate/orm/6.0/quickstart/html\\_single/](https://docs.jboss.org/hibernate/orm/6.0/quickstart/html_single/)

[https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html\\_single/](https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/)

<https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>