Pizza Delivery Manager

Analysis and Design Document

Student: Bagoly Szabolcs-Marton

**Group: 30234**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

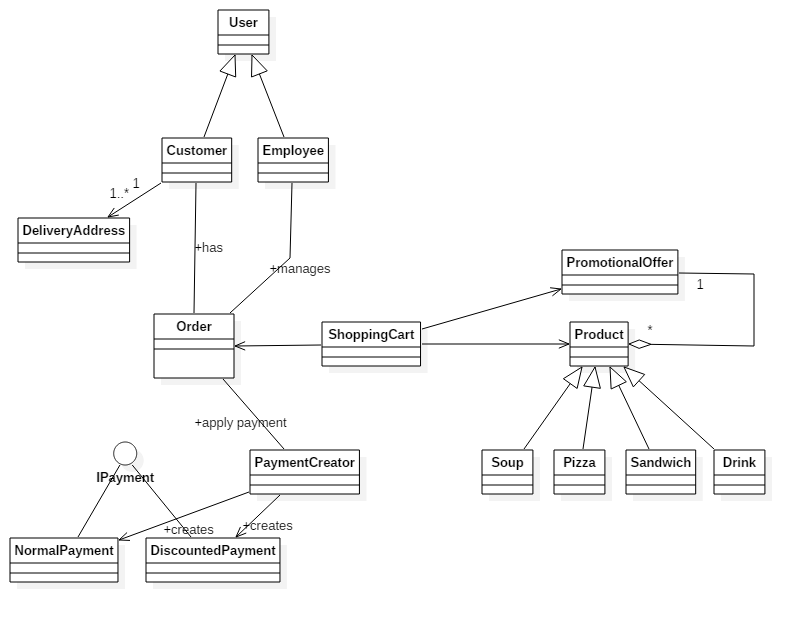
Design and implement a client-server application used to record orders, handle payments and manage users’ activity in a pizza delivery process. The system allows the clients to choose a product or more products and place the order, specifying name, delivery address and mobile phone number. When an order is placed, it must be processed by an employee. After delivering the product, the order is considered completed.

To place an order, first of all clients have to register. Then the account can be accessed using a username and password or using automated Facebook/Google login process. If a user has placed orders several times in the past, automatically apply a discount for loyalty. The order status should be updated by an employee to inform the client, in which state the order is: waiting for processing, preparing product or delivering.

# Elaboration – Iteration 1.1

# Domain Model

The application has two separate clients (customer and employee) which mainly operate on the same data. A customer creates new orders, and fills the shopping cart with products or promotional offers (which are a single product or a group of products. For the order a total price should be calculated based on the customers’ previous activity and voucher usage. For this task, different payment objects are created, which return a price and eventual applicable bonuses.



# Architectural Design

## Conceptual Architecture

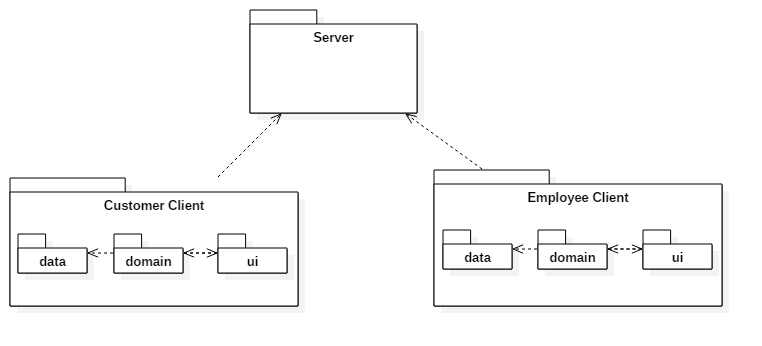
The system is based on client-server architecture. Basically, the system has two types of clients and almost all of the data processing is performed at the client side applications. The main responsibility of the server-side application should be to connect to the database, to store and to retrieve data on client request.

The implementation of the client system is based on the Model View Presenter (MVP) architectural style. Using MVP a greater separation of concerns is reached the data model being completely separated from the view. The MVP also facilitates automated testing, because of binding the three parts only with interfaces. That means that different parts that implement the same interface can be changed, without further modifications in the code.

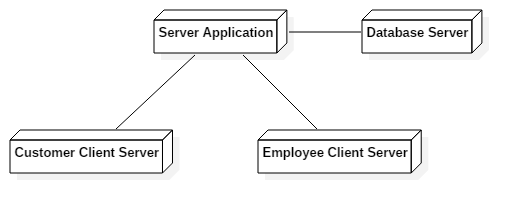
Related to the Android specific implementation, every different screen (Activity) will have one or more presenter which processes the user activity. The information requests from the data model are achieved using Interactors from the presenter, preferably in an asynchronous way, so the user interface will be updated when the requested information arrives, without blocking the user to search across currently available data. The MVP architectural style will provide a greater separation of concerns and cleaner code for further development.

## Package Design

The package diagram below illustrates the whole system presented in the previous section. At top level the server side and two client side applications can be observed. Inside the client side packages, the separation into further packages offers the grouping of the data accordingly in order the implement the MVP pattern: data model, domain model and view model.



## Component and Deployment Diagrams

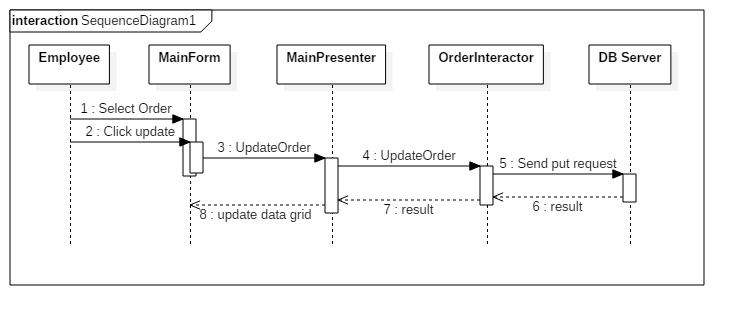


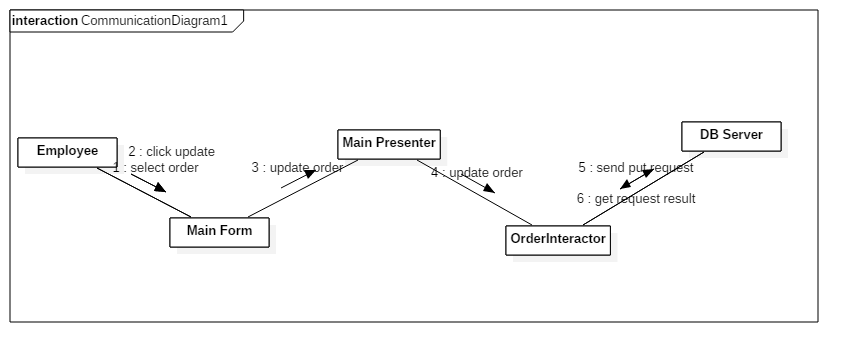
# Elaboration – Iteration 1.2

# Design Model

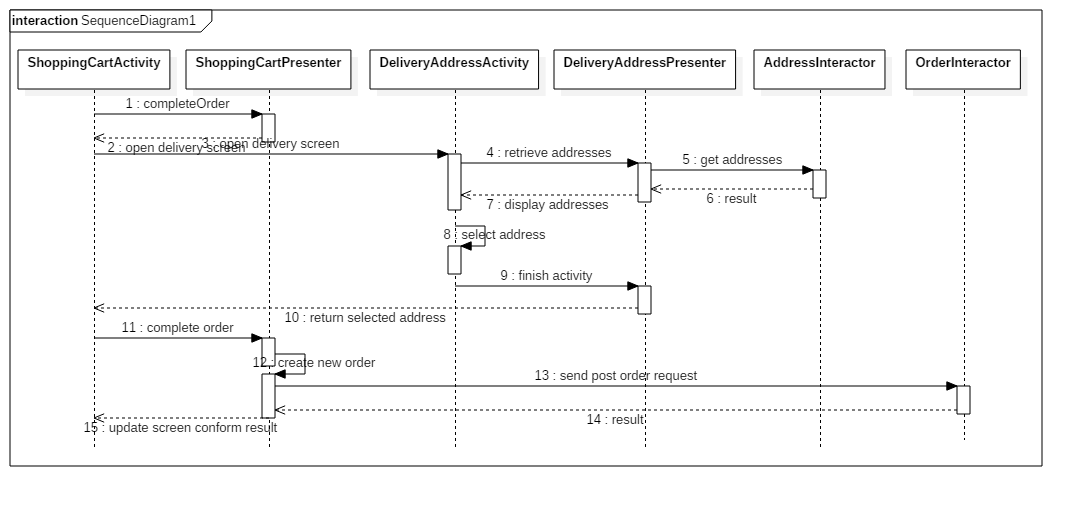
## Dynamic Behavior

One significant scenario is updating the status of an order by an employee. For this operation, the employee has to select an order from the existing list of orders displayed on the screen, then click on the corresponding button. The system will automatically detect which is the next state of the order and send an update request to the server with the correct information.





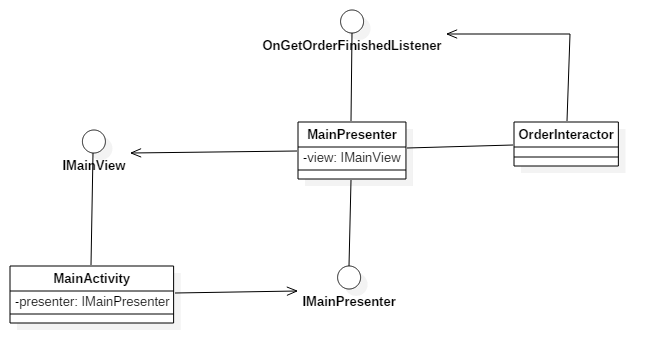
Another significant scenario is the placement of a new order. The client has to select one or more products. Then access the shopping, cart, select a delivery address and the place the order via a HTTP interactor and post the new order to the DB server. The following sequence diagram presents the process considering that the shopping cart screen is already opened, products are selected. The next steps in order to complete the order is presented below.



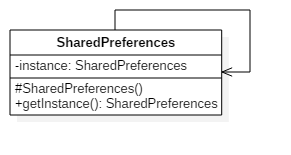
## Class Design

In this section the applied design principles will be presented:

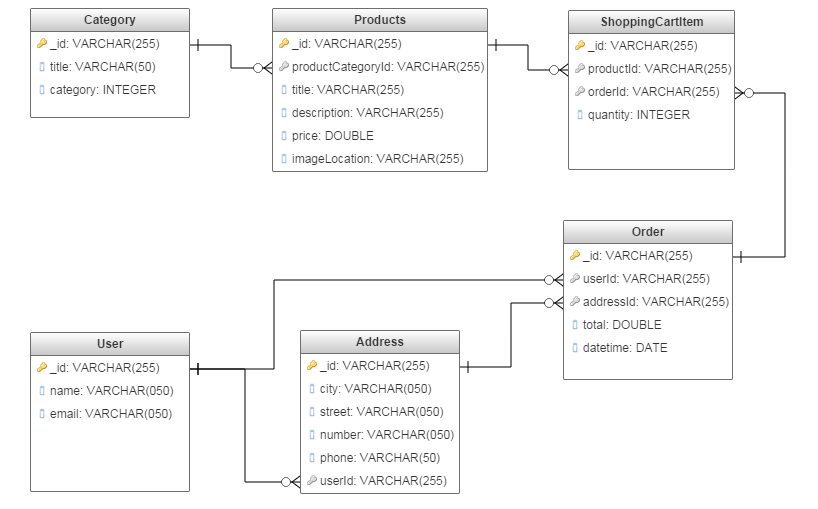
Designing the Android application, the Model View Presenter pattern was followed. Conform this pattern, every single screen is implementing an IView interface and has reference to a IPresenter interface. Every presenter associated with a screen is implementing the corresponding IPresenter interface and is using one or more interactor classes in order to make modifications to the data stored on the server. Every interactor has OnOperationFinished methods which will be implemented by the presenter.



Another popular pattern used was the Singleton. This pattern is applied when creating the SharedPreferences manager class for an Android client, where persistent settings will be saved. In SharedPreferences class we would store consistent settings of an application set by the user. When something depends on user option, relevant settings from SharedPreferences class should be extracted. Since the same information should be contained in every instance of this class, it would be a good practice to make a single instance of it.



# Data Model



# Unit Testing

# Elaboration – Iteration 2

# Architectural Design Refinement

Besides, the classical packages that are used in to implement the MVP pattern (data, domain, ui), additional packages were added in order to have a clear view about the architecture. The *utils* package is containing utility classes used in more packages, and in the *datamodel* package POJO classes are stored. These classes are used in order to serialize and display the retrieved data from the database server.

# Design Model Refinement

# Construction and Transition

# System Testing

# Future improvements

The present system can be extended, to not only manage the incoming orders, but the entire food store. All utilized ingredients should be introduced to the system, and when an order is being prepared, the stock of the used ingredients should automatically decrease. When the stock number of a specific ingredient reached a low threshold, the corresponding employee should be notified to make new orders for ingredients.

Regarding the client application, special features could be added, like selecting the ingredients of a specific product, or make a personalized pizza based on the selected ingredients and other specifications. Also, payment handling could be included into the system.

# Bibliography