IKEA WEB APPLICATION

Project documentation

Student:

* Udrea Bianca

**Group:**

**30433**

**Contents**

[I Project specification](#_heading=h.gjdgxs) **3**

[1.1 Domain Model Diagram](#_heading=h.30j0zll) 3

[II Use-Case model](#_heading=h.1fob9te) **3**

[2.1 Users and stakeholders](#_heading=h.f2irg1azp7rm) 4

[2.2 Use-Case identification](#_heading=h.3znysh7) 4

[2.3 UML Use-Case diagram](#_heading=h.2et92p0) 4

[III Architectural design](#_heading=h.tyjcwt) **5**

[3.1 Conceptual architecture](#_heading=h.3dy6vkm) 5

[3.2 Package diagram](#_heading=h.1t3h5sf) 5

[3.3 Class diagram](#_heading=h.4d34og8) 5

[3.4 Database (E-R/Data model) diagram](#_heading=h.2s8eyo1) 6

[3.5 Sequence diagram](#_heading=h.17dp8vu) 6

[3.6 Activity diagram](#_heading=h.3rdcrjn) 6

[IV Supplementary specifications](#_heading=h.26in1rg) **6**

[4.1 Non-functional requirements](#_heading=h.lnxbz9) 6

[4.2 Design constraints](#_heading=h.35nkun2) 6

[V Testing](#_heading=h.1ksv4uv) **7**

[5.1 Testing methods/frameworks](#_heading=h.44sinio) 7

[5.2 Future improvements](#_heading=h.2jxsxqh) 7

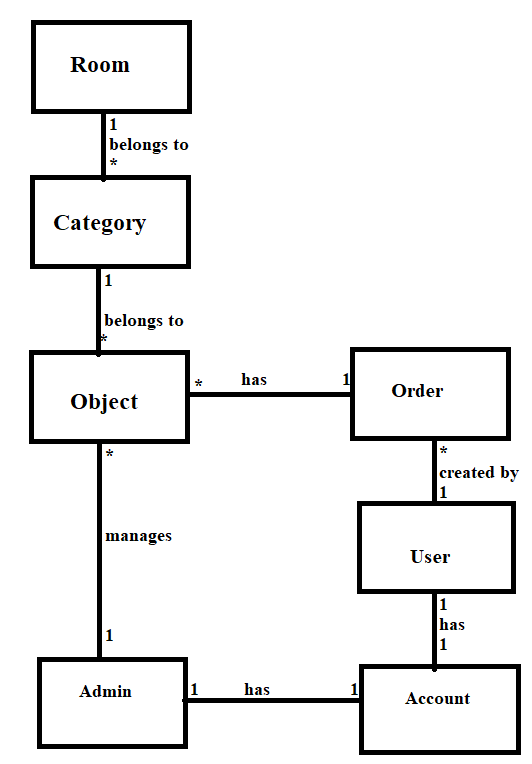
[VI Bibliography](#_heading=h.z337ya) **7**

# I Project specification

This project aims to realize a web application similar to the current Ikea website, following its division in “room” categories. Keeping in mind that we will have 2 types of users: admin and regular users, the application will provide easiness to access the Ikea products, register, sort products, add the items to a cart and even place orders. Special actions such as adding products or manage orders are going to be done by the administrator. This project wants to achieve an easy to use interface for all users, access to information and in general a clear presentation of the application.

## 1.1 Domain Model Diagram

The following diagram shows the links between models in our application and the actions of the admin and the user. Starting from the Object, it belongs to a Category, and the Category to a Room. Our application will follow the Ikea division by rooms, having 4 specific ones: Bedroom, Bathroom, Living Room and Kitchen. Each room has their certain categories, such as bedroom containing: bed, nightstand or bed sheets. And each such category will have specific objects. They will be created and managed by the Admin, and included in an Order. The order is created by a User.



# II Use-Case model

A Use Case model shows different types of users, their actions and responsibilities over the application. In our application, the 2 different types of users are: Admin and the regular User.

## 2.1 Users and stakeholders

The Admin has responsibilities of creating, reading, updating and deleting users, orders and products. A user is considered anyone who uses the application, whether it is somebody from inside or outside the organization. A stakeholder is anyone who impacts or is impacted by an organization’s actions or products. So customers, users, and anyone inside your organization with an interest in your product is classified as a stakeholder.

## 2.2 Use-Case identification

1:

NAME: REGISTER

LAYER: USER – GOAL

MAIN ACTOR: USER and ADMIN

MAIN SUCCESS SCENARIO: Both user and admin provide information for creating an account, they would have to provide a valid name, email address and password.

EXTENSION: In case one of the fields isn’t properly entered, an exception occurs and the user or admin are informed, in order to type again a valid field.

2:

NAME: VIEW PRODUCTS

LAYER: USER-GOAL

MAIN ACTOR: USER and ADMIN

MAIN SUCCESS SCENARIO: After the login step, both types of users will be able to see the main page which has all the products displayed.

EXTENSION: If the login step is not performed correctly, meaning the username or password aren’t typed correctly, the user wouldn’t be able to see the products.

3:

NAME: SORT PRODUCTS BY PRICE

LAYER: USER-GOAL

MAIN ACTOR:USER and ADMIN

MAIN SUCCESS SCENARIO: After accessing the main page, the user may want to order the products by their price so that it is easier to find the perfect choice for them.

EXTENSION: All products from all categories and all rooms are ordered ascending according to their price field.

4:

NAME: ADD ITEM TO CART

LAYER:USER-GOAL

MAIN ACTOR:USER

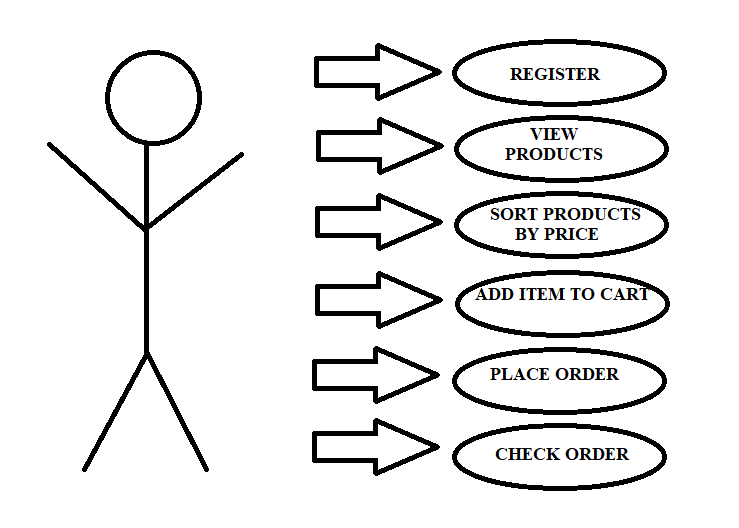
MAIN SUCCESS SCENARIO: If the number of objects that the user wants to buy is available, the object is successfully added to the cart.

EXTENSION: If the number of objects is not available, the user should be informed.

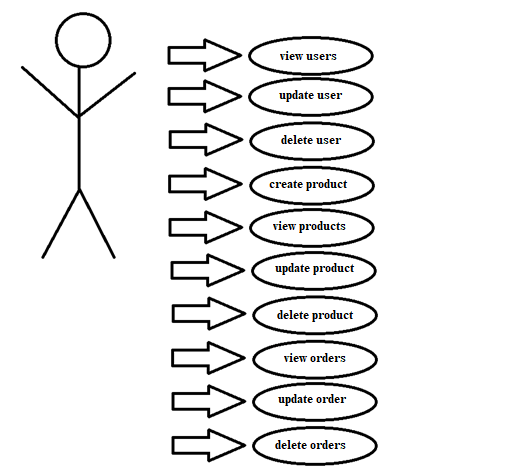
## 

## 2.3 UML Use-Case diagram

USER:



ADMIN:



# III Architectural design

The following chapter introduces the architectural design. It has a broader scope than design patterns, and has to correctly support the implementation of the project. The definition of an architectural pattern is: a general, reusable solution to a commonly occurring problem in software architecture within a given context.

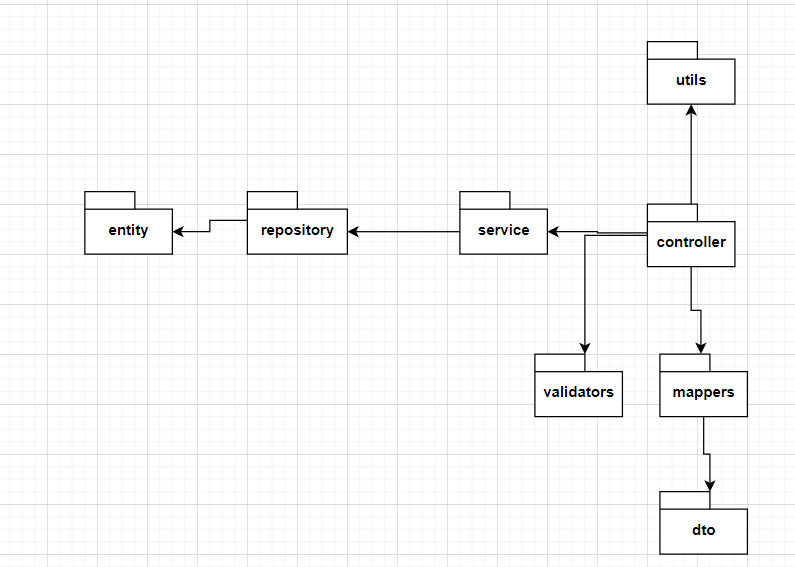
## 

## 3.1 Conceptual architecture

Our application is developed based on a web application, with a layered structure of type MVC and connected to a database. A web application is a software application that runs on a remote server, coded in browser-supported language such as HTML as this language relies on the browser to render the program executable. Our project is a layered application because its structure is divided into modules with similar functionality. Layers are stacked on top of each other so that every layer can communicate only with the one adjacent to it. It also retrieves and saves data into the database based on the MVC structure, having the Model, View and Controller components.

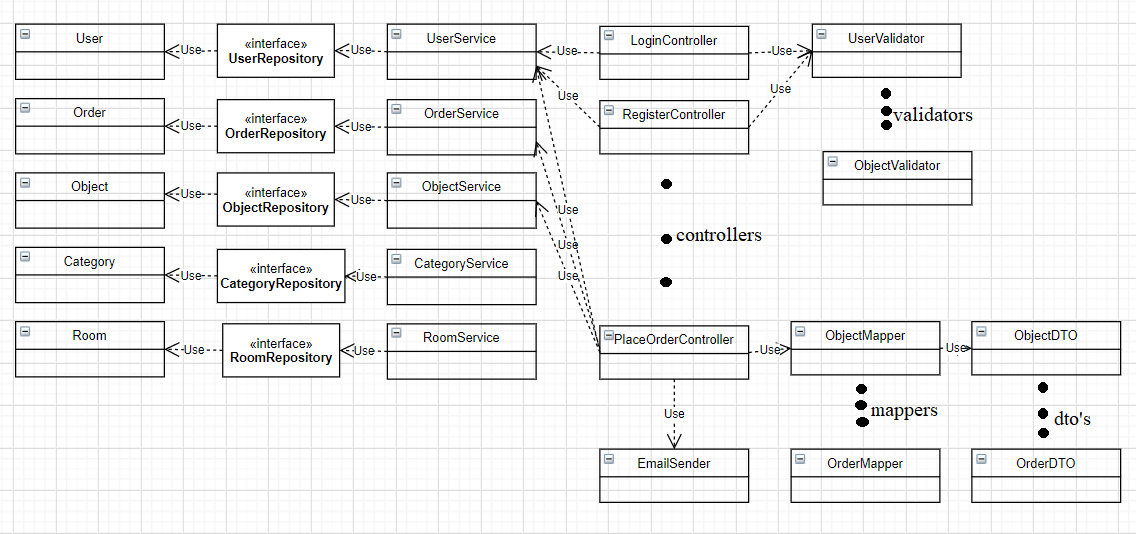
## 3.2 Package diagram

The following diagram corresponds to the division in particular packages for our application. Each package has a specific utility and will be used by certain packages. Following the layered architecture, these packages will be placed accordingly, meaning the ‘Controller’ package will call the ‘Service’ package, then ‘Repository’ and finally ‘Entity’. The order is respected throughout the entire project.

**

## 3.3 Class diagram – this is not the final form

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The diagram below doesn’t show all classes, only some are featured for representation.

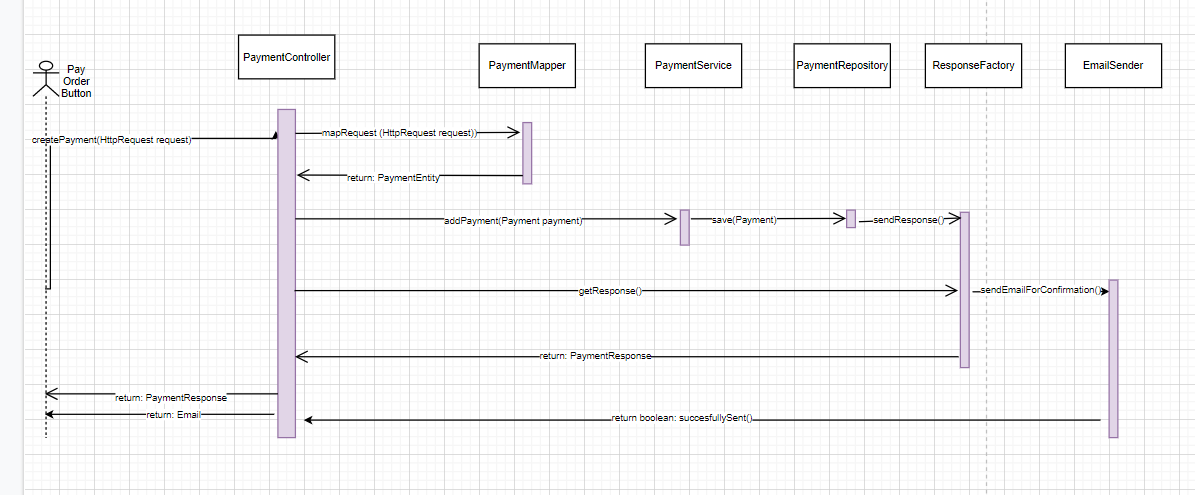


## 3.4 Database (E-R/Data model) diagram

## 

## 3.5 Sequence diagram

## Sequence Diagrams captures: the interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams) and high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams). The diagram below captures the sequence of steps of achieving the payment of the order in our application.



## 3.6 Activity diagram – for placing an order

The following diagram shows the steps up to placing an order on the Ikea website. Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modeling how a collection of use cases coordinate to represent business workflows.

# 

# IV Supplementary specifications

The following chapter introduces requirements or constraints that may be added to our project, depending on our plan or our customer’s wishes. A constraint refers to a requirement that limits the available options for developing it.

## 4.1 Non-functional requirements

Non functional requirements specify the quality attribute of a software system. They judge the software system based on responsiveness, usability and other non-functional standards that are critical to the success of the software system.

**Data integrity** is the maintenance of the accuracy and consistency of data over its entire life-cycle, and is a critical aspect to the design, implementation and usage of any system which stores, processes, or retrieves data. So, in our case, it is very important to be sure that data is correct, as we keep track of all products of the Ikea company, and all orders.

**Platform compatibility**: It must work on Windows. Building software that does not satisfy the platform constraint means you have failed to design a software system that satisfies stakeholders' key concerns.

Moreover, **performance** refers to the amount of useful work accomplished by the system. In our case, there are certain moments when our website would be accessed by far more customers, during sale times, ‘Black Friday’ or holidays. It is important that this is handled properly, all orders and products’ information to be correctly recorded.

**Availability** represents the total time a functional unit is capable of being used during a given interval or the total interval. In our case, our website will be available 24/7 as it shouldn’t impose any time restrictions.

Also, **data backup** is a copy of computer data taken and stored elsewhere so that it may be used to restore the original after a data loss event. I believe this is an important feature to achieve for any application, and in our case to make sure that, even when under an unforeseeable event, data loss is prevented.

Not to mention, **usability** is the ease of use and learnability of a human-made object such as a tool or device. Usability is key to our application since it is a program open to all people and should be easy to use for everybody. Also, an important aspect is that the harder and unclear it is to use, it may even influence the sales of the products.

**Durability** is the ability of a physical product to remain functional, without requiring excessive maintenance or repair, when faced with the challenges of normal operation over its design lifetime. We want a durable application since the Ikea company is a long-lasting business and it will be a 24/7 open website for a long period of time.

Last but not least, **quality** is a conditional and somewhat subjective attribute and may be understood differently by different people. Consumers may focus on the specification quality of a product/service, or how it compares to competitors in the marketplace. This way, compared with other competitors, our application will have to do similarly in terms of goal achievements.

## 4.2 Design constraints

Design constraints usually describe conditions on how the application will be implemented; they provide accuracy.

Tool and language constraints are often met, our application could be constrained to be implemented in Java and on the Intellij platform. For our project, we might be restricted to use a specific library or framework, some companies might have a list indicating which open source software may be used. We will use the Spring framework.

The project will be restricted to use a relational database for storing the data of the application. Each operation done by the user of our program, is going to activate procedures like retrieving data from the database, save it or modify it. Any operation such as updating an order or simply viewing users is going to activate that. We will also be imposed that we validate each data inserted by the user and confirm that every operation was or not done successfully. This is important as it shows the data was correctly modified in our system, all products and orders information is updated in the right way.

The application will be required to have been completely tested manually. This can be seen as providing precision and rightness to our program. Moreover, our program will be developed using Java 8 or a latter version.

A multitude of design constraints can be further added and our project can easily support numerous requirements.

# V Testing

*< Se va discuta la laborator./>*

## 5.1 Testing methods/frameworks

## 5.2 Future improvements

# VI Bibliography