

**Hairdresser application**

**Name: Șimon Bianca Andreea**

**Group: 30232**

**Table of Contents**

[Deliverable 1 3](#_Toc67946327)

[1. Project Specification 3](#_Toc67946328)

[2. Functional Requirements 3](#_Toc67946329)

[3. Use Cases Model 3](#_Toc67946330)

[3.1. Use Case Identification 3](#_Toc67946331)

[3.2. UML Use Case Diagrams 6](#_Toc67946332)

[4. Supplementary Specification 6](#_Toc67946333)

[4.1. Non-functional Requirements 6](#_Toc67946334)

[4.2. Design Constraints 6](#_Toc67946335)

[5. Glossary 7](#_Toc67946336)

[Deliverable 2 7](#_Toc67946337)

[1. Domain Model 7](#_Toc67946338)

[2. Architectural Design 7](#_Toc67946339)

[2.1. Conceptual Architecture 7](#_Toc67946340)

[2.2. Package Design 8](#_Toc67946341)

[2.3. Component and Deployment Diagram 8](#_Toc67946342)

# Deliverable 1

## Project Specification

The purpose of this project is to design and implement an application for an hairdresser, where we have 2 types of users: the administrator and the clients. The clients would have to create an account and then they could make an appointment, choosing the services they want to have. The administrator will accept or will reject the appointments, and also will update the details about hairdresser and services.

## Functional Requirements

The client user can perform the following operations:

* Create an account
* Log in
* View or update account information
* Make an appointment
* Cancel an appointment
* View the history of the appointments

The administrator user can perform the following operatons:

* Create an account
* Log in
* Add, delete or update services
* Accept or reject appointments
* Cancel appointments
* View the history of all appointmens

## Use Cases Model

### Use Case Identification

Use case: register

Level: user-goal level

Primary actor: the user

Main succes scenario: the user creates an account -> the user is added to database

Extensions: the user already exists -> an error message is displayed

Use case: log in

Level: user-goal level

Primary actor: the user

Main succes scenario: the user successfully loggins

Extensions: no such user exists / incorrect password -> an error message is displayed

Use case: update account information

Level: client-goal level

Primary actor: the client

Main succes scenario: the client successfully updates the account information -> the information is updated in databse

Extensions: - the information is unchanged -> the account has the same information

Use case: make an appointment

Level: client-goal level

Primary actor: the client

Main succes scenario: the client successfully make an appointment

Extensions: -

Use case: cancel appointment

Level: client-goal level

Primary actor: the client

Main succes scenario: the client cancels the appointment -> the appointment is deleted from database

Extensions: - the client cancels the appointment by mistake-> the client has to make another appointment

Use case: view the history of the appointments

Level: client-goal level

Primary actor: the client

Main succes scenario: the client sees the history of the appointments

Extensions: -

Use case: add service

Level: administrator-goal level

Primary actor: the administrator

Main succes scenario: the administrator successfully adds a new service -> the service is added to databse

Extensions: - the service already exists -> an error message is displayed

Use case: delete service

Level: administrator-goal level

Primary actor: the administrator

Main succes scenario: the administrator successfully deletes a service -> the service is deleted from databse

Extensions: -

Use case: update service

Level: administrator-goal level

Primary actor: the administrator

Main succes scenario: the administrator successfully updates the information of a service -> the service is updated in databse

Extensions: - the information is unchanged -> the service has the same information

Use case: accept/reject an appointment

Level: administrator-goal level

Primary actor: the administrator

Main succes scenario: the administrator successfully accepts/rejects an appointment-> the appointment is added to database if it is accepted

Extensions: -

Use case: cancel appointment

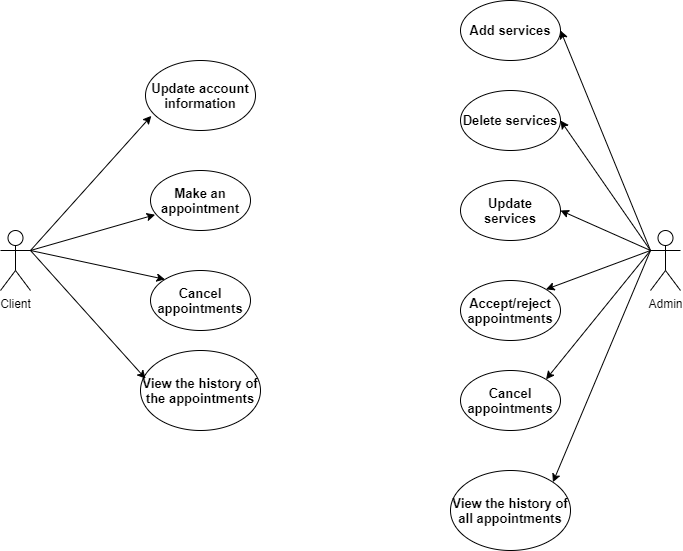
Level: administrator-goal level

Primary actor: the administrator

Main succes scenario: the administrator cancels the appointment -> the appointment is deleted from database

Extensions: - the administrator cancels the appointment by mistake-> the administrator has to make another appointment

### UML Use Case Diagrams



## Supplementary Specification

### Non-functional Requirements

!!One of the non-functional requirements is that all the inputs of the application must be validated before submitting the data and saving it in the database. The information about the clients will be used when they will make appointments.

Another NF requirement is that when sending notifications about appointments, the server must send them within 10 minutes from the moment when the administrator accepted or rejected them.

The third NF requirement that is suitable for this application is that datas will be protected in a database. Every user will have certain rights and will have access to certain information.

The last NF requirement is that every user who has minimal knowledge of using a computer, should manage to use the application, make an account, and also an appointment.

### Design Constraints

!!Due to the fact that it is used Java as programming language, the graphic interface used will be the one offered by it. Spring will be used as Framework and MySQL as database.

## Glossary

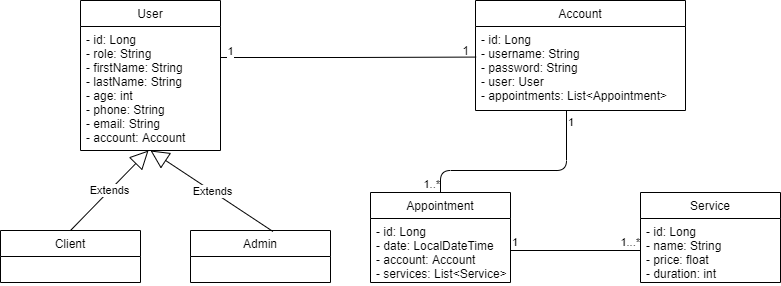
------------------------------------------------------------------------------------------------------

# Deliverable 2

## Domain Model

The data models used in my project are: User, Admin, Client, Account, Appointment and Service.

The User class represents the users of the application, while the Admin and the Client classes define the type of the user. Every user has an account. The clients can make appointments choosing different services, while the administrators can add, update or delete services.



## Architectural Design

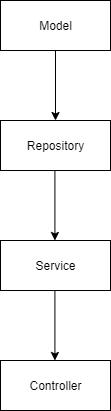
### Conceptual Architecture

The architectural pattern used in this project is Layers because is the most common and also this pattern closely matches the traditional IT communication and organizational structures found in most companies.

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role and responsability within the application. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request and doesn`t need to know or worry about how to get some informations from the below layers.

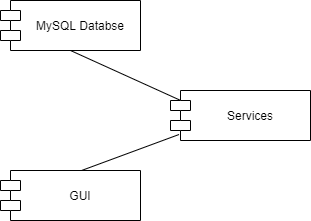
Another powerful features of the layered architectural pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains tot hat layer. This type of classification makes it easy to build effective roles and responsability models.

### Package Design

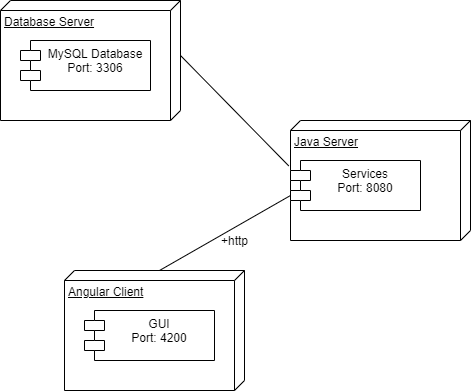


### Component and Deployment Diagram

Component Diagram:



Deployment Diagram:



​