MedPat: A Doctor - Patient Platform

Analysis and Design Document

Student: Turcu Lucian Andrei

**Group: 30432**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 19/Apr/18 | 1.0 | First iteration. Domain Model, Architectural Design, Deployment diagram | Turcu Lucian Andrei |
| 24/Apr/18 | 1.1 | Design model, Data model | Turcu Lucian Andrei |
|  |  |  |  |
|  |  |  |  |

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

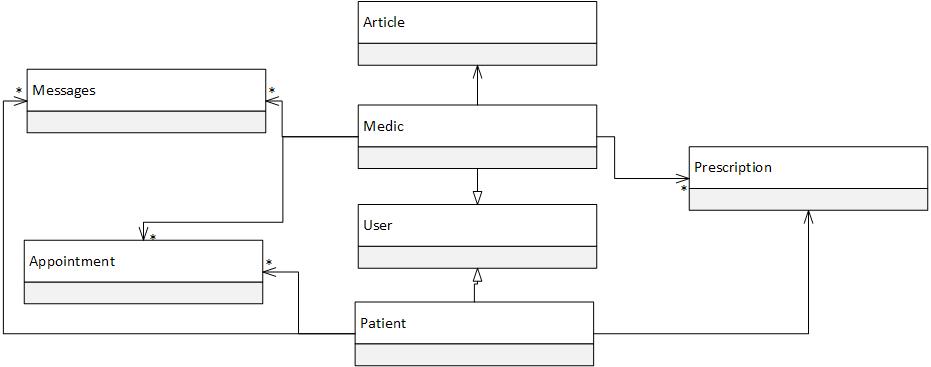
# Project Specification

MedPat is a platform whose purpose is to facilitate the communication between medical staff and their patients. Using MedPat, a medic can offer basic medical consulting remotely and write prescriptions for a patient. It also gives specialized users the possibility to write articles on medical subjects, which may be of interest to both medical and non-medical users. The patients can book appointments to a specific doctor, keep track of prescriptions and request consulting from a doctor.

# Elaboration – Iteration 1.1

# Domain Model

This is the general representation of the relevant concepts that have to be modelled in the software application, including the relations between them and the multiplicity factors.

**

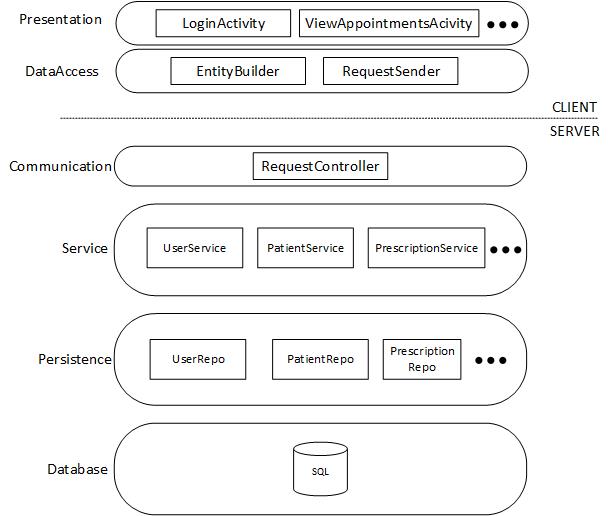
# Architectural Design

## Conceptual Architecture

The system is based on a client-server architecture. The client is the Android application, through which the users interact with the system. It will implement the Builder design pattern to instantiate objects using data received in the server responses and Dependency Injection to easily bind the components.

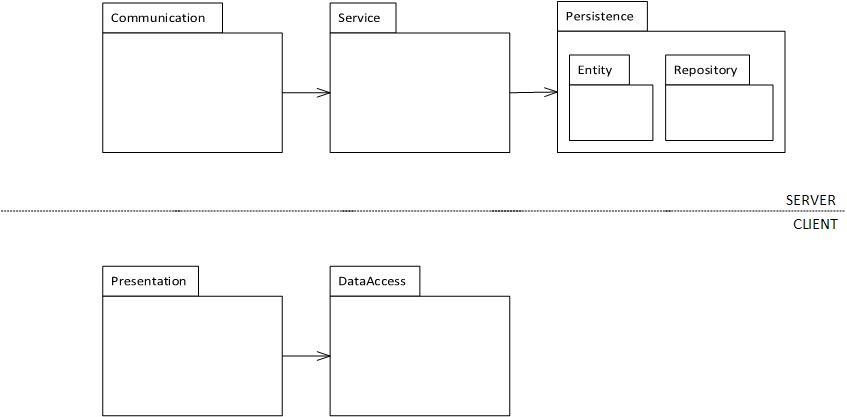
The desktop application processes the client’s requests and retrieves data from the database. This way, multiple users can use the application at once, using a common data set. The RequestController class is representing the entire server system, according to the Façade design pattern.

The server application’s architecture is a layered one, to achieve separation between the different levels of functionality. The layers are repository, service and communication. Between the layers, communication is only available from a layer to the one directly above or below.

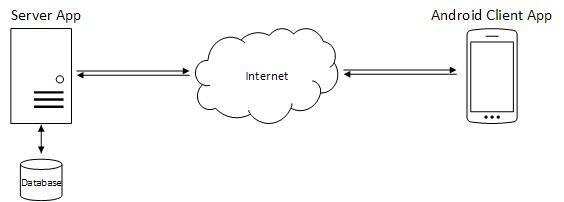


*<layered architecture>*

## Package Design

**

## Deployment Diagram

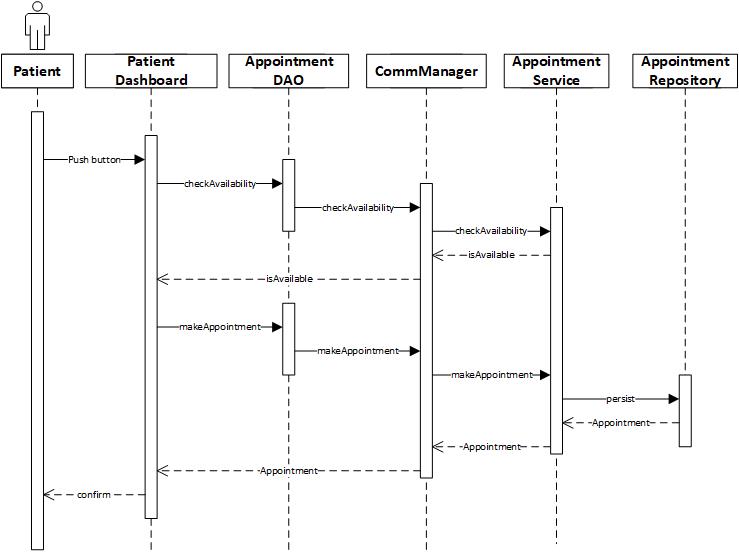


# Elaboration – Iteration 1.2

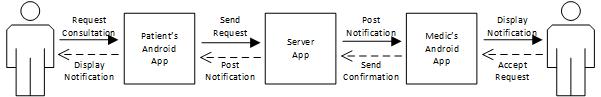
# Design Model

## Dynamic Behavior

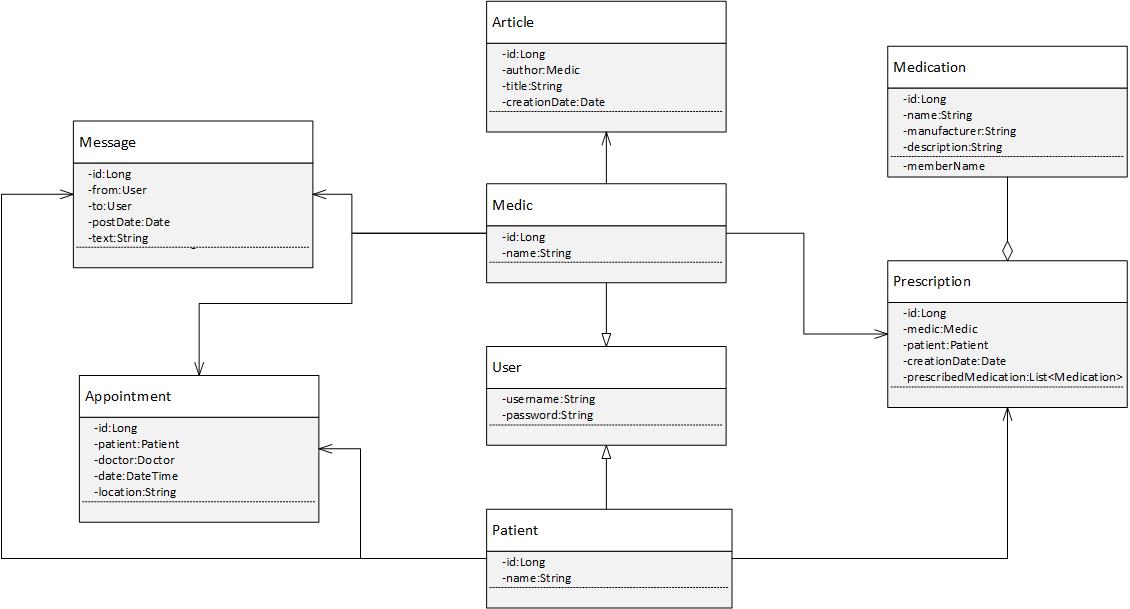
*Sequence diagram for the “Make Appointment” scenario*

**

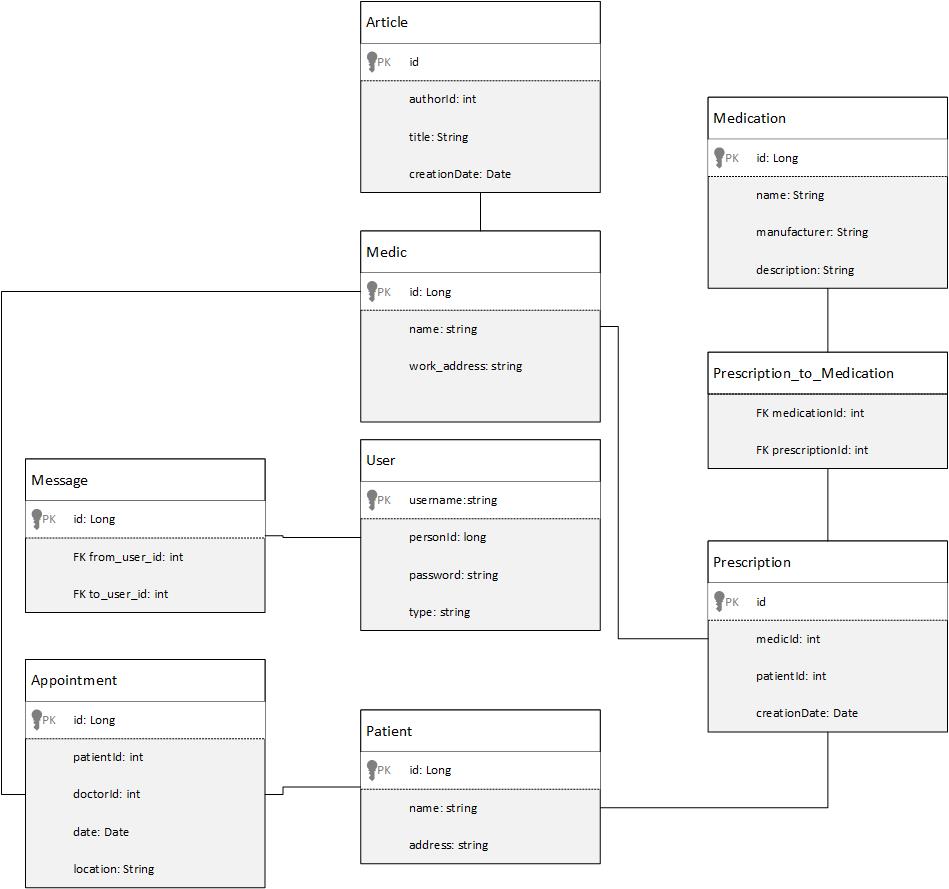
*Communications diagram for the “Request Remote Consultation” scenario*



## Class Design

**

# Data Model

**

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

# Bibliography