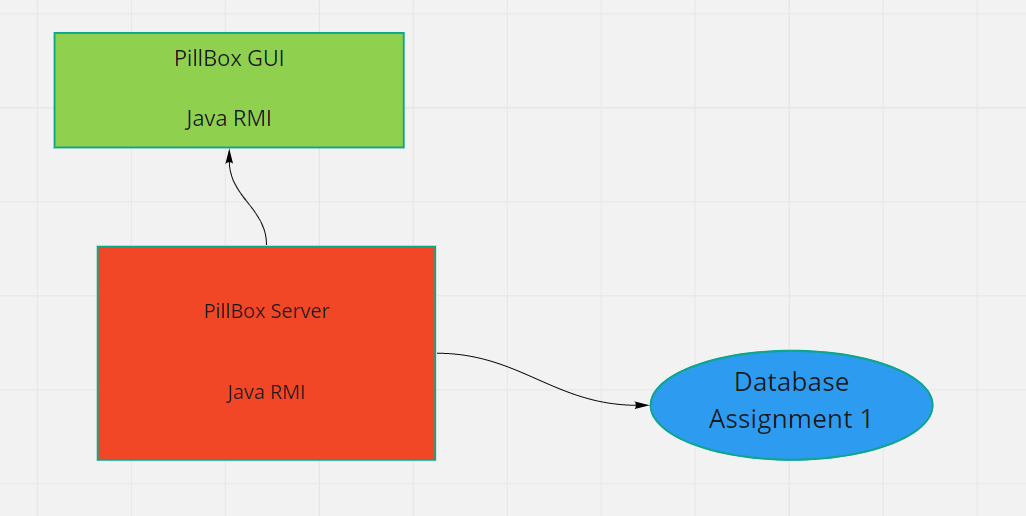
**Medication Dispenser**

-Distributed systems-

**Name**: Péntek Tamás

**Group**: 30642



**Figure 1. Conceptual description of the distributed system**

This documentation presents a solution for a medication dispenser system which has a desktop user interface. The conceptual description of this complex system can be found on the Figure 1.

This system will alert a patient when she/he has to take the medication. The client application will alert the server application when the medication was taken or when the patient didn’t take the medication. The pill dispenser client application communicates with the server side using RPC (Remote Procedure Call), exactly via JSON-RPC. The system has 3 main parts: pill dispenser client application, server application and database.

The client application is represented by a Spring application which has a graphical user interface written in Java Swing. This app is a desktop application, so we don’t have a webpage for it, only a desktop graphical user interface. The client gets the patient’s medication plan using RPC and for this is used the JSON-RPC dependency. The JSON-RPC API needs only a small configuration and near this we have to declare an interface, which will be the base of the communication, via this interface messages are sent between client and server.

The medication plan is loaded in the client app at a predefined time during each day and it downloads the medication plan for the next 24 hours. This app displays a list of medication that have to be taken at a predefined intake interval.

Each medication has a “Taken” button and when the patient presses this button the current medication is deleted from the list and the server is informed that the patient takes the medication.

When the current time passes the intake interval of a medication and the “Taken” button was not pressed, the server is informed that the patient didn’t take the medication.

The server application is also a Spring application and it communicates with the client application using JSON-RPC and accesses the database using Hibernate. This application also has an interface which is used for communication with client. The server also has the implementation of this interface, because of this, the application can access the medication plan from the database and can send this to the client app or can save the status (Taken, Not taken) of every medication from medication plan.

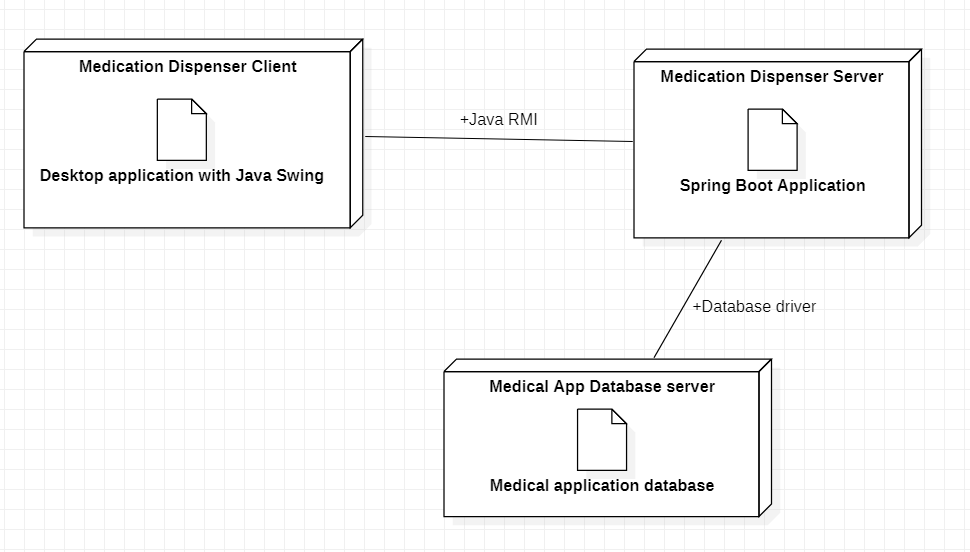
The database used for this system has been created for Assignment 1 and in this database is stored the medication plan of each patient and the list of medications which every patient takes. The selected data arrives from database to the client app using Hibernate between the server application and database server and using RPC between server application and client application.

On the next diagram (Figure 2.) is presented the Deployment diagram of the distributed system. The first part of this diagram is the relational database server, on which are stored the medication plans and details about patients and medications. In this case the database server is Postgres, which is an open-source database server and it has an add-on for Heroku Cloud, so it is useful when the application is planned to be deployed on cloud.

The second part is represented by the server application of the system, which is implemented using Spring Boot. The communication between server application and database is done by database driver, which offers an API to simplify the communication between these two parts.

The server communicates with client application via JSON-RPC which is an implementation of the RPC (Remote Procedure Call) that uses the Jackson library to convert Java objects to and from JSON objects. The JSON-RPC is used to send the medication plan of the patient, every day at a predefined time and to get the information from client app that the patient has taken the medication or not. This information is saved in the Postgres Database.

The third part of the system is represented by a desktop application which has a Swing user interface. On this application appears the medication plan of the patient for the next 24 hours or for the period of intake interval. The patient has to press the “Taken” button when takes a medication. This action is sent to the server application using JSON-RPC.



**Figure 2. UML Deployment diagram**

The pill dispenser client and server application are written in Java and is built using maven and the selected RPC (Remote Procedure Call) API is also written in Java. The system’s database is the same that was presented for Assignment 1. This distributed system is deployed on Heroku cloud using Docker. To run, to test the application is much better using cloud than localhost, because in this way the application can be accessed from everywhere, without location restrictions.

To use this part of the project, we don’t have to access the web application, we just run the Pill Dispenser Client application. For this, in the database we need to store at least one patient, which has a medication plan and some medications which are included in the medication plan. These details can be saved in the database using a doctor account. There is no possibility to create a doctor via user interface, so it can be done by inserting direct in the database or by using a tool like Postman, where we can make a POST method with a JSON, which contains details about doctor account (for example HOST\_URL/doctor/insert with a body: {"id":"45774962-e6f7-41f6-b940-72ef63fa1943","account":{"userName":"dr","password": "1234", "accountType":"DOCTOR"}}). After a doctor is created, we simply log in as doctor and create some medications and a patient, to which we add the created medication plan. Then we have to open the Pill Dispenser Client Application where we can see a list of medications that a patient has to take at current time, based on the intake intervals of the medications. For demo purposes, the medication plan is loaded in every minute, so in this way we can test much faster all the functionalities of the Pill Dispenser application.