# Smarthomes

1. Introduction

The main target of this project is to implement an application that is able to do the management of a smart-home, from the very basic house chores to the more complex tasks like offering the owner information regarding the traffic, weather or even the news based on his preferences that he can update at any given time.

1. Technologies used

Languages and frameworks used are : Java 8 with Spring MVC and numerous other dependencies that will be detailed in a future section, for the back-end server and Javascript ES6 with React for the client . As deploying technologies we are using : CloudFoundry and Heroku, the application becoming a distributed system and the MySQL database is also deployed on both services, through the ClearDB as binding for the both cloud services, in this way we make sure that ACID set of properties is fully respected. The communication between the client and server is made possible because the use of the HTTP communication protocol, the information being encapsulated in JSON or in XML, application implementing both REST and SOAP paradigms.

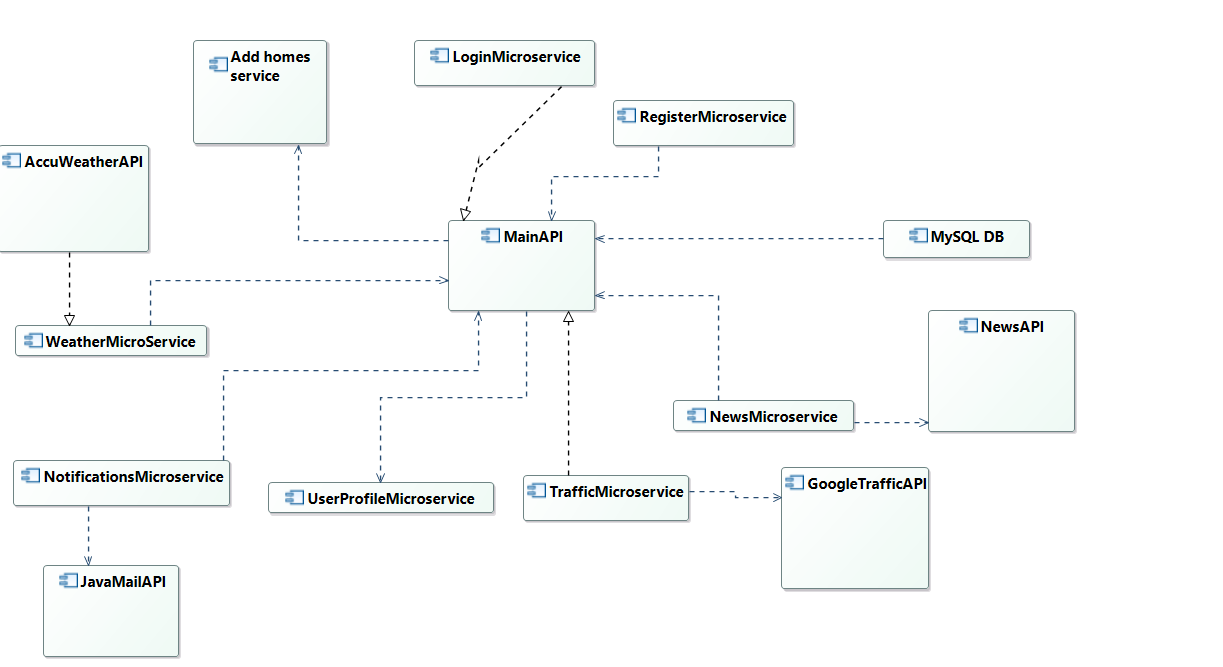
* 1. Server

Server is made up for microservices, each one being coded using Java 8 with SpringMVC as framework and Eureka Discovery for the communication between them using HTTP Requests. Information between client and server is carefully validated using custom annotations and regular expressions, before being inserted in the database. We also make use of dependencies like Quartz for queuing different jobs, RabitMQ in order to create an active queue for jobs , Executor, ThreadPool and Process in order to parallel process data which comes from python scripts ( python was choose for this task because of the previous experience of the developer with this scripting language and the ability to be interpreted by Java because of different libraries).

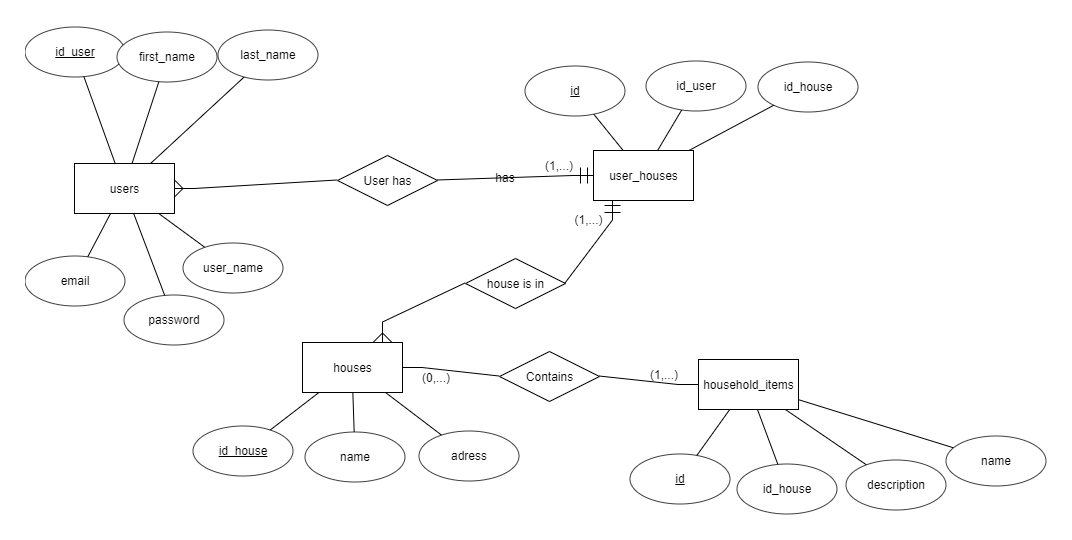
* 1. Client

Client is made from a mock-up server using Javascript ES6 and React as framework

1. Application architecture
   1. Detailed diagram of the application



* 1. ERD Diagram of the database used



* 1. Server micro-services
     1. Register-Service

This is the service responsible for the registering of new users. User will be asked to fill a form that will require : first name, last name, username and password which will be encapsulated in a JSON and send to the server which will validate the data using custom annotations and regular expressions. Password needs to contain at least 8 characters , an uppercase letter, an lowercase letter, a number and a special character and email needs to be in the correct format e.g : [test@yahoo.com](mailto:test@yahoo.com) . If the data is correct it will be created a new entry in the Users table using an object of type JpaRepository . Otherwise, a custom exception will be thrown and logged on the DEBUG level using Log4j and the client will be notified of which constraint he did not respect.

3.3.2 Login-Service

This is the service responsible for logging in existing users. User will be asked to enter his username and password that after will be encapsulated in a JSON and send to the sever. Using native queries we will determine if the credentials are correct or if the user does not exist in the database and a JSON with the corresponding status and message will be sent to the client side and if the credentials are correct user will be redirected to the main page, otherwise not.

* + 1. Group-Homes-Service

This is the service that gives the user the option to add houses to his inventory or household items. Furthermore, he can group homes and household items in each home. We make use of the @Embedded and @Embeddable annotations when we are creating the pivot table between users and house, this approach had been chose in order to further use the respective entities, after we make sure that the respective house or household item actually exists in the database.

* + 1. Parallel-Processing-Service

Just for learning purposes we have implemented a service that does a parallel process the output of 2 python scripts. The output is retrieved using the Process class in order to execute the command that actually runs the scripts and StringBuilder to proper construct the output and return it. The actual processing is done using a ThreadPool object and an ExecutorService one.

* + 1. Soap-Service

Another just for learning purposes service in which we implement a basic SOAP GET Request on hard-coded data defining in the respective XML file the behaviour of the response and the request itself.

* + 1. Quartz-Service

Implemented a basic job that prints a message and a basic trigger that will fire at every 5 seconds in order to get myself used with the Quartz. Also, implemented the possibility to make the injections through the @Autowire annotation by setting the ApplicationContext to be a bean factory of type AutowireCapableBeanFactory

* + 1. News-Service & Traffic-Service & Weather-Service

Implemented an API mash-up in order to give the users information regarding the daily or trending news, weather for today or the traffic condition between his home and other work or any other place. Each service makes a GET Request to the corresponding API : GoogleNews, TrafficMap, AccuWeather with the token key for each one of them and receives as response a JSON containing the necessary data which is sent to the client side in order to be printed in a more pleasant way.

* 1. Security

Each field of each entity that will receive input from the user is secured for SQL Injection attacks and XSS Attacks using custom annotations and regular expressions inside each constraint validator class . If the user tries to insert a malicious piece of code, a BadRequestException will be thrown and logged on the DEBUG level and the user will be notified about it.