SmartE
SMART ENERGY FOR YOUR HOME
DESIGN DESCRIPTION

Version 1.6.4

Revision History

Date	Version	Description	Author
2015-11-05	1.0	Initial Draft	Eugen Družin, Marko Vojić
2015-11-11	1.1	All sections added	Eugen Družin
2015-11-12	1.2	Introduction and background provided	Ondrej Kollar
2015-11-13	1.3	Added System structure, Sequential diagrams	Marko Vojić
2015-11-13	1.4	Technologies, System architecture, Mobile mock-up	Eugen Družin
2015-12-23	1.4.1	Spell checking, adding names to the figures	Elena Kyorova
2016-01-13	1.5	Update	
2016-01-19	1.6	Technologies, Mobile application	Eugen Družin
2016-01-20	1.6.1	Technologies, Mobile application, High-level system structure	Eugen Družin
2016-01-20	1.6.2	High-level system structure, System architecture, Communication and integration	Marko Vojić
2016-01-20	1.6.3	Revision document	Eugen Družin
2016-01-20	1.6.4	Proofreading	Nathan Chape

Table of contents

1 Introduction

This document describes the design of the application and it is updated during the realization of the project. This includes a short background preview, an architecture design, the technologies used, the software design, frontend design as well as backend design. The last section will cover the mobile application.

1.1 Purpose of this document

The purpose of this document is to expose the design decisions that have been made by team SE4YH during the SmartE project which is being developed as part of the Distributed Software Development course simultaneously at Mälardalen University situated in Västerås, Sweden and the University of Zagreb situated in Zagreb, Croatia.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction,* describes contents of this guide, documentation used during the developing process etc.
- Section 2, *Background and objectives*, overview of the project and high-level description of functionalities
- Section 3, *High-level system structure*
- Section 4, *Communication and integration*
- Section 5, *Technologies*
- Section 6, *System architecture*
- Section 7, *Mobile application*

1.3 Intended Audience

This document is intended for developers that need to follow this structure during the development process, even if some details may be changed further on. Moreover, the stakeholders can check our progress and read about the decisions we have taken.

1.4 Scope

The document concerns the design of the SmartE Project. It provides a description of the structure and style of the overall system. It shows the design decisions we have taken.

1.5 Definitions and acronyms

In the following tables we will present and explain definitions and acronyms or abbreviations that will be used in the document.

1.5.1 Definitions

Keyword	Definitions
REST API	API that adheres to the REST architectural constraints

1.5.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
SE4YH	Smart Energy for Your Home
FER	Faculty of Electrical Engineering and Computing, University of Zagreb
MDH	Mälardalen Högskola, Sweden
UML	Unified Modeling Language
UI	User Interface
GUI	Graphical User Interface
REST	Representational State Transfer
API	Application Programming Interface
JSON	JavaScript Object Notation
IDE	Integrated development environment
MVC	Model view controller

2 Background and objectives

2.1 Overview

The purpose of the project is to develop a system that allows users to define their own preferences and rules for the management of energy consumption by their appliances (e.g., conflicts, priorities, goals), and to manage the appliances in the house depending on them.
http://www.fer.unizg.hr/_download/repository/SE4YH_ProjectPlan_v1.1.docx

2.2 High-level description of the functionalities

This project aims to build a secure, flexible system that allows a user to manage and monitor the smart devices that are in a user's house without any knowledge in programming that is needed in order to create openHAB rules. List of predefined rules for the specific type of device is given to the user who can than modify rules options and apply them in order to ensure that the application fits user profile and preferences. List of rules and supported devices depends on openHAB and will grow in number and complexity. Besides this major functionality we want to give the user full control over the devices as well as the option to monitor them and put them into groups in order to minimize the need of the openHAB user interface.

3 High-level system structure

On the figure 3.1 is the design of our system which consists of 3 main components: server, client application and home user appliances.The server is running our SmartE application and it must be the same home server/computer where openHAB is running. User application, the entry point for user to have a look into the system and he will be able to use it in local area network and over the world wide web (insecure way). This document won't focus on the home appliances because all the communication with them is done by openHAB.

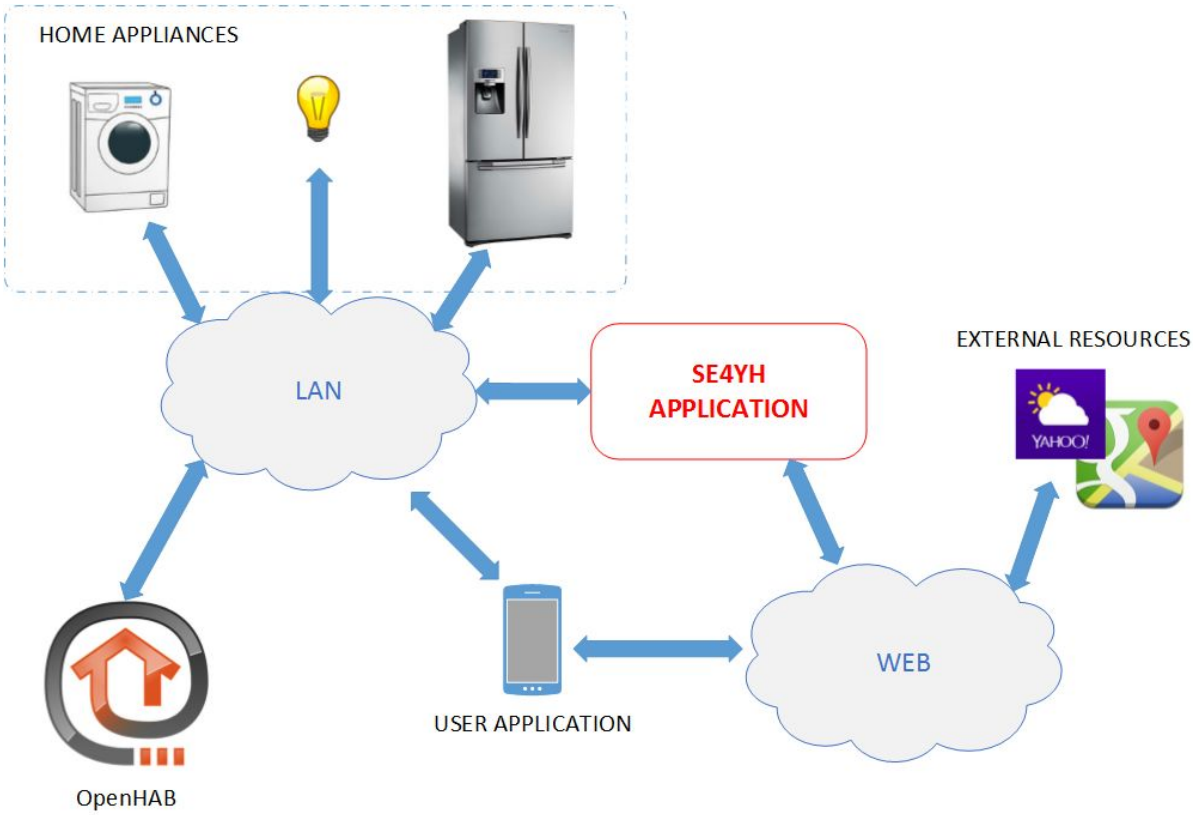


Figure 3.1: High-level system architecture

3.1 Communication infrastructure

OpenHAB is a powerful technology software for home automation. We will use it to simplify communication with home appliances. OpenHAB has built-in communication protocols for variety of devices. To SmartE, OpenHAB will provide a standardized virtual representation of home appliances to simplify the control of them.

3.2 SmartE application

SmartE application will contain the main functionalities such as rule management and automation of home appliances. The application will control the appliances through OpenHAB REST API and also provide content for the user application with its own API. The application should also communicate through the web with other services like Yahoo Weather to provide more information significant for the system.

3.3 User application

The user application provides the functionalities for monitoring and controlling the home appliances, adding new rules in order to conserve energy, easy to use for basic users but also with some additional functionalities for more advanced users. We will develop for the android platform. Other optional platforms are web and iOS platforms.

3.4 External resources

This part can provide our mobile and server applications useful data which can be used to extend the functionality of the application and the possibility to make new features.

4 Communication and integration

We can divide the communication into two parts: client-server communication and communication between two server applications SmartE and openHAB. These parts will be described in next chapters as well as interesting features and communication flow.

4.1 SmartE - openHAB

The main communication between openHAB and SmartE application is done by using openHAB restful API. OpenHAB is running on the same machine as SmartE application. Because of that, the openHAB restful service is listening on loopback IP address with port value of 8080. Using this communication SmartE application gets devices information and perform actions on them.

Apart from restful communication SmartE is using one way communication for rule insertions. SmartE opens and edits openHAB configuration files in order to store new rules, and make changes on them. It can also create new configuration files in order to separate the logic. This feature allows us to simple usage of the openHAB rule engine.

4.2 SmartE - client application

Restful API on SmartE application is the main and only connection to our client application. It is a two way communication. Two sides are exchanging data in the form of standard JSON data packages. Communication can be done over LAN and WWW and its flow is presented on figure 6.1 with green lines.

4.3 SmartE API

SmartE API is designed for the communication between SmartE application and client application using JSON format. In order to use it requests and responses must be formatted in the right way. This information can be found in SmartE API documentation which can be downloaded at this address: https://bitbucket.org/se4yh/dsd-project/wiki/Server%20API

Also the API returns different error codes for different errors so it is easy for debugging and handling occurred errors. Error codes and error description can be found on this address: https://bitbucket.org/se4yh/dsd-project/wiki/ERROR%20CODES

4.4 Communication security

If we want to use our application over the world wide web we have to ‘hide’ and secure user appliances to avoid unauthorised access to them. Because of this openHAB should be restricted to LAN only and we have to provide some security protocols to our communication between client and server application.

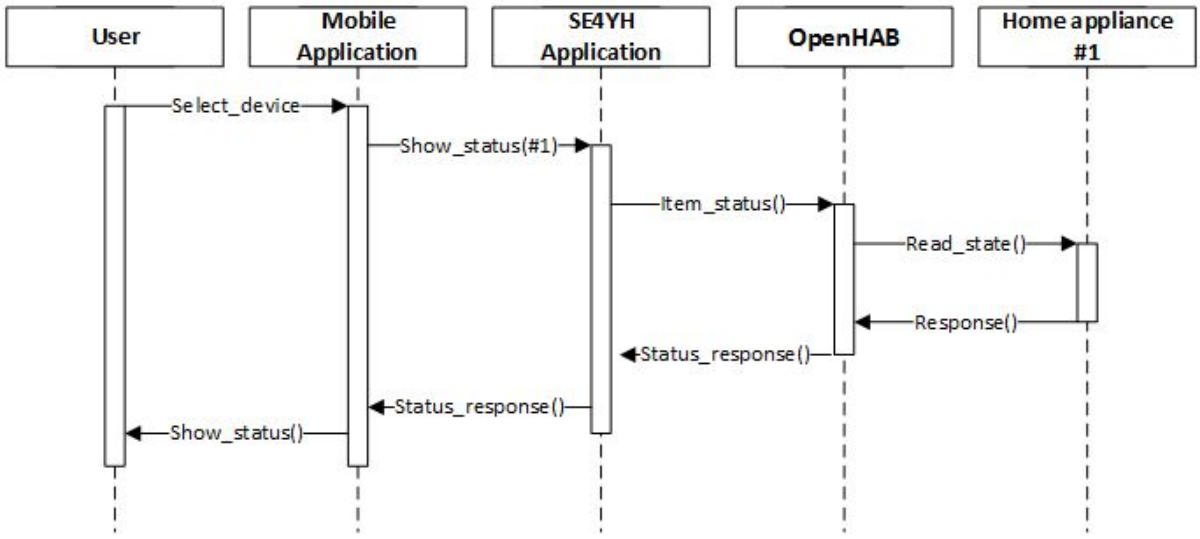
To ensure communication security between SmartE and client application, the client must attach a token in every call, the exception is the login call. After a successful login the client receives a token from the server and attaches it in every request to server in header as X-AccessToken.

4.5 Sequential diagrams

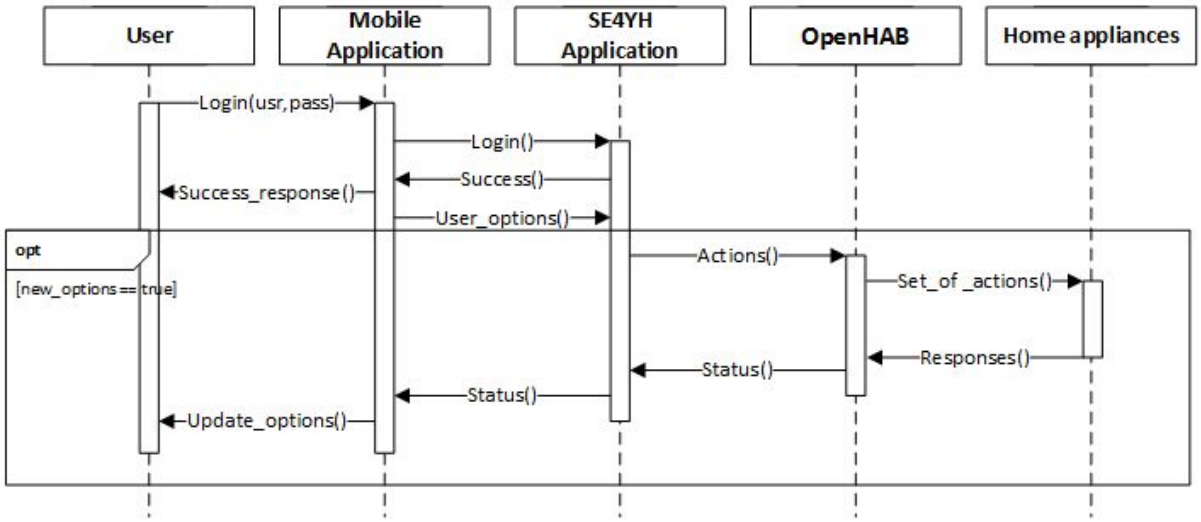
These sequential diagrams show the communication flow between parts of the system and represent some of the use cases.

Use case UC7 - *As a user I want to be able to change the status of a selected smart device.*

Use case UC6 - *As a user I want to be able to view the status of a selected smart device.*



Use case UC1 *- The user can connect to his SmartE System. Either he is at home or remotely.*



5 Technologies

5.1 OpenHAB

The main core of this project is based on using the OpenHAB[1] technology. It enables us to connect and control different predefined home appliances from light, climate, ventilation and heating to TV and remote controls. Connecting to that many different devices is enabled using OpenHAB bindings. Bindings are not only limited to devices but also supports protocols like HTTP. Commands to bindings are provided using events. There are mainly two types of events: commands which trigger an action or a state change of some device and status updates which inform about a status change of some device. Events enables easy transfer between devices similar features but different connection protocols.

5.2 PHP

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. It is used for server side application and also for device simulators.

5.4 Laravel

Laravel is a PHP web application framework, intended for the development of web applications following the MVC architectural pattern. Some of the features of Laravel are a modular packaging system with a dedicated dependency manager, different ways for accessing relational databases, utilities that aid in application deployment and maintenance.

5.3 Java

Java programming language is used to build the user mobile application. Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented,

5.4 Android

The user mobile application will be built for the Android mobile operating system. Android is a mobile operating system (OS) currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smart phones and tablets.

5.5 Espresso

The Espresso[2] testing framework, provided by the Android Testing Support Library, provides APIs for writing UI tests to simulate user interactions within a single target app. A key benefit of using Espresso is that it provides automatic synchronization of test actions with the UI of the app you are testing. Espresso detects when the main thread is idle, so it is able to run your test commands at the appropriate time, improving the reliability of your tests. Heavily used in testing client application.

5.6 Dagger 2

Dependency injection framework for Java and Android used for developing client application. Dagger 2[3] is the first to implement the full stack with generated code. The guiding principle is to generate code that mimics the code that a user might have hand-written to ensure that dependency injection is a simple, traceable and performant as it can be.

It was used for testing and to ease further changes in client application data modeling.

[1] http://www.openhab.org/
[2] https://google.github.io/android-testing-support-library/docs/espresso/
[3] http://google.github.io/dagger/

5.7 Git

Git is used for source code management system in software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. We used *git-flow,* a Git extension to provide a better branching strategy and release management[4].

5.8 Crashlytics

Tracking application stability is very important to ensure product lifetime and good user experience. We add Crashlytics, free crash reporting solution that offers great statistics and analysis of every crash that occurred in our application. Knowing crash cause on every device makes it easier to solve.

[4] http://nvie.com/posts/a-successful-git-branching-model/

6 System architecture

Basic system architecture is shown on the figure 6.1. System architecture can be divided in 3 parts. First one is the server part which is the main part and contains: SmartE application, openHAB application and database. Second part of the system is Android client application and finally, the third part, is the home appliances. In next chapters we will describe each component of these parts except for the user appliances which are handled mostly by openHAB and for the interaction we are using openHAB representation of them.

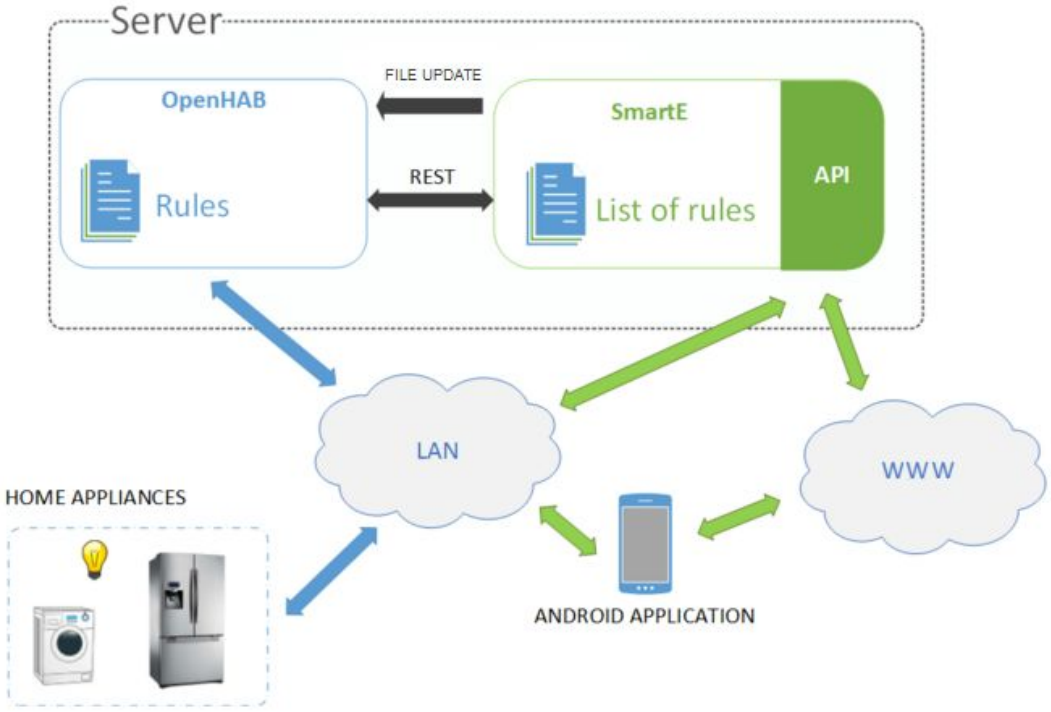


Figure 6.1 Basic system architecture

The server side of the system contains two applications: openHAB and SmartE.

OpenHAB supports many devices from different vendors and offers a presentation of them on an ‘abstract’ level in the form of items so it is easier for the SmartE application to handle them in that way. Also, it has great rule engine which is also used in our project.

The SmartE application is the core of the system. It is a connection between the client application and openHAB. It does many functions like handling API calls from the client application, interacting with openHAB to collect data from home appliances and sending actions (commands) to them, storing user rules in the openHAB rule configuration files and communicating with database.

6.1 OpenHAB

OpenHAB is a software which allows a user to control and automate behaviour of home appliances. Key components of the openHAB are: rule engine, restful API, bindings and event bus. We are using a restful API for communication with SmartE application. The rule engine is being used to achieve implementation and execution of SmartE predefined rules. The event bus and bindings are used for monitoring and testing of the system. Bindings translate events (commands) from the event bus to control specific devices using different protocols. For testing and simulating our system, some mock up devices were made and they are connected using openHAB Bindings as shown on the figure 6.2.

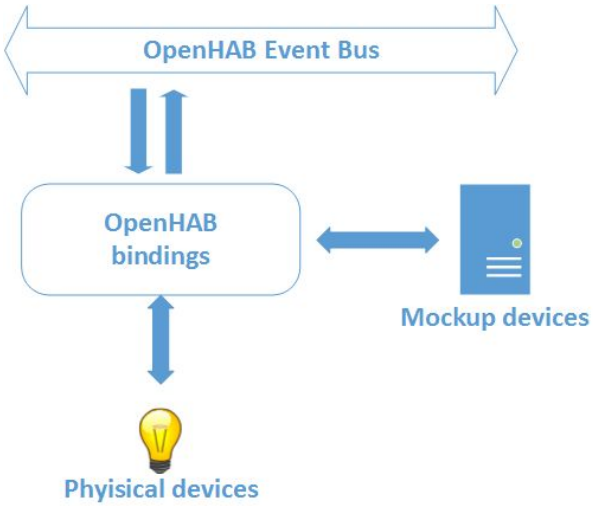


Figure 6.2 OpenHAB Bindings

6.2 SmartE

The SmartE application is the core of the system. It is a connection between the client application and openHAB. It does many functions like handling API calls from the client application, interacting with openHAB to collect data from home appliances and sending actions (commands) to them as well as storing user rules in the openHAB rule configuration files and communicating with database.

6.3 Android application

Our custom application built for Android will provides the user interface to see connected devices and rules as well as defining new rules. It communicates with OpenHAB to collect data and send commands. Commands are going to be generated by defined set of specific rules. Rules will depend on user preferences and the current state of the system (all devices).

The client application is built for mobile use on the Android operating system. It allows a user to see devices, rules and groups (each as a list) defined for his/her house. Groups can be created and deleted using the application, also, the user can assign different devices to desired group(s). The application offers users the functionality to create different rules for different devices. Each rule applies to only one selected device. Rules can be edited and deleted. Users can see the list of connected devices and the details of each device. For each device that has editable items (device properties that can be changed/set), the user can perform actions on those items. To use the application user must enter valid address of the server and login in with correct email and password.

6.4 Database

The database is used for preserving data needed for a functional system. We are using MySQL, an open-source relational database management system with entity-relationship data model, figure 2.3. There are typical entities like user and token and specific entities like items, rules, properties… etc. Entity rules have a specific role, it stores our predefined rules that will be transformed into a rule usable by the openHAB with dynamically entered user values via client application for that rule.

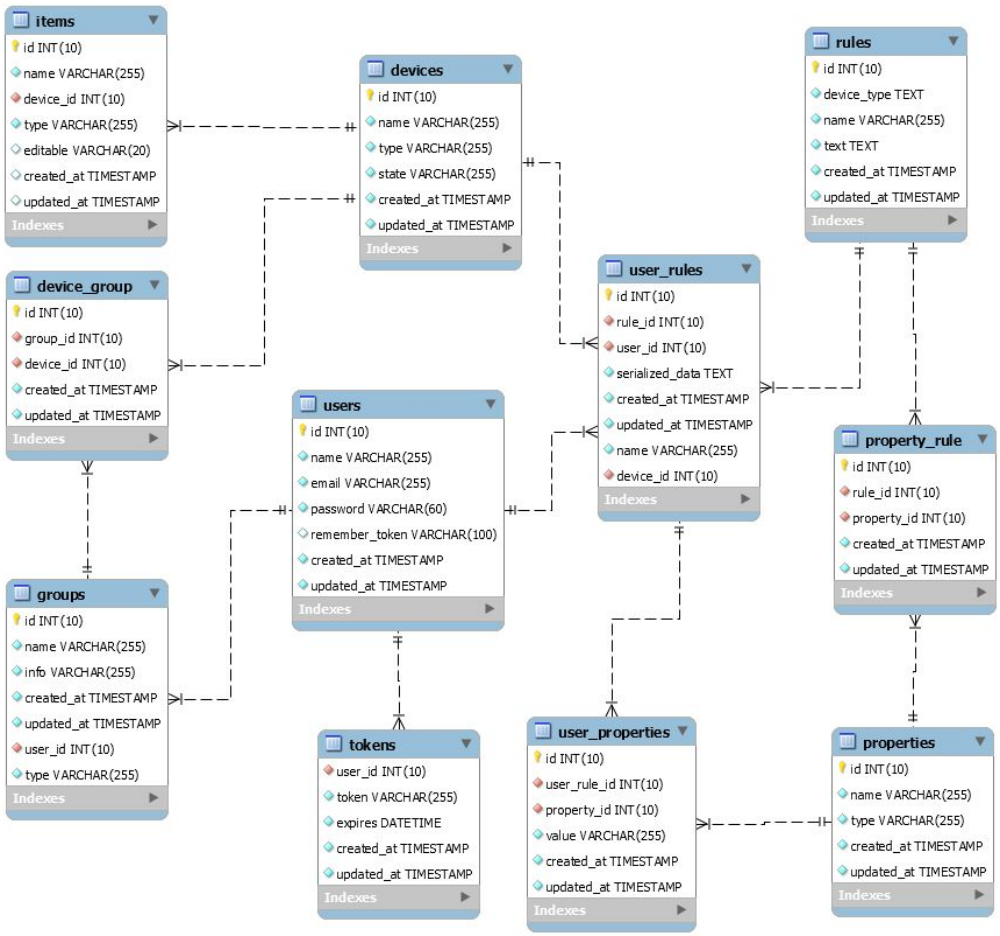


Figure 2.3: Database model

7 Mobile Application

7.1 Initial mock-up

For the mobile application on Android mock-ups were built to define each screen with basic functionalities. The start screen, *figure 7.1*, will provide two input boxes for user name(1) and password(2). User can login(3) if he entered valid user name and password or he can reset the password (4).
If the user name or password is wrong application must show appropriate message(1), *figure 7.2*.

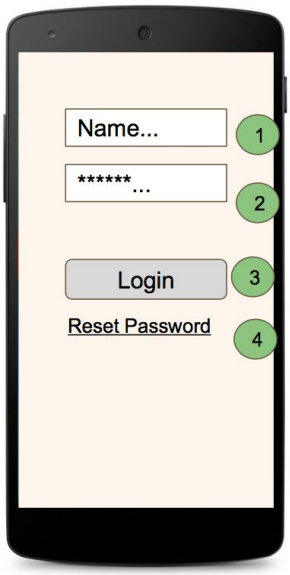


Figure 7.1: Login mock-up

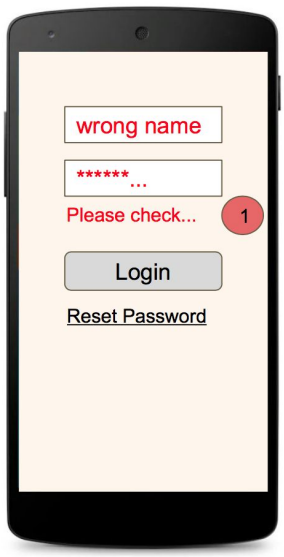


Figure 7.2: Login error mock-up

After successful login, *figure 7.3*, each screen will have a top Toolbar with a button for opening the Navigation Drawer(1), title of current screen(2) and Action button in right corner(3) that will provide different actions depending on the current screen. Rest of the screen will be used to display main content e.g. list of the devices.
After opening Navigation Drawer, *figure 7.4*, user will get list of actions(2) with appropriate icon(1).

Figure 7.3: Groups mock-up

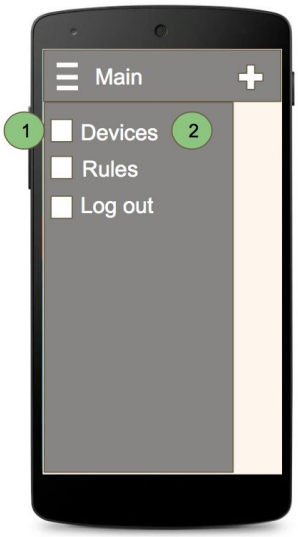


Figure 7.4 Navigation Drawer mock-up

7.2 Implementation

Final mobile application is implemented for Android OS. To launch the application, the device must have Android 4.4 (API 19) or higher.

The application consists of several screens which allow the user to see different groups, rules and devices defined in current implementation of openHAB for logged user. The navigation of the application is implemented using Navigation Drawer patter as predicted in mock-up. For the better understanding client application each screen will be described.

7.2.1 Login Screen

This is the starting screen of the application, *figure 7.5*, where the user has to login with defined user name and password in order to enter the application. Also, they can change the address of the server, address of our development server is predefined in that box. Users can select an option for automatic login, to remember entered user data: user name, password and server address, and skip the login screen on every application start up.

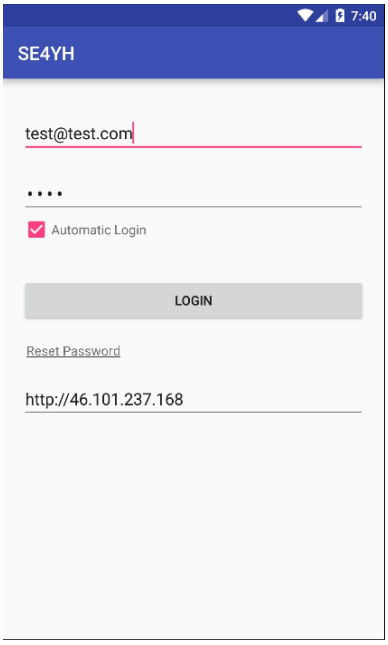


Figure 7.5: Login screen

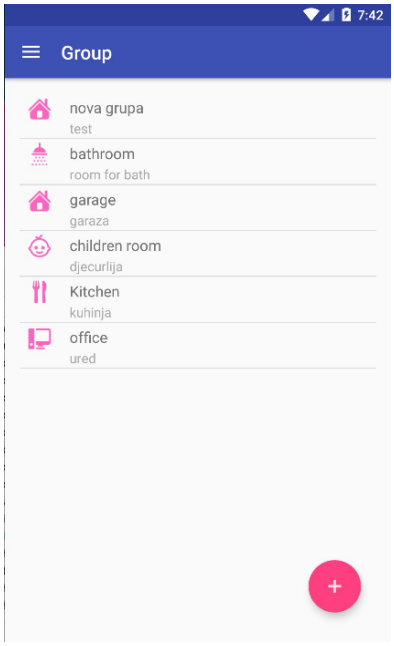


Figure 7.6: List of groups

7.2.2 Group Screens

After a successful login the user will see a list of all defined groups, *figure 7.6.* In the right bottom corner is button to create a new group. To create a new group the user must define name and optionally add, a description and group type, *figure 7.7*. The user can select each group from the list and see all devices added to current group, *figure 7.8*, and add new devices, *figure 7.9.*

Users can delete groups, using long press on list of groups, items will change and show check boxes enabling user deleting groups. After deleting the list of groups will be refreshed with new data.

The next option the user can do is open Navigation Drawer, *figure 7.10*, and navigate to other screens of the application.

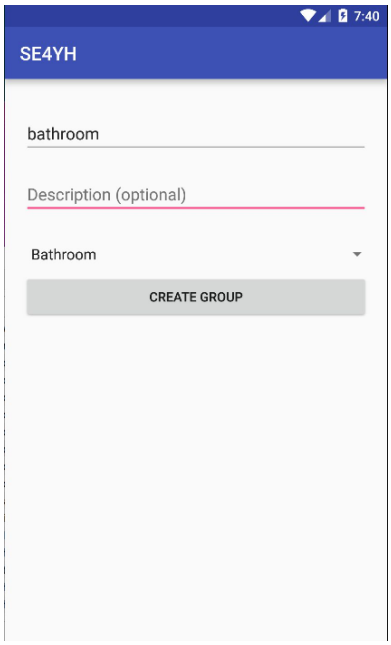


Figure 7.7: Add a new group

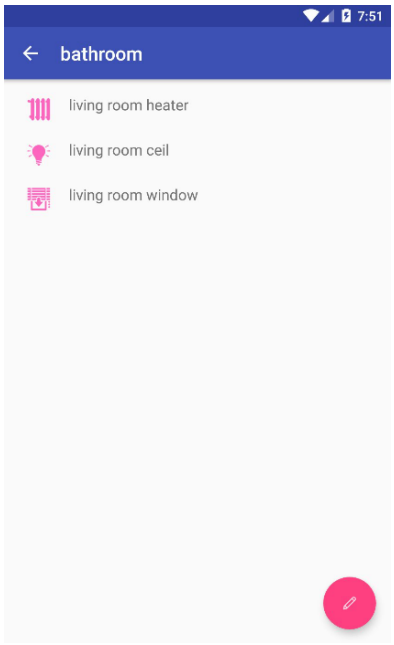


Figure 7.8: List of devices added to the group

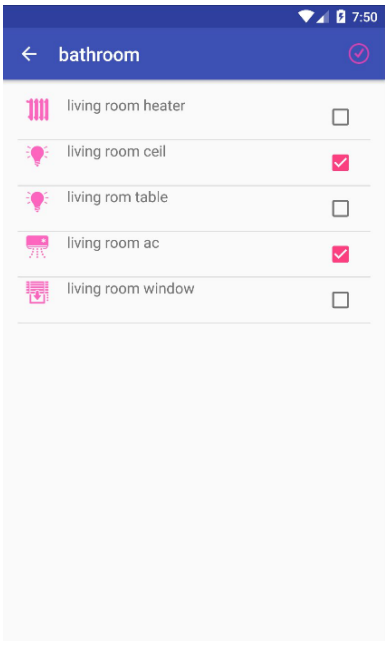


Figure 7.9: Add new devices to the group

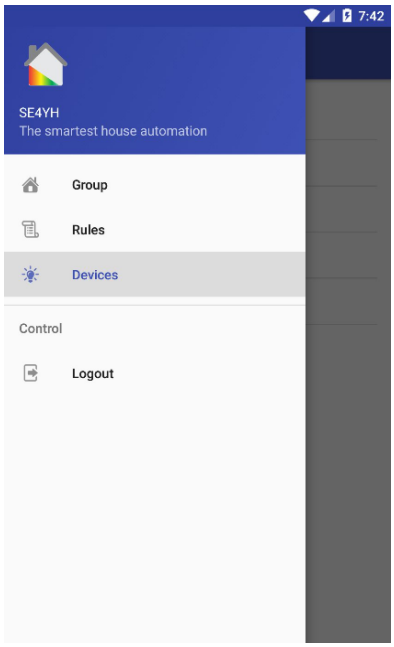


Figure 7.10: Navigation Drawer

7.2.3 Devices Screens

After navigating to Devices, a list of all devices binded in the openHAB will be shown, *figure 7.11*. Users can select each device, then see all of the device's details: name, type, a list of groups it belongs in and items specific for each type of the device, *figure 7.12.* If the device has configurable items, user can change and send the action, top right corner, to openHAB. After the action is sent, the device details are refreshed and applied action will be seen in details.

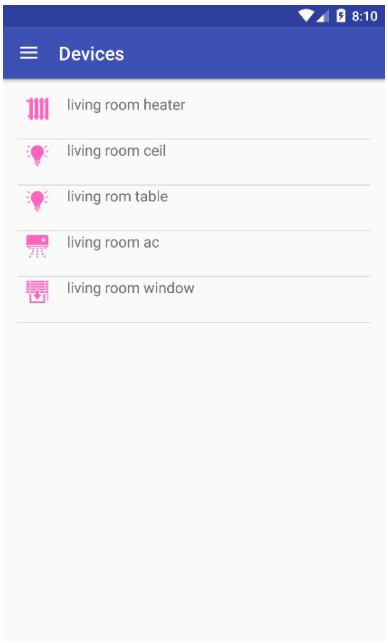


Figure 7.9: List of devices

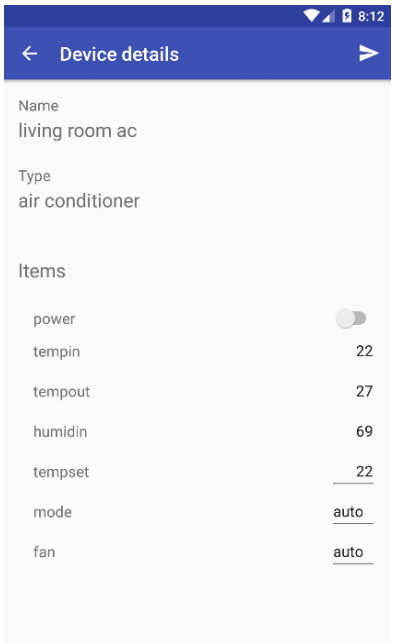


Figure 7.10: Device details for Air Conditioner

7.2.4 Rules Screens

The rules screens are the most useful and complex part of application, it can be accessed selecting Rules from the Navigation Drawer. First, there is a list of all existing rules for all devices, *figure 7.11*.

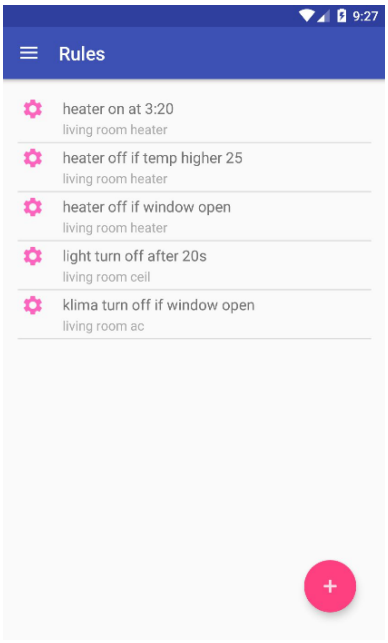


Figure 7.11: List of rules

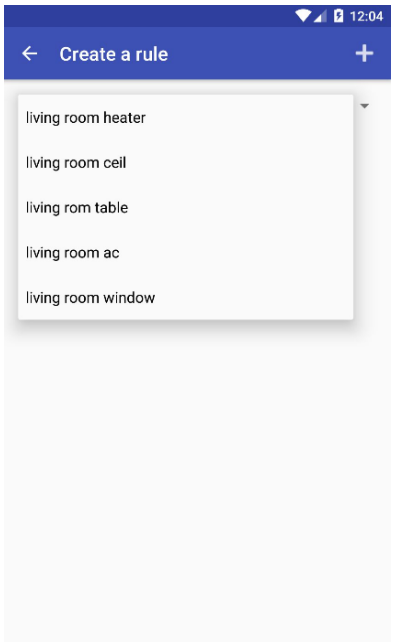


Figure 7.12: Select device to create a rule

The user can create his own new rule by clicking on the add button. Then user must select a device that he wants to create rule for from list of all devices defined in openHAB, *figure 7.12.*

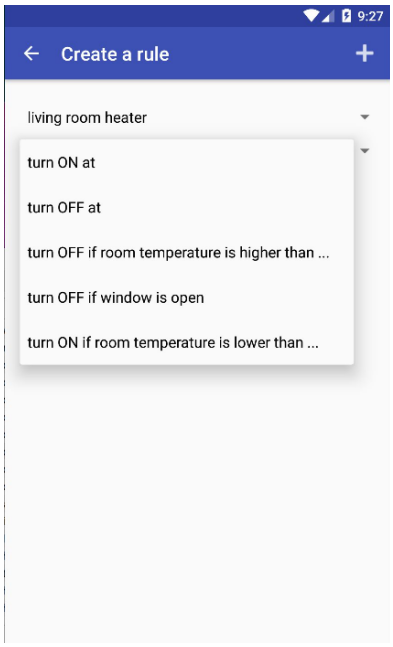


Figure 7.13: Create a rule for device type heater

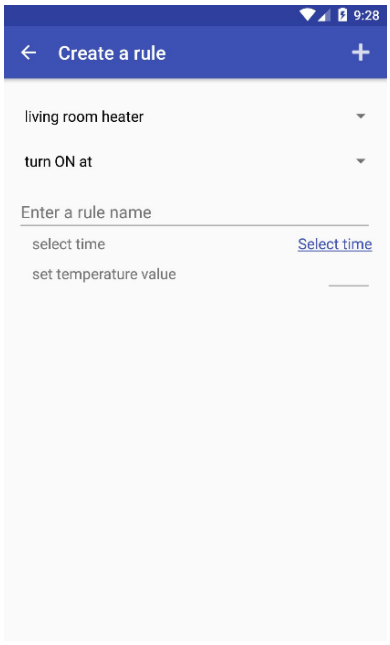


Figure 7.14: Create rule Turn ON at for heater

After selecting the desired device, he will be offered a list of predefined rules. Each device will have different number of predefined rules, e.g. device type heater will have 5 predefined rules,as shown on *figure 7.13*:

- turn on at set time with entered temperature
- turn off at set time
- turn off if window is open
- turn off if room temperature is higher than set temperature
- turn on if room temperature is lower than set temperature

Device type air conditioner have same rules but different logic for last two, cooling turns on if temperature is higher and turns off if temperature is lower than set.

There is also device type light with three rules:

- turn on at set time
- turn off at set time
- turn off after set amount of seconds

For device type contact, used as window contact, there are no rules because it can just detect state. It dose not have any item that can change or set value.

Each predefined rule will have different values to configure, e.g. rule *turn ON at* for device type heater will have item desired *time* to turn on at and wanted temperature for heater, *figure 7.14*.

Users can afterwards edit rule, just select the desired rule from the list of rules. There is an option to delete rules, the user must long press the list of rules and items will offer check boxes for selecting rules, *figure 7.15*. After selecting and deleting rules, new list of rules will be fetched.

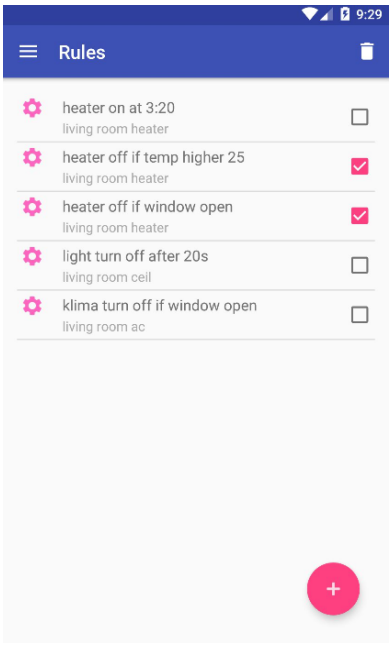


Figure 7.15: Deleting rules

7.2.5 Logout

The user can logout from application using Navigation Drawer, *figure 7.10.* After loggin out, all stored user data like: user name, token, server address will be deleted and user will directed to *Login Screen*.