Technical Project Report - Android Module

SmartDomotics

Course:	44139- Computação Móvel
Date:	Aveiro, Tuesday, 7 November
Authors:	72398: Nelson Reverendo 72447: Álvaro Martins
Project abstract:	As technological strides are made, we are discovering ways to automate our homes. This report describes the implementation of a home automation mobile application for the end user (owner of the house). This app enables the user to control different features of his house like lighting systems and to view live sensor data (temperature sensors, etc). Also, it is context aware which means it can locate the user's phone within the premises of the house by means of beacon technology.

Table of contents:

1 Introduction

In an era in which technology assumes a significant weight in society, it is essential to develop new technologies that facilitate people's daily lives. Therefore, the concept of domotics arises within the scope of automating routines and tasks of a home. SmartDomotics is a domotics application for smart control of a house.
This application allows the user to monitor the house state and actuate on lights and alarms, it also silently monitors environment sensor values and raises a notification on alarming situations.

2 Application concept

SmartDomotics is a domotics application for smart control of a house, and as such the users will be owners of the house and/or family. They will use the app primarily in the house, however, use outside of the house is also possible. The app will be used for house state monitoring and for interacting with the house actuators(lights, alarms …)

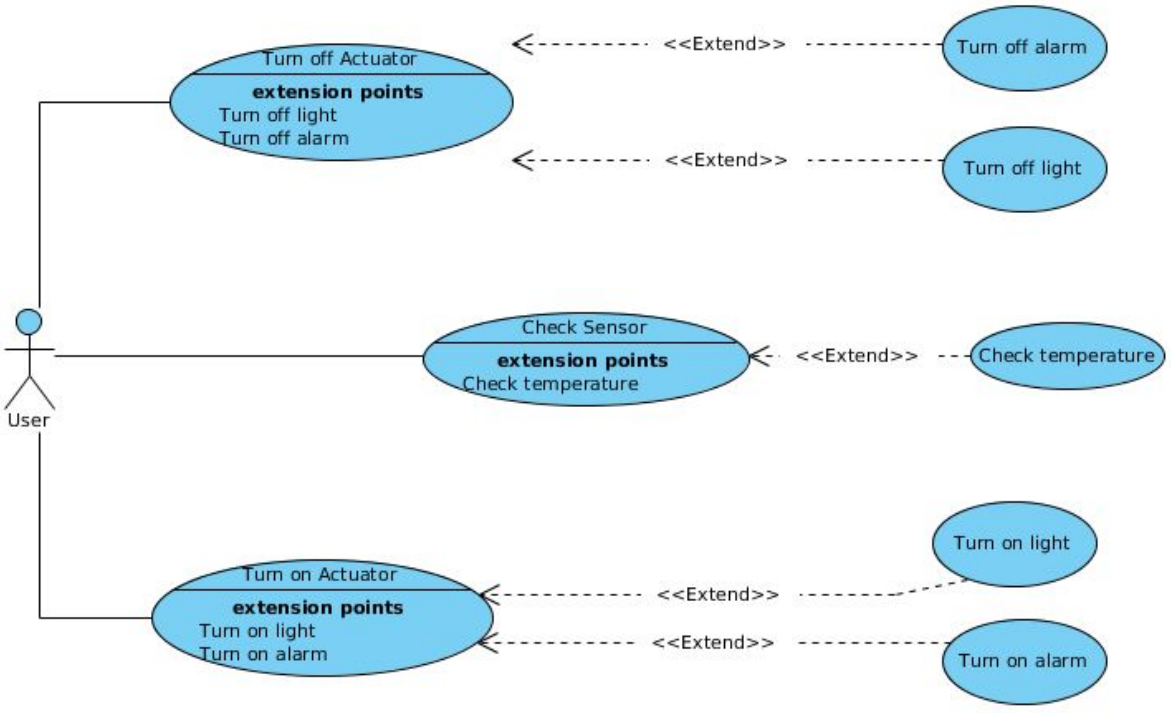

Figure 1. Example use cases for SmartDomotics application

Features

- Actuate (send commands to a remote controller and change lights/alarms)
- Read environment sensor values and raise notifications on alarms
- Context awareness

3 User experience design process

The app context is domotics, it allows the smart control of a house, as such the app icon is a house:

All used icons in the app come from: https://material.io/icons/

The color palette used was the android studio default for Android lollipop(5) as it seemed appropriate and we didn't feel the need to change it.

In earlier prototypes we used a Sidebar for division navigation but the need of change came quickly as it wasn't very user friendly changing the division on a navbar, so now we use a menu at the bottom

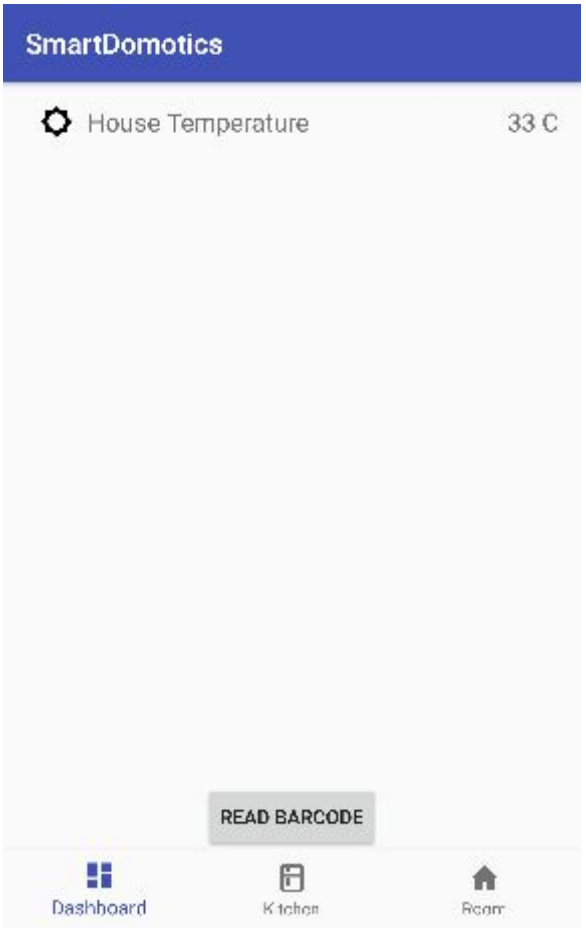

Figure 2. Dashboard from final version

When the applications starts up the beacon background searching starts, and if a suitable beacon is found the application changes to the matching division.

4 Architectural plan for the solution

Having the need for several asynchronous communications, such as beacon scanning and backend communication, we opted for the use of background services.

SmartDomotics has two background services, the beacon scanner that searches for beacons in the background(until found or timeout) and the ZeroMQ subscriber that awaits changes published by the backend.

For easy configuration with backend, the application also requires the reading of QR codes(will be presented by the backend) for this a google mobile vision example was adapted.

On a domotics context the application should use local (LAN) communications with the backend as much as possible, this forced us to stray from firebase and other cloud solutions and implement our own persistence model, communication model and synchronization method.

Communication

For communication message passing through tcp/ip was used, more specifically using ZeroMQ. To fit our domotics context we needed a communication model that allowed the app to communicate with the backend but also allowed the backend to publish to all the apps online, so two ZeroMQ patterns were used: REQ/REP* and PUB/SUB*. The application contains a subscriber that subscribes to two topics, the database change topic and the sensors topic, which allows the app to be up to date with the database while online. When the app sends a REQ to the backend, it simply publishes the change so that all online apps have access to the change in real time.

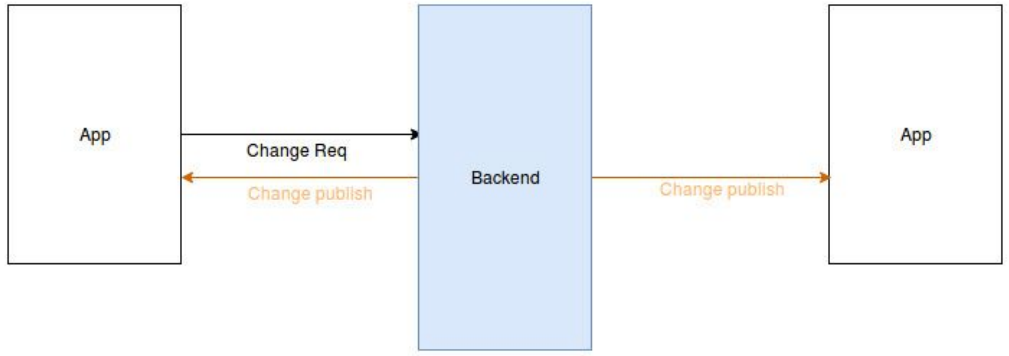

Figure 3. Communication architecture diagram with the REQ/REP and PUB/SUB patterns

However this raises a problem. This communication model does not guarantee that my app will be synchronized if it is offline during a change, so a synchronization REQ was implemented that allowed the app to synchronized on startup. The synchronization comes on the REP from the backend and does not require the backend to publish the synchronization as it would flood the network unnecessarily.

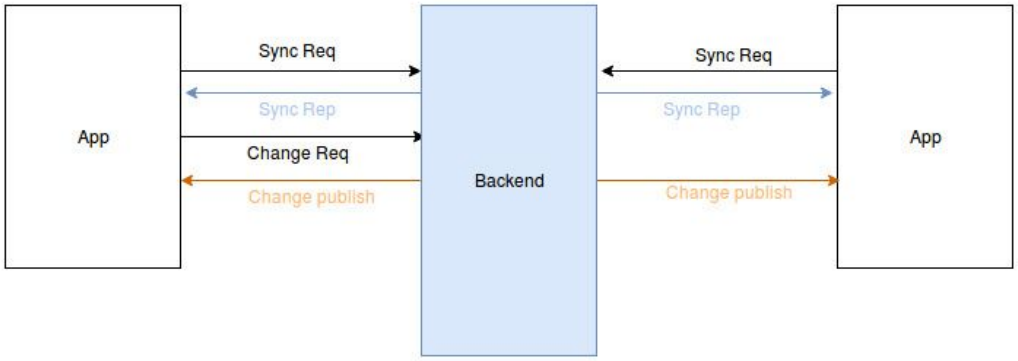

Figure 4. Communication architecture diagram with sync

Persistence

For persistence Android Room was chosen as it allowed to use an abstraction layer over SQLite simplifying the development process with no compromise on usability. Room also enabled the development of a single persistence solution for both backend and mobile application because of the use of Android Things for the backend.

Three major components in Room were used:

- Entities
  - This component defines the application persistent entities.
  - 4 entities were used, Division, Log, Boolean Entity(defines a light or a alarm, basically anything that has a ON/OFF status) and Sensor Entity(defines a environment sensor).
- Data Access Object
  - This component contains the methods used for accessing the database.
  - As such a DAO was created for each entity.
- Room Database
  - This component contains the database holder and serves as the main access point for the underlying connection to the app's persisted, relational data.
  - Allow the application to access the DAO.

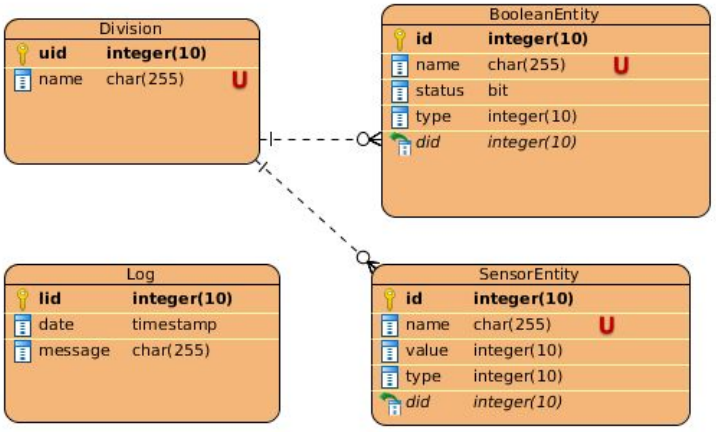

Figure 5. Application room entities

Android Architecture Components

Besides Android Room two more Android Architecture Components were used:

- ModelView
    - This component allowed to manage UI-related data in a lifecycle conscious way, observing changes in LiveData components.
    - Each Fragment has a ModelView that manages the fragment data and updates the UI only when necessary.
- LiveData
    - This component allows to call observers whenever data changes in a lifecycle conscious way.
    - The data in the data repository is of type LiveData.

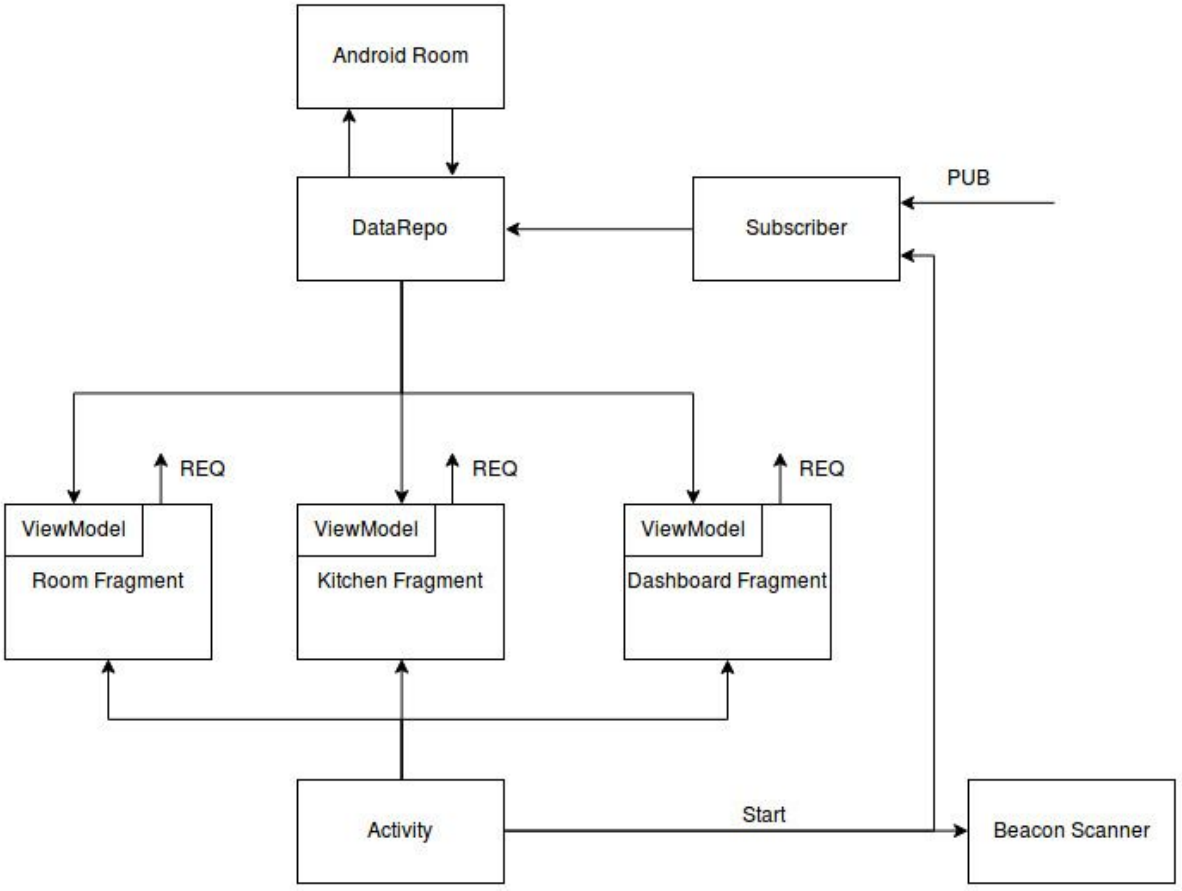

Figure 6. Overall application architecture

5 Implemented solution in Android

The technologies used in development were:

- ZeroMQ
    - Message passing through tcp/ip.
    - this was used for the communication model.
- Google mobile vision + zxing
    - this was used for QR code generation and reading
- Kontakt io SDK
    - beacon development SDK
    - used in the beacon functionalities of the app
- Android Things
    - is an Android-based embedded operating system platform by Google
    - allows to run nearly unmodified android applications on RPI3
    - is used for backend and device aggregator

When the app is started, by default it starts at the dashboard, however if a suitable beacon is found it changes to the corresponding division.

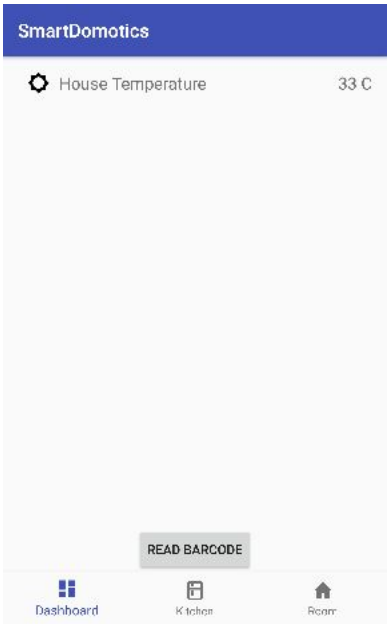

Figure 7. Dashboard, default user view

In the Dashboard we present “universal” sensors and/or actuators(universal as they don’t belong on a particular division). In this case only the temperature sensor doesn’t belong in a division.

If the user clicks on the READ BARCODE button, a UI with a camera will appear that will read the backend QR code to setup the system, in that UI the user must click when the camera presents the QR value, after clicking it returns to the Dashboard UI.

If the user clicks on the Kitchen menu tab the UI will change to the kitchen.

Figure 8. Kitchen UI

In the Kitchen tab we present actuators and sensors that belong in the kitchen division. In this case we have the smoke alarm and the kitchen light. The user can click to turn ON/OFF each of the actuators.

If the user clicks on the Room tab the UI will change to the Room.

Figure 9. Room UI

Much like in the kitchen the room UI presents sensors and actuators that belong in the room. In this case we have a light and a chandelier, both of which the user can click to turn ON/OFF.

6 Backend

As backend a raspberry pi 3 running Android Things was used.
As such much of the resources used in the application development were able to be reused in the backend development.
The rpi3 is mainly used for data persistence and log storing. It is also a device aggregator and is the main component that communicates with the sensors and actuators, i.e. all mobile applications communicate with the rpi3 and the rpi3 will communicate with the actuators and reads the environment values.
The data persistence model using Android Room was reused in the backend development, and most of the ZeroMQ code was also similar to the used in the application.

Figure 10. Rpi3 Android things UI seen in the emulator

The backend UI contains the QR code to be read by the app, and also contains the state for each of the actuators.

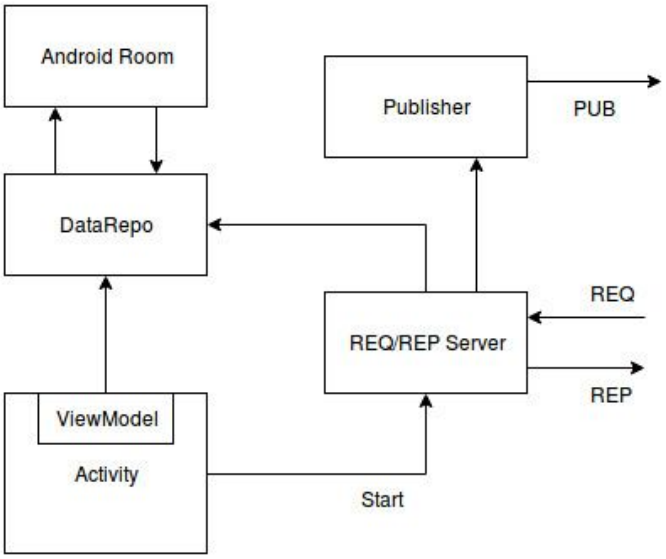

Figure 11. Rpi3 Android things architecture

7 Conclusion

All of the application features were successfully implemented, however while developing the application and the backend several problems arose, most of them distinguishable between backend and application.
On the application several problems appeared:

- Understanding how ViewModels worked and how to interact with them, this was solved by creating a Data Repository between the Database and the ViewModels
- DataBase synchronization between backend and application was a challenge as it required to pass Database state between messages and in the meanwhile allowing the UI to update

There was another problem, though not related with the application requirements, while trying to implement Music Controlling as the backend was using Android things, this was not solved.
On the backend(Android Things):

- The biggest challenge on the backend was implementing the typical backend services, such as the publisher and req/rep server, because typically Android is the client, no material was found for using android as backend. This was solved by trial and error and adapting server examples on other languages and implementing in java that is valid for Android.
- Android things configuration initially was very time wasting and error prone.

Overall the application checks all of the requirements however more features could be implemented that would fit in a domotics environment.

8 References and resources

-- as of 07 Nov 2017

https://developer.android.com/things/hardware/raspberrypi.html

https://developer.android.com/things/hardware/raspberrypi-io.html

https://developer.android.com/training/data-storage/room/index.html

https://developer.android.com/training/run-background-service/create-service.html

https://www.novoda.com/blog/minimal-zeromq-client-server/

https://github.com/kontaktio/kontakt-beacon-admin-sample-app

https://github.com/googlesamples/android-vision/tree/master/visionSamples/barcode-reader/app/src/main/java/com/google/android/gms/samples/vision/barcodereader

Glossary

REQ/REP - request reply pattern
PUB/SUB - publisher subscriber pattern
JSON - JavaScript Object Notationt http://www.json.org/

Project resources

Project resources for the Android module:

- Code repository: http://code.ua.pt/projects/cm-2017-2018-g01
- Ready-to-deploy APK:
    - Android things: https://meocloud.pt/link/9f976df8-0a5e-4361-86e1-2d5502e0a861/rpi3SmartDomotics.apk/
    - Android app: https://meocloud.pt/link/f8b12caf-0559-4a1c-a276-492a72a11e3f/SmartDomotics.apk/