VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in de Toegepaste Informatica

OPPORTUNITIES FOR DISTRIBUTED PHYSICAL/DIGITAL USER INTERFACES

DAVID CINO

Academic year 2017 - 2018

Promoter: Prof. Dr. Beat Signer
Advisor: Audrey Sanctorum

Sciences and Bio-Engineering Sciences

Abstract

In this thesis we present a project combining two current research fields namely Distributed User Interface and the Internet of Things. The main purpose of this project is the creation of a web application distributed across multiple devices. The web application called InteHome aims the management of home appliances such as the lighting, speakers, smart TV and the security appliances that are parts of the Internet of Things technology. The system is built especially with connected devices and sensors. Even more, the application is a multi-user system, we tried to demonstrate the potential of these research fields by making the application distributed across various devices but also across different users. With this application we tried to have a positive contribution to this new research field. Furthermore, we provided at the end of this thesis short guidelines for other persons who want to create this kind of application. Finally, in this thesis one can see the methods we used in order to create such systems and the benefits it generates.

Acknowledgements

This thesis would not have been possible without the help of several persons. Firstly, I would like to express my gratitude to my supervisor Mrs Audrey Sanctorum. She was always present to guide me, and her guidance helped me to realise my project. Her patience, comments and corrections were precious. Thank you.

Last but not least, many thanks to the people who supported and encouraged me during these years: my girlfriend for her presence, my parents and the rest of my family for their encouragement and support.

Contents

1 Introduction

Nowadays new technologies take a great part of our daily life. A lot of research is done trying to make users' life easier. People want an easier and faster way to perform daily actions since they are more occupied and have less time to lose with daily actions. With the expansion of the World Wide Web the number of connected devices increased drastically. This evolution created new horizons to let people manage their life in a faster and easier way. On the other side, the evolution of web technologies also increased monstrously. RESTful APIs made the development of complex systems, with different physical objects, easier and more powerful.

In this thesis we try to create new opportunities with existing technology fields. We focus on two research fields, namely distributed user interfaces and the Internet of Things and try to show the potential of these technologies in an innovative manner. Therefore, we created a home automation system that combines distributed user interfaces and the Internet of Things. The potential of distributed user interfaces in an Internet of Things environment is the most important aspect of this thesis. The goal of this thesis is to combine both fields of research by creating a web application that is distributed across multiple devices and integrates the concepts of IoT. Distributed user interfaces are nowadays almost not involved at all in home automation systems. This was exactly the challenge of this project.

Our solution can manage the audio, TV videos, lighting and the security of the home using directly the web application called *InteHome*. The system is used by different users. So, it is necessary to create a multi-user system in which their interfaces are distributed. We also developed the system in a way that data is produced by the users and the devices available in the system. Finally, in the administrator panel, we provide statistical information with the data that the system produces making this thesis as complete as possible.

1.1 Contributions

In this thesis we present the web application *InteHome* which contributes to the distributed user interface and the Internet of Things research fields. We made investigations in both research domains to acquire knowledge to build an innovative web application for an Internet of Things environment. The goal of the application is to facilitate some daily actions at home. Currently a lot of researches are done in both research domains separately. This thesis is to the best of our knowledge the first research that combines both fields and home appliances, in order to make an innovative home automation system where the user interface is distributed across multiple users.

1.2 Thesis Outline

The remainder of this thesis is structured as follows. First, related work in the domains of distributed user interfaces and the Interface of Things have been investigated. After the related work, our proposed solution is presented incorporating multiple aspects learned during the exploration of related work. In the fourth chapter meaningful scenarios for the different sections of the web application are presented. The most important part of this thesis is presented in the sixth chapter, after describing the methodology, the implementation of the system as well as its technical details, technology choices, hardware and database are discussed. After the implementation chapter we present also some guidelines and recommendations for such systems. Next, other interesting functionalities and ideas for future work are given in the last chapter. Finally, we give some conclusions for this thesis.

2 Related Work

Before the implementation, related existing works needed to be investigated. We investigated works in the domain of distributed user interfaces (DUIs) and the domain of Internet of Things (IoT). Both research domains are the core of the final prototype.

2.1 Distributed User Interfaces

A lot of new devices emerged with modern technologies, and this leads to new interaction designs. Distributed user interfaces are nowadays an active field for research. All references have another definition for distributed user interface. A definition given by Elmqvist is synthesising DUI definitions of follows [5]:

'*A distributed user interface is a user interface whose components are distributed across one or more of the dimensions input, output, platform, space, and time*'

Elmqvist's work gives an overall overview of the concept of distributed user interfaces. He describes DUIs as a subarea of the human-computer interaction (HCI), where the components are distributed across one or more dimensions.

He gives five dimensions namely, input, output, platform, space and time. Input, means computational devices which can give input into the system, for example, a computer with a keyboard also called *input redirection*. Output is mostly the graphical device who displays the results to the user also called the *content redirection*. Next to these two basic dimensions we have the platform dimension: a distributed user interface can work on one or multiple computing platforms (i.e. operating systems or networks). The space dimension is also very important, DUIs can work in the same physical space or in different spaces all over the world. Finally, the time dimension describes either that interface components are working asynchronously (distributed in time) or synchronously. While Elmqvist lists the multiple dimensions for distributed user interfaces, Paternò and Santoro's research [6] provides a logical framework with a set of aspects that helps people to analyse the potential of applications in a multi-device environment. This is useful for the evaluation and the creation of new multi-device environments.

Paternò and Santoro's analysis is divided in several aspects. Most of these aspects are not limited to distributed user interfaces but they address multi-device user interfaces in general. The first aspect proposed in this proposal is the *user interface distribution*. This aspect analyses the ability to distribute the user interface across different devices at a given time even if it is duplicated. The second aspect is the *user interface migration*. Here the continuity of an application when switching from one device to another is analysed. A user should be able to change from one device to another without losing the current application state. Migration and distribution are two different concepts. The *user interface granularity* is the aspect which analyses which part of the user interface is distributed. It can be the whole user interface or parts of it. Components (inner functionality of a user interface) can also be distributed. For example, instructions given in a command line interface with the result shown on a large screen. The dimension *Trigger Activation Type* analyses how the distribution of the interface has been triggered. In the domain of distributed user interfaces, the time is also an important dimension to consider. Here two different time effects can occur, it can be immediate or deferred. The *user interface adaptation* is the faculty of a user interface to change when the device is changing too, and the transformation done by the interface to make this adaptation possible. This aspect is also bound with the *user interface granularity* as explained previously. Finally, the architecture of the application is also important in this domain. It is important to know which architecture the application is running. Mostly the architecture can be either peer-to-peer or server/client.

A well-known project which illustrates the concepts of distributed user interface and migration is the Deep Shot framework from Chang and Li [2]. Chang and Li have focused on one important scenario, namely how to go from one device to another without leaving the task that people are performing. They firstly need the internal state of the application or website using e.g. the URL. Then copying it by hand or use another medium like email.

The problems with these methods are that desktop applications do not provide the current state and website URLs may also not give the state, you should navigate through the website to find the right state. Deep Shot gives a solution to these problems.

With the Deep Shot application users can migrate tasks by exchanging the state of applications between mobile phone and computer using the phone's camera. The user must install Deep Shot's software on the phone and the desktop and then take a picture using the camera, the software can determine what the application or website state is and open it on the other device using a digital vision algorithm. This scenario is demonstrated in Figure 2.1
.

Figure 2.1: Deep Shot application interface and workflow

Four scenarios, explaining the different manners to interact between devices using the Deep Shot application are given. The first scenario is the migration of information from a computer to a mobile phone.

Secondly, the reverse of the first scenario, here information is also sent but from the mobile phone to a computer. The third scenario is exchanging information between two computers by using a mobile phone as a bridge. Finally, it is possible to distribute content between two phones by taking a picture of the other phone screen.

The interaction between a phone and a desktop is very useful. The Unified Remote Project is like Deep Shot a popular application founded by Bergqvist and Berglund[1]. They are two Swedish students that began the project in 2010. Their initial goal was to create a mobile application to control remote desktop programs.

The first version of the application was an Android application that contained more than ten remote controls for programs like Spotify, VLC, Windows Media Player, PowerPoint, and so on. The second version released in April 2011 was a full version with a lot of new features like computer mouse and keyboard control, screen viewer and remote Wi-Fi/Bluetooth connection. A last update was in 2014, the user interface is more user-friendly and the application is available on different platforms and support for controlling all possible operating systems. In Figure 2.2 we can see the new user interface.

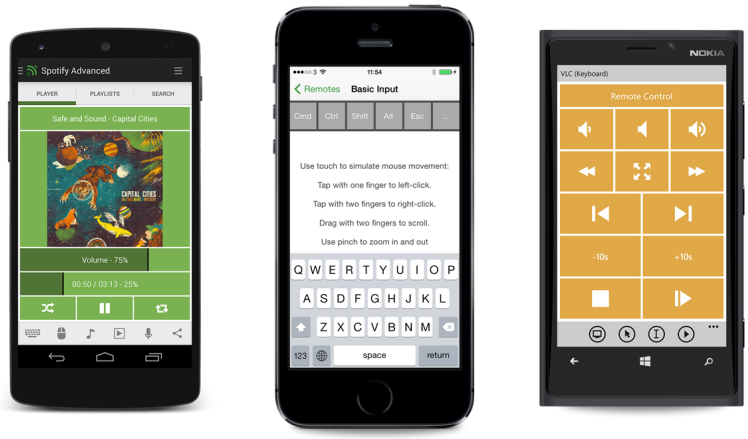

Figure 2.2: Unified Remote Project

[1] https://www.unifiedremote.com/

The PowerPoint remote controller is a very useful and well-known application in this domain. This is also what Demeure et al. did in their project [1]. CamNote is a distributed plastic slide viewer (presentation software).

CamNote is like the Microsoft PowerPoint software, with extra features, it has been developed during the CAMELEON project. CamNote can be distributed across a computer and/or a pocket computer and is composed of three main components.

The first component is a slide viewer with the possibility to insert translucent videos provided by cameras. This component is working only on computer. A note editor and viewer is also a component only available on a computer.

The user can enter comments on each slide. The remote controller component is distributed across the computer and the pocket computer. This component allows the user to navigate through the presentation.

CamNote's interface can be rotated (Figure 2.3), which makes the user interface more flexible and makes transitions between computer and pocket computer more effective and realistic. The user can wave the slide viewer window until it disappears from the computer and appears into the pocket computer. The opposite is also possible, when the window is shrunk, it returns automatically to the base configuration.

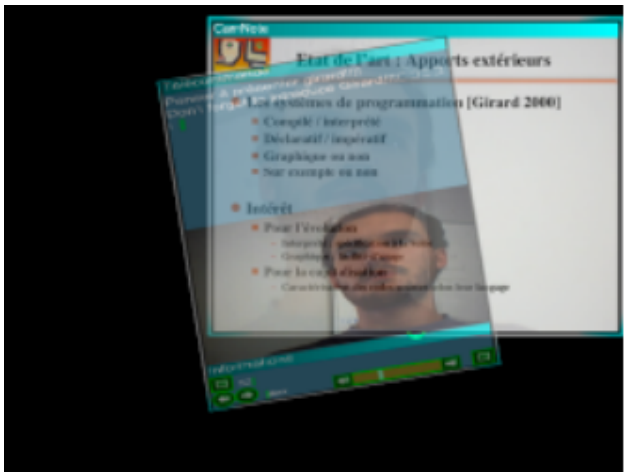

Figure 2.3: CamNote interface

Distributed user interfaces can be used to improve collaboration meetings. Co-interactive table is a system founded by de la Guía et al. to facilitate the

group work process in a collaborative environment [3]. Co-interactive table is composed of A4 format panels, each panel shows the different operations that the user can do during a meeting.

The operations are illustrated by pictures and the functionality offered by a RFID tag is hidden under the picture. Each user has also a mobile device equipped with a RFID reader and the co-IT application.

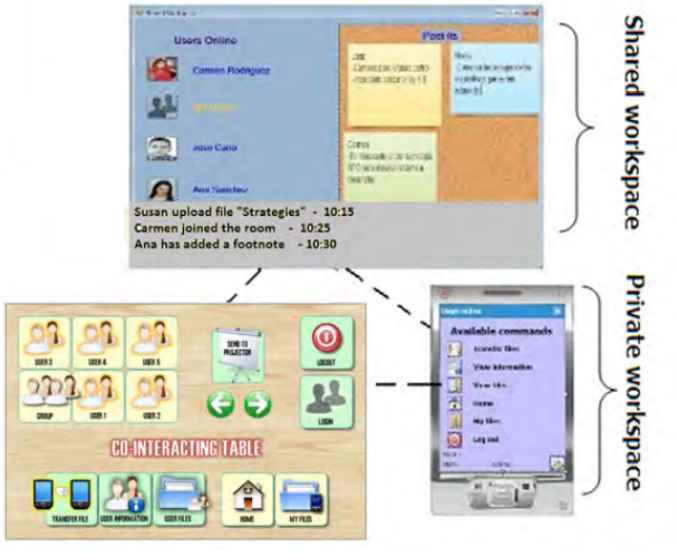

Figure 2.4: Co-IT, the shared interface and the private interface with all their functionalities

The functionalities are provided by the panels. The user must first log in to associate the panel with them, in Figure 2.4 we can see the interface a user gets after the login. The user can then share files with one or multiple participants and save information. Users can see information and files from other participants by selecting them on the panel, the idea is to facilitate the communication between participants when they do not know each other. The user is also able to see all files received during the meeting and send their interface to the projector making their screen visible for all participants. At the end of the meeting the user can simply log out to end their association with the panel.

2.2 Internet of Things

The Internet of Things is nowadays a very popular research field. Some works around the Internet of Things were explored. The Internet of Things is actually a network of physical devices which are connected to the internet such as lamps, vehicles and home appliances. In this work investigation we try to get important information about this research domain, and acquire knowledge for our own prototype.

Smartwatches can be interesting devices to use in office spaces to support and digitally augment interactions. Bernaerts et al. [12] designed and developed an application running on smartwatches to make employees' life easier by giving them the possibility to lock and unlock office doors, get office room information and notify when the employee wants to enter in the office. The founders set the focus on precise use cases, namely how employees enter an office room and how they get information about the room. Based on these use cases the developers implemented three physical gestures to perform the three most important actions in an easy and fast way (Figure 2.5). The first gesture is a virtual knock the gesture is the same as a real knock. Second gesture is turning the wrist, like turning a key to open or close a door, this gesture is used to open and close the doors. The last gesture is swiping the arm to go to the home screen and scan the room.

Figure 2.5: Different gestures

The system also provides a lot of feedback necessary to make the system useful. For example, when a user wants to open the door of a room the

owner of the room can see which action was performed and he can choose if he lets the user enter or not. The owner can also be in the room and send a message to the user's smartwatch if they are busy or occupied. All these kinds of actions are also sustained with audio feedback in order that the user can perfectly know if the gesture was executed or not.

Smart TVs also play an important role in IoT-environments. In the survey from Yusufov and Kornilov [10] the potential of Smart TVs is analysed. Smart TVs can be used for data storage, for example, to save films and series and freeing disk space on the laptop. Of course, smart TVs can act as a visualisation device for external resources, for example, to share video content from a smartphone using a 'share to TV' button. For interacting with the IoT-infrastructure a Smart TV can also be interesting. A Smart TV can be used as remote, voice or gesture input controller. To manage something at home, such as the temperature or room lighting. Due to his large dimension a Smart TV can also be an external data processor for other devices. It can be interesting for making computations faster for other devices. Another way to use Smart TV's is to let them in charge of numerous routines.

An example given in this survey [10] goes as follows, a person comes to the home at 6 o'clock and has the habit to make some coffee. The Smart TV can automatically launch a command to the coffee machine at 17:55 to prepare a coffee, and thus freeing the user from this routine. Smart TVs can also collect and distribute precious data via sensors, these sensors can be added externally or embedded in the TV. TVs can collect various data like temperature, sound, smoke and water, which is then sent to the server. Camera streams are often used by TVs for security purposes, using them, for example, as face recognition device or as surveillance device.

After the analysis of the different functionalities that a Smart TV can provide, the authors give an overview of the platforms and middleware available on the market.

A middleware acts as a layer between the hardware and software and is determined by the platform, which is different from vendor to vendor. The platforms API's based on the middleware provides different roles that a smart TV can perform.

	Storage	Visualization	Interaction	Processing	Source of data
Samsung	Good	Good	Good	Average	Good
Panasonic	Good	Good	Good	Average	Good
LG	Good	Good	Good	Average	Good
Mediaroom	Average	Bad	Good	Average	Good
Google TV	Good	Good	Good	Good	Good

Figure 2.6: Platforms and middleware's rating [10]

In Figure 2.6 platforms and middleware are rated depending on the roles that they can perform. Good means that the platform can act in all use cases according to the role. Average means that the platform is limited in a particular role. And bad means that the platform cannot act in this role. From the table above, we can conclude that the Google TV is most suitable to act in an IoT-environment. Note that the Google TV's operating system also allows to run applications in background. Which makes notifications easier and makes the data processing very efficient.

Retrieving data from smart objects is nowadays very easy with the expansion of the World Wide Web. RESTful APIs are suitable for systems to communicate with each other, using a request and response communication over the internet. Vukovic [8] did a study about API's that makes the extendibility of IoT environments effective. RESTful API's power is that they provide highly consumable services that can be used and reused by different clients. They also create interconnectivity between different services and applications making developing across different application domains possible.

Vukovic describes firstly the main benefits of the IoT. IoT is useful for analytics that is derived by monitoring and analysing the information generated by the physical world. Our physical world is going to be actuated by these smart devices. These objects are also useful for security purposes, for example, in transportation road system and railway.

RESTful application programming interfaces (APIs) are for many developers a rapid manner to program highly consumable services. The power of reuse of these services is also a non-negligible aspect of APIs. APIs are meaningful for IoT since it provides a lot of benefits like data access and data normalisation, adding additional layers of security for IoT devices, versioning and management of devices.

The role of humans in the Internet of Things environment is often neglected. But humans can also use their own sensors in an environment. Humans can use their senses in three different ways: automatically, explicitly and implicitly. Vukovic explains that humans can also provide different sensing data. Automatically, with their devices sensors. Explicitly through manual data input of low-level sensors, such as the temperature. And finally, also implicitly in this case also through manual input. An example given by Vukovic is when a user is swiping a RFID tag pass on a gate, it can generate useful data. From the Internet of Things' perspective, the focus lies on the incredible amount of data produced by millions of sensors. The data is then processed, filtered and analysed. The result of the analysed data, provided by both mobile devices from people and IoT devices, influences people, systems and organisations by gaining insights in these data. RESTful APIs play an important role by unifying these data access through different sources and give us the capability to make context-aware and adaptive applications.

The Internet of Things has a lot of benefits in this world but a lot of domains are overlooked. One of the domains that is often neglected and where technology can provide a considered solution is the Internet of Things for people with disabilities. Domingo [9] made a research about and for people who are dependent of their families and are not economically active and socially included due to their handicap and a lack of services/support for these persons. In 2011 the World Health Organisation (WHO) estimates that 15% of the wold population are living with disability. Domingo believes that the Internet of Things can offer the assistance and support necessary for a good quality of life.

The Internet of Things architecture is divided in three layers. Each layer has their own proper main functionality. The first layer is the perception layer. This layer is responsible for identifying objects and gathering information. In this review assistive objects are introduced. They try to gather information from the environment of the disabled person. One of the future components is nano-sized artificial retina implants which will capture images from the visually disable person and send the data back to the smartphone. In the perception layer the RFID technology can be very useful. Another given example that helps blind persons is the smart RFID cane. The cane can help blind people to find their way by distributing RFID tags in areas. It can be used to direct the disabled person to walk into the centre of the pavement for example. Data from the cane is then processed and sent back as a voice message. Obstacle detection based on ultrasonic sensors can also

be added to the cane. For people who are hearing impaired, there also exist RFID-based objects. One example given in the review is RFID-tagged games for children to learn how to use sign language. Also, for hearing impaired there exist external or internal assistive implants that improve the hearing quality. Finally, a lot of sensors like smart doorbells, smoke detectors, for example, can help deaf persons to notify them when an event occurs or when there is a danger. Persons with physical problems can also have solutions thanks to the technology. A lot of new body sensors, actuators and neurochips make movements for paralysed persons possible. Some sensors are attached to the nerves and can detect the intention of the disabled person. Thanks to micro-implants some region of the body can be stimulated by electrical impulses and digital command data. Another solution is the exoskeleton for humans, intentions of the user are recorded and the skeleton can perform the action that the user was thinking.

The second layer discussed is the network layer. This layer is made up of private wireless networks, Internet and network systems. The goal of this layer is to transmit the information from the previous layer. Most of our Internet protocols are made for classic networks and not for mobile networks. There is a need that communication protocols fit the requirements of the Internet of Things.

The last layer is the application layer. This layer is accessed by applications and monitoring stations. They provide services like RESTful APIs where data can be accessed. The most important operations are performed in this layer and it also provides important functionalities like authentication, billing and service management.

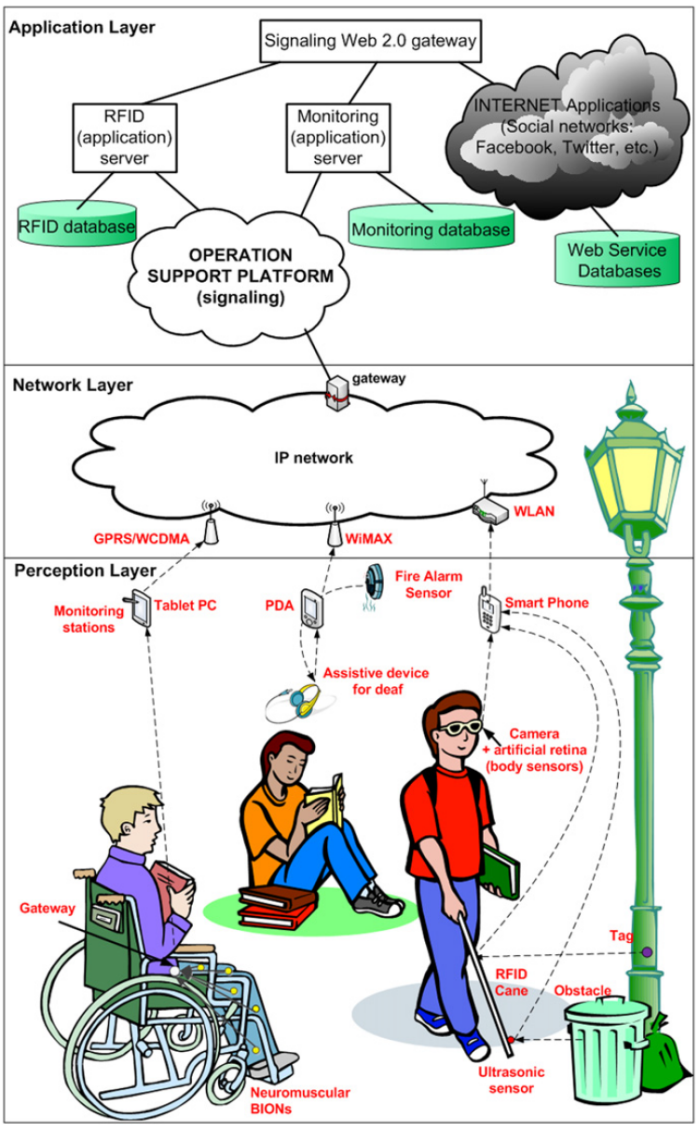

Figure 2.7: Three-layer architecture for the IoT

Next, some application scenarios for disabled persons are given based on this three-layer model. For people who are visually impaired an RFID based system can help while doing shopping. The smart cane introduced previously can help blind persons to navigate through the supermarket. The floor of the shop must provide RFID tags for navigation information. Also, products can provide RFID tags to get some information about the product itself but also from the ingredients, temperatures, etc. The data provided by the database is then sent to the monitoring station and returned as a vocal message.

The school scenario gives some examples where young persons with a handicap can learn and play in an interactive and intelligent environment. Also in this scenario the RFID technology is very useful. They can be used for blind people to find the right book in the library and then read them thanks to the text-to-speech module. Mentally disabled persons can also use the RFID technology to scan some objects and get the sound it emits. With the technology of augmented reality screens, we can imagine using them to make some objects much more real, for young persons with mental problems it can be very positive.

The last scenario which is given in this survey is at home. In this scenario RFID tags are also suitable for blind persons. You can attach RFID tags on different elements such as clothes to get information about colours, material, washing program and so on. Other goods like washing machine, refrigerator, thermostat, TV with their embedded computers can help disabled persons. These persons can use voice commands to enable some apparatus at home. Robotic systems are also a solution for physically disabled persons making everyday movements easier.

While Domingo focused his research on the Internet of Things for disabled persons, Parkash et al. [11] set their focus on smart street lights. The public service can also have great benefits using technology to maintain and to manage the street lighting. This research proposed an intelligent street lighting system which can decrease costs to -70%. In this research they find out that street light is one of the biggest energy expenses for cities. One of the reasons is that street lights are still working with a manual switch system. Lights are turned on the evening and turned off the morning, between the on-and-off timing a lot of energy is wasted. The authors of the research find a solution to this problem using the Internet of Things technology. The principle of this project is simple. It consists in using IR sensors when an obstacle is passing by the street is detected the lights turns on. When the lights turn on the resistance decreases and in the dark, it increases. The data produced by the system is displayed on a webpage in real time.

The current system needs more manpower for manually switching on/off the lights and it consumes more energy while the intelligent street lighting system requires less manpower and there is a real diminution of the energy consumption, CO2 emission, light pollution and maintenance costs.

This research helped us a lot to acquire the knowledge needed for the realisation of our project. Now, we know what distributed user interfaces are. We know the various ways to perform the distribution of user interfaces. The technologies used by other researchers were analysed to know what and how to use it. We walked over various smart home appliances such as a smartwatch and a smart TV. We will also use these appliances in our system. Finally, we read some papers about APIs. This software architecture is powerful for the communication with smart objects. So, it was necessary for us to know how to implement this architectural structure.

3

Solution

The proposed solution of this thesis integrates the two research domains presented in the related work, namely the domain of distributed user interfaces (DUIs) and the domain of the Internet of Things (IoT). We created a prototype of a web application for managing an entire home automation system that includes IoT devices as well as the user interface distribution functionality. The goal of this prototype is to show new opportunities and the power of distributed user interfaces combined with the Internet of Things.

InteHome is the name that we gave to the prototype. *InteHome* is a web application and it stands for 'Intelligent Home'. With this web application, we provide a fully innovative manner to work with devices at home. We tried to regroup different physical objects to make one central system. The application is split up into five different parts: the administration panel, the security part, the lighting part, the audio part and the video part. Each part will be explained later in more details.

The web application is used to perform multiple tasks at home by managing different devices. Of course, not everyone can have access to the web application and manage these objects. Therefore, to secure our system we have a login page (see Figure 3.1).

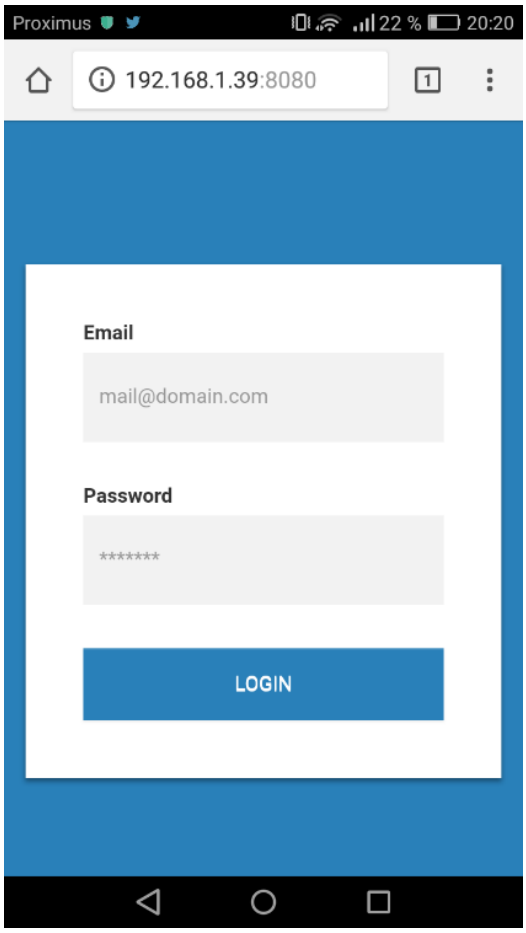

Figure 3.1: Login page

The login is composed of the email and password. If the users are registered in the system, they have access to the application. There exist three different account types. The *administrator account*, which has only access to the administration panel. Next, there is the *complete account* and the *minimal account*. The difference between both is the content of the application. The complete account has access to the security section while the minimal account has not. Imagine you have children, you do not want the child playing with the security settings. This is the reason why there are different account types. Notice that this multi-user system has a considerable influence on how the user interface is distributed according to the account type. The application is also distributed across different devices. The only constraint is that the device must contain a browser to access the web application. This multi-user system with different account types is innovative in the home automation domain. Other systems do not provide multiple ac-

count types that can manage one instance of an object. Generally, one has to make a single account in the application. Next, we will discuss the hardware that was used and the global infrastructure of this prototype.

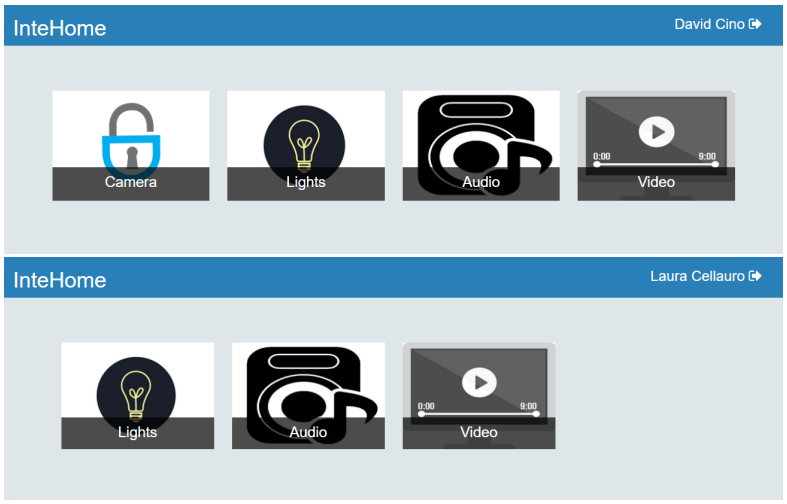

Figure 3.2: Home screen of the web application with the two different account types: the top picture demonstrates the *complete account type* and the bottom one demonstrates the *minimal account type*

3.1 System Infrastructure

For the comprehension of the system and the devices available in this system, in Figure 3.3 one can see the system infrastructure explaining the composition of the local network with their available hardware, objects and devices.

Figure 3.3: System infrastructure with their devices, objects and sensors

To make the video and audio part working properly, the system needs to be connected to the internet. Therefore, we have a router who is connected to the Internet and who gives to the devices a private IP address. The central unit and brain of the system is a raspberry pi model 3[1]. It contains the web server, the database and the storage. The raspberry pi is connected to the home Wi-Fi. The speakers are directly connected to the raspberry pi with an auxiliary cable. The camera can be connected with USB 2.O cable or via the local network. Finally, the PIR motion sensor is connected to the raspberry pi using GPIO connectors.

For the lighting system, we use TP-Link HS110[2] smart plugs. The smart plugs are connected via Wi-Fi to the local network. Thus, we send requests from the raspberry pi to these smart plugs over the local network. We also

[1] `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`
[2] `https://www.tp-link.com/en/products/details/cat-5258_HS110.html`

have multiple devices that are used to access the application such as smartphones, tablets and desktops. Smart TVs are a little bit different since they can only be used to display videos (see video section). Finally, the Fitbit Ionic smartwatch is used to manage the home lighting. This smartwatch works only if it is coupled to a companion (smartphone).

3.2 Administration Panel

The administration panel can only be accessed by the administrator. The user interface is accessible from different devices thatwho contains a browser. The main functionalities of the panel are managing the users of the system and displaying important data produced by the devices and sensors. Some devices, such as, smart plugs produce useful data and are easily accessible using their APIs. In the panel we display data like light consumption that was provided from the smart objects, further we also display data retrieved from the database which provides data about users, requests and events. Finally, the administrator can add, delete and update a user. One of the most important roles of the administrator is to give the right account type to the user.

3.3 Security

This section contains the security of the home. The only account type who has access to this section is the *complete account type*. From this section the user can activate or deactivate the alarm system. When the alarm system is activated by a user, the motion sensor turns on. When the sensor detects a movement in their view angle, it fires the siren on the home speakers. Automatically, a picture is taken by the camera and distributed with information, such as the timestamp, to all *InteHome* users. The user can then view the pictures on their device and delete them when finished with this notification. The user interface is distributed since each user can manage the single motion sensor instance. On top of that, the picture is distributed to all users connected to *InteHome*.

Users can also watch the stream of the camera. Each user can have access to the camera stream at the same time. The camera stream is thus distributed across multiple interfaces and devices. The user who is looking to the camera stream has the possibility to take a picture and distribute it with other users. The interface has three different compartments: the camera stream,

the alarm switch and the notifications. In the notification compartment we show only the last five notifications whether from the alarm system which is sent automatically when the motion sensor detects a movement or from another user who sent a picture from the camera.

3.4 Lighting

This part of the system allows users to access the home lighting system. The user can turn on/off lights. Each user who has access to this section can see the power state of each light in real time like in Figure 3.4. Thus, here we also have a distribution of the interface. The physical object, which is the light in this case, has only one instance and this instance is distributed across the different user interfaces. Furthermore, this part of the system is distributed to a smartwatch application. This feature is new for the home automation domain, from the smartwatch you can also manage lights and get the real time power state. Even better, when swiping on the light that we want to manage, the interface is directly distributed to the smartwatch. The smartwatch opens the application like in Figure 3.5, with the appropriate light that we want to manage. This is a nice feature that shows the power of distributed user interfaces.

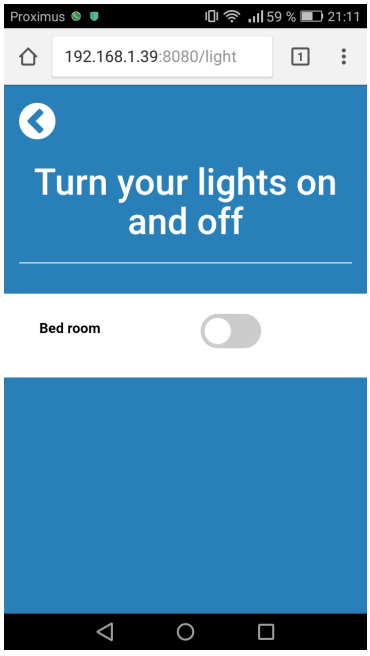

Figure 3.4: Light section with the switch to turn on/off the bed room light

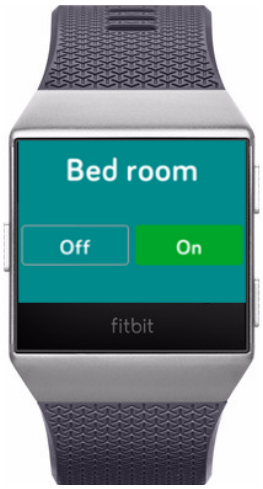

Figure 3.5: Smartwatch application to manage home lights

3.5 Audio

This part of the application is also a great part showing different aspects of distributed user interfaces and physical objects playing an important role. The user searches for a music in the search bar and the result comes to a list view. The list view is composed of a response from the YouTube API. So, we receive a list of songs or more properly audio streams. When the user clicks on the audio stream in order to listen to it, it he wants to listen one instance is played on the home speakers. Each user who has this section part of the application open on his their device can see an icon appearing on the interface, it means that an instance of a song is playing. The instance of the songs is thus distributed across multiple devices and users. The icon has two utilities, firstly informing users that a song is currently playing on the speakers and secondly allowing each user to stop the song by can clicking on this icon to stop the song instance. If the song finished ends,of course this icon disappears. While the song is playing users can search for another song and play this new instance, the previous instance is then overridden with the new one.

Since all requests for a song are recorded in the database we also established for each user a top five favorites list we can see this list in Figure 3.6. The user can click directly on one of these songs to play it on the speakers. On the right side we also provide also a button to distribute the favourite list with other users. So, when one clicks on the button their favourites come to the search list view of other users.

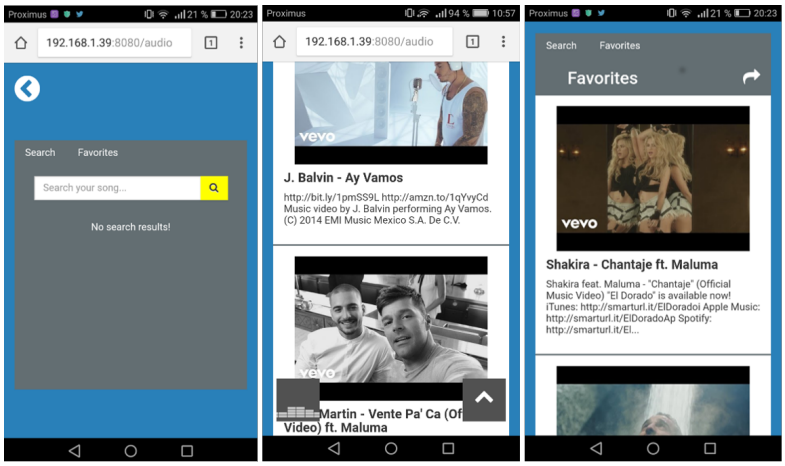

Figure 3.6: Audio section

3.6 Video

This section works a little bit like the audio section. In this section the user is searching for a video, the YouTube API is then queried and the result is shown in a list view like in Figure 3.7. When the user clicks on one of the videos, the YouTube video is automatically distributed to the smart TV interface. Of course, the user should be connected to the application on their smart TV. The system detects automatically that the device is a smart TV. Therefore, no account is required, it displays the correct interface automatically. The smart TV interface can be seen in Figure 3.8. Each user can then change the instance of the video by searching for a new one and then launching the desired video.

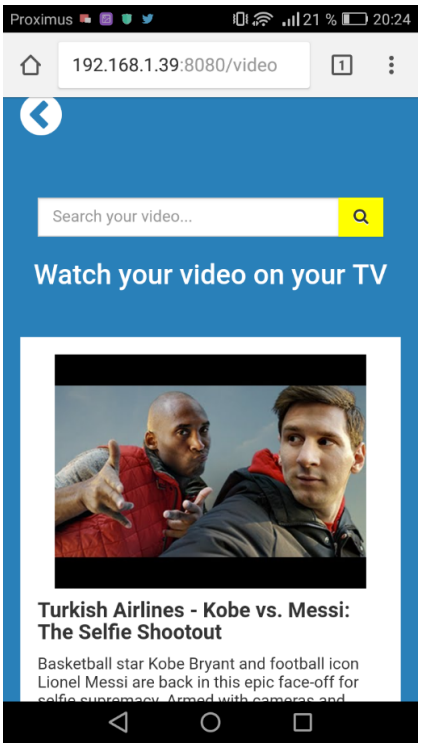

Figure 3.7: Video section with one result in the list view

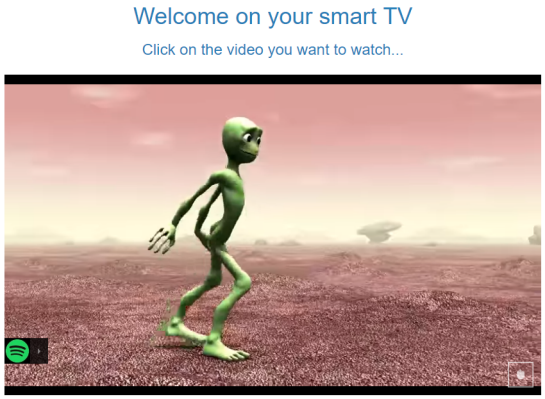

Figure 3.8: Smart TV interface with a video distributed by a user

4
Scenario

The scenario is a strong tool making the complexity of a system more understandable. Telling stories about the system ensure that people have a global perception of the system requirements and functionalities. In order to write coherent scenarios, we needed a good understanding of the system requirements. Another aspect that was important when writing scenarios is to know on which kind of users we want to focus. That is why we used fictive users that are in accordance with our focus users. In these scenarios we made a story about a family using our system. Marc is the father of the family and the administrator of the system. With his wife Clelia, they have two children Arnaud and Sophie. Arnaud is 20 years old and Sophie is 19 years old. Finally, Marc also has a 20-year-old nephew named Bruno.

4.1 Security

Sophie, the daughter, wants to connect to *InteHome*. She gives her identifiers, and she goes to the security section of the system. In this section she can watch the live stream. She takes a picture because she sees a strange person in front of the home's door. By swiping on this picture, she distributes the picture to her brother, mother and father. While she is occupied with the camera, she can also check the notifications (pictures) that the system and her brother sent/are sending. She decides to activate the alarm system

because there is nobody at home and her father has forgotten to activate it. Yesterday, the siren turned on because the system had detected a thief forcing the doors lock. The thief was scared and so left the house.

4.2 Lighting

Marc works late every day. When he comes home he first wants to turn on the light of the living room and not the light of the hallway because his wife is sleeping. In order to do this, he goes to the platform on his smartphone and he can turn on the light. He wants to take a shower before sleeping but his phone battery is low, so he distributes the light interface to his smartwatch and he puts his smartphone to recharge. He can now easily go to the bathroom, turn on the light of the bathroom and turn off the light of the living room with his smartwatch.

4.3 Audio

Sophie and Clelia have the habit to listen to songs on Sunday mornings while Marc and Arnaud are not at home. With *InteHome* they can listen to the songs using the platform. Sophie goes to the platform and goes to the audio section. In this section she can see which music she has listened to previous Sunday but now she wants to listen to some new songs, she can search on the platform and select a new title. Clelia wants to listen to Sophie's favourite songs. For that, Clelia asks Sophie to distribute their favourite songs. Sophie clicks on the distribute button, and their favourite songs are displayed in Clelia's interface. Clelia can now simply click on the desired song and listen to it on the home speakers.

4.4 Video

Arnaud is a fan of the soccer player Cristiano Ronaldo and just like him, he plays soccer. Before every game, he likes to watch videos of their idol. With *InteHome* he can easily watch videos on their Smart TV and search for videos on the web application on their smartphone, when he found the desired video he clicks on it and the video is distributed on the TV screen.

4.5 Administrator Panel

Marc is the administrator of the system, only him has access to the admin panel. Every month he looks to the dashboard to see if all goes well. He looks to the number of lights' request that were done this month in order to estimate his electricity consumption. Today Marc decides to add Bruno, his nephew to the system, because he often comes during the week. He accesses the administrator panel with the right credentials, he goes to the user section and adds Bruno as an user. Marc allows Bruno to access the light and audio system section by giving him the right permission. He also set Bruno's email and password. Bruno is now able to manage the lights and audio system at home.

5 Methodology

Before beginning with the implementation of this project a lot of work has been done to brainstorm about the methodology to adopt to begin with the perfect mindset and with a clear method of work.

The research and analysis of other works in the same domain is necessary to understand what it is about. We did a lot of investigation in order to understand the principles of distributed user interfaces and get crucial information about the working of such technology. Other works explaining the Internet of Things were investigated and were very beneficial for the progress of the project.

For such project it is very important to know what the project is about and what the goals of the project are. To know perfectly what the project should do, it was important to define and to break apart the requirements of the project.

One of the methods used to know what the different functionalities of the application are, is to make mock-ups of the user interface. With these graphical illustrations of the user interface it is easy to see what the application will look like and what the different functionalities are that the application must provide. It also provides a guideline when building the application, we

know perfectly how to navigate through the application and how to organise them. Derived from the mock-ups we split the application in different sections. That offers a better global vision of the application. These sections are also the base in the making of use cases and scenarios. You can easily imagine what the user can do with the app and describing meaningful scenarios and use cases.

5.1 Goal

The goal of this thesis is to demonstrate the potential and the opportunities when we combine distributed user interfaces and the Internet of Things. By combining these two research fields, we wanted to create an innovative web application that can be used in a certain environment, in our case the IoT environment. The purpose of our system is to make our users' life easier. Partly this is what the Internet of Things stands for. Internet of Things appliances want to make their users' life easier and faster to manage. But we want to go a step further, by combining this research field with distributed user interfaces. We can create systems that change people's way of living.

InteHome is the name we give to our system. It is a home automation system which user's interface can be distributed. The management of home appliances can be distributed across different users and devices. We added innovative functions to the application, making user's life easier. A good example, which demonstrates the potential of this research, is when a person wants to watch a video on his smart TV. The way a person looks for a video nowadays is too long and boring. It is mainly due to the remote controller that takes a lot of time to write a search tag. With our system users can easily search on their smartphone or tablet and distribute the desired video on their smart TV. It is precisely this kind of feature we want to develop. But without the combination of distributed user interfaces and the Internet of Things it would be not possible.

5.2 Users

We developed a multi-user system, which means that the user is central. For this, it is very important we know precisely what kind of users we focus on. Concretely, we focus on users who have a house and who need a system making the management of home appliances easier. Our users must not have any disabilities such as blindness, deafness or significant mental disabilities.

However, our application can provide an advantage for users who have slight physical problems since they have to move less in the house to perform daily actions like turning on a light. In general, the application aims users that want to perform daily actions easier and faster. Furthermore, the application is beneficial for users who want more security at home. Since our application is a multi-user system it can be beneficial for a family. Children too are potential users since the application provides an alternative account for children or users who cannot access more serious functionalities like the home security management.

Finally, we made a persona representing a typical user of our application. Despite the fact that the application has a wide range of potential users. Our persona in Figure 5.1 is a typical user where we focus on. The persona helped us to be always concentrated on the end-user. Furthermore, personas help decision-making in the design of the system.

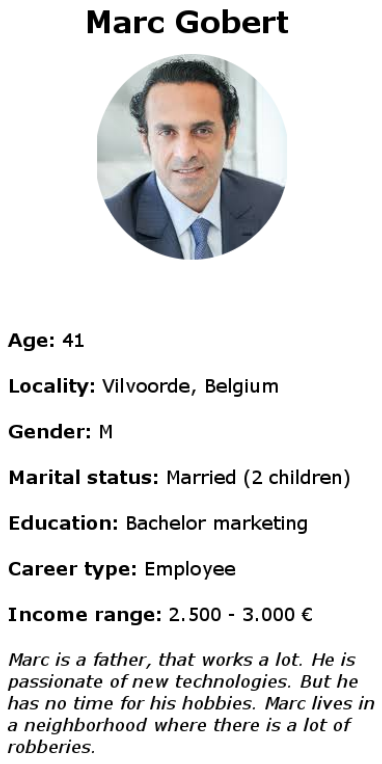

Figure 5.1: Persona

5.3 Requirements

Functional and non-functional requirements need to be carefully selected in order to ensure a good course of the development phase. Before we began with the implementation, we took long brainstorm sessions to define the requirements of the web application.

5.3.1 Functional Requirements

Functional requirements will describe different functionalities of the application. They describe what the system should do depending on the situation. The requirements of our web application are listed as below:

- A user must be connected into the right platform when logged in with the right email and password.
- The right interface must be displayed when the user clicks on a section.
- When the user clicks on the logout button, the user must be disconnected and the login page must be displayed.
- In the security section the camera stream must be displayed.
- When the user clicks on the 'distribution' button, the picture must be distributed to all users.
- The notifications in the security section must be deleted when the user clicks on the delete button.
- The alarm system must turn on or off when switching the alarm system state.
- The user should be able to change the light's power state.
- The power state must be automatically updated for each user.
- When a search tag is entered, results from the YouTube API must be shown.
- The home speakers must be playing the song, when the user clicks on this song.
- Each interface must be automatically updated with the state of the home speaker.
- A user should be able to watch their top 5 favourites.
- A user should be able to distribute their favourites with other users.
- A user should be able to change the current audio stream.
- A user should be able to stop the audio stream.
- A user should be able to search for a video.
- A user should be able to launch a video on the smart TV using their device.

- A user should be able to change a video on the smart TV using their device.
- The administrator should be able to add, modify or delete a user.
- The system must generate various information about the data in the database.

5.3.2 Non-Functional Requirements

Non-functional requirements describe how the system will behave when performing a functionality or what the limits of this functionality are. Non-functional requirements will also specify the quality or characteristics of the system. Our non-functional requirements are listed as below:

- When a user is logged in, the information is saved in sessions.
- When a user distributes a picture, other users should have the picture within 3 seconds.
- Each request from a user must be saved automatically in the right table of the MySQL database.
- When a user changes the light's power state, the light must react within the 5 seconds.
- The system should automatically detect which interface to show according to the device type and screen.
- When the user set the wrong credentials an error message must be shown.
- When a user browses the wrong page a 404-error page must be shown.
- Users with a *minimal account type* should not be able to access the security section.
- The system should manage up to 5 requests at the same time.
- The alarm system must detect an intrusion in less than 1 second.
- The system should be available 24/7, it means all and every day.
- The system should make a difference between user requests and system requests.

5.4 Style Guide

When implementing the user interface, we followed some style guidelines to ensure a user-friendly interface. For the user interface design, we principally

based on the Google Material Design[1]. However, our application must be designed for completely different platforms also such as IOS and Android, so some modifications have been done to let our user interface fit for different platforms and devices. Material Design uses movement of elements as feedback for the user. We also tried to implement this principle. For example, in the home page when a user goes hover a section with their mouse the caption of the picture animates until it takes 100% of the picture size.

More generally, Material Design adopts the 'card' motif as the main motif for user interfaces. This motif is also recurrent in our application. Nowadays with the expansion of different devices types there exists a wide range of screen sizes. So, it is very important that our application fits within all device screens. One of the most important principles from Google's Material design is the grid-based layout. This design principle aims to change the user interface depending on the screen size. To fulfil this principle, we used the twelve columns grid system of the well-known CSS framework Bootstrap[2]. This allow us to create a responsive design for small and large screens. All interface elements such buttons, containers, layout are made using Bootstrap components which are surely responsive.

The colours of the application are also a design principle that we not neglect. We used a consistent colour palette for all pages and components of the application. Our interface is principally consisted of the colour light blue. We also used a flashy yellow colour for search buttons, setting the focus on the search functionality. Google's Material Design devote importance to usability guides for persons who not perceive colours well. We also take care of the usability of the interface by creating a contrast between design elements or text with the background. It makes the application comfortable and pleasant to use. Icons are also important, we used Material icons from the iconic text and CSS toolkit Font Awesome[3]. Their icons are scalable and perfect for any screen types. Their icons are adapted to the Bootstrap framework and meet the guidelines of Google's icons. Our application respects also the layout of Google by providing a header at the top of the page. Left of the header we set the application title and right the logout icon.

Furthermore, our application is designed in accordance with various style guides for distributed user interfaces provided by Elmqvist et al. [4]. Elmqvist

[1] `https://material.io/`
[2] `https://getbootstrap.com/`
[3] `https://fontawesome.com/v4.7.0/icons/`

said that a distributed user interface should not require more affordance than a non-distributed user interface. Nevertheless, some design guidelines were respected when implementing the interface. Consistency is one of the challenges when making a distributed user interface. Our application also has a consistent interface, each instance has the same interface and share mainly the same states. Elmqvist presents the synchronisation between different interfaces as an aspect not to neglect. Each instance of our application is continually synchronised with the most up-to-date state. Melchior, Vanderdonckt and Van Roy did also some research about the design of distributed user interfaces [7]. We agreed most of their sayings. Each interface shares the same instances of the interface or objects. For example, when a song is playing on the speakers each user can stop the song or change the song since the instance of the song is distributed and shared in a consistent way in each user's interface.

5.5 Prototyping

Such application should be easy to understand. For this, it is important that the different views are consistent. The navigation through the different pages is very consistent since we provide only one way to navigate through the pages. The user can access each section from the home page. To access another section the user has to go back to the home page, this way of navigation is very intuitive for users. In Figure 5.2 one can notice that the navigation through the different pages is straightforward.

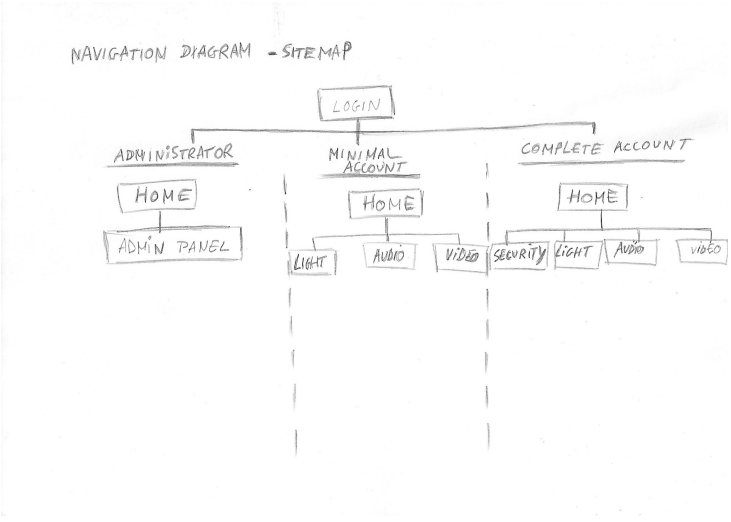

Figure 5.2: Sitemap of the application

Many other graphical elements were added to make the application intuitive and enjoyable to use. Firstly, we provide a lot of feedback to let the user know what should happen. When music is currently playing we add a floating animated icon to the bottom left of the page. Text elements are also added to critical places in the application. For example, in the smart TV interface we added the text 'click on a video' to let the user know that he must click on a video to display it on the TV screen. These kinds of little details make the application intuitive to use and therefore faster to understand. Secondly, we provide a sticky back-to-top button in the video and audio section when the user looks for a video or song. This feature is especially useful for long search lists in small screens. This feature is an example making the application attractive.

Since our application is running on different Internet of Things devices such as a smartwatch and smart TV, the interface automatically adapts to the device that request the application, for example the home page of a smart TV is different from the home page of a smartphone. Furthermore, the smart TV does not require any login.

Finally, with all knowledge and inspirations we acquired from the Google's Material Design guides and research, we made firstly on paper the sketches of how the user interface of our application will look like. In Figure 5.3 we present the mock-ups that we did before implementing the application.

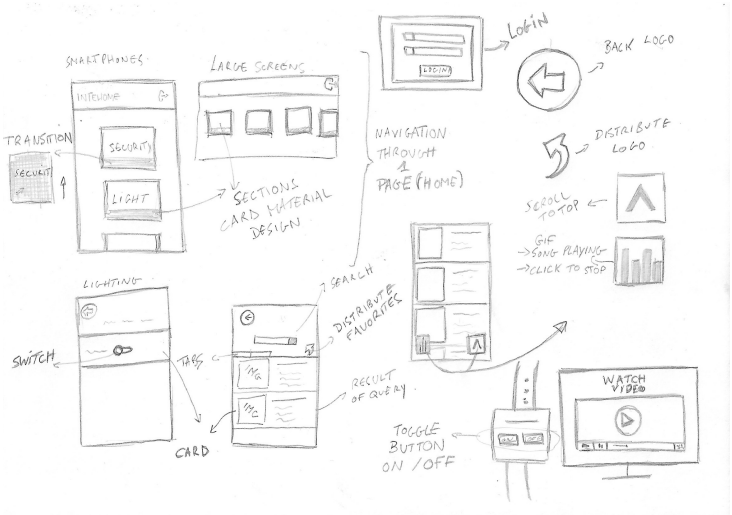

Figure 5.3: Drafts of the user interfaces and their graphical elements

One can notice that these mock-ups are very similar to the final version of our application. Most of design guides that we found interesting were implemented. We also set the focus on a responsive design making our application available on different devices. We also tried to be consistent in our choices for example always the same icon to go back. We also always used the same icon to distribute something with other users.

6 Implementation

In this chapter we present a crucial part of this thesis. We tackle the question "how to actually build the *InteHome* web application and global system?" Details, choices and issues about the development are explained in detail in this chapter. Important libraries and source code elements are highlighted in order to understand on what the project focus namely distributed user interfaces and the Internet of Things.

6.1 Important Hardware

The infrastructure of the system is composed of several devices and objects. The central unit of the system is a Raspberry Pi model 3 with Raspbian running on it. The Raspbian operating system is a lightweight distribution from Linux especially for the Raspberry Pi. We installed NodeJS for the web application and MySQL for the database. The Raspberry Pi also has two devices connected to it, the camera and the speakers.

For the lights we used a smart power plug from TP-Link. To let the smart power plug work properly we need to connect it to the local Wi-Fi connection. We made this connection using the *Kasa*[1] application from TP-Link.

[1] `https://www.tp-link.com/us/kasa-smart/kasa.html`

Once the connexion established, the smart plug receives an IP-address. Using `tplink-smarthome-api` package, we can send requests to the power plug to perform some actions such as turning the power plug on or off and to get information such as the current power state.

Another important device in our application is the passive infrared (PIR) motion sensor. This sensor detects movement in an angle of 120 degrees. This sensor detects motion by measuring the infrared light radiating from objects, in other words, it detects the heat energy that objects produce. The motion sensor is a module that we connect to the Raspberry Pi GPIO pins. When the sensor detects heat radiation it sends a 5V to the Raspberry Pi. The output sent by the sensor can be 1 or 0. Furthermore, the sensitivity and the detection delay is modifiable on the sensor itself.

6.2 Technologies

In this section we present the technologies we used for the implementation of this project.
Initially, we hesitated between two completely different programming languages namely PHP and NodeJS. PHP is a very popular server-side scripting language that can be used for general purposes but more particularly for web development since HTML content can be written in PHP files. NodeJS is also a server-side runtime environment which is actually JavaScript for servers. Initially JavaScript was a programming language for the Front-End (client-side) only. But this event-driven programming language is now also available for the Back-End. Some important differences between these two server-side languages are listed in Table 6.1. The origin of these information is from the official website of PHP[2] and NodeJS[3].

PHP 1	NodeJS
Single thread	Multithread
Packages difficult to find through the web	NPM packages directly accessible
Synchronous (blocking I/O)	Asynchronous (non-blocking I/O)

Table 6.1: Differences between PHP and Nodejs

[2] `http://www.php.net/`
[3] `https://nodejs.org/en/`

The technology choice depends above all on what we need in the solution. In our case NodeJS is more suitable since it provides a lot of packages which gives a lot of innovative solutions. One of these packages written in JavaScript, which is very important for us, is Socket.IO[4].

It gives us the possibility to send messages from the server to the client but also from the client to the server. This technology is very useful for sending real-time information between clients and thus a great solution for distributing the user interface.

Based on the requirements we previously defined, NodeJS has been chosen to implement our system. NodeJS is an event-driven non-blocking environment, which means that a user could perform different tasks at the same time. The callback function is triggered when the task is completed with the status successful or unsuccessful. Once the callback is called, the returned data can be processed in this function and return a result or call other tasks. This concept is interesting for the web application that we developed since it gives us the possibility to manage various objects at the same time. Finally, NodeJS can manage dynamic web pages efficiently.

6.3 Application Infrastructure

The web application has been made with the open-source framework Express.js[5]. Express is a lightweight web application framework for NodeJS. This framework generates automatically a folder structure and an application architecture. The folder structure can be seen in Figure 6.1, notice that we add two additional folders namely *background* and *models*.

[4] `https://socket.io/`
[5] `https://expressjs.com/`

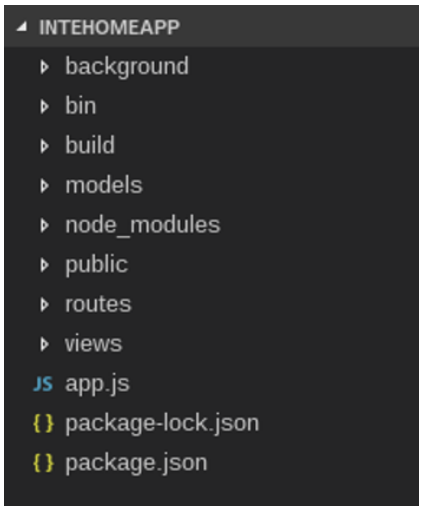

Figure 6.1: Folder structure of the application

The root of the application is the `app.js` file. In this file we initialise the server on port 8080 and we initialise the routing system. This file is actually the starting file of the whole application. In The `package.json` file we can retrieve all NPM packages we installed with their version. We can also add some application configurations.

When the user requests a URL, the request is redirected to the appropriate file in the routes folder. We have five possible routes, namely index, camera (for the security section), audio, light and video. In these files the request will be processed and then the views will be rendered according to the request.

In Figure 6.2 we can see the different EJS files which are the views (pages) of our application. Some pages like the index page are split up in different files making the code easier to understand and to maintain. In the *components* folder we can retrieve different parts of a view like the header and the footer, these parts are included in all pages of the application. Additionally, note that the `error.ejs` file is the view that will be rendered when a user makes a mistake such as requesting an inexistent URL or when a fatal error occurs. These pages also include JavaScript and CSS files from the *public* folder.

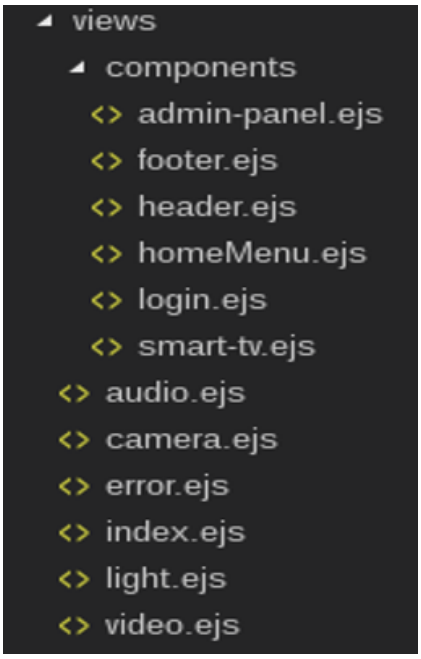

Figure 6.2: View files, split up into different EJS files

Finally, the *background* folder contains the files `motion-detection.js` and `tplink-smart-plug.js`. These two files are separated from the core of the application because they run separately from the application. The *models* folder contains two files namely `Database.js` and `User.js`. These files contain the schema for the database and for a user object. They will determine how the data will be saved in the application. The database class is also used to initialise the connection to the MySQL database and perform SQL queries.

6.4 Database

The database design for this kind of application is very important. Our database is a MySQL relational database. Before designing the database, we must know which kind of information we want to save. Our database model is entirely designed to make the multi-user system easy to manage. A long time has been spent thinking about the different tables and their columns. The database allows us to access, update and delete data within its tables using a structured query language named SQL. The relations between the tables are designed so that important queries for our system are easy to build and to maintain.

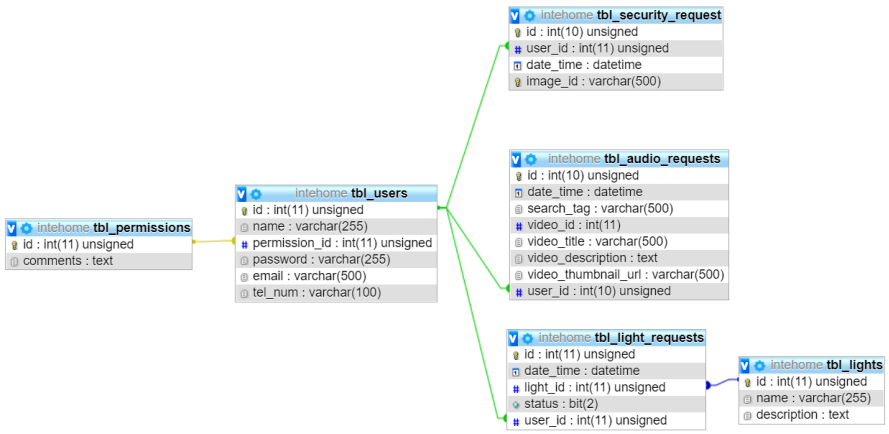

Figure 6.3: Entity Relationship Diagram

From Figure 6.3 we can deduce that the most important table is *tbl_users*. This table is the cores of our database and of our application. Since there are three different account types, each user need to have permission. In *tbl_permissions* we saved these three account types with a unique identifier as primary key and extra comments to explain what this account type is about. From the user we need a unique primary key identifier, the name of the user, their password, their email address and finally their phone number.

Notice that we only save information that is essential for the functionalities of the system. Even better, the database was modelled according to the functionalities of the web application. This is also why there are relationships between the user which is central and the various requests we can do in our application. The only one who is not available in our database is video requests since saving video requests in the database is not relevant for our requirements. Of course, this database model can be expanded, saving and making more complex data available. However, we only save the data that we really need for the web application.

When a user switches the light's power state, we create a new light request. This request is automatically saved in *tbl_light_requests*. We create a new unique identifier and we save the date and time of the request together with the status which is actually the power state (0 for off and 1 for on) and the user identifier (*user_id*) to know which user made the request. The attribute *light_id* is a foreign key associated with the table *tbl_lights* which is actually a table where all available lights are listed. At this moment we have only

one light available in our system, but imagine we could add another one. It is easier to manage the system this way.

Since using the smartwatch do not require any login, we cannot know which user made the request. To handle this problem, in *tbl_users* we set the *user_id* on 0 which is the *user_id* of the system administrator. So, when we see that a light request was made with *user_id* 0, it means that the request was made using the smartwatch.

Just like light requests, users can make an audio request when listening to songs using our system. When the user clicks on a song, an audio request is created in *tbl_audio_requests*. Here, we also save some information about the request. Firstly, general information such as ID, user identifier to know who made the request, the date and time, the search tag the user used to obtain the link to the song and the most important is the *video_id*. Which is the unique ID from YouTube for a video. Why *video_id* and not *audio_id* will be clear in the next Section 6.5. Finally, we save other information about the video that is provided by YouTube such as title, thumbnail link and description. We save this information in order to make the creation of a top 5 favourite songs possible without requesting again the YouTube API.

The last request a user or the system can do is a security request. This table saves, like every table, the ID, the date and the time. Even if we have the same table two different requests can be saved in this table. The request can be made by a user, this means the user took a picture from the camera and distributed it to other users, or the request was made by the system itself, when the alarm system was on and the motion sensor detected somebody. We used the column *user_id* to make the difference between both requests. When the automatic alarm system makes a request, the *user_id* will be set on 0. Otherwise the *user_id* will be the user's identifier which means that a user made a request. In these two situations a picture will be taken, so with the *image_id* we can retrieve the corresponding picture in our image folder.

6.5 InteHome

For a good implementation structure, it is important to split the list of functionalities in phases. Each phase corresponds with a section of the application and regroups functionalities and requirements that belong together. During the implementation we adopted an agile development process. Each functionality and requirements were developed independently from each other. After

the implementation, the functionality/requirement was tested and changed if necessary. This work cycle was done until the functionality was complete and running correctly. After, the same work cycle is applied on the next functionality. After the implementation of the phase each functionality is again tested, reviewed and corrected if necessary. The testing phase was mainly done using exploration testing, trying to cover all different possibilities and issues of the feature. When an issue was encountered it was fundamental to find an alternative. In this kind of project, it is very important to always find a solution to any potential problems. Most of the time further investigation is required to find the right solution.

6.5.1 Login System

When the user wants to connect, he goes to the index page. The router goes to the index routes and makes a GET request asking for the index page. This request method can be seen in Listing 6.1. One important thing occurs here, we firstly check if someone is already connected. For this we check the `session.user`, if this session is empty it means that nobody is logged in. In this case we give the variable `connected` the Boolean value: false. Otherwise it means that the user is already connected, the `connected` variable will have the value: true. After checking if the user is already connected or not we render the index page. When rendering this page some data are transferred to the EJS view, namely the title of the page, the connected variable, the device type, and the user object in the session. Notice that we used external packages for using `sessions` and gather information about the requesting device. These packages are `client-sessions` and `express-device`.

```
router.get('/', (req, res, next) => {
  var connected =  (req.session && req.session.user) ? true : false;

  var ag = agent.parse(req.headers['user-agent']);
  console.log(ag);
  console.log(ag.device.toString());

  res.render('index',{title: "Home",connect: connected,device: req.device.
      type,user: req.session.user});

});
```

Listing 6.1: GET method for the index page

In the index page we check various parameters. Firstly, we check the `connected` parameter. If the value is true it means that the user is connected to the application. So, in this case we check the user object's `permission_id`. Depending on the `permission_id` we know which interface we have to render

either the administrator panel if the `permission_id` is 1 or the home page with its menu. Otherwise, if the `connected` variable is false it means that nobody is connected. So, then we have to check to the device type. If the device type is a 'TV' then we have to display the interface for the smart TV otherwise we display the login page.

When the user clicks on the 'login' button, a POST request is sent to the `/index/login` URL that can be seen in Listing 6.2. Here we will check if the user who wants to log in sets the right email and password. We make firstly a database instance, then we query the database asking if a user with this email and password combination exists. If it returns a user object, we save it in the session and we redirect the user to the index page which will again perform the actions that we described previously. If the user does not exist, we render again the login page with an appropriate error message. This workflow can be seen in Figure 6.4.

```
router.post('/login',(req,res)=>{
  var db = new Database();
  var user = db.getUser(req.body.email,req.body.password).then((user)=>{
    db.closeConnection();
    if(user){
      req.session.user = user;
      res.redirect('/');
    }else{
      res.render('index',{title: "Home",connect: false,device: req.device.
          type,user: req.session.user,login_error: "Email or password is
          incorrect!"});
    }
  }).catch((err)=> setImmediate(()=>{throw err;}));
});
```

Listing 6.2: POST method when the user wants to log in

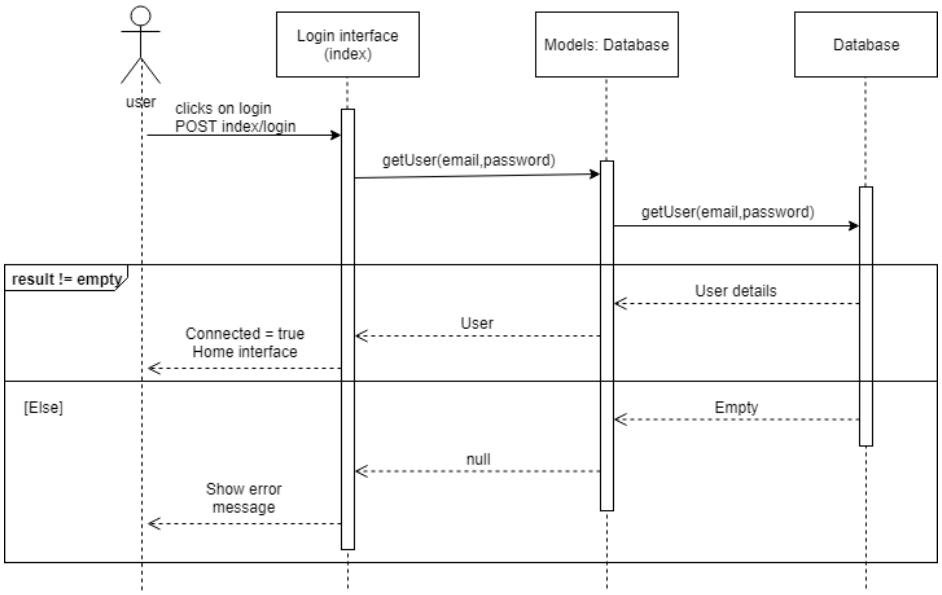

Figure 6.4: Sequence Diagram for login POST method

When the users wants to log out, he has they only have to click to the 'logout' button. It will make a POST request to `index/logout`. The session is then reset and the user is redirected to the index which will then render the login view.

6.5.2 Audio Section

When the user clicks on the audio button of the home page's menu. The user makes a GET request to `/audio`. Just like other requests we check if the user is connected, if not he is redirected to the index page. If the user is connected, the audio page is rendered. Here the `isPlaying` is an important parameter that is passed to the audio view. This parameter always has the state of the home speakers. Passing this parameter to the view permits to the interface to know if a song is currently playing or not.

When a POST request is submitted to `/audio`, it can have different meanings. Firstly, we look to the request body's `id` parameter. If it is `undefined`, it means we are searching for a song. Otherwise, if the `id` is set it means we want to play a song. This can be seen in Figure 6.5 In the case that the user is looking for a video, he posted the search tag. With this search tag we browse the YouTube API. We use the package `youtube-search`, after passing the options like personal key, type of data and the number of results. We can pass our search tag and it returns a JSON object with a list

of video information. This JSON object is then passed to the view so that he can organise and display the results. If the request body's `id` parameter is not `undefined`,hen it means that the user wants to launch a song. In this case we have to first check if a song is currently playing, if it is the case we must reset the audio stream by removing all event listeners, resetting the `decoder` and set the `Speaker` and `isPlaying` variables to false. After this we can play a new YouTube stream from the `requestURL` which is composed of the concatenation of the YouTube URL "*https://youtube.com/watch?v=*" and the request body's `id` variable. A YouTube audio stream is created by using the NPM package `youtube-audio-stream`. The audio stream is then piped to a new speaker object and then we finally save the request in the database and we emit to all sockets the message `startPlaying`. In the client side, the interface knows that a new song instance is created. This process can be seen in Listing 6.3.

```
if (Speaker && audioStream && isPlaying) {

    audioStream.removeAllListeners();

    audioStream.emit('end',{error: 'ending music'});
    decoder = new lame.Decoder();
    audioStream = null;
    Speaker = false;
    isPlaying = false;
    }

    if(!isPlaying){
    audioStream = youtubeStream(requestUrl).pipe(decoder);
    audioStream.on('format', function (format) {
      Speaker = new speaker(format);
      isPlaying = true;
      this.pipe(Speaker);
      res.json({isPlaying: isPlaying});

      saveRequest(searchTag,req.body.id,req.body.title,req.body.description,
          req.body.thumbnail,req.session.user.id);
      req.app.io.sockets.emit('startPlaying');
    });
```

Listing 6.3: Process occuring when the user make a request for a new song

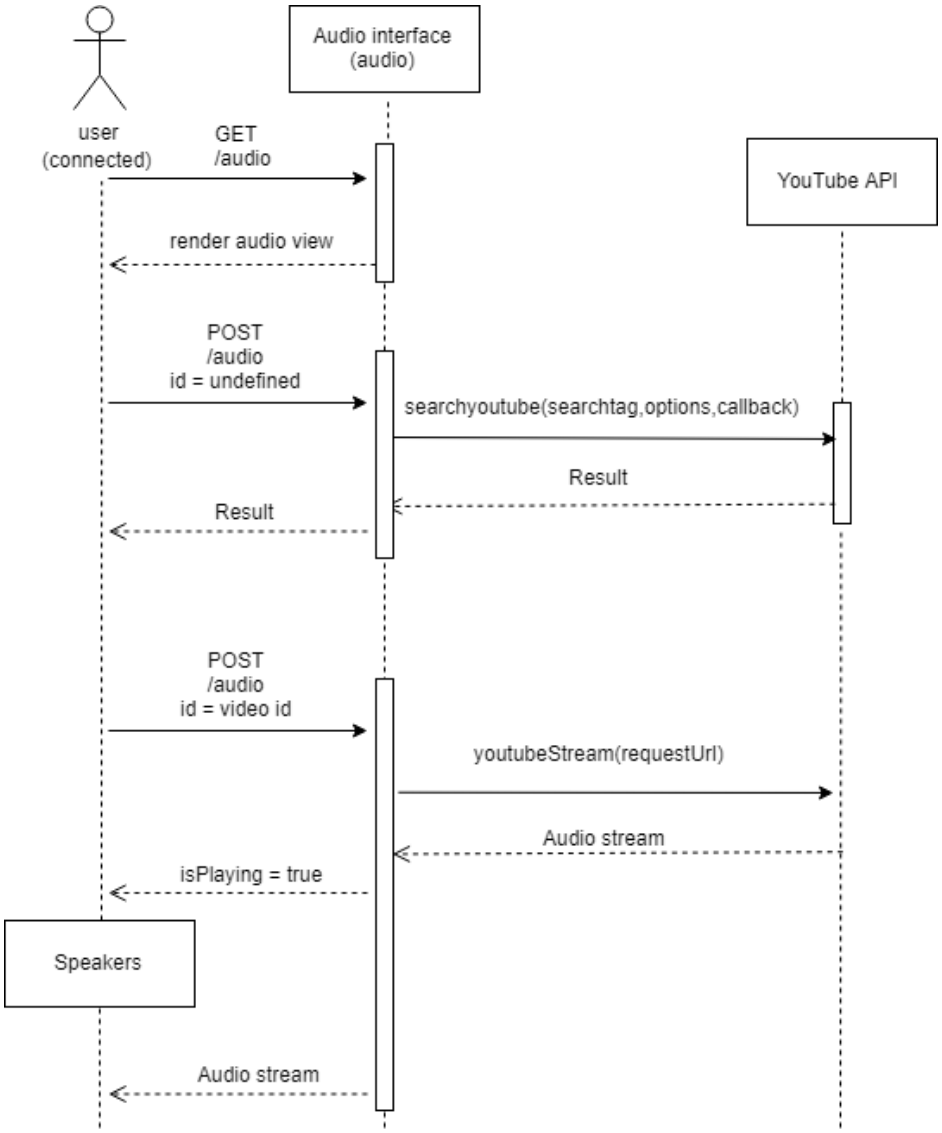

Figure 6.5: Sequence Diagram for audio requests

In the audio section we also retrieve the favourite songs. When the user clicks on the 'favourites' tab, a POST request to the server is made. Then the user can query the database to get a list of audio requests that the user made in the past and where the `user_id` is equal to the user who made this request. To distribute the favourites to other users we use web sockets. In the client side, we send the data to the server and the server makes a broadcast to all sockets which will then receive the data containing a list of songs.

6.5.3 Lighting Section

In the lighting section we demonstrate that Express is very efficient for creating a little API. This section is composed of two GET methods and two POST methods. The first one is the `/light` this will render the view and it will transfer the light power state to the view in order that the view set the right state to the switch button.

In order to get the power state, we created an object named `tplink`. This object uses the NPM package `tplink-smarthome-api`, the goal of this object is to provide some functionalities that we can call from everywhere in our code such as `on`, `off` and `isDeviceOn`. The package is mainly used to communicate with our smart power plug, firstly to discover the smart plug in the network and secondly to send or get information.

The second GET method is `/light/API/bedroom/getPowerState`. This request will send a JSON object, the current state of the smart power plug by using the `isDeviceOn` function of the `tplink` object. This request is mostly used by the smartwatch in order to get the current power state when the smartwatch application is launching.

The two POST requests are `/light/bedroom/on`, this method can be seen in Listing 6.4, and `/light/bedroom/off`. The goal of these two requests is very similar. One is used to turn the light on and the other to turn the light off. The request is first saved in the database. After that, the `tplink` function `on` or `off` is used accordingly. Finally, a message is emitted to all sockets in order to update each interface with the new state.

```
router.post('/bedroom/on',(req,res,next) => {
  console.log(req.device.type +"------>request␣for␣light␣api␣on");
    var uid = (req.session.user) ? req.session.user.id : 1;
      var db = new Database();
      db.insertLightRequest(1,1,uid).then((res) =>{ // 1 = lamp id , 1 =
          state of the lamp (1 = true = on)
        tplink.on();
        req.app.io.sockets.emit('br-light',{powerState: true});
      });
      res.json({status: 'ok'});
});
```

Listing 6.4: API URL for turning the light on

6.5.4 Video Section

This section is very similar to the audio section. The `/video` GET method is only used to render the video view. When the user enters a search tag, we use, just like for the audio part, the `youtube-search` NPM package to make a query to the YouTube API. Next, we will receive a JSON object from YouTube containing information about the video. We then render this data object to the view which will organise them using jQuery on the client side.

When the user clicks on the desired video, we emit from the client side a message using sockets to the server, the server will then redirect this message containing the video ID to the smart TV socket. In Listing 6.5 we can see what occurs when the socket receives the message, it will namely extract the video ID and create the HTML `iframe` object or replace the current source with the new YouTube URL.

```
$(document).ready(function(){
    var socket = io.connect('http://10.42.0.160:8080');
    socket.on('launchVideo',function(resp){
        $("#videoCadre").attr("src","https://www.youtube.com/embed/" + resp.
            video_id);
    });
});
```

Listing 6.5: Smart TV client receives a message from the server containing the video ID

6.5.5 Camera Section

The camera section is a little bit more complex due to their the different functionalities. The Back-End is mostly done but we did not finish the Front-End yet. The alarm system is mainly dependent of on the motion sensor. For the communication with the GPIO pins of the Raspberry Pi we used the NPM package `rpi-gpio`. This package provides functions that we can use to work with the sensor such as the `setup` and `destroy` function. Using the `setup` function, we can initialise the pin we want to listen to. Once the `setup` function is called, it means that the sensor is on. When the input pin detects an electric charge, it means that the sensor has detected someone, and the event can be called. In the callback function we firstly save this request into the database, we take a picture from the camera, we emit a message to all sockets and finally we launch the siren audio file into the home speakers.

When we call the `destroy` function, it means that we turn the sensor off. We made an object `motion`, who actually uses these functions to make an abstraction and use it everywhere in our application. We created then the function `isActivated` which returns `true` or `false` and this value will determine the state of the sensor. When a change is done, we use web sockets to broadcast the new state of the sensor to all users, just like in the light section.

Finally, pictures taken either automatically from the system or from a user, are saved into the *image* storage. Each week we call the `deleteImage` function who will delete all images from the storage that are out dated.

6.5.6 Administrator Panel

The administrator panel is the less complex section of this application. When the page is requested, we query the database to get all relevant information such as requests amounts, users and alarm system information. When he users wants to create, delete or update, the user does a POST request which will then generate the appropriate query, with the data obtained from the request, for the MySQL database.

6.6 Fitbit Ionic Application

The Fitbit Ionic smartwatch is used to manage the lighting system of the home. Since the Fitbit do not has a browser to launch the web application, we had to built an application in JavaScript. Before explaining the development of the application, it is necessary to understand how this smartwatch works.

The smartwatch needs to be coupled with the smartphone via Bluetooth. The Android or IOS Fitbit[6] application needs to be installed on the user's smartphone. Once the security code is introduced into the Fitbit application, the smartwatch is recognized and the first synchronisation occurs.

Actually, the smartwatch does not have access to the internet. It is the companion that uses their Internet connexion for browsing some resources from the internet and send it to the smartwatch via the Bluetooth connexion.

[6] `https://www.fitbit.com/be/setup`

To develop an application for the Fitbit we did a Fitbit account for Fitbit Studio[7]. Fitbit Studio is an environment where we can create applications for Fitbit's smartwatches. Once we are logged in we can create a new application. The application that we did for the smartwatch is called `FitbitThesis`. The application's folder architecture is important when building a Fitbit application. First, the *app* folder, it contains the `index.js` file who will run into the smartwatch. The *companion* folder should also contain a `index.js` file who will run into the smartphone. Next, we have the *common* folder, this folder is used for some code that can be shared between the application code and the companion code. It prevents to write duplicate code. The last folder is the *resources*, it contains the `index.gui`, `widgets.gui`, `styles.css` and `icon.png` files. This folder is used for the graphical content of the application such as text, buttons, images etc. Finally, we have also a `package.json` file, it is generated automatically when you create an application and contains various application settings.

Fitbit provides also to the developers a large list of APIs for the smartwatch and smartphone. When we are coding, we just need to import the right package and you can use the API. In the `index.js` that runs on the smartwatch we import two packages namely `document` and `messaging`. `Document` is necessary to get elements from the interface. `Messaging` package is used to send data from the smartwatch to the smartphone. In our application the `document.getElementById` function is used to get the buttons from the interface and save it into a variable. With this variable we can listen to the click event, and send data from this socket to the companion socket. In the index file for the companion, we listen to message entries from other sockets. When a message is caught, we look to the JSON data, the data provides the status that we wanted, depending on the button that was clicked.

Afterwards, depending on the status, the functions `fetchOn` or `fetchOff` will be called. These functions make a simple web request to the URLs of our web application which will then turn the light on or off. The `fetchOn` and `fetchOff` functions can be seen in Listing 6.6. Notice also that when the application launches the function `fetchGetPowerState` is requested which returns the current power state of the light.

[7] `https://studio.fitbit.com/`

```
function fetchOn(){
  console.log("in fetch function on");
  fetch(`http://192.168.1.39:8080/light/bedroom/on`, {
    method: "POST"
  })
  .then(function(res) {
    return res.json();
  }).catch(err => console.log('[FETCH]: ' + err));
}

function fetchOff(){
  console.log("in fetch function off");
   fetch(`http://192.168.1.39:8080/light/bedroom/off`, {
    method: "POST"
  }).then(function(res) {
    console.log(res.json());
  }).catch(err => console.log('[FETCH]: ' + err));
}

function fetchGetPowerState(){
  console.log('in fetch function get powerstate');
  fetch(`http://192.168.1.39:8080/light/bedroom/getPowerState`).then(
      function(res){
    console.log(res.json());
    let data = res;
    if (messaging.peerSocket.readyState === messaging.peerSocket.OPEN) {
          messaging.peerSocket.send(JSON.stringify(data));
    }else{
      console.log('send powerstate -> error socket not open!');
    }
  }).catch(err => console.log('error: ' + err));
}
```

Listing 6.6: Fitbit Ionic web requests

The application's interface is composed of a text element and two square toggle buttons. The interface is a SVG file where elements we want to use are defined in the widgets.gui file.

6.7 Issues

This section presents the issues we encountered during the development process of the system. The first issue we encountered is in the *InteHome*'s audio part. Initially, the goal was to pick up audio streams and information from the SoundCloud API[8]. SoundCloud is a website specialized in music tracks. But during the implementation we were unlucky because the API was not more available anymore for clients. Thus, we had to find an alternative to this problem. The solution was to use, just like the video section, the YouTube API. Users of our application should search for a title

[8] `https://developers.soundcloud.com/`

or artist and then the YouTube API is queried. With the NPM package `youtube-audio-stream`. We can extract the audio stream from the YouTube video and distribute it with the home speakers.

With the smartwatch Fitbit Ionic we also encountered two different issues. The application in the smartwatch works only if it is coupled to a smartphone. Furthermore, the smartphone must be synchronised with the smartwatch continually to let the application work properly. The problem is that the synchronisation does not work every time.

So, it is pretty hard to launch the application without problems. Another issue related to the smartwatch is that we wanted to distribute the user interface from the smartphone to the smartwatch by clicking on a button or swiping on the screen. However, with the Fitbit Ionic it is not possible to include this feature, since it does not provide a service to get requests from the outside. Request can only be sent from the smartwatch, via the companion (smartphone), to the server (unidirectional).

7 Guidelines

Before and during the development phase we followed a lot of guidelines for such projects. For both research fields, whether it was for distributed user interfaces or the Internet of Things, most of the research provides guidelines for other developers. We also give some specific guidelines for persons who want to develop these kinds of systems.

7.1 System Infrastructure

The first thing to know is what the system must do. Based on the system requirements, one can think about different hardware that must be present in order to accomplish these requirements. A lot of smart objects are available nowadays, but one must always think about how this object works in our own system. Some objects do not provide an API which makes the object difficult to access from an application, or they use other communication protocols such as Bluetooth, radio frequency, RFID and NFC. Sometimes one can see that the smart object desired is not existing already. In this case, one can create it with all available micro-sensors and micro-computers such as the Raspberry Pi or Arduino.

So, it is important to know how smart objects work and to analyse the communication protocol and the accessibility from outside. However, always

try to find an alternative solution. Nowadays each problem can have one or more solutions, always try to find the best way to solve a problem. Communication between a web application and a smart object is better with Wi-Fi. The application is easily accessible on the local network using the IP-address or using a local domain name. Furthermore, when one is working on a web application we recommend connecting it through the Wi-Fi because Wi-Fi is used by all devices.

7.2 Technologies

We highly recommend making good choices about technologies to use, such as the right programming languages. Generally, we recommend using NodeJS for the back-end technology in addition with Socket.IO. For such system the combination of NodeJS and Socket.IO is very powerful, resolving real-time problems in an efficient way. So, we used a server-client architectural pattern but one can even use a peer-to-peer architecture, which is as well suitable for distributed user interfaces. NodeJS is suitable for this kind of project, because it has a lot of NPM packages who are well documented and can be easily used directly in the application.

7.3 Style Guidelines

When designing an application, one must hold in mind that the design must be intuitive and easy to understand. We suggest following the Google Material Design guides, most of their design guidelines are based on usability. Following their guidelines ensure an effective, intuitive and enjoyable user interface. Furthermore, one needs also to reflect about how to distribute elements of the application across different devices. One needs to think about what will be distributed and how. Create always an area in the user interface where distributed elements can end up. The best way to handle distributed user interface is to create transitions and feedback so that the user knows what happens. In general, one does not invest more effort to a distributed user interface than a non-distributed user interface, because principles and style guides are the same for both kind of interfaces.

7.4 Implementation Recommendations

For the implementation of a distributed user interface in an Internet of Things environment, we firstly suggest thinking about how to communicate from the client to the server in an efficient way. Most of our actions are triggered from a HTTP post request from the user, for example, switching the light's power state triggered from a post request. In general, this way of working is very similar to APIs. We provide some requests that the different users can do to trigger an action in the server side. Since we are speaking about distributed user interfaces it is important that the instances of the physical objects are in the server side and not in the client side. In this way it is easier to access the instances of the object, and also more efficient.

Concerning the user interface, when one wants to synchronise their interface with someone else's, sockets are necessary to make the communication faster. Furthermore, sockets allow to send data to the server and the server can then propagate it to other devices creating the illusion that devices are communicating directly together. In this way, one can change the instance of an object and other interfaces are automatically up-to-date. Another opportunity is to send data from one device to another so that it can change its interface depending on the data that it receives.

In these kind of applications, it is important to generate data. These data can then be used for different purposes such as tracking the number of requests. Even more, data can eventually be shared with other users or systems. Without a well-designed database it is not possible. So, it is important to reflect a lot of time when designing the database. Different devices and objects, configurations and user's information can be saved in the database. The database is a paramount part of the application. Also for redundancy purposes the database can be useful. When the server is interrupted, we can save current states of the objects and of the interface making the application more reliable.

Finally, the best suggestion we can give is to make a lot of research about these technologies and about the devices and objects one wants to use.

8 Future Work & Conclusion

In this chapter, we discuss some new ideas that can be added in the future. Finally, some conclusion about the work performed in this thesis is given.

8.1 Future Work

While this project has demonstrated the potential of distributed user interfaces in the context of the Internet of Things, many other opportunities are left principally due to a lack of time. In this section we present other features that could be adopted in this project, which will certainly have a positive impact on this project.

8.1.1 Framework

Currently all different sections are static, which means that you cannot add or delete some sections. In the future we can imagine that the administrator can add or modify some sections, for example, by adding a new section for the thermostat directly from the administrator panel without modifying any source code. Another example using the administrator panel, is adding new lights and smart power plugs into the system.

8.1.2 Notifications

A nice gain could be notifications for security purposes. In the current prototype the user must be logged in and in the right section to see some notifications such as camera pictures from other users or the alarm system that switched on. But we can imagine in the future that users can have real notifications from the system which alert them that something occurs. Moreover, by clicking on the notification we could distribute the message and see more information on the smartwatch or open directly the web application without login needed.

8.1.3 Speech Commands

Speech commands are nowadays common in home automation systems. So, in our system we can also imagine adding speech commands to make the users' life easier. We could add speech commands to switch lights on and off. Another example using the voice is to stop the audio stream that is currently playing.

8.1.4 Go Beyond the Local Network

The prototype works currently only in the local network. We could later make the application available on the internet. So, we do not have to be connected to the local network to manage the different parts of the system. With this new requirement we could see the camera stream from everywhere, for example.

8.2 Conclusion

We presented an innovative system for home automation integrating the two research domains of Internet of Things and Distributed User Interfaces. Our system, *InteHome* demonstrates how user interfaces can be distributed across different devices using different applications introduced in the Solution Chapter 3. Making physical object instances such as speakers available in the interface of any user.It demonstrates the potential of distributed user interfaces since this only instance is distributed across different interfaces. IoT devices, such as a smartwatch, Smart TV and a smart power plug, have been integrated to the system as well. However, the most challenging part of this thesis has been to distribute the web application with these smart objects. Very few research were done on the combination of these two fields presented in this thesis. Therefore, based on the experience acquired when building the

InteHome system,in Chapter 7 we introduce some guidelines for developers of such hybrid systems combining IoT and DUIs. With this, we hope to create new horizons in the world of technology.

To make this project possible, we first adopted a clear methodology by listing the different requirements and functionalities that must be implemented in the web application. It helped to make a planning of work and a clear vision of the system to implement. Secondly, a lot of research was done to gain insight about the technologies in the scope of this thesis. The research was divided into two parts, one for distributed user interfaces and another for the Internet of Things. This was necessary to understand some aspects of distributed user interfaces and the working of the Internet of Things. The way the project looks like is defined in the solution Chapter 3 of this thesis. We described the different sections of the system. With this description other persons should understand the purposes of the system and its benefits. The major part of this thesis is the implementation of the system. A lot of new technologies was used to make the system work properly. The choice of the programming languages, libraries, user interfaces and hardware are fully presented in this chapter just like the details of the source code and the issues encountered during the implementation phase. During the long brainstorm sessions and implementation, we found new interesting ideas and functionalities that could be developed in the future.

Finally, the results of this thesis and project demonstrate the power of both research fields, Distributed user interfaces and the Internet of Things. Thanks to the work delivered before the implementation phase, we had enough knowledge about the subject since we had implement most of the functionalities and requirements defined previously in the methodology chapter 5. We demonstrate that NodeJS and Socket.IO are powerful tools for developing distributed user interfaces. And since NodeJS provides a lot of packages to manage IoT objects, it is even more effective for an Internet of Things environment.

Bibliography

[1] Demeure Alexandre, Calvary Gaëlle, Balme Lionel, and Coutaz Joëlle. CamNote: A Plastic Slides Viewer, January 2005. supported by the European CAMELEON R&D Project IST-2000-30104.

[2] Tsung-Hsiang Chang and Yang Li. Deep Shot: A Framework for Migrating Tasks Across Devices Using Mobile Phone Cameras. In *Proceedings of CHI 2011, Conference on Human Factors in Computing Systems*, pages 2163–2172, Vancouver, Canada, May 2011.

[3] Guía E., Gallud J., Tesoriero R., Lozano M., and Penichet V. Co-Interactive Table: a New Facility to Improve Collaborative Meetings. In *Proceedings of the 12th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI 2010)*, pages 3–6, 2010.

[4] Fisher Eli Raymond, Karthik Badam Sriram, and Elmqvist Niklas. Designing the Distributed User Interface: Case Studies on Building Distributed Applications. *International Journal of Human-Computer Studies*, 72(1):100–110, January 2014.

[5] Niklas Elmqvist. Distributed User Interfaces: State of the Art. In José A. Gallud, Ricardo Tesoriero, and Victor M.R. Penichet, editors, *Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem*, pages 1–12. Springer-Verlag, London, 2011.

[6] Paternò F. and Santoro C. A Logical Framework for Multi-device User Interfaces. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS 2012* , pages 45–50, Copenhagen, Denmark, June 2012.

[7] Melchior Jérémie, Vanderdonckt Jean, and Van Roy Peter. A model-based approach for distributed user interfaces. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2011)*, pages 11–20, Pisa, Italy, June 2011.

[8] Vukovic Maja. Internet Programmable IoT: On the role of APIs in IoT: The Internet of Things (Ubiquity symposium). *ACM Transactions on Computer-Human Interaction*, 2015(3), November 2015.

[9] Domingo Mari Carmen. An overview of the Internet of Things for people with disabilities. *Journal of Network and Computer Applications*, 35(2):584–596, March 2012.

[10] Yusufov Murad and Kornilov Ivan. Roles of Smart TV in IoT-environments: A survey. In *Proceedings of the 13th Conference of Fruct Association*, pages 163–168, Petrozavodsk, Russia, April 2013.

[11] Bapurajan S., Yoshiura N., Prabu V., Rajendra D., and Nirosha K. Internet of Things Based Intelligent Street Lighting System for Smart City. *International Research Journal of Engineering and Technology(IRJET)*, 3(2):7684–7691, 2017.

[12] Bernaerts Yannick, Druwé Matthias, Steensels Sebastiaan, Vermeulen Jo, and Schöning Johannes. The Office Smartwatch âĂŞ Development and Design of a Smartwatch App to Digitally Augment Interactions in an Office Environment. In *DIS Companion 2014 Proceedings of the 2014 Companion Publication on Designing Interactive Systems*, pages 41–44, Vancouver, Canada, June 2014.