OULU UNIVERSITY OF
APPLIED SCIENCES

Xiaoyun Li

Kamill-Maximilian Simon

MIRROR-BASED INFORMATION DISPLAY FOR HOME

MIRROR-BASED INFORMATION DISPLAY FOR HOME

Xiaoyun Li, Kamill-Maximilian Simon
Bachelor's Thesis
Spring 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Authors: Xiaoyun Li, Kamill-Maximilian Simon
Title of Bachelor´s thesis: Mirror-Based Information Display for Home
Supervisor: Kari Laitinen
Term and year of completion: Spring 2019 Number of pages: 68 + 1 appendix

---

The objective of this thesis was to build a smart mirror system which would show information to the users while also reflecting their own image.

For building the mirror, a monitor was used along with a Raspberry Pi system and a two-way acrylic mirror. The Raspberry Pi is connected to the monitor through an HDMI cable and the mirror is acting as a layer in front of the mirror for showing the information displayed on the monitor while reflecting the images. The software of the mirror was developed using ElectronJS, a JavaScript library for building desktop applications.

As a result, the mirror was built, and the software was developed. They were assembled together to act as a real mirror.

Keywords: Smart mirror, Raspberry Pi, smart home

PREFACE

This thesis was made at Oulu University of Applied Sciences. During the thesis work we learned a lot about technologies like API, ElectronJS, node packages, building a frame, Raspberry Pi.

The supervisor of this thesis was Kari Laitinen, who helped a lot in writing it, giving ideas, providing resources and information about styling and conveying information and Kaija Posio for helping to correct the grammar and writing.

Oulu, 05.05.2019
Xiaoyun Li
Kamill-Maximilian Simon

CONTENTS

VOCABULARY

API	Application Programming Interface
BE	Backend
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DB	Database
FE	Frontend
GB	Gigabyte
GHz	Gigahertz
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HTML	Hypertext Markup Language
IR	Infrared
JS	JavaScript
JSON	JavaScript Object Notation
LAN	Local Area Network
OS	Operating System
RAM	Random Access Memory
SD	Secure Digital
SQL	Structured Query Language
UI	User Interface
USB	Universal Serial Bus
UV	Ultraviolet

1 INTRODUCTION

The purpose of this thesis work was to create a functional smart home system. The smart home system includes a smart mirror, a monitor for displaying information, and a Raspberry Pi minicomputer. These will collaborate to create a so called “Smart mirror” [1], which will display useful information to the users.

The smart mirror is capable of displaying a multitude of information, such as time, weather, the state of the lights in the home, and football match results.
The bulbs which are compatible with this system are the Philips Hue lights, which are controlled by using an API.

The main idea to create this project sparked from the desire to combine many components together in a unified, simple to use system. The mirror also acts as a way to make the system more fun to use and more aesthetically pleasing.

Throughout this thesis work, we improved our software development skills, learned more about APIs, and also learned a bit about wood cutting and frame making.

In this thesis work glass and wood have been used. The Raspberry Pi works to enhance the current technology into a modern technology.

2 USED TECHNOLOGY

2.1 Node.JS (Kamill)

“Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server side and client-side scripts.” [2]

The advantage of using Node.js

1. It is an open source framework. It has hundreds of libraries or modules free of charge. In this case, the authors found more than one library to control Philips Hue light easily in the most popular package manager website. Additionally, there are 4 features in this smart Mirror project. Using a library provides a better performance and it is less time consuming, as the authors are not familiar with every device, for example Philips bridge. Thus, it was decided to use a specific library to control Philips Hue and the Amazon Echo system, Alexa.

2. Node.js is a cross-platform runtime environment for developing server-side and networking applications. Node.js can be used to produce dynamic web pages, not only for the frontend side but also for the server side. As the smart mirror does not have the same complexity as dynamic websites, it does not need a heavy and complex server language to run. Node.js is a light but powerful language. Secondly, every developer could understand easily the entire stack, and corporate it easily.

3. It supports Electron framework. Node.js is one of the best and competitive eco-system for Electron.

The disadvantages of using Node.js

1. Some of the Node packages found in the repository are poorly documented or have no support from their creator, meaning that the API compatibility is lackluster, since Philips has changed the way their Hue API works during recent times.

2.2 Electron (Xiaoyun)

Electron is the main framework for the frontend side in this project. As this project is not a website or a mobile web application, it is critical to find a framework which fits in this mirror project (See Figure 1).

"Electron is a framework for creating native applications with web technologies like JavaScript, HTML, and CSS. It takes care of the hard parts so you can focus on the core of your application." [3]

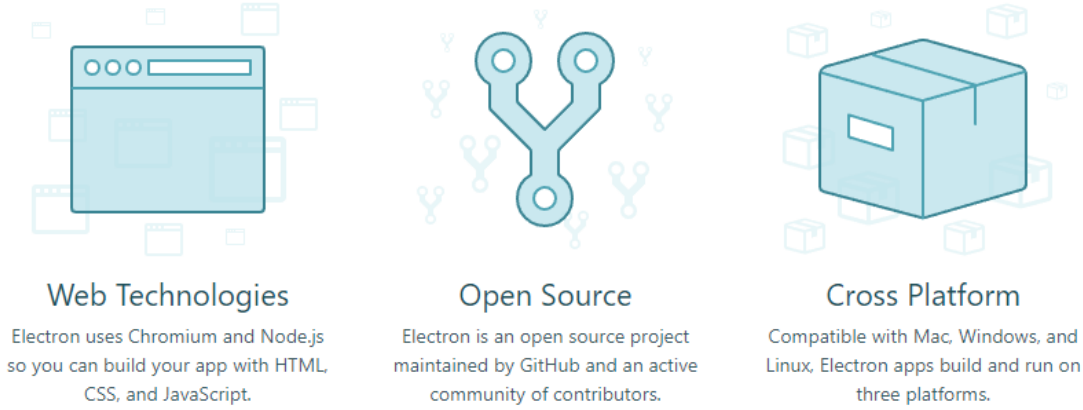


FIGURE 1. The Light Flow of Two-Way Mirror [3]

Electron is an interesting tool to create applications by using HTML, JavaScript, and CSS. For this project, the authors needed to create one desktop application to show the information and data. Instead of creating a website, desktop applications are more powerful than websites. For instance, the authors would like to add the Alexa Voice Service and Video Play service into this project in the future. Thus, a simple website cannot fulfill potential features implementation.

With Electron, it saves time and energy to create a real application based on authors' web technology knowledge.

Electron is open source. There are more open source UI toolkits and UI component for building applications. For instance, PhotoKit is one of UI toolkits, offering macOS style components. Currently, there are not many UI components in this project. However, if there are more features and components added into the project, PhotoKit will be the first option to add.

2.3 Raspberry Pi (Kamill)

The Raspberry Pi is a small computer which is commonly used for projects where low computer processing power is sufficient.

There are multiple models of Raspberry Pi released. Raspberry Pi Zero is the cheapest model. It has lower processing power than other models, thus it is ideal for simple projects.

In the Smart Mirror project, the chosen model is the Raspberry Pi B+. It has a slot for a micro SD to hold data and the OS, an HDMI connectivity for a display, multiple USB ports, an Ethernet port for a faster Internet connection and a 3.5mm audio for sound output. A wireless connection is also included, in case an Ethernet cable is not available.

This model was chosen because it fits the scope of the project well. It has a display connection which helps displaying the information on the Smart Mirror, a wireless connectivity to communicate over the network to allow a remote control, and enough processing power to handle the running of the software, while outputting graphics.

FIGURE 2 shows a Raspberry Pi Model B+ with its circuits and ports visible.
In terms of specifications, this model has 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU, 1 GB RAM, On-board wireless LAN - dual-band 802.11 b/g/n/ac, On-board Bluetooth 4.2 HS low-energy.

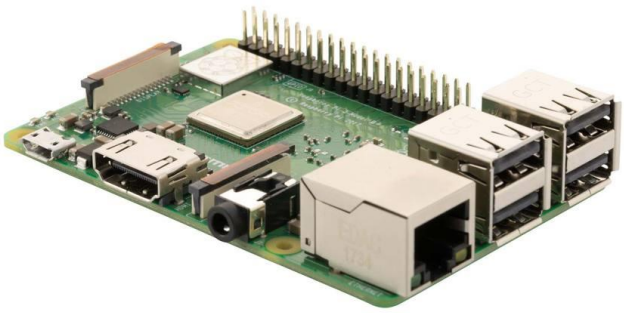


FIGURE 2. Raspberry Pi B+ [4]

2.4 Philips Hue (Kamill)

Philips Hue lights are a variety of smart lights which can be controlled remotely over the network. The main controls for the lights are setting the state to on/off, setting the brightness of the bulb. The lights can be controlled with multiple

devices, the most common being a phone application. Smart assistants such as Amazon Echo, Google Home and Apple's HomePod can also control the lights when receiving voice commands.

The Hue ecosystem has Zigbee 3.0 support, meaning that it is compatible with 3rd party devices with Zigbee support. Amazon Echo plus is an example for this.

The basic setup for the Philips Hue light is a Hue bridge, which is connected through an Ethernet cable to a router and a set of any number of smart bulbs. The bridge makes it possible to handle the communication between the controller devices and the bulbs. This is possible for devices connected to the same network, although it is also possible to change the state of the lights outside the network if a Philips account is linked to the bridge and the device.

Philips has released a variety of lights for indoor and outdoor use. Some bulbs are only capable of outputting white color, while others are capable of the output of 16 million colors by mixing multiple colors.

FIGURE 3 below shows a room lit by multiple Philips Hue lights.

FIGURE 3. Philips Hue room setup [5]

FIGURE 4. Philips Hue Bridge [6]

FIGURE 4 shows a Philips Hue bridge, which acts as a controller for the network of bulbs.

In this project, Philips Hue color bulbs are used for developing and testing. This allows testing with multiple colors and is enough for the basic usage.

2.5 Philips Hue API (Kamill)

Philips Hue API is a REST API created to allow the control of Philips Hue bulbs. A developer account is needed to access it.

The API supports GET, POST, UPDATE, PUT operations which allow setting the state of the lights on or off, setting the brightness of the lights, grouping lights, and changing the colors of the lights. For security reasons, a username is required for these operations. This can be obtained by pressing the link button on the Philips Hue bridge and making a request to the API.

2.6 neDB (Kamill)

neDB is a JavaScript database. It includes multiple methods to save the data, such as persisting it in the memory or saving it to a file in a JSON format.

The API of neDB is a subset of MongoDB's API, making it a NoSQL database. It supports operations, such as insert, delete, update.

The reason for choosing this as a data storage is that it is commonly used with Electron.js, the main framework used for creating the Smart Mirror app and it is suitable for small scale data, which aligns with the purposes of this project.

2.7 Dark Sky API (Xiaoyun)

Dark Sky API is one of the most significant weather API providers. It offers not only a daily and weekly forecast, but also accurate to-the-minute weather information. In developers' perspective, it is easy to use with a detailed document. The free tier gives 1,000 call per day, and $0.0001 per call if over the limitation. This platform updates weather information approximately every 3 minutes, which improves the accuracy of the data.

There are 3 features which attracted the authors to choose this API.

It is free of charge within 1,000 calls every day. As the mirror project only needs every 3 minutes to call the API, thus everyday it would be about 480 calls per day, which is a good deal for saving the cost of this project.
The API returns weather conditions in the requested units. For example, in temperature, the default unit is the Fahrenheit temperature, but in Europe, the default unit is the Celsius temperature. Adding a unit request in the API would solve this issue easily. In addition, the language can be changed in the same way as the temperature.

It returns a weather condition requested by time. It returns the particular date weather information, including the past and the future. It is a good feature for the mirror project to offer the specific date weather data searching, or the past week weather display.

2.8 Sports API (Kamill)

This project required a sports API to display sport competition results on selected matches. The criteria for choosing an API are listed below:

-Cheap price: A cheap price is required to test the data and check if it fits the scope of the project and can be implemented with the ease of using the Electron framework. Preferably, the API is free but a low cost per month is also acceptable
-Calls per minute: Calls per minute is also an important factor in the API. The user wants to get fresh data of sport competitions, thus a low restriction on calls can come as detrimental. The preferred number of calls is between 10-30, meaning that as much as an API call can be made every 2 seconds.
-Competitions: The competitions from which the API feeds data on. As many competitions as possible is preferred from a multitude of sports, such as football, tennis, darts. If the list of competitions is limited to a low number of coverages, it is considered what that list includes. It is preferable to have popular competitions which appeal to a broader range of users.
-Update interval: The update interval is also to be considered when choosing a sports API. For example, data which is updated every 2 minutes or more frequently is preferable over APIs which update the data less frequently or just contain the results of the competitions.

After doing some extensive research, the chosen API is the Football Data API.

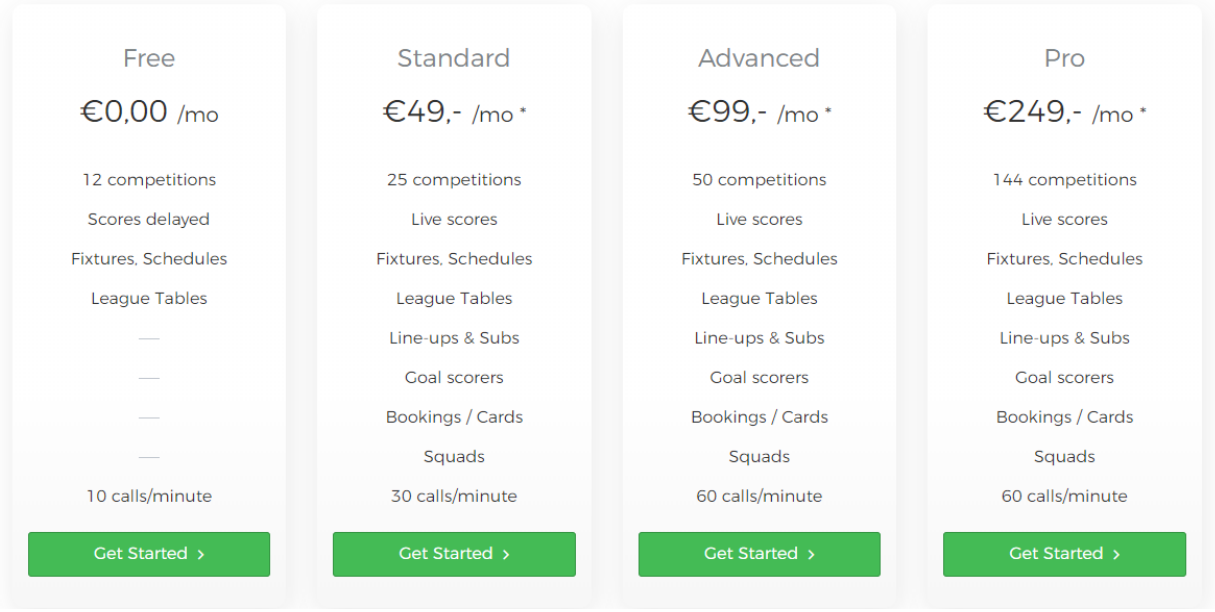


FIGURE 5. Football Data API pricing plans [7]

FIGURE 5 above shows the prices of different Football Data API plans.

This API fits the criteria of pricing, it has a free version which can be used initially for testing. The scores are delayed, meaning that the API only shows the competition results at the end of the match, but it is suitable to check if it fits the project.

Calls per minute criteria are also reached by this API, with 10 calls/minute it is possible to have the data refreshed every 6 seconds.

The competitions broadness criteria did not fit, but other APIs are more costly and do not provide a good free testing opportunity, as a result this API is used.

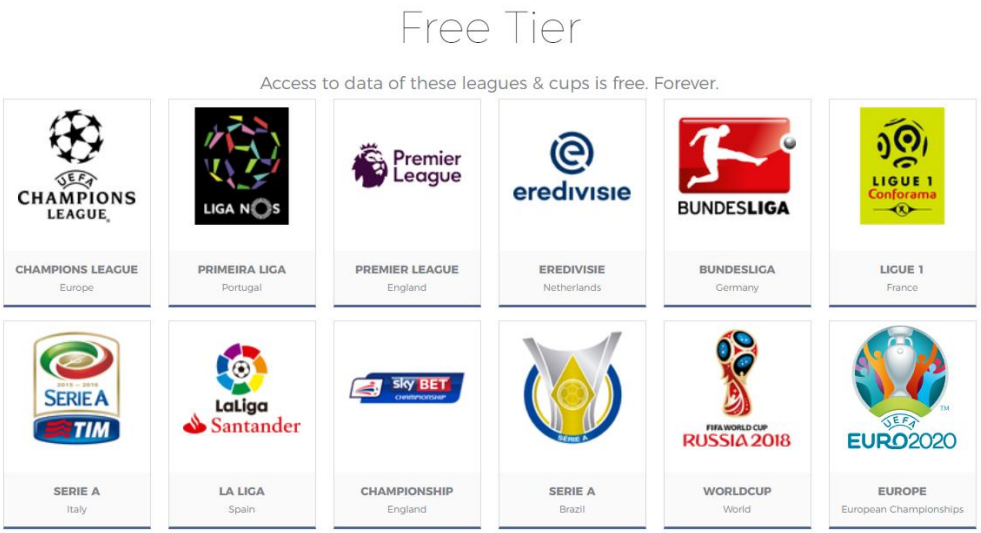


FIGURE 6. Available competitions in the free plan [8]

FIGURE 6 shows all the competitions which are available in the free plan of the Football Data API.
Initially, the free plan is used in developing this application, but later when more features are implemented, an upgrade is possible to the standard plan, which offers live results of the competitions, provides a broader range of tournaments, and increases the limit of API calls per minute.

2.9 Github and Gitkraken (Kamill)

"GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code." [9]

Github was the choice for using a version control in this project. It was already familiar before and it offers an easy way to create repositories and collaborate with multiple persons on them.

With Github it is easy to manage the code and quickly share it between multiple computers. A repository consists of multiple folders and files which can be accessed by anyone who is a collaborator in the private repository.

For the scope of this project a private repository was created. The next task was to plan how the workflow of the project will work. This was decided to work in the following way:

1. A master branch is the main branch of the repository. This will be used as a starting point after cloning the repository to a system.
2. When a feature is planned and ready to be implemented, a new branch can be started from the master branch. The naming of the branch can describe the feature which it will implement.
3. The feature is implemented in the feature branch. There can be multiple commits on that branch until the feature is finished.
4. After the feature is implemented in the feature branch, a pull request is made from the feature branch to the master branch. The other collaborator in the repository can review this pull request and comment/accept it.
5. Once the pull request has been accepted, the feature branch can be merged into the master branch.

Github's other useful functionality is a version control. In case the code breaks something or something is not working, the code can be reverted to a previous state where it was working well.

Following this workflow, it can help improve the project code and stability. If multiple persons are checking each other's code and suggesting better alternatives or conventions, it can improve the ease of implementing new features in the future.

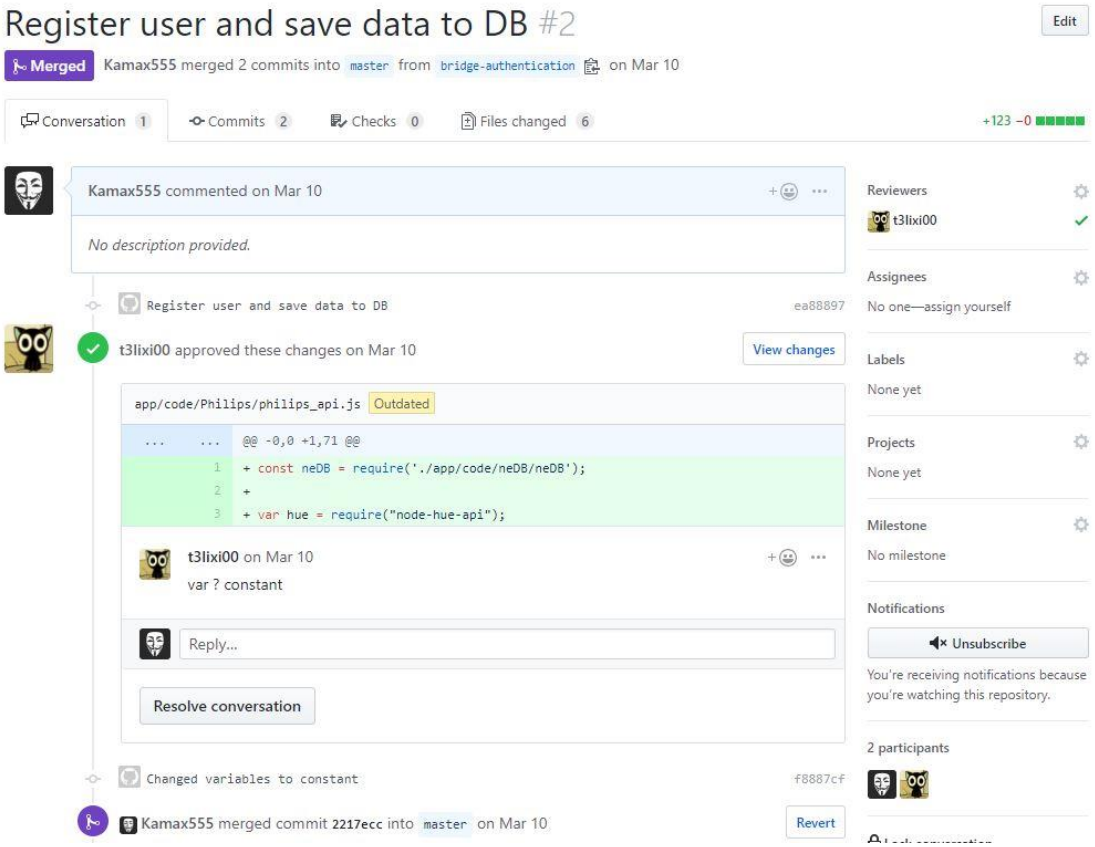


FIGURE 7. A pull request in the repository

In FIGURE 7 a pull request can be seen from the repository. This pull request asks to merge the authentication code to the bride from the feature branch named “bridge-authentication” to the “master” branch.

As it can be seen from the figure, there is a comment suggesting that a variable type can be changed from “var” to “constant”. Following this suggestion, the line of code can be changed and when it is accepted, it can be merged to the “master” branch.

Github has multiple tools for managing branches and pull requests. In the described case, there is a possibility to add comments to the pull request, suggest some alternative code, accept the pull request. In case when alternative code is suggested, the code can be changed easily from the UI on the Github repository’s webpage. In case the pull request is accepted, it can also be merged easily by pressing a button on the webpage.

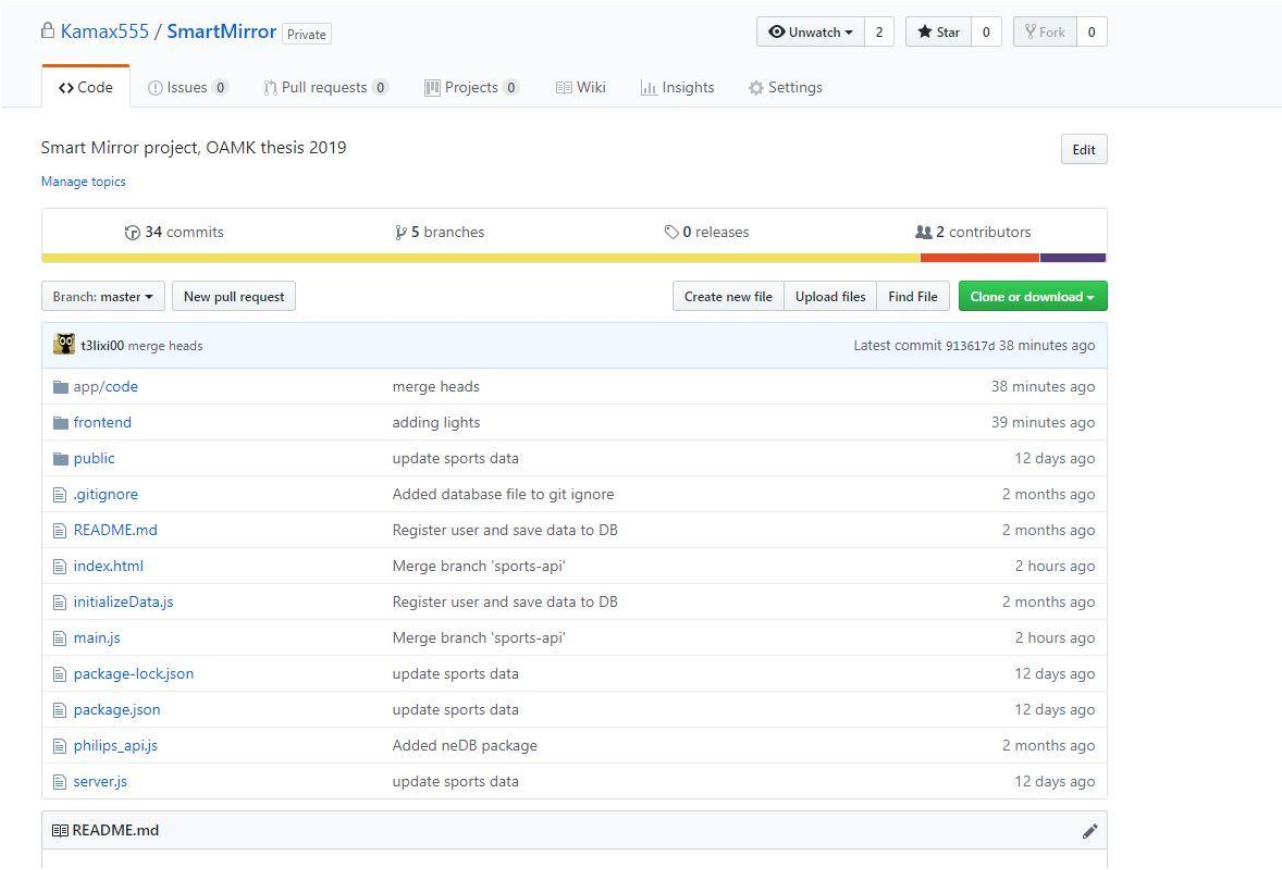


FIGURE 8. Github repository of the project

In FIGURE 8, a screenshot of the Github repository can be seen. There is useful information on that page, such as how many commits have been made to this repository, how many branches are there in the repository, how many contributors, how many percentage of each programming language is used in the code, and a list of all the files and folders in the repository.

Gitkraken is a Git client which helps perform git operations easily. It has a GUI interface for managing branches, making commits, and merging branches.

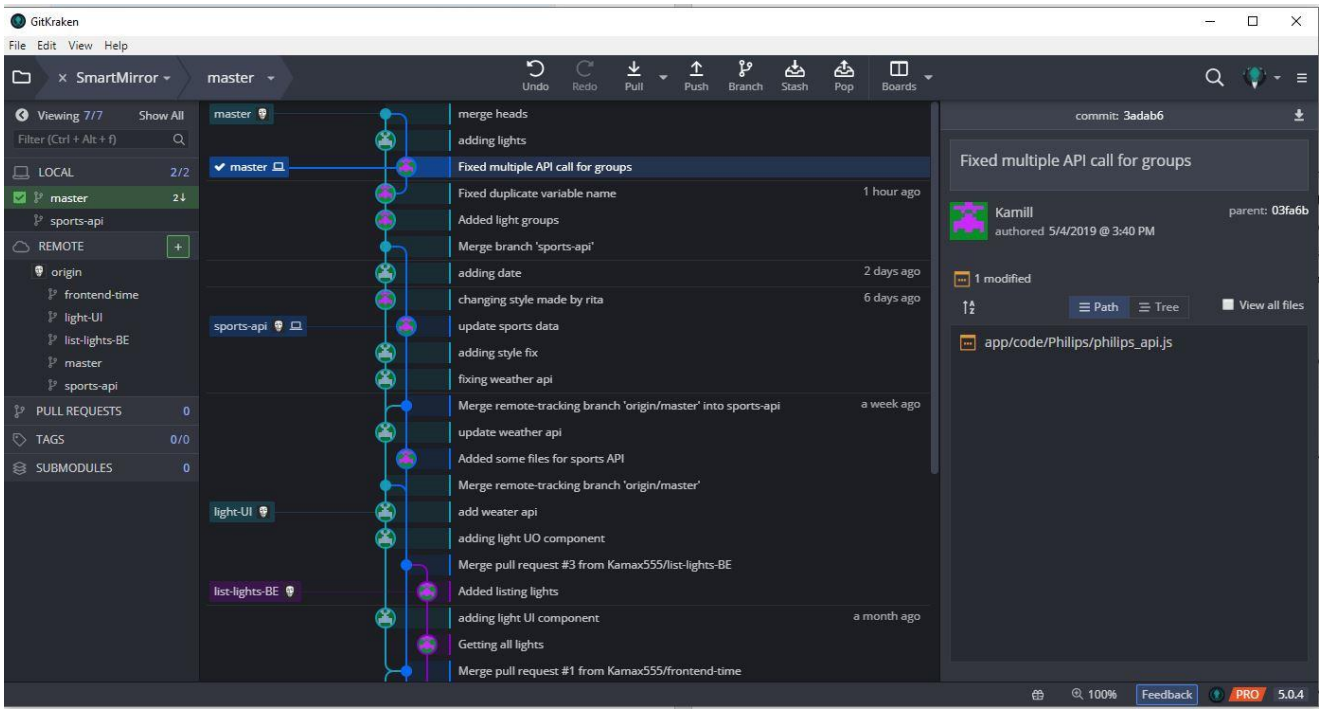


FIGURE 9. GUI of Gitkraken

FIGURE 9 shows the GUI of the Gitkraken client. In there, the project repository is opened. There is a tree in the middle to show the commits, branches and how they relate to each other. If some branches are ahead of the master branch, it will show as a fork in the interface. On the left side, the UI displays the branches which were used in this repository. It is also possible to switch branches there by double clicking a branch.

On the right side, there is more information about the selected commit. It shows the commit message and the files which were modified in that commit.

Merging branches can be done by right clicking on a branch and selecting another branch to merge into it from the menu.

Gitkraken also has a conflict resolver built into it. In case merging a branch to another one causes some conflicts, it will have the option to resolve those conflicts right from the client. Gitkraken was chosen for this project to make the repository management easier and less time consuming compared to the command line management.

2.10 Trello (Kamill)

“Trello is an incredibly versatile tool for project management. Its flexibility allows for it to be a simple tool for personal organization or a powerful engine for product development with large teams. In this article we share how we use Trello, along other interesting use cases from different startups.” [10]

Trello was used in this project as a SCRUM board to keep track of the tasks which had to be done and the overview of the project.

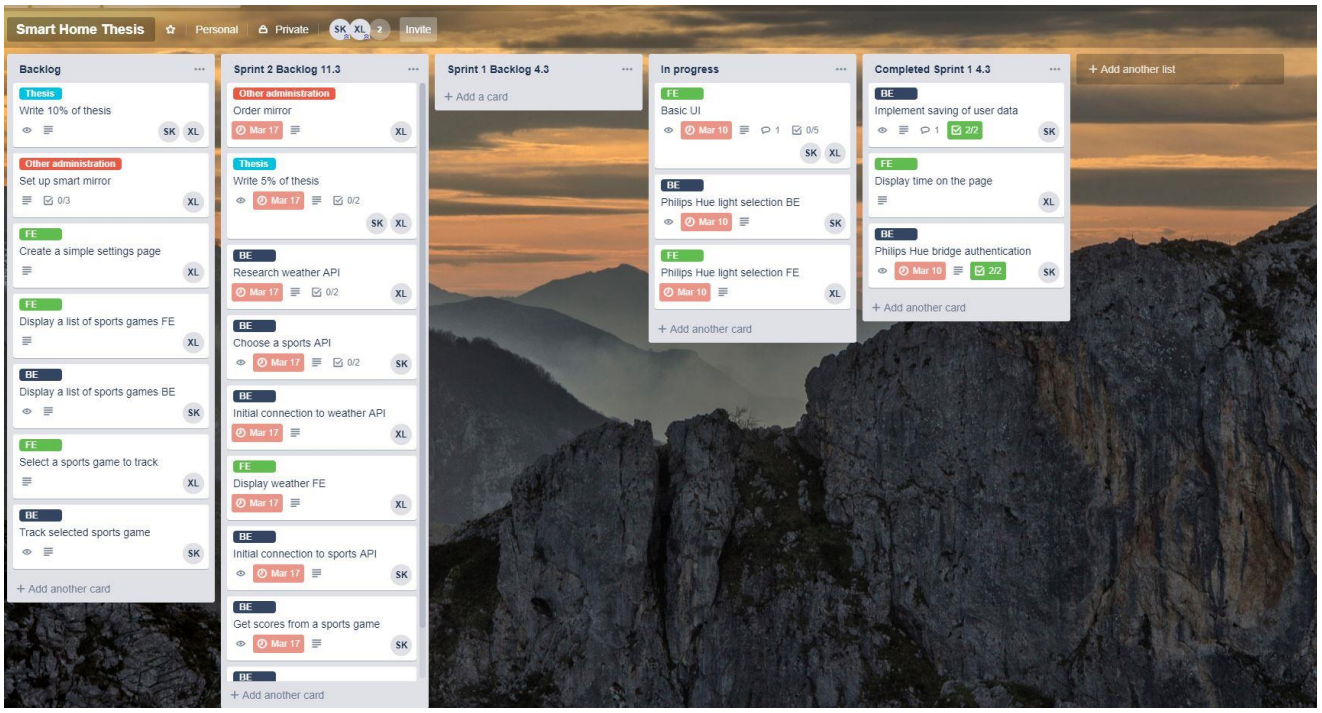


FIGURE 10. Screenshot of early version of Trello board

In FIGURE 10 the Trello board for the project is shown. This is the state it was in the first few weeks. On the first column there is a “Backlog” column. This column shows all the tasks that need to be completed to reach the aims and finish the project.

The second column is “Sprint 2 Backlog”. This column contains all the tasks that need to be done during the second week of development.

The “In progress” column has the tasks which are currently on progress and which the project members are working on. The “Completed Sprint 1” column shows the tasks which were completed in the first week of development.

The tasks have colored labels to show which category they belong to. There is “FE”, “BE, “Thesis”, and “Other administration” tasks. FE tasks are tasks which concern the frontend development. BE tasks are task which concern the backend development. Thesis tasks are tasks which require a thesis specific work, such as writing the document, taking notes. Other administration tasks are tasks which require anything other than developing or writing the thesis. These can be tasks, such as ordering a mirror or building a frame.

3 MIRROR HARDWARE

3.1 Mirror Hardware layout (Xiaoyun)

The smart Mirror is also called as a magic mirror. The smart mirror is highly customized, there is no specific rule for the mirror features or appearance. Due to the lack of time and financial support, only a basic prototype was made in this project. There are four basic hardware components used for this smart mirror project.

1. Raspberry Pi
2. Monitor / Monitor screen
3. Two-way mirror
4. Frame

Depending on the requirement of the smart mirror, it is possible to add a speaker, a microphone, motion sensors and other peripherals. For instance, with a microphone and speaker, a voice control system can be constructed. In the chapter 7 Future development possibilities, there will be more detailed information and plans for this project.

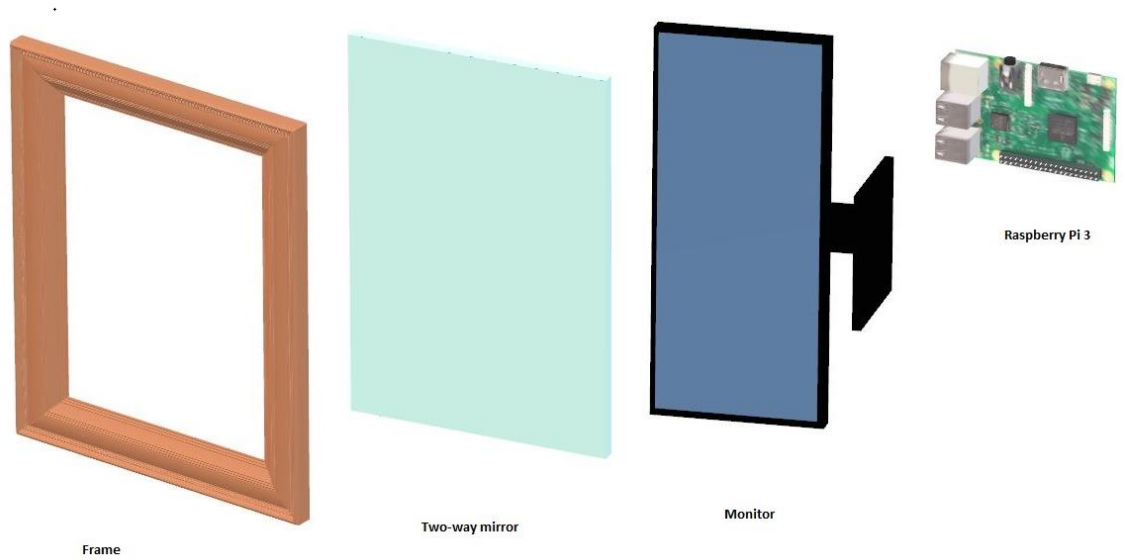


FIGURE 11. Smart mirror structure

Hardware contains all the necessary physical material. Each material should be put on over another. FIGURE 11 shows the structure of the smart mirror [11]. The frame is supposed to be on the top to hold the glass and monitor. Raspberry Pi is required to stick on the back to monitor and connect with it.

If the IR frame is added into the smart mirror, it is supposed to be between the frame and the two-way glass. Other peripherals can be added next to Raspberry Pi.

3.2 Mirror Physical frame (Kamill & Xiaoyun)

The physical frame for the mirror is required to hold the monitor and the mirror together and make the smart mirror look more aesthetically pleasing [12]. In this section, the building process of the frame will be presented.

3.2.1 Planning the frame

Building the physical frame required some planning and shopping for materials necessary to build it. There were 3 main things to consider:

1. The size of the frame
2. The materials needed to build the frame
3. The tools needed to build the frame

Unfortunately, after attempting to build a frame, the results were not satisfying enough to use it in a real-case scenario. However, the building process took some time, therefore it will be documented here.

3.2.2 Building the frame

When starting to build the frame, the monitor size and the mirror were measured. There was one issue in the beginning. The monitor was wider than the mirror, thus calculating the size needed for the frame became more difficult. The monitor width and the mirror height had to be measured. along with the thickness of the two objects combined. Once the measurements were done, some estimations could be made for the frame.

The authors' mirror size is 30 inch, which is 46cm wide and 61cm long. The monitor is 64.5 cm wide. In order to hold each element, the inner frame size is 41.8 cm * 57.8 cm at least. FIGURE 12 shows the size of our mirror.

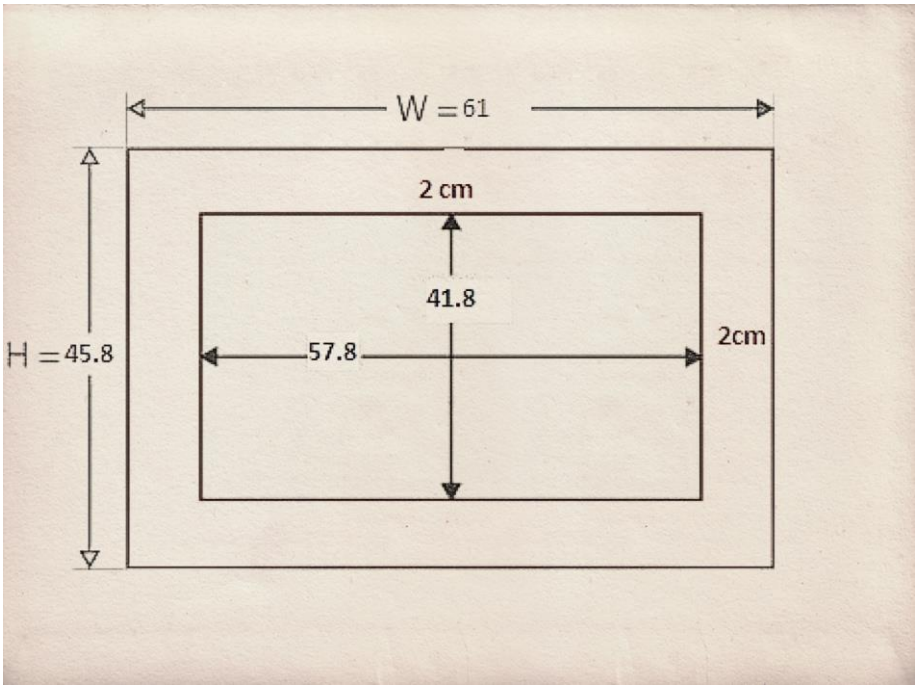


FIGURE 12. Frame Size

The next step was to buy materials for building the frame. Some wooden planks, glue, nails and screws were bought. The planks will act as the base of the frame and the screws, and nails will hold the frame together, along with some help from the glue.

The main structure of the frame is the following:

1. An outer frame, which is deep enough to hold the monitor and mirror together
2. An inner frame which holds the whole structure together.

FIGURE 13. Workshop room for cutting wood

In FIGURE 13, there is a picture of the room were the frame was being created. It had all the tools needed for this purpose.

First, the wooden planks had to be cut because their length was too long. The length of the wood had to be in accordance to the measurements made before. Secondly the wood edge had to be cut in a 45-degree angle to make it possible to connect the wooden pieces together.

FIGURE 14. Wood cutter machine

In FIGURE 14, the machine, which was used to cut the wooden planks, can be seen. When it was turned on, its blades rotated, cutting through wood. The handle helped to push down the blade onto the plank.

After the wood had been cut, the frame was ready to be assembled. This was supposed to happen by nailing the four wooden planks together and inserting a screwdriver between them to hold the wood together. This is the step which failed when building the frame.

The idea of the frame would have been the following:

1. The monitor and the mirror are held together to the frame
2. The Raspberry Pi is attached to the side of the frame
3. The cables connecting the Raspberry Pi and the monitor are arranged behind the frame

Although, the frame was not successful, it can be included as a future project to create it and use it with the mirror and the monitor. This will make the mirror hangable on the wall or usable in a vertical position.

3.3 Mirror Selection (Xiaoyun)

The most important material in this project is the mirror. As it was necessary to leave a monitor behind the mirror to show the data, it was necessary to select a two-way glass.

A two-way mirror is also called one-way mirror or semi-transparent mirror. It has 2 sides. On the front side, it reflects light, allowing people to see their own reflection, on the back side light goes through the glass, enabling people to see through it, and it could be transparent if the back side is dark enough (see FIGURE 15).

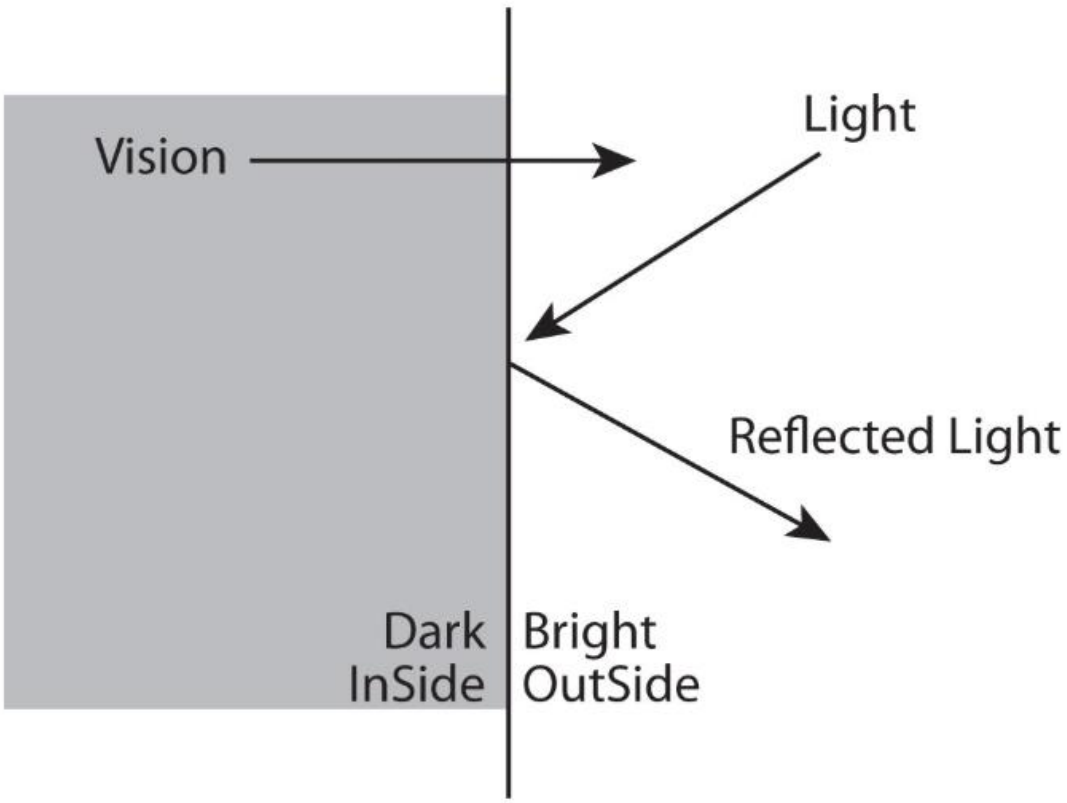


FIGURE 15. The Light Flow of Two-Way Mirror

3.3.1 Difference between real mirror and two-way mirror

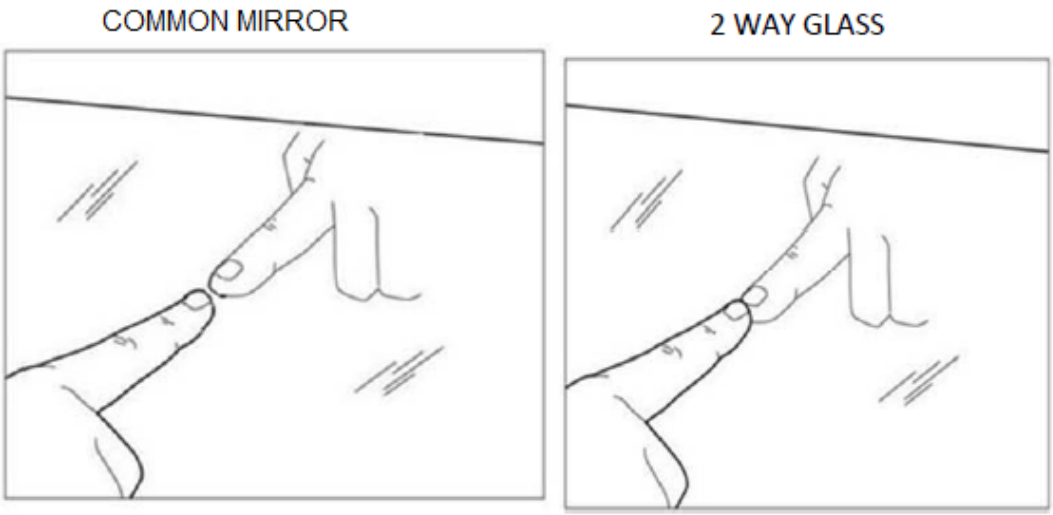


FIGURE 16. Comparison of mirrors [13]

Common mirrors reflect most of the light, but in some case some light could pass through. Generally, a mirror is coated with silver and two or more layers. Thus, the reflectivity is higher than that of a two-way mirror and the transmissivity is lower than that of a two-way mirror.

As shown in FIGURE 16, the way of checking a two-way mirror from a common mirror is to place a fingernail against the reflective surface. If there is a gap between the nail and reflection, it is a common mirror. But if the nail is directly touching the reflection, it is most likely a two-way mirror.

3.3.2 Two-way mirror comparison

Based on material and cost, there are various types of glass in the current market. If the size is 12” x 24”, Table 17 shows the difference between different types of mirrors [14].

TABLE 17. Difference Among Different Mirrors

	Visible Reflectance	Visible Transmittance	Thickness	Price
Glass Two Way Mirror	70%	11%	1/4" / 6mm	$99.99
Acrylic 2-way mirror	70%	30%	1/4" / 6mm	$136
Glass Smart mirror	70%	30%	1/4" / 6mm	$138
Dielectric Tv mirror	30%	70%	1/4" / 6mm	$345
One normal glass with two-way mirror film	99%	5%	1/4" / 6mm	$45

Except those factors shown in the table, we still need to consider e.g. size, tactile, and environment.

For the size, if it is larger than 24cm by 48cm or longer than its width, it is better to not consider an acrylic mirror because acrylic material is plastic and bends in a large size.

Touchable: Acrylic material is easy to scratch; the mere coating is more fragile.

Environment: If the environment where the mirror is left is wet like a bathroom, the wetness could damage the mere coating.

Frameless: An acrylic mirror generally has a sandy edge.

Monitor size: If the monitor’s size is smaller than that of a mirror. For a dielectric mirror, it is obvious to see the screen’s edge, because of high transparency

Based on all of the reasons listed above, it was decided to use a Two-Way Glass Mirror. Thus, the authors can build one mirror, which is touchable, 27cm long and easy to purchase in the local store.

3.4 Preparation of the monitor (Xiaoyun)

The monitor was chosen based on the size and cost. Cheap monitors can be easily found in a second-hand shop. The monitor used in this project is Asus 27 inch. Generally, it is better to assemble the whole mirror with a monitor screen, where the monitor's bezel is removed, as the bezel has thickness, which affects the mirror performance. If an IR frame is needed, the thickness of bezel could affect the touch quality of an IR frame.

However, in this project, the monitor back bezel is glued to a monitor. At least, the front bezel was removable. It is easier to glue the glass and monitor together, and there is potential chance to insert an IR frame inside the project. The back bezel of monitor, if it is more than 4cm thick is taking more space in the wooden frame.

4 MIRROR USER INTERFACE DESIGN (XIAOYUN)

Generally, a User Interface refers to the design of software or applications. In this chapter, there will be user design for this application and hardware design like the arrangement of cables and mirror frame design.

4.1 Mirror Layout Design

A Smart Mirror Project has a variety of examples on the Internet, however, most of them represent a simple and clean designed interface [15]. Also, in this mirror project, the mirror will be above the monitor, but the mirror transparency is not as good as that of the monitor, thus, color from the mirror is lighter than from the monitor. For a better viewing experience, white color is the main font color, as a contrast to the dark background. As for the UI tool, Balsamiq wireframes is suitable for this project, which focuses more on the structure and content, avoiding the usage of unnecessary colors in the project.

There were two versions of UI layout design in the beginning. The mirror order was delayed due to the difficulty of purchase. Thus, there were two basic models, depending on the mirror size. In the first model, shown in FIGURE 18, the mirror size is bigger than the monitor size.

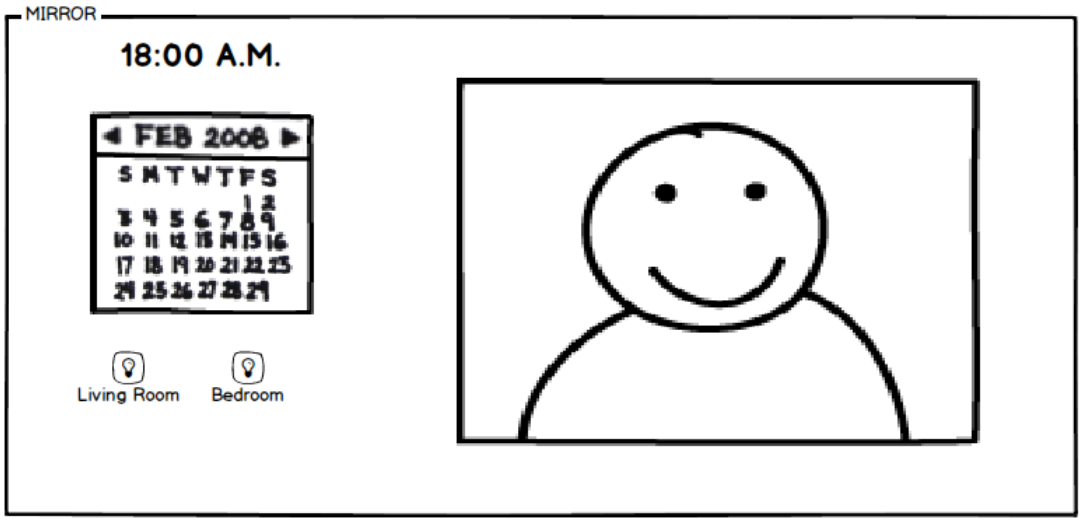


FIGURE 18. Version one UI Mockup

In this model, the mirror is the whole subject, and it is divided into 2 areas. On the left side, the monitor will hide behind the mirror. On the other side, it is for users to use as a common mirror. The advantage of this idea is that users can have the normal mirror area to use without distraction and interruption. Relatively, there is enough space, but it is difficult to show all necessary features only on the left side of the mirror. Thus, it needs more detail design in the left area. But the basic information will be shown, especially the status of lights. For example, if the living room lights are off, the bulbs color will be black, otherwise, the bulb color will be yellow.

The second layout (FIGURE 19) is when the mirror size is the same as the monitor size and the monitor is square.

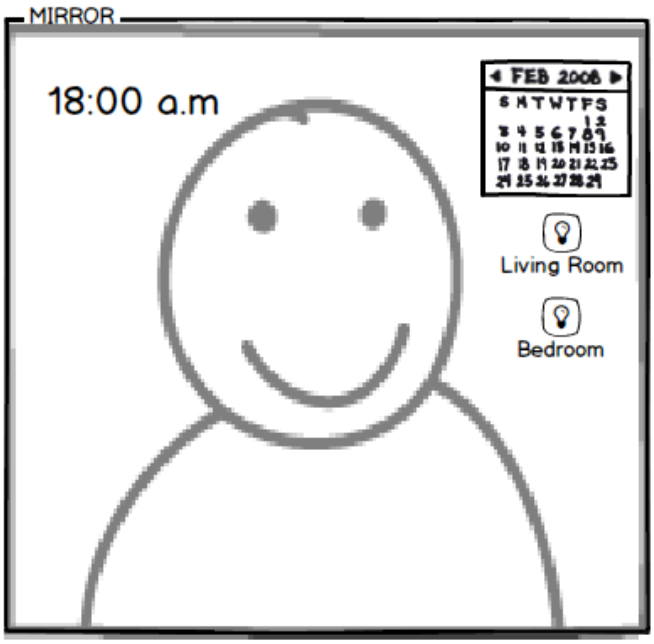


FIGURE 19. Version two UI Mockup

As for this design, obviously, there is more space to place the UI components, but it interrupts users' experience to use it as a normal mirror. So, it is critical to choose where to put those UI components. As the picture above is showing, time is an independent subject, short and simple as well. Thus, it fits on the left corner to give the time information to users without disturbing. On the right corner, it will be the calendar and lights data. The middle area is for users as a common mirror. The advantage of this model is that there is more spare space for an information display. Because of these small areas, the bigger UI component cannot fit in this mirror, for example, there will be weather information and sports information needed to be shown. Thus, it is necessary to redesign the calendar and other features, making those small and tidy enough.

After the authors got the mirror in hand, it changed their previous plans. The mirror is 46cm * 61cm, which is a standard 30inch monitor size with 4:3 ratio. The monitors that were prepared are two 21inch monitor (1:1 ratio), which are much smaller than this mirror. As the limitation of Raspberry Pi where only one HDMI connector is available, a single 21inch monitor is not matching the size of the mirror. Thus, a third monitor was purchased, with the size of 27inch. Currently, the monitor is having the same width as the two-way mirror, but it is shorter than

it. The third version UI design came after a 27inch monitor and mirror were settled.

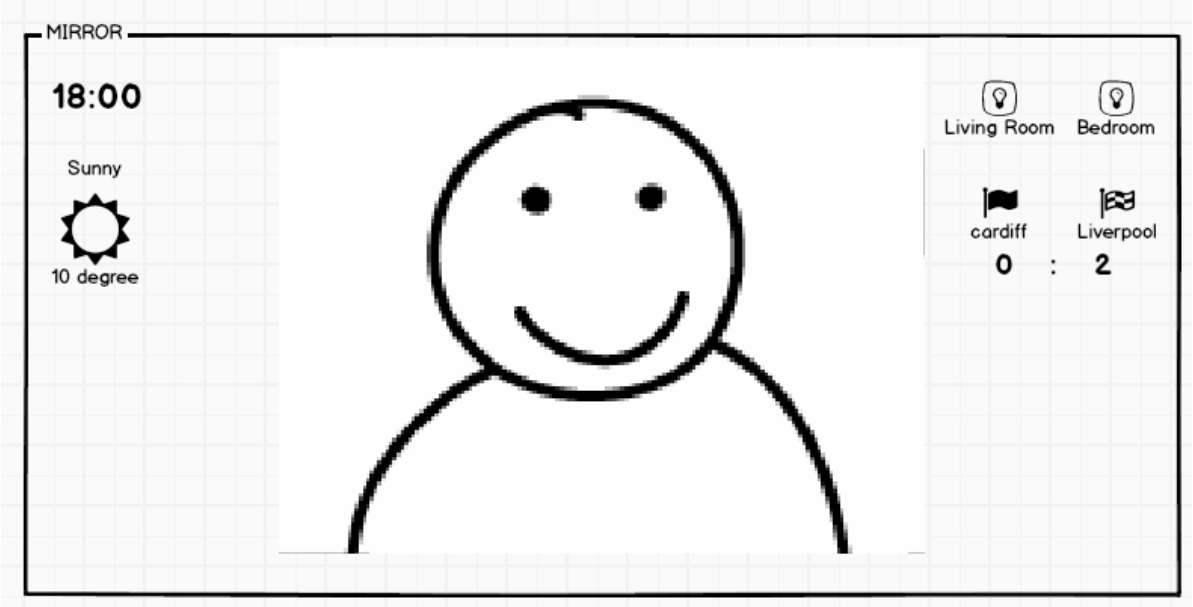


FIGURE 20. Version three UI Mockup

Due to the same width for both the monitor and screen (see FIGURE 20), it offers more space in the middle area to display. Time and weather information is represented on the left corner. On the right side, room lights information is shown. There is sports information added below lights, making it convenient for users to gather sports data.

If there is more information needed to be added, there is still some amount of space available on the mirror.

4.2 Weather Interface Design

Weather information is one of the necessary and valuable data in the project. The design of a weather user interface is devoted to being informative and clear. Therefore, the layout is important to let users just to have a quick glance to get needed data.

FIGURE 21 shows parsed data gotten from the Dark sky API, which locates in Oulu.

```
{
  "latitude": 65.01236,
  "longitude": 25.46816,
  "timezone": "Europe/Helsinki",
  "currently": {
    "time": 1555955676,
    "summary": "Overcast",
    "icon": "cloudy",
    "precipIntensity": 0,
    "precipProbability": 0,
    "temperature": 47.79,
    "apparentTemperature": 44.62,
    "dewPoint": 39,
    "humidity": 0.71,
    "pressure": 1020.22,
    "windSpeed": 6.81,
    "windGust": 16.17,
    "windBearing": 235,
    "cloudCover": 0.98,
    "uvIndex": 0,
    "visibility": 6.22,
    "ozone": 300.78
  },
  "hourly": {
    "summary": "Mostly cloudy until tomorrow morning.",
    "icon": "partly-cloudy-night",
    "data": [
      {
        "time": 1555952400,
        "summary": "Overcast",
        "icon": "cloudy",
        "precipIntensity": 0,
        "precipProbability": 0,
        "temperature": 49.49,
        "apparentTemperature": 46.88,
        "dewPoint": 39.54,
        "humidity": 0.68,
        "pressure": 1020.23,
        "windSpeed": 6.36,
        "windGust": 14.55,
        "windBearing": 239,
        "cloudCover": 1,
        "uvIndex": 0,
        "visibility": 6.22,
        "ozone": 301.47
      },
```


FIGURE 21. Weather API Data

The weather data includes a large amount of information, such as latitude, current time, current summary. It is grouped separately by currently-group, hourly-group and daily-group. Inside of each group, it contains time, summary, humidity wind information and UV Index.

Based on simple design concepts [16], only the most valuable data is shown. Thus, data is chosen from currently-group. To make sure of the accuracy of the weather, weather information will be updated every 3 minutes. Additionally, a weather summary, icon and temperature are grouped into a weather UI component.

Considering visual and usable aspects, the weather component needs to be pleasant-looking and clear. Thus, there will be less text, one big icon to attract users' attention and one temperature information for users to read.

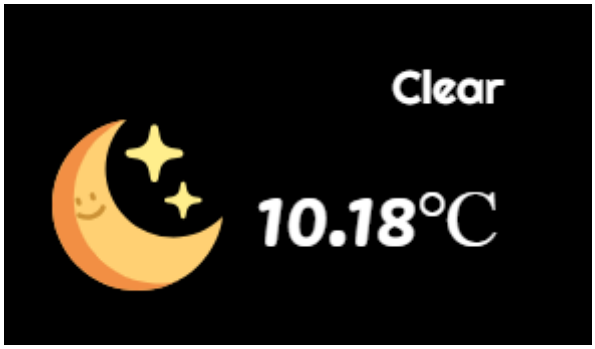


FIGURE 22. Weather UI Component

FIGURE 22 shows one weather UI component sample, whereas the temperature and summary may differ, depending on the location. The top text of this UI component is the weather summary, such as sunny, rainy, snow. The icon is the main object to attract attention as well as explain a summary by a visual image. The temperature is displayed below the icon to offer additional information.

According to the Dark sky official document, there are only 10 icon values defined. Thus, the following property of icons are clear-day, clear-night, rain, snow, sleet, wind, fog, cloudy, partly-cloudy-day, or partly-cloudy-night. Each icon property can have each icon as a match. In case of future utility, there are still 37 icons available.

Icons are downloaded from web FLATICON, which is one of the biggest free icons database[17]. See FIGURE 23.

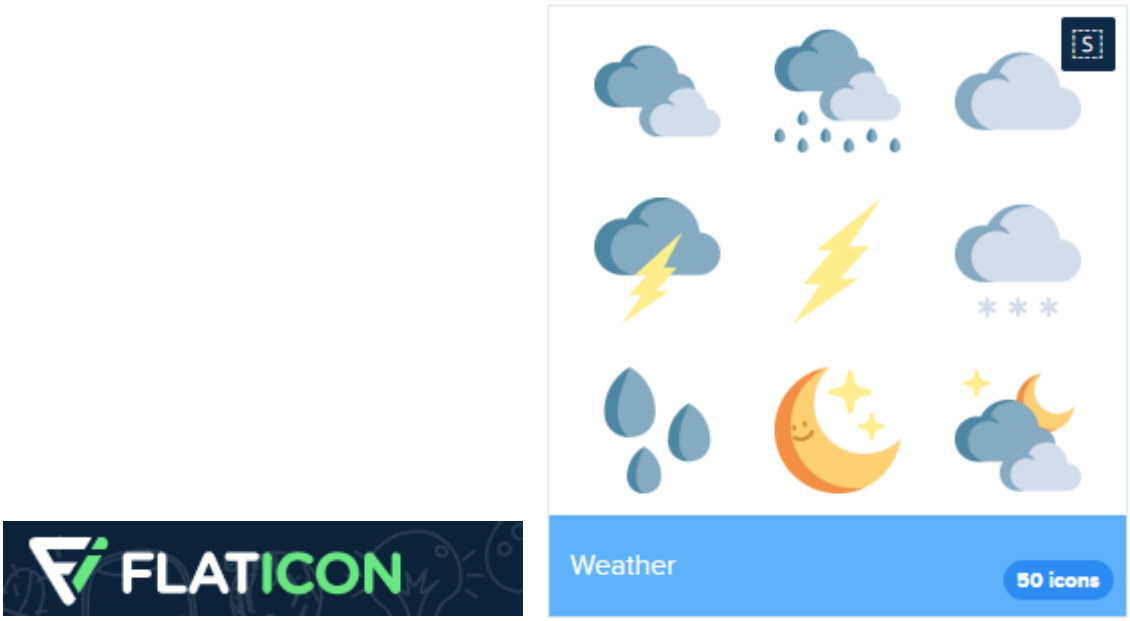


FIGURE 23. FLATICON and icon package [17]

This icon package is selected because of the high contrast and curvy style. As the monitor is placed below the mirror and the two/way mirror having low transparency, the high contrast color can be seen easily without an additional light. And a curvy style could give users a pleasant and relaxing feeling.

4.3 Light Interface Design

In UI design, one of the important refactors is consistency. *'Consistency in UI design is concerned with making sure elements in a user interface are uniform. They'll look and behave the same way. This helps constantly prove a user's assumptions about the user interface right, creating a sense of control, familiarity, and reliability.'* [18] Based on the weather UI design, the lights component keeps the same layout and style (see FIGURE 24).

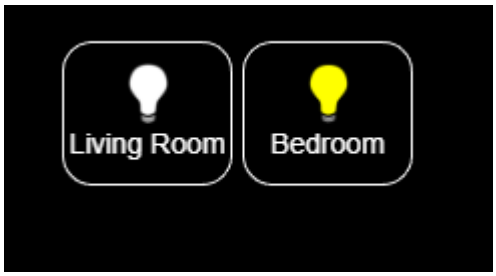


FIGURE 24. Lights UI Component

For a future use, lights components are designed as clickable button, which can be clicked. The purpose of border is to help a distinguish button from a non-button object. Icons are simple bulbs images and the color of bulbs describes whether those bulbs are on or off.
To keep the same curvy style, borders of two lights buttons are set to

border-radius: 15px;

which makes it rounder and cuter.

The amount of lights is fetched from the Philips Hue bridge API. Depending on the number of registered Philips bulbs, the frontend controller will display the same number of lights UI components. The detail will be shown in the next chapter.

4.4 Sports UI design

The last UI design is for sports. In the beginning, information related to tennis was decided to be added into this mirror, but as there is no free tennis API available, thus, football information is implemented instead.

The data returned from the football API has 24 objects, containing a live match, finished match and upcoming match. As the amount of data is large and the mirror has little space, it is not possible to fit everything on the mirror. The first solution to solve this issue is to make a dynamic data loop display. (FIGURE 25)

FIGURE 25. Example Of dynamic data display

The data will be changed every 30 seconds. In total, 12 minutes is spent for 24 groups of data. However, the disadvantage is that not many users have patience to wait for the desired match to display. Even if this design is improving users' interactive experience and keeping the whole structure clean, it still gives users a bit of an inconvenient experience.

Thus, the alternative option is used to insert a short table on the mirror. As for the size of the data, only the latest 5 matches are selected to be shown on the front side. In addition, there are no icons of teams shown in the mirror because the colors are diminished from this mirror material.

FIGURE 26 below shows the final design:

FIGURE 26. Sports UI

4.5 Font Family

The entire project style is tending to curvy, round, and cute. So is the Font style.

In this mirror, the main UI components are time, weather, lights and sports. FIGURE 27 shows mainly time and weather components. Fonts can be divided into numerals and text. Numerals represent more important data than text. Thus, for keeping the curvy style, the Font family Pacifico (see FIGURE 28) is selected for numerals. Meanwhile, the Font family Righteous serves for text.

FIGURE 27. Mirror Left side Design

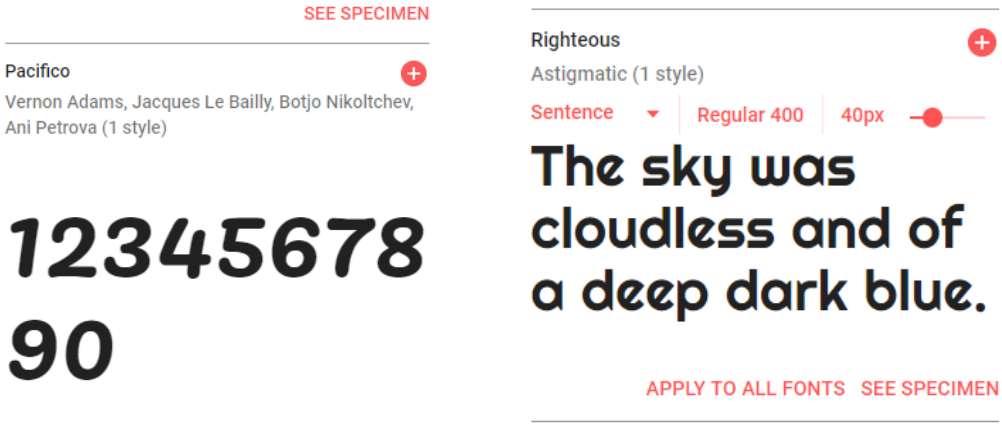


FIGURE 28. Font Pacifico and Righteous

5 FRONTEND DEVELOPMENT (XIAOYUN)

This chapter will explain the detail and logic about the Frontend implementation.

5.1 Front Page Layout

On the front page, the Bootstrap grid system is used as one frontend framework (The other frontend framework is Electron). Bootstrap provides the grid system to align and layout elements. It allows 1 to 12 columns and unlimited rows on the page.
FIGURE 29 shows the layout for this project.

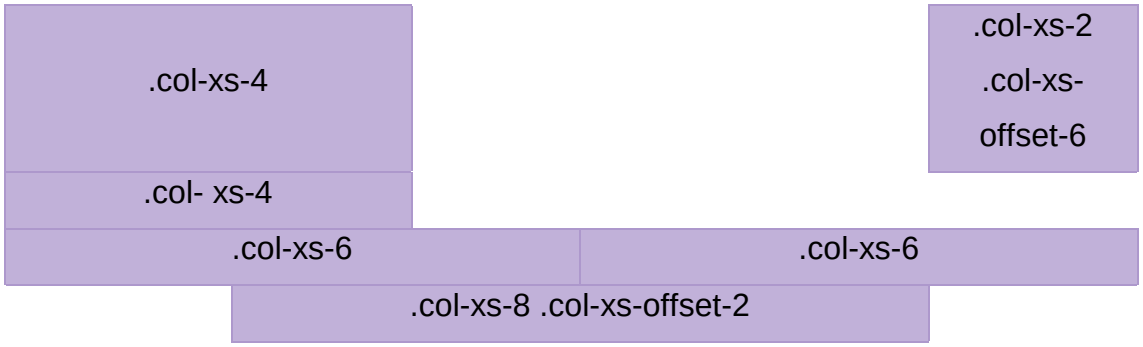


FIGURE 29. Smart Mirror Layout

In FIGURE 29, the purple color cell represents elements' spots. Generally, there are 12 columns in a row. ".col-xs-4" means three equal-width columns across. ".col-xs-offset-6" means six equal-width columns across to left.
In the first row, there are 2 areas for showing the date time and lights UI component. The left cell is for time and date. The other side is showing the lights status. In the middle (none purple area), it is left for blank. The second row is only for weather information. The last row shows sentences. With the Bootstrap grid system, it is easy to locate each element in its own location.

5.2 Time implementation

As the project was planned, the whole mirror was regarded as home-use furniture. Thus, time and date were the first planned and required feature in this project. Due to the simplicity of this single element, there is not much design for it. Only one JavaScript file is necessary to create time and date nodes when the browser is rendering the page.

```
function showCurrentTime() {
    const date = new Date();
    const months = ['January', 'February', 'March', 'April', 'May', 'June','July', 'August','September','October','November','December'];
    let hour = date.getHours();
    let min = date.getMinutes();
    let sec = date.getSeconds();
    let year = date.getFullYear();
    let month = months[date.getMonth()];
    let day = date.getDate();

    min = checkTime(min);
    sec = checkTime(sec);

    function checkTime(i) {
        if (i < 10) {i = "0" + i};
        return i;
      }
    var t = setTimeout(showCurrentTime, 500);

    document.getElementsByClassName('left-column')[0].innerHTML = hour + ':' +min + ':'+sec;
    document.getElementById('currentDate').innerHTML = day + ', ' +month + ', '+year;

}
```

FIGURE 30. Time JavaScript File

FIGURE 30 shows the code that returns time and date. The code of time function is inspired by W3Schools. The date function is written by Xiaoyun. The basic logic is to get the date and time from the Data Object and the transfer the data into different format. For instance, in the function date.getMonth(), the month data which returned is number 5. After months[date.getMonth()], month 5 has been transferred to “May”. Gathering the date and time data, then this function will create a nodes element on the front page to show the needed data.

Nevertheless, there is a drawback, time is based on the current location and it is not changeable. Thus, the smart mirror can only show the local time, not time in other time zones.

5.3 Weather Implementation

Weather is another important feature in this project. It is used in daily life, therefore it is becoming absolutely necessary.

Weather data is coming from Dark Sky API. To implement this feature, the first step is to connect this application to the weather API [19].

```
const api = 'https://api.darksky.net/forecast/fd665229bfc0f056e6c81e8c3363d2f9/65.012360,25.468160?units=si';

const axios = require('axios');

const weatherJs = require('./frontend/Js/weather');

module.exports = {
    getWeather: function() {
        axios({
            method: 'get',
            url: api,
            responseType:JSON
        })
            .then(function(response){
                let data = response.data;
                console.log(data);
                weatherJs.getWeatherData(data);
            });

        setInterval(this.getWeather, 180000);
    }
}

module.exports.getWeather();
```

FIGURE 31. Weather API connection

FIGURE 31 shows the API connection code, which is made with JavaScript and module axios [20]. Axios is making the Promise [21] call from Dark Sky API. As it is a promise, this application achieves an asynchronous transmission. Whereas, the request to the Philips API is a promise as well. After it gets a response, it will call a getWeatherData() function from a module WeatherJs on the frontend side. Meanwhile, the getWeather() function will update every 3 minutes to get the latest data.

After getting the data from the API, the second step is to fetch the data on the front page. There is one JavaScript file on the frontend side named weather.js. In total, there are three functions in this weather JS file. The first function is used to be called in the backend side. Just because the request to the weather API is a promise, the frontend side cannot get data immediately. It caused a promise

pending issue. To avoid it, the getWeatherData function is exported in the weather JS file and imported in weatherApi JS. Thus, when the data returns, the function getWeatherData will be called and it will call to the createWeatherNodes function in the following way

```
“module.exports = {
  getWeatherData: function (data){
      createWeatherNodes(data);
   }
}”
```

The second function createWeatherNodes (FIGURE 32) is used to create necessary nodes elements on the frontend side and to give values into these nodes. Because of the new data coming every three minutes, there is one “if” programming condition to filter the old weather data out and to replace the latest data.

```
function createWeatherNodes(data) {

    let weatherWrapper = document.createElement('div');
    let weatherIcon = document.createElement('img');
    let weatherWord = document.createElement('p');
    let temperature = document.createElement('p');

    weatherWord.innerHTML = data.currently.summary;
    weatherWord.setAttribute('class','weatherWord');

    weatherIcon.src = getWeatherIcon(data.currently.icon);

    temperature.innerHTML = data.currently.temperature + '&#8451;';
    temperature.setAttribute('class', 'temperature number');

    weatherWrapper.appendChild(weatherWord);
    weatherWrapper.appendChild(weatherIcon);
    weatherWrapper.appendChild(temperature);

    let weatherRow = document.getElementById('weather');
    if(weatherRow.childElementCount!==0) {
        let oldWeatherRow = document.getElementById('weather').childNodes[1]
        weatherRow.replaceChild(weatherWrapper, oldWeatherRow);
    }else {
        weatherRow.appendChild(weatherWrapper);
    }

}
```

FIGURE 32. Function of creating nodes

The third function (FIGURE 33) getWeatherIcon is choosing the relative images based on the weather status and returning them to the front page.

```
function getWeatherIcon(data) {
    switch(data) {
        case "clear-day":
            return "frontend/Images/weather/clear-day.svg";
            break;

        case "clear-night":
            return "frontend/Images/weather/clear-night.svg";
            break;

        case "rain":
            return "frontend/Images/weather/rain.svg";
            break;

        case "snow":
            return "frontend/Images/weather/snow.svg";
            break;

        case "sleet":
            return "frontend/Images/weather/sleet.svg";
            break;

        case "wind":
            return "frontend/Images/weather/wind.svg";
            break;
        case "fog":
            return "frontend/Images/weather/fog.svg"
            break;
        case "cloudy":
            return "frontend/Images/weather/cloudy.svg";
            break;

        case "partly-cloudy-night":
            return "frontend/Images/weather/partly-cloudy-night.svg";
            break;
        case "partly-cloudy-day":
            return "frontend/Images/weather/partly-cloudy-day.svg";
            break;
    }
}
```

FIGURE 33. Function of Icons selection

The last step is to align each weather element and text together and rearrange the size to match the screen size. This is achieved by the home.css file.

5.4 Lights Implementation

Lights implementation is similar to weather implementation. Both of them share the same data processing. Firstly, promises will trigger the frontend function after it gets data from the API. Secondly, data will be parsed and processed into frontend files. After filtering the unnecessary data, the valuable data will be inserted into node elements and fetched into web rendering. The only difference is that the weather UI component is grouped into 1 unit. This unit group is composed a of weather icon, a weather status, and temperature. But for light components, there are unknown groups, and it depends on the amount of users' Philips' lights and users' lights setting. In this case, there are 4 Philips lights which are grouped into bedroom and living room (JSON data can be seen in appendix 1). Thus, data is divided into bedroom and living room separately. In case of adding new lights in the future, looping through data is compulsory to get all necessary data. In this case (FIGURE 34), a data object name is detected if there is "room" in the name, this object will be filtered into the needed data.

```
getLightsData: function(data) {
    let lights = [];
    for(let x=0; x<data.length; x++ ) {
        if((data[x].name).includes('room')){
            lights.push(data[x]);
        }
    }
    addLight(lights);
}
```

FIGURE 34. Lights data filter function

After the needed data is available, data will be parsed into node elements created by JavaScript. One issue which was found during the development is that using appendChild (nodes) always requires removing the old elements. As a result of the mechanism of lights setting, new node elements are coming every 10 seconds. In case of showing duplicated elements, the solution shown in FIGURE 35 is used to clean those old elements.

```
function addLight(data) {
    let lightNodes = document.getElementById('lights');
    if(lightNodes.childElementCount !==0) {
        lightNodes.innerHTML = '';
    }
        insertLightNode(data);
}
```

FIGURE 35. Detection of the old elements

Before inserting the latest elements into the template, this function will check whether old elements exist or not. If they exist, they will be wiped out.

5.5 Sports Implementation

Sports data is the largest data on the frontend side. There are 24 objects in one API call, and the data structure is different from other data. One object is separated into awayTeam, homeTeam, and score. (FIGURE 36).

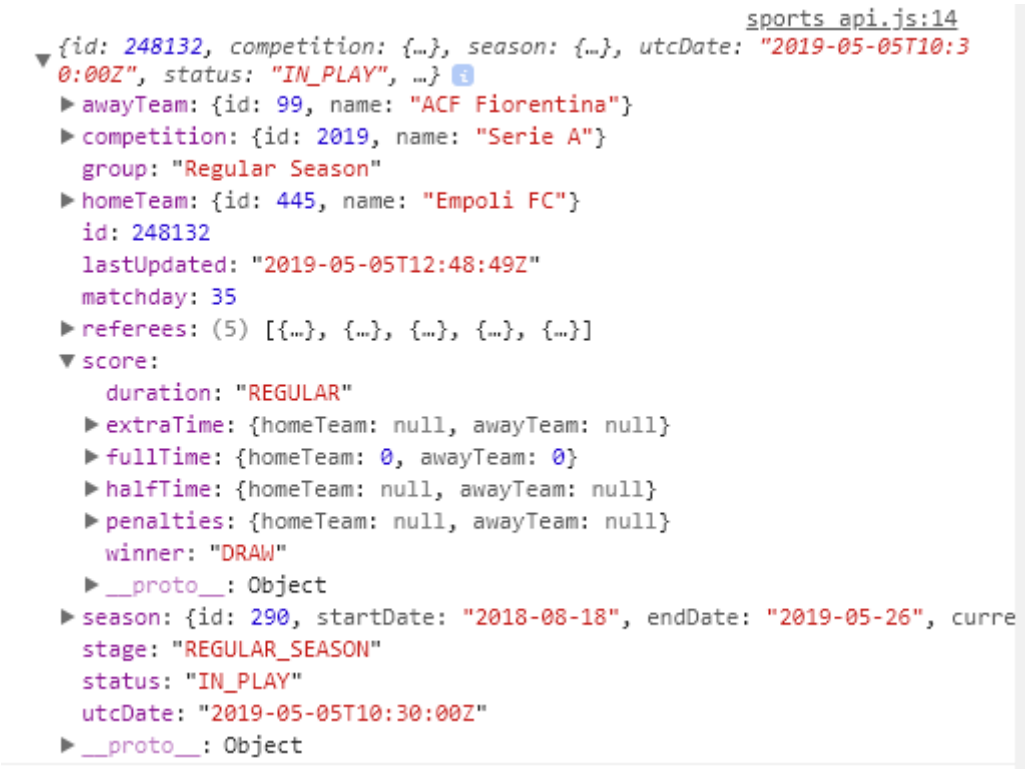


FIGURE 36. One object in Sports data

For gathering the fitful data, it needs to be extracted and replaced. As data has 2 main groups, awayTeam and homeTeam, 2 node elements are created based on them. According to each team, a function will get and insert a relative name and score into node elements. (FIGURE 37).

If a match is not started , score data is null in the data API. For a better user experience, a score will show “In playing” instead of null.

```
function addScore(match){
    globalMatch = match;

    let x = 'awayTeam';
    let y = 'homeTeam';

    let awayTeam = createTeamIcon(match.awayTeam, x);
    document.getElementById('football').appendChild(awayTeam);

    let quote = document.createTextNode(':');
    document.getElementById('football').appendChild(quote);

    let homeTeam = createTeamIcon(match.homeTeam, y);
    document.getElementById('football').appendChild(homeTeam);
}

function createTeam(data,whichteam) {

    let teamName = document.createElement('p');
        teamName.innerHTML = data.name;

    let score = document.createElement('p');
    let scoreValue = globalMatch.score.fullTime[whichteam];
        score.innerHTML = checkScore(scoreValue);

    let team = document.createElement('Div');
        team.appendChild(teamName);
        team.appendChild(score);

    return team;
}
```

FIGURE 37. Sports data transformation

6 BACKEND DEVELOPMENT (KAMILL)

6.1 Setting up the project

The project development was started by planning the required hardware and software for the development of the app and running the app.
The necessary hardware was a glass mirror, a monitor for displaying information, a Raspberry Pi for running the app and another computer for development.
The necessary software was the Raspbian operating system to run on the Raspberry Pi and Visual Studio Code to write code.

The first step on setting up the project was to turn on the Raspberry Pi and connect it to a display, mouse, and keyboard. The Raspberry Pi came with a preinstalled OS on a micro SD card, thus completing the OS setup was also a necessary step. When the OS booted up, the required packages, such as Node.js, had to be installed along with the Electron framework. There was one issue on the Raspbian operating system, which made it incompatible with Electron version 4 framework, thus Electron 3 had to be installed. Raspbian has planned support for Electron 4, but for the purpose of this project the third version was enough.

After the framework had been set up, a simple test was made to check the Raspberry Pi capability of running apps written in Electron. The test was successful. As a result, the Raspberry Pi was set up and ready to run the application in this project (FIGURE 38).

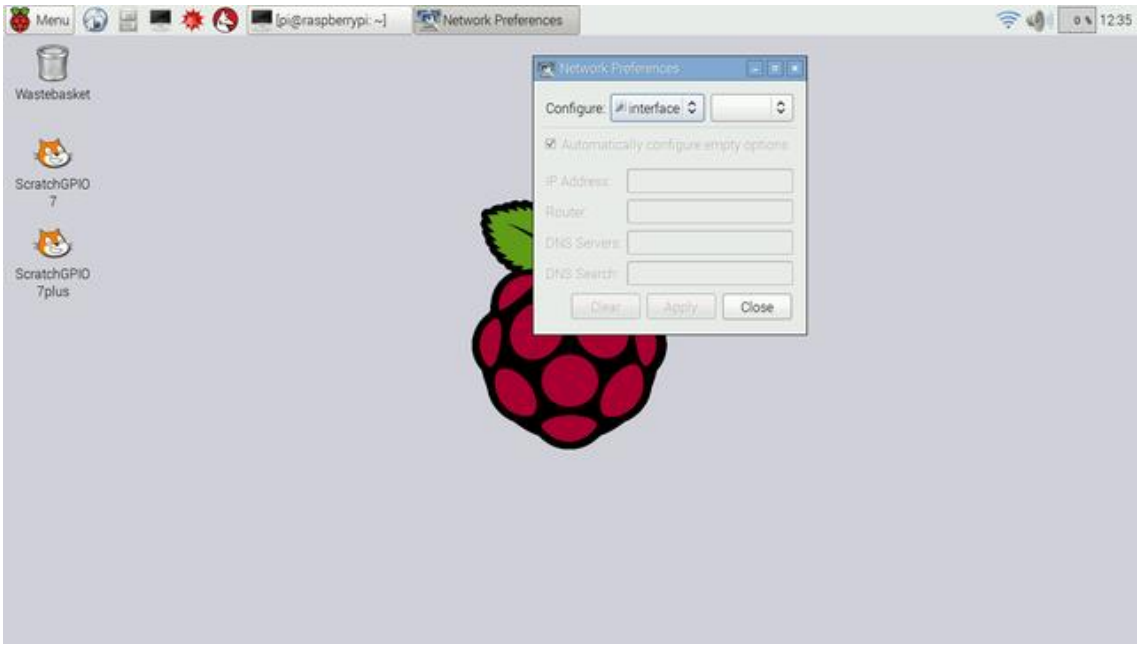


FIGURE 38. User interface of Raspbian OS

6.2 Starting development

The development on Raspberry Pi was not possible as it has a low processing power, thus another solution had to be considered. The development was made on a PC using Visual Studio Code as the main software (FIGURE 39).

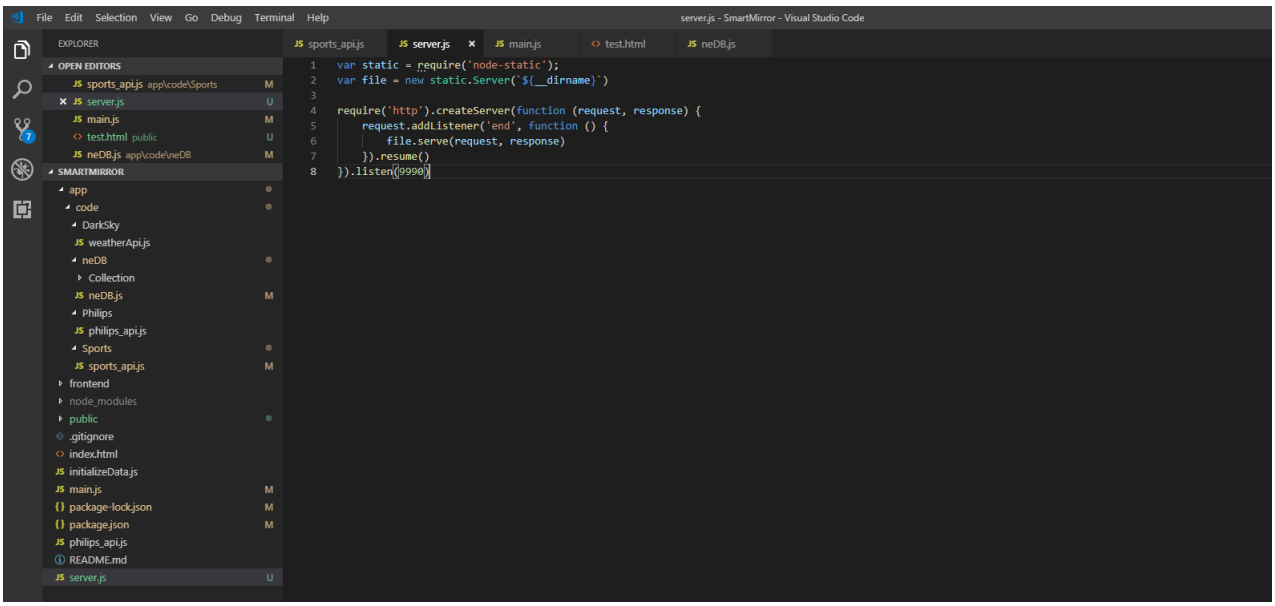


FIGURE 39. Example of development on Visual Studio Code

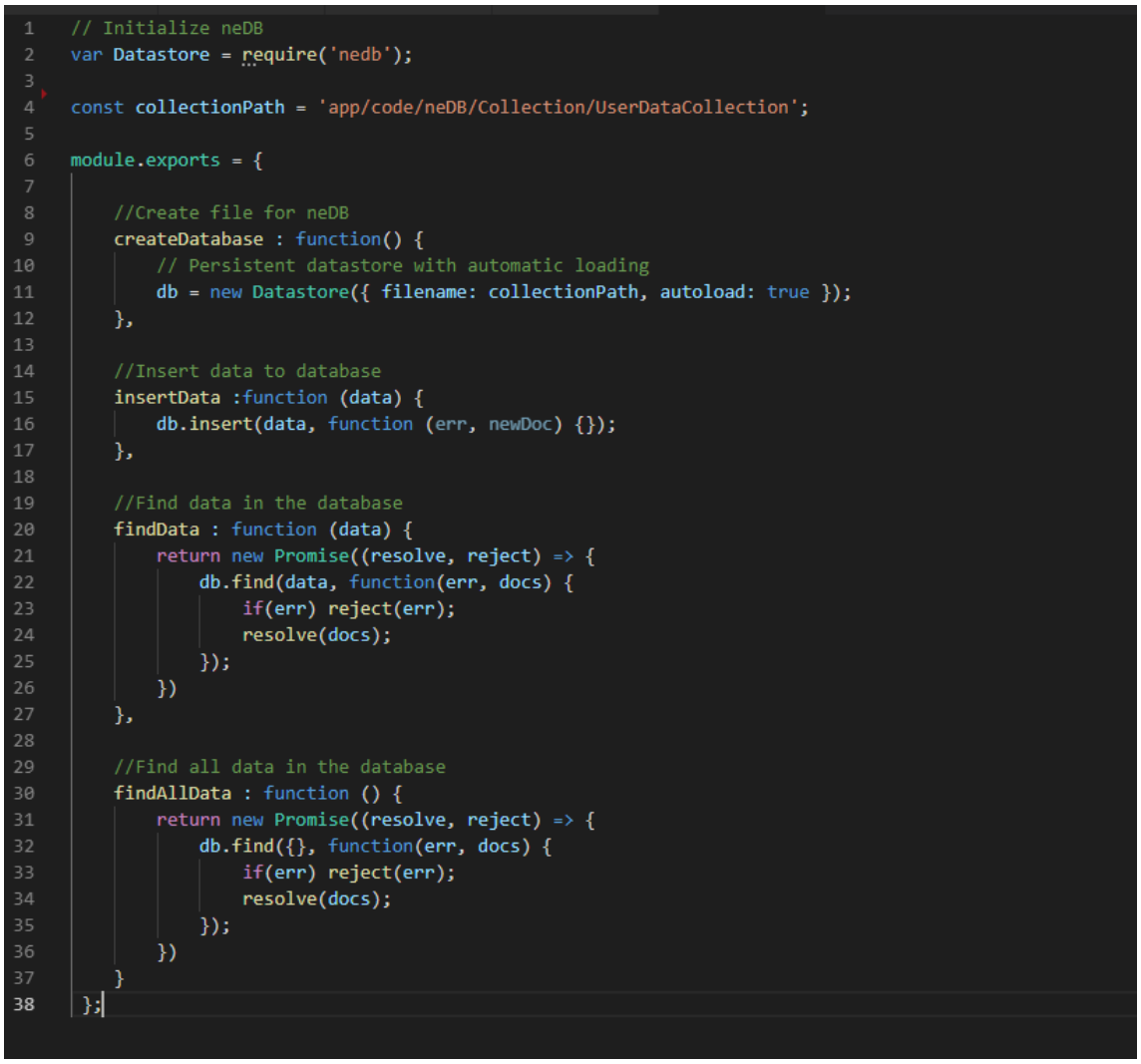
A git repository was set up to have a version control and to synchronize the files between the PC, Raspberry Pi and other computers that were used for the development.

6.3 Setting up the database

The most important step in the beginning was to set up a database which can hold user data. This was necessary since the application needs authentication for the Philips Hue API and its user's username needs to be saved.
For this purpose, the neDB JavaScript database was selected since the data is small, and Electron works well with this NoSQL DB.
When setting up neDB, some aspects had to be considered:

1. Which method of saving data is to be used
2. How to make the code reusable everywhere for easy database access

In the first case, it was decided to use a text file as a memory storage for the application. This means that the database will store all the data which is inserted in the JSON format to a file specified in the code. If the file does not exist, neDB will create one. Using this form of data storing makes it easier to test the application while developing, since the data is easily viewable when opening the file where it was written to.



```
// Initialize neDB
var Datastore = require('nedb');

const collectionPath = 'app/code/neDB/Collection/UserDataCollection';

module.exports = {

    //Create file for neDB
    createDatabase : function() {
        // Persistent datastore with automatic loading
        db = new Datastore({ filename: collectionPath, autoload: true });
    },

    //Insert data to database
    insertData :function (data) {
        db.insert(data, function (err, newDoc) {});
    },

    //Find data in the database
    findData : function (data) {
        return new Promise((resolve, reject) => {
            db.find(data, function(err, docs) {
                if(err) reject(err);
                resolve(docs);
            });
        })
    },

    //Find all data in the database
    findAllData : function () {
        return new Promise((resolve, reject) => {
            db.find({}, function(err, docs) {
                if(err) reject(err);
                resolve(docs);
            });
        })
    }
};
```

FIGURE 40. Reusable code for database operations

In FIGURE 40, it was created a JavaScript file which can be imported in any other JavaScript file for quick, reusable code. In this file there is a function for all the common database operations, such as creating a database, inserting data in the database, finding data in the database by a specific keyword, and finding all the data in the database.

The file where the data is stored is specified at the beginning of the code.

While using these functions, there were some issues in passing the variables around to other files, since accessing the database is promise based. The solution for this was to return a promise and use that to manipulate variables.

Calling the create database function is necessary in all the files where this code is used. This does not mean that the database is created each time, it is just a way to start doing database operations.

The function, “insertData”, gets the data as a parameter and inserts it to the database. This data can be in an object format.

The function, “findData”, accepts some data object as a parameter and will go through the database to check if the specified data exists in the database. The find all function will return the whole collection in the database.

6.4 Setting up Philips Hue API

The first thing to be considered when setting up the Philips Hue API was to use some node package while doing the API operations. Searching for that, there appeared to be multiple node packages for that purpose. The criteria for choosing a package were:

1. Making development easier
2. Compatible with the latest Philips Hue API
3. Well maintained

After some research, it was chosen a node package which is compatible with the latest API and seems to be still maintained.

The first challenge was to register the users when opening the app for the first time. Each API call needs a username to be passed on for security reasons. The solution for this was to check if a username already exists in the database or not. If it exists, it is used for making the API calls, if it does not, a new user is registered. Registering a new user requires pressing the link button on the Philips Hue Bridge. The interface should warn the user about this in case the button was not pressed. The API call will return a message in case that did not happen.

After pressing the link button, the registration can go on and the username can be saved into the database. On later API calls, that username can be used to authenticate the user.

```
//Search for bridges
var searchBridges = function(bridge) {
    if (bridge.length < 1) {
        return "No bridges found";
    } else if (bridge.length === 1) {
        //Search for field "bridgeIP" in the database
        bridgeIPData = neDB.findData(this.findBridgeIP).then((docs) => { return docs });

        bridgeIPData.then(function(result) {
            //If there is no bridge IP in the database yet, add it
            if (result.length < 1) {
                let foundBridgeIP = bridge[0].ipaddress;
                let insertData = {
                    bridgeIP: foundBridgeIP,
                    useBridge: true
                };
                neDB.insertData(insertData);
            }
        });

    } else if (bridge.length > 1) {
        return "Multiple bridges found. Choose one.";
    }
};
```

FIGURE 41. Searching for Philips Hue bridges

In FIGURE 41, a search is performed for Philips Hue bridges on the local network. If a bridge is found, its IP will be saved into the database. If multiple bridges are found, the user should have the ability to choose which bridge they want to connect to.

```
//Register user to bridge

userExists = neDB.findData({useUsername: true}).then((docs) => { return docs });

//Check if user exists, if not register one and save it to the database
userExists.then(function(result) {
    if(result.length < 1) {
        bridgeHost = neDB.findData(this.findBridgeIP).then((docs) => { return docs });

        bridgeHost.then(function(result) {
        var host = result[0].bridgeIP,
        userDescription = "Test device";

        //After generating user, save it to the database
        var saveUserResult = function(result) {
            let insertData = {
                username: result,
                useUsername: true
            };
            neDB.insertData(insertData);
        };

        var displayError = function(err) {
            console.log(err);
        };

        var hue = new HueApi();

        hue.registerUser(host, userDescription)
            .then(saveUserResult)
            .fail(displayError)
            .done();
        });
    }
});
```

FIGURE 42. Registering new users

In FIGURE 42, a function is called to search the database for usernames. If there is no username in the database, an API call will be made to the Philips Hue API to register a new user. This requires the bridge link to be pressed beforehand.

The Philips Hue API generates a username automatically and returns it. This username consists of numerous numbers and letters. When the username is generated, it will be saved into the database for a future use.

```
//List all lights

bridgeHost = neDB.findAllData().then((docs) => { return docs });

bridgeHost.then(function(result) {
    let bridgeIP;
    let bridgeUsername;

    //Get bridge IP and username
    result.forEach(function(data) {
        if(data.bridgeIP) {
            bridgeIP = data.bridgeIP;
        } else if (data.username) {
            bridgeUsername = data.username;
        }
    });

    var displayResult = function(result) {
        result.lights.forEach(function(light) {
            console.log(light.name);
        });
    };

    var host = bridgeIP,
        username = bridgeUsername,
        api;

    api = new HueApi(host, username);

    api.lights()
        .then(displayResult)
        .done();
});
```

FIGURE 43. Listing all lights

In FIGURE 43, an API call is made to the Philips Hue API to get all the lights which are connected to the bridge. Each light bulb is an object and contains a name which makes it easier to identify it. The object also includes other useful information, such as the state of the lights (whether they are on or off), the brightness, color.

The result from this call is passed to the frontend page where information about the bulbs can be displayed.
There is no way to get real data from the API. As a result, the state of the bulbs is updated every 10 seconds.

6.5 Setting up the sports API

Similar to setting up the Philips Hue API, it was also considered whether to use a node package or not when setting up the sports API. After some searching, it came to light that packages for sports APIs are scarce. As a result, API calls need to be made using an AXIOS call.

```
function getCompetitions() {

  const headers = {
    'X-Auth-Token': 'TOKEN'
  }

  const URL = 'http://api.football-data.org/v2/matches'

  axios.get(URL, {headers})
  .then(function(response){
    response.data.matches.forEach(function(match) {
      console.log(match);
    });
  });
}
```

FIGURE 44. Simple Sports API call

In FIGURE 44, a simple API call is made to get the recently played football matches. The result will come organized by football leagues and teams that played.

A token needs to be passed to the API call. This token can be obtained when registering for the API usage.

After the results are fetched, they are looped through and then they can be used in frontend to display results and competitions.

In case the API call limit is reached, this call will return an error.

6.6 Setting up weather API (Xiaoyun)

The weather API configuration is one of the simplest configurations in this project. According to the official document of Dark Sky, only secret key is needed inside an API call (Secret Key is the key which Dark Sky offered to allow the API calling).

Besides the API key, a customized parameter is critical to be added into URL. The first parameter is location coordinates. In Oulu, a coordinate is defined as 65.0121° N, 25.4651° E, searched from Google [22]. The second parameter is a temperature unit. The default unit in Dark Sky is the Fahrenheit scale. To switch to the Centigrade scale, unit setting is required to insert into URL.

For getting an asynchronous call, a promise is used as the main function. As for reducing the issue of promise delay, the frontend function is injected into the promise. After data is caught from the API, the frontend function getWeatherData will be triggered and the data is passed into the weather JS file. For keeping the accuracy of weather data, there is one setInterval function to make an API call every 3 minutes (See FIGURE 45).

```
module.exports = {
    getWeather: function() {
        axios({
            method: 'get',
            url: weatherAPI,
            responseType:JSON
        })
            .then(function(response){
                let data = response.data;
               // console.log(data);
                weatherJs.getWeatherData(data);
            });

        setInterval(this.getWeather, 180000);
    }
}

module.exports.getWeather();
```

FIGURE 45. Simple Sports API call

7 FUTURE DEVELOPMENT POSSIBILITIES

During the development of the software and building the mirror, many ideas came up about implementation possibilities. This chapter will focus on the most interesting ideas that came up but did not end up being implemented in this thesis work.

7.1 Remote controlled mirror

In its current state, the mirror is not a touch screen and cannot be controlled without connecting a mouse and keyboard to the Raspberry PI. While this is a good way to get information from it, there is no possibility to input information which could open up to further development possibilities.

This idea can be implemented by creating a node server to which a smart phone can be connected from the same network. The connected smartphone will act as a controller to the mirror. To keep the connection alive and make it possible to establish a communication between the node server and the Electron app, a socket needs to be used. The socket will listen to inputs and control the output on the mirror. The mirror will display a QR code or an IP address where the phone can establish the connection to the mirror. A mobile optimized interface will show up on the phone and multiple actions can be performed there.

Implementing this idea will open to multiple development possibilities discussed in-depth in the next sections.

7.2 Integrated smart light control

Once the remote connection has been established to the mirror, the Philips Hue lights can be controlled by pressing a button on the mobile interface. That will make an API call to trigger the light bulb state changing from on to off or vice

versa. The mirror interface will also update to reflect the changes in the light bulb state.
Controlling the colors or the brightness of the light will also be a possibility in this case. That could be a slider on the UI for the brightness control or a color palette where the colors can be selected.

7.3 Light reacting to sport results

This feature could be implemented to make the smart lights react to sports results in real time. The user could choose a football match to follow. When the football match starts and a team scores, the light can change color or blink.

7.4 Touch screen mirror

The mirror can be made a touchscreen by attaching an IR frame to its original frame. The IR frame will register the inputs of the users on the screen and allow pressing buttons or performing various other actions.

7.5 Voice Control system

The voice control system is achievable in 2 ways. First, to integrate with the Amazon Echo system Alexa. Or using the tiny JavaScript library named annyyang [23]. The annyang library supports node JS and light weight.

8 CONCLUSION

This whole project is built by developers' interest. During the implementation, there were many issues and difficulties. Even though some of them have been fixed and solved, there are still some drawbacks in the prototype. But from developers' perspective, it is still a milestone for combining our knowledge with a real life experience.

The purpose of this thesis work was not only to make a smart home system but to practice skills and train our mind. In the near future, this project will continue to be developed. The next step will be adding an IR frame to achieve a tangible mirror device and integrating it with the Amazon Alex system to achieve Voice Control.

Currently, the smart mirror is showing multiple information. There are still more ideas coming into this project, like playing YouTube videos, and a mobile remote control. The ideas will never end.

After this thesis, we got more passion about the IoT development and encouraged our friends to join us. We hope that we can make more smart devices to serve our real life.

REFERENCES

1. Cohen Evan. 2019. Smart Mirror. Date of retrieval 05.05.2019 https://docs.smart-mirror.io/

2. Node.js. Date of retrieval 10.04.2019 https://en.wikipedia.org/wiki/Node.js

3. Electron Official document. 2019. Date of retrieval 05.05.2019 https://electronjs.org/

4. The Pi Hut. Raspberry Pi 3 Model B+. Date of retrieval 10.04.2019 https://thepihut.com/products/raspberry-pi-3-model-b-plus

5. Coolshop. Philips Hue - Starter kit E27 Richer colors 2018. Date of retrieval 10.04.2019 https://www.coolshop.co.uk/product/philips-hue-starter-kit-e27-richer-colors-2018/AD7F8B/

6. HeikoAL. The Philips Hue Tap. Date of retrieval 10.04.2019 https://pixabay.com/photos/philips-hue-bridge-smarthome-3146129/

7. Football-data. 2019. Pricing. Date of retrieval 04.05.2019 https://www.football-data.org/pricing

8. Football-data. 2019. Free Tier. Date of retrieval 04.05.2019 https://www.football-data.org/coverage

9. Github. Date of retrieval 04.05.2019 https://en.wikipedia.org/wiki/GitHub

10. How we effectively use Trello for project management. Date of retrieval 04.05.2019 https://wpcurve.com/trello-for-project-management/

11. Kore. 2017. Building an intelligent voice controlled mirror. Date of retrieval 05.05.2019 https://medium.com/@akshaykore/building-an-intelligent-voice-controlled-mirror-2edbc7d62c9e

12. Hoffmann Robert. 2018. Building a big MagicMirror with metal frame – The summary, part list and prices. Date of retrieval 05.05.2019 http://robstechlog.com/2017/06/25/building-a-big-magicmirror-with-metal-

frame-the-summary-parts-prices/?fbclid=IwAR26o-Tgg7NqOF8-vLKan3ulm1RstAdeSV_lM6st2kaVpxkyGhu4VE7t0O4

13. Mir Rehman Rayees. 2011. How to detect 2-Way Mirror and Hidden Camera at any place. Date of retrieval 05.05.2019 https://mirrayees.blogspot.com/2012/10/how-to-detect-2-way-mirror.html

14. Two Way Mirrors. 2018. Acrylic Vs Glass For Smart Mirror Project(2018). Date of retrieval 05.05.2019 https://www.youtube.com/watch?v=tOC48aQ3JKo

15. Rahul, Jain. 2017. Prototyping Smart Mirror—UX Case Study. Date of retrieval 05.05.2019 https://blog.prototypr.io/prototyping-smart-mirror-ux-case-study-da20571c4428?fbclid=IwAR1TgM0pLBJmAktw-z6_NFJEUGaPVC25oHz_SrMVhkRsxFV8Jt-Fu2ViKhc

16. Rouse Margaret. 2005. User Interface (UI). Date of retrieval 05.05.2019 https://searchmicroservices.techtarget.com/definition/user-interface-UI?fbclid=IwAR1vkkl_AJglLq6l2gtnPRGUF4IKlxWpmJ7UTRzliHxm1D9gjwxLUpbJ4iQ

17. Weather Icon Pack. Date of retrieval 02.04.2019 https://www.flaticon.com/packs/weather-165

18. De la Riva Maria. 2018. Consistency of UI design. Date of retrieval 04.04.2019 https://careerfoundry.com/en/blog/ui-design/the-importance-of-consistency-in-ui-design/

19. Morelli Brandon. 2017. Build Weather App. Date of retrieval 15.04.2019 https://codeburst.io/build-a-simple-weather-app-with-node-js-in-just-16-lines-of-code-32261690901d?fbclid=IwAR0ayvh0rhnR0hg4epG5PHKS8OrCaHfy8XLm2Z0a-O-EkSHGYGCz57TY69c

20. Eschweiler Sebastian. 2017. Getting Started With Axios. Date of retrieval 05.05.2019 https://medium.com/codingthesmartway-com-blog/getting-started-with-axios-166cb0035237?fbclid=IwAR1AvKGJ2zxY2fuvHBfKhWe-Z-RvCZDM-c0DYCevWBxp_uBuZWrK849E1uI

21. Archibald Jake. 2019. JavaScript Promises: an Introduction. Date of retrieval 05.05.2019 https://developers.google.com/web/fundamentals/primers/promises?fbclid=IwAR3EG80iikdixsQn3HpvVL789G1Wea7T9MVMl5GIIyxtzIdx6TtEcmUc23o

22. World Atlas . 2019. Where Is Oulu, Finland? Date of retrieval 05.05.2019 https://www.worldatlas.com/eu/fi/15/where-is-oulu.html Date of retrieval 5-May-2019

23. Annyang Speech Recognition Library. Date of retrieval 05.05.2019 https://www.talater.com/annyang/?fbclid=IwAR2-M5ff8MM5n-lOC5KVtLpofb_KN0VTisKH2PK8cfypfB8UJpA0gk1Sffc

Philips Lights JSON DATA


```
[
  {
    "id": "0",
    "name": "Lightset 0",
    "type": "LightGroup"
  },
  {
    "id": "1",
    "name": "Living room",
    "lights": [
      "1",
      "2"
    ],
    "sensors": [],
    "type": "Room",
    "state": {
      "all_on": false,
      "any_on": false
    },
    "recycle": false,
    "class": "Living room",
    "action": {
      "on": false,
      "bri": 1,
      "alert": "select"
    }
  },
  {
    "id": "2",
    "name": "QuickHue: Switch Lights",
    "lights": [
```

```
      "1",
      "2"
    ],
    "sensors": [],
    "type": "LightGroup",
    "state": {
      "all_on": false,
      "any_on": false
    },
    "recycle": false,
    "action": {
      "on": false,
      "bri": 1,
      "alert": "select"
    }
  },
  {
    "id": "3",
    "name": "Bedroom",
    "lights": [
      "3",
      "4"
    ],
    "sensors": [],
    "type": "Room",
    "state": {
      "all_on": false,
      "any_on": false
    },
    "recycle": false,
    "class": "Bedroom",
```

```
   "action": {
    "on": false,
    "bri": 1,
    "hue": 7676,
    "sat": 199,
    "effect": "none",
    "xy": [
      0.5016,
      0.4151
    ],
    "ct": 443,
    "alert": "none",
    "colormode": "xy"
  }
},
{
  "id": "4",
  "name": "Entertainment area 1",
  "lights": [
    "4",
    "3"
  ],
  "sensors": [],
  "type": "Entertainment",
  "state": {
    "all_on": false,
    "any_on": false
  },
  "recycle": false,
  "class": "TV",
  "stream": {
    "proxymode": "auto",
```

```
   "proxynode": "/lights/4",
    "active": false,
    "owner": null
  },
  "locations": {
    "3": [
      0.1,
      0.19,
      0
    ],
    "4": [
      0.05,
      0.21,
      0
    ]
  },
  "action": {
    "on": false,
    "bri": 1,
    "hue": 7676,
    "sat": 199,
    "effect": "none",
    "xy": [
      0.5016,
      0.4151
    ],
    "ct": 443,
    "alert": "none",
    "colormode": "xy"
  }
},
{
```

```
    "id": "5",
    "name": "Group for wakeup",
    "lights": [
      "3",
      "4"
    ],
    "sensors": [],
    "type": "LightGroup",
    "state": {
      "all_on": false,
      "any_on": false
    },
    "recycle": true,
    "action": {
      "on": false,
      "bri": 1,
      "hue": 7676,
      "sat": 199,
      "effect": "none",
      "xy": [
        0.5016,
        0.4151
      ],
      "ct": 443,
      "alert": "none",
      "colormode": "xy"
    }
  }
]
```