

PiMirror

CA400

Technical Guide

PiMirror: Smart Mirror with Facial Recognition

Kieran Turgoose: 14355046

Supervisor: Dr. Suzanne Little

End Date: 20/05/2018

Contents

1. Introduction	3
1.1 Abstract	3
1.2 Glossary	3
2. General Description.....	4
2.1 Motivation.....	4
2.2 Research.....	4
2.3 System Functions	6
3. Design.....	7
3.1 Component Diagram	7
3.2 Context Design.....	8
3.3 Data Flow Diagram.....	8
3.4 Android UI Design	9
3.5 Process Management.....	10
4. Implementation.....	11
4.1 Facial Recognition	11
4.2 MagicMirror Modules.....	14
4.2.1 Core Module File: ModuleName.js:	15
4.2.2 The Node Helper: node_helper.js:.....	16
4.2.3 Weather Forecast:.....	18
4.2.4 Gmail:	20
4.2.5 Google Calendar:	20
4.2.5 Cryptocurrency:	21
4.2.6 Dublin Bus:	22
4.2.7 News Headlines:.....	22
4.2.8 Configuration Files:	23
4.3 Android Application	23
4.4 Component Installation	34
5. Problems & Resolutions	36
6. Results.....	37
6.1 Facial Recognition Accuracy Tests.....	37
6.2 Climate/Scenario-based Testing for Facial Recognition Python Code	37
6.3 Scenario-based Testing for Android Code	38
7. Conclusion	39
7.1 Future Work.....	39
7.2 Summary.....	39

1. Introduction

1.1 Abstract

PiMirror is a Raspberry Pi operated Smart Mirror with an accompanying Android application. An attached USB webcam allows for the utilisation of facial recognition to switch between designated user profiles. PiMirror is comprised of a monitor behind two-way glass, allowing for the interface to be shown through the mirror, as well as the mirror image itself. The interface can be used to display numerous modules of the user's preference. These modules include: 5-day Weather Forecast; Dublin Bus Real-Time Information; Cryptocurrency stock price tracking; Gmail inbox; Google Calendar events; and a scrolling News Ticker. The profiles allow the interface of the mirror to change to each user's pre-set module configuration, which are established from the Android app.

1.2 Glossary

SSH: Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. It allows the user to connect to and control the Raspberry Pi through Linux shell commands over a network.

Raspberry Pi: Raspberry Pi is a small computer, ideal for developing small practical projects.

Internet of Things (IoT): The network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to connect and exchange data.

Eigenfaces: An appearance-based approach to face recognition that seeks to capture the variation in a collection of face images and use this information to encode and compare images of individual faces in a holistic manner.

Fisherfaces: An enhancement of Eigenfaces. Especially useful when facial images have large variations in illumination and facial expression.

Local Binary Patterns Histograms: A simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighbourhood of each pixel and considers the result as a binary number. Considered robust in handling lighting variations.

MagicMirror: MagicMirror is an open-source modular smart mirror platform that uses a plugin system and Electron as an application wrapper. This allows it to be used without web server or browser installs.

Facial Recognition: A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source.

OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez.

DOM Object: The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML

document as a tree structure wherein each node is an object representing a part of the document.

2. General Description

2.1 Motivation

The motivation for this project originated through the desire to develop some form of an Internet-Of-Things (IoT) project. During my INTRA period in third year spent at SAP, a primitive Smart Mirror was shown during a developer's showcase, displaying a basic interface and some minor modules. These modules included date, time and a static news bar at the base of the screen. This was the main inspiration for pursuing further research into this project.

This system can be a great addition to the ever-growing market of IoT household objects. We are seeing more and more IoT additions lately, from light bulbs to thermostats, to video and voice enabled doorbells. PiMirror can be added to this market and provide use to the everyday adult, both functionally as a mirror, and as an information interface. Likely to be used in either an entrance hallway, bathroom or bedroom, PiMirror will save the user time throughout their day, most importantly in the mornings. When the user wakes up and uses their mirror they will be updated of the weather for the day, any important emails/scheduled events, important news headlines and how much time until their bus arrives. Developing this system was aimed at giving the user freedom to go about their routine without having to find all this information for themselves.

2.2 Research

Smart Mirror Interface:

After seeing the aforementioned Smart Mirror during my INTRA placement, a large amount of the remaining portion of my summer was spent researching possible avenues with which to pursue this project. While researching these in-market Smart Mirrors, a lot of similarities were found. Most of them used an Android interface, to allow users to just plug in an Android enabled computer and have the mirror functioning almost as an enlarged tablet. Other more home-made versions went down the route of using Javascript and other similar technologies to host the interface. It was during this research that the open-source platform MagicMirror was found. As it is open source, there was the ability to use their already developed backend technology and just focus on the frontend design of the specific modules themselves. It is Raspberry Pi friendly, and considering the goal of developing a lightweight IoT project, this was considered the best starting point for the project.

Facial Recognition:

When researching the facial recognition portion of this project, due to the nature of it being used for loading personal information, there were multiple factors that were of key importance. These were accuracy, performance, and security. In terms of this project the least important was the security portion, due to this being mainly a proof-of-concept, and as hardware was very limited. With this in mind, possible facial recognition algorithms were researched, that were both reliable enough with their accuracy, and also lightweight enough to run from a Raspberry Pi.

Due to its rising popularity, there were many services available for this purpose, including those provided by Google, Amazon, Microsoft, and OpenCV. Due to prior basic experience

with python and OpenCV, this was the initial choice as it is known to be lightweight enough for a pi to handle. The three algorithms OpenCV provides are – Eigenfaces, Fisherfaces, and Local Binary Patterns Histograms. Eigenfaces was considered the most basic, as it is mainly focused on remembering facial “features”. However, it was stated that it is heavily affected by lighting. Fisherfaces is an evolution of Eigenfaces, it extracts the useful characteristics that separate one person from another. This means that one person’s feature will not be more dominant than another. This too was affected by lighting though.

The final algorithm - Local Binary Patterns Histograms (LBPH) – was considered to be the best when recognising faces in inconsistent lighting, and so therefore seemed the most complete of the three algorithms.

Raspberry Pi:

The Raspberry Pi is the computer that powers the PiMirror interface, and also processes the facial recognition code and therefore had to be a model powerful enough to perform these tasks. The Raspberry Pi 3 Model B was the newest and most powerful model yet, and so was the obvious choice for the task. It had built in Wi-Fi too which was an added bonus. There is also plenty of documentation online regarding implementing OpenCV with Raspberry Pi’s so this made the choice of hardware clear.

Android:

With no prior experience in developing using android, this project consisted of a lot of continuous research into the capabilities of the android development, and what was actually achievable considering the timespan. Since android is mainly a combination of Java and XML, there was some considerable comfort in using this technology considering previous university modules have taught significant Java skills. While researching android, and attempting to find a way of connecting to the pi itself from the app, a package called “JSch” or “Java Secure Channel” was found. This allowed connections from the app to the pi via SSH, which would allow configuration files to be changed as expected. This was the main hurdle when considering the complexity of the app itself, and so seemed a valid and practical solution.

Javascript:

There was no prior experience developing using this language. The use for Javascript as part of this project was for the front-end development, i.e. the modules to be displayed on the PiMirror interface. Online documentation and tutorials were consulted for basic syntax research. Whereas the research into developing the actual structure of the module files was provided by the MagicMirror platform itself, who provided basic developer documentation at the following address:

<https://github.com/MichMich/MagicMirror/blob/master/modules/README.md>”.

Module APIs:

Further into the development of this project, once the modules were decided, certain APIs were needed to provide the data for certain modules. The weather, cryptocurrency, and Dublin bus modules all needed APIs to gather up-to-date information. Whilst the news ticker needed RSS feeds for certain news organisations. These were all relatively easy to find, with the weather one being the hardest as it was not just up-to-date data that was required, but also a weekly forecast.

Camera:

In order to enable the functionality of the facial recognition, a high-quality camera would be needed, as the clearer the pictures the better chance of high-quality recognition.

Researching this led me to the Logitech C920 HD Pro, which is considered a very good quality USB webcam that is utilised a lot by the online content creating community and has a good, and reliable reputation. Wi-Fi cameras were considered due to the neater look it would provide, but were decided against due to cost limitations and unnecessary additional coding features.

2.3 System Functions

Mirror Configuration & Interface:

The interface for PiMirror is supplied by MagicMirror as mentioned above. It allows for custom developed modules to be displayed and connected to the provided backend system. The modules are selected via a "config.js" file, but as PiMirror is designed using multiple users, a new user-specific config file is created when the user first saves their module choice with the format: "(*user email address*)config.js".

Modules:

Six custom modules have been developed for PiMirror:

1) Weather Forecast:

Provides the user with the current time, date, temperature, wind speed, and time of sunset/sunrise. The location for the weather is entered by the user in the app. The user must create an account with <https://openweathermap.org/>. This will generate an API key that they must enter when first configuring their app, if they wish to use this module.

2) Dublin Bus:

Provides the user with up-to-date bus times for up to three selected bus stops. These are selected by the user from the app.

3) Gmail:

Provides the user with the sender and subject line of their last five emails for whichever address they logged into the app with.

4) Google Calendar:

Provides the user with a list of their next ten upcoming events on their Google Calendar. The user must either make their calendar public, or enter their private ".ics" file path into the app.

5) Cryptocurrency:

Provides the user with a visual representation of up to ten cryptocurrencies, with their respective changes in the last hour, twenty-four hours, and seven days. There is also a graph of the weekly change. The currencies to be shown are chosen from the app.

6) News Headlines:

Provides the user with an up-to-date feed of news headlines from their selected news agency. The user has a choice of three networks to choose, these are selected from the app.

Android Application:

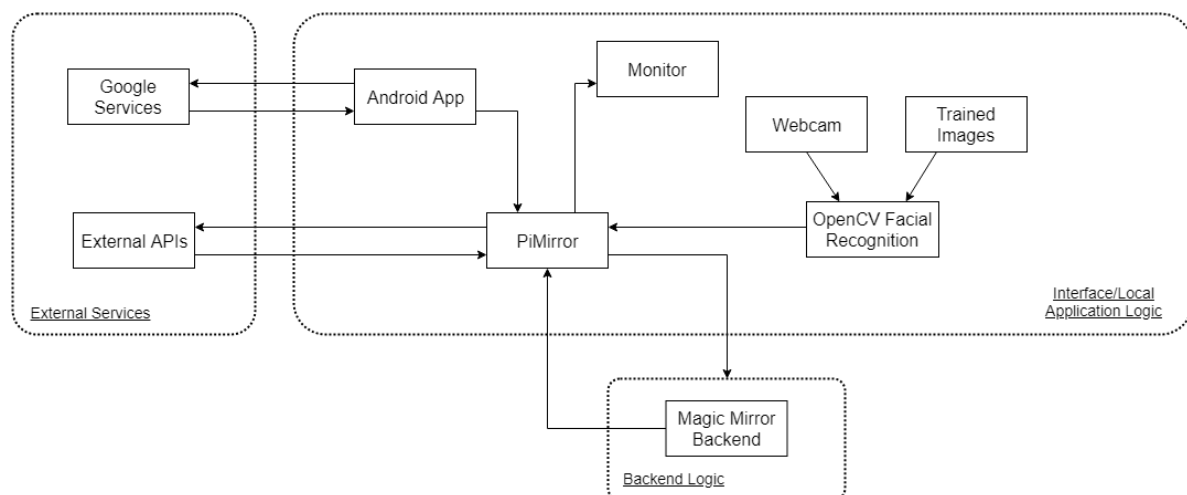
The android application allows users to login using their Gmail account, and select which modules they wish to display, in which position on the mirror, and what customisation that is module-specific they wish to show. E.g. the currencies they wish to track in the cryptocurrency module. They must provide the IP address and SSH password for their pi before connecting. They can also turn mirror interface on/off directly from the app.

Facial Recognition:

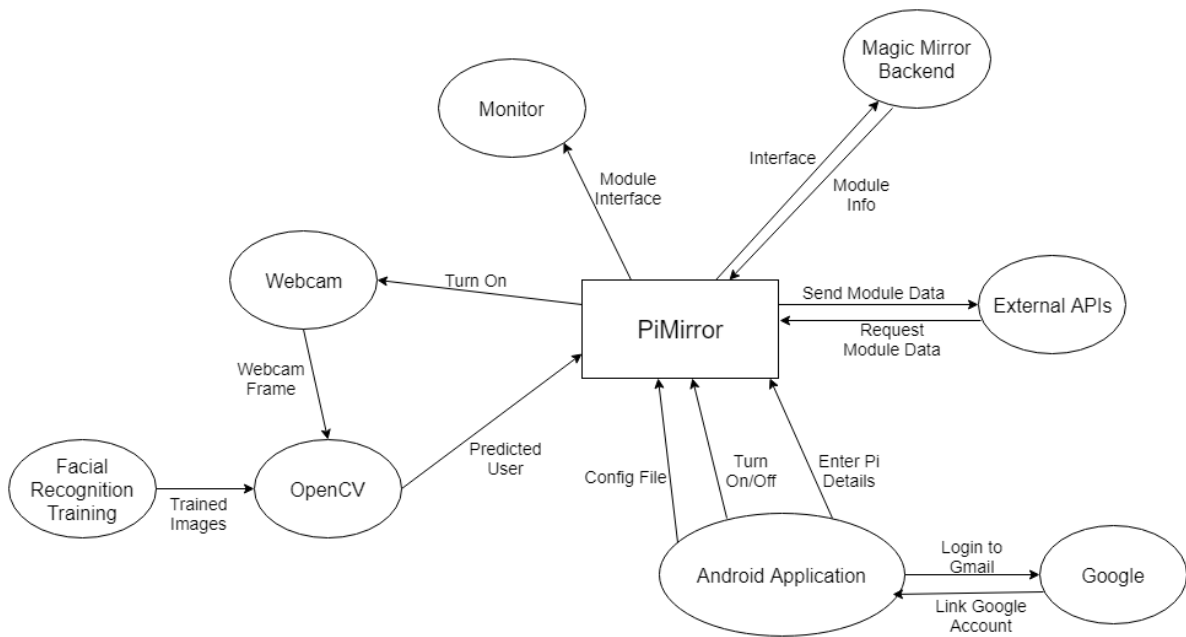
The facial recognition is used to switch users depending on who is currently in-front of the mirror. When a user whose face has been trained on the system stands in-front of the mirror they will be detected and their last saved module configuration from the app will be loaded. After a certain period of time, if no user is detected, the interface is shutdown. This is to help protect the user's privacy, and allows for complete hands-free start up and shutdown of PiMirror.

3. Design

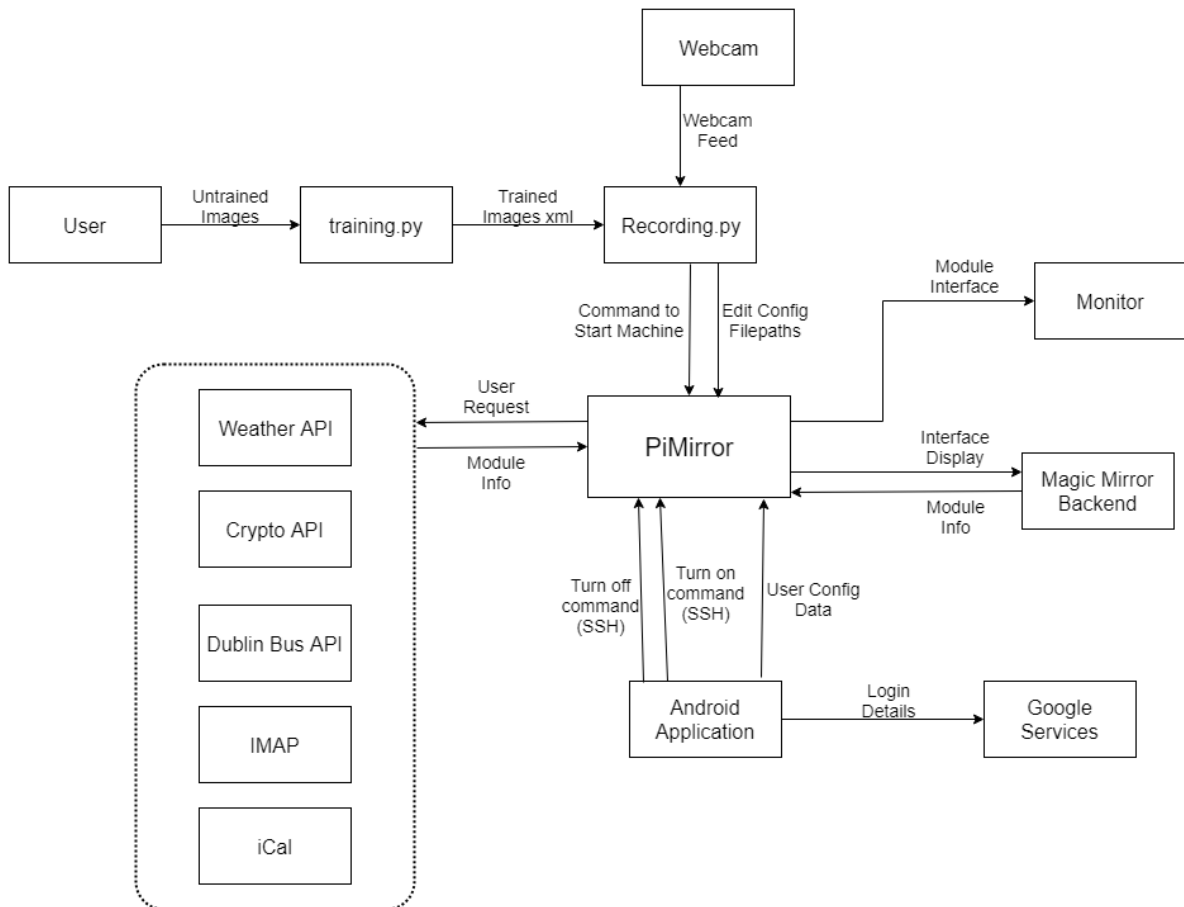
3.1 Component Diagram



3.2 Context Design

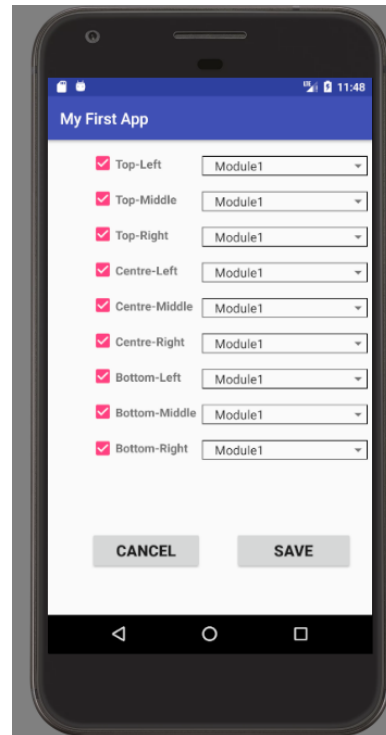
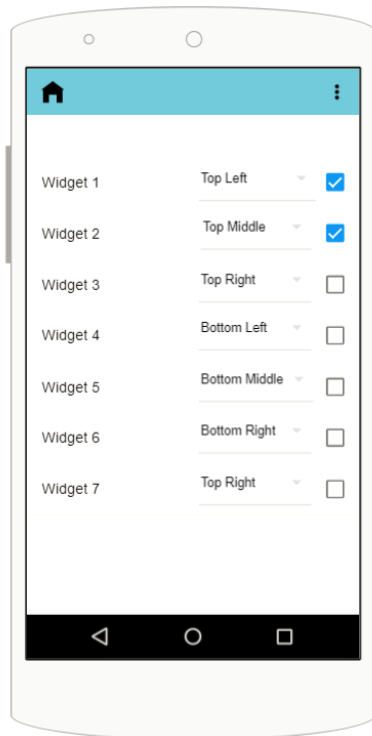


3.3 Data Flow Diagram



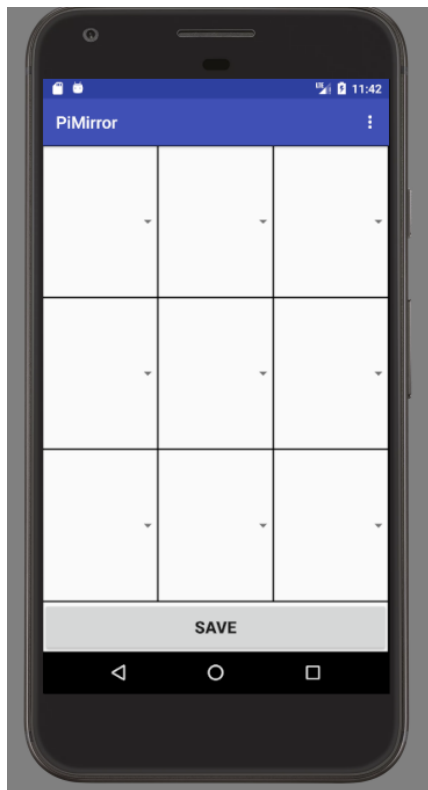
3.4 Android UI Design

The design for the home page of the Android application undertook a few variations through the course of the development project. Originally, as can be seen from this image taken from the functional specification, my intended design was as seen on the left, below. Therefore, when the development phase actually began I attempted to implement this design. Which led me to the UI shown on the right.



However, during one meeting with my supervisor I was asked the simple question: “With no limitations, how would your ideal app be designed?”, to which I answered that I would prefer a grid pattern on screen that is representative of each of the six sections of the PiMirror interface itself. After this encouragement to change my design I quickly developed a new version of the UI, representing exactly this, see below. After this was developed, I undertook a UI testing process, whereby I asked three fellow students which of the two designs they preferred. Each one responded with the new, changed design, stating that it seemed much more intuitive, albeit perhaps not quite as pleasing on the eye as the old version. With this feedback, I decided to abandon the old UI and continue on with the new one.

Eventually, I discovered that due to limitations of the MagicMirror software itself, the middle row of the 3x3 grid was unusable, and therefore the final UI is shown on the right, below.



Also, when designing the application, I always took into major consideration, reducing the number of “clicks” to each potential end goal. The modules being selectable and saveable from the home screen, and the rest of the functionality located one tap away in the top-right corner menu makes this as effective as possible.

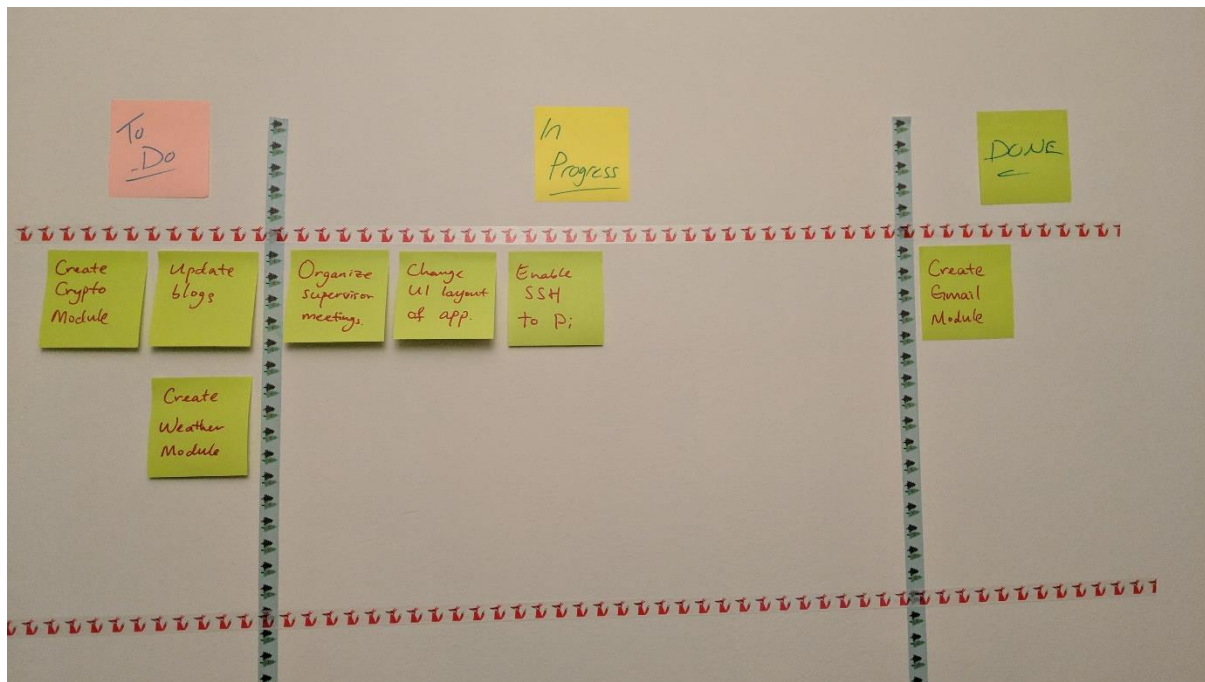
3.5 Process Management

The methodology used for this project was largely agile and followed the main ideas of Kanban. Kanban has three basic principles:

1. Visualise
2. Limit Work in Process (WIP)
3. Manage Flow

The visualisation aspect was done by simply using sticky notes and pasting them on a work board. This set-up is shown in the below image, where there is a “to-do”, “in progress”, and “done” section. This board represents a two-week period where I would assign a number of tasks that were required to be completed. This helped to keep track more efficiently of what was to be done.

Limiting WIP meant to limit the number of tasks that are being currently worked on, i.e. the “in progress” section. Using this methodology prevented multiple tasks all being worked on simultaneously, which hinders the efficiency of completing those tasks. Instead the focus is put on an individual task, and it is therefore completed more quickly due to more focus and concentration being placed on a single task or problem. This helped tremendously to reduce the chances of becoming overworked or overwhelmed at the multiple facets of this project.



4. Implementation

The implementation of this project's functionality is split between three major components:

1. The Facial Recognition Python Code:

- a. Training of data
- b. Predicting data
- c. Launching the mirror

2. The PiMirror modules:

- a. Weather Forecast
- b. Gmail
- c. Google Calendar
- d. Cryptocurrency
- e. Dublin Bus
- f. News Headlines
- g. Configuration Files

3. The Android Application:

- a. Google Sign-In
- b. Connecting to the pi
- c. Selecting modules
- d. Writing configuration files
- e. Saving application state

4.1 Facial Recognition

Training of data:

Training the user's pictures for use with the facial recognition was initially done on the Raspberry Pi, but due to speed and performance issues was quickly moved to being executed from a Windows laptop. Four main users were trained on the system, two of them were people known to the developer – including the developer himself, and the other two

were of more famous people, Elvis Presley and Ramiz Raja. These images for the famous people were taken from online, whereas the others were pictures taken from a combination of smart phones and the webcam to be used with the PiMirror. Each person's images are stored in separate folders so they can be correctly labelled when training the data.

Training the data itself was done using OpenCV and their Local Binary Patterns Histograms (LBPH) methodology. This was explained in more detail in the research section. The python code used to train the file takes all the images from each folder and depending on the folder number, will assign a label to each of the pictures in that folder. The labels to be applied are listed at the top of the folder.

```
# Existing subject names
subjects = ["", "kieranturgoose@gmail.com", "fourthyearemail@gmail.com", "Elvis Presley", "Ramiz Raja"]
```

The two main users are given emails as their names because this will be what they use with the app and configuration files on the pi to identify themselves without having to add additional profile information about themselves. Before applying the labels, the faces are found and highlighted from each image using "lbpcascade_frontalface.xml", which is again from OpenCV. This allows the face itself to be trained, not the additional details from the image.

When all the labels have been assigned to the images, the OpenCV function "LBPHFaceRecognizer_create()" is used to create the data to be trained and it is then trained before the result is written to an xml file that we must then export to be used on the Raspberry Pi with the actual facial recognition software.

```
# Train on pre-loaded images
print ("Training...")
faces, labels = prepare_training_data("training-data")
face_recognizer = cv2.face.LBPHFaceRecognizer_create()
face_recognizer.train(faces, np.array(labels))
print ("Training Done!")
face_recognizer.write("face_model.xml")
```

Predicting of data:

The actual prediction of users comes in real-time on the Raspberry Pi, with the USB webcam used as the input. The webcam continually feeds frames from its video stream into the "predict" method which uses the same "lbpcascade_frontalface.xml" method to find the face within the frame. The face then has a box drawn around it and a prediction is attempted.

```
# Used for face recog
def predict_detect(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier('/home/pi/opencv/data/lbpcascades/lbpcascade_frontalface.xml')
    faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 15)

    if (len(faces) == 0):
        return None, None

    (x, y, w, h) = faces[0]
    return gray[y:y+w, x:x+h], faces[0]
```

As the prediction is made a confidence is also calculated along with the label. If the confidence of the person is below 100 – this is mainly for demo reasons, most confidence values ranged between 40 and 60 – then a label of that user's name is shown and the process of deciding whether to start or stop the mirror begins.

```

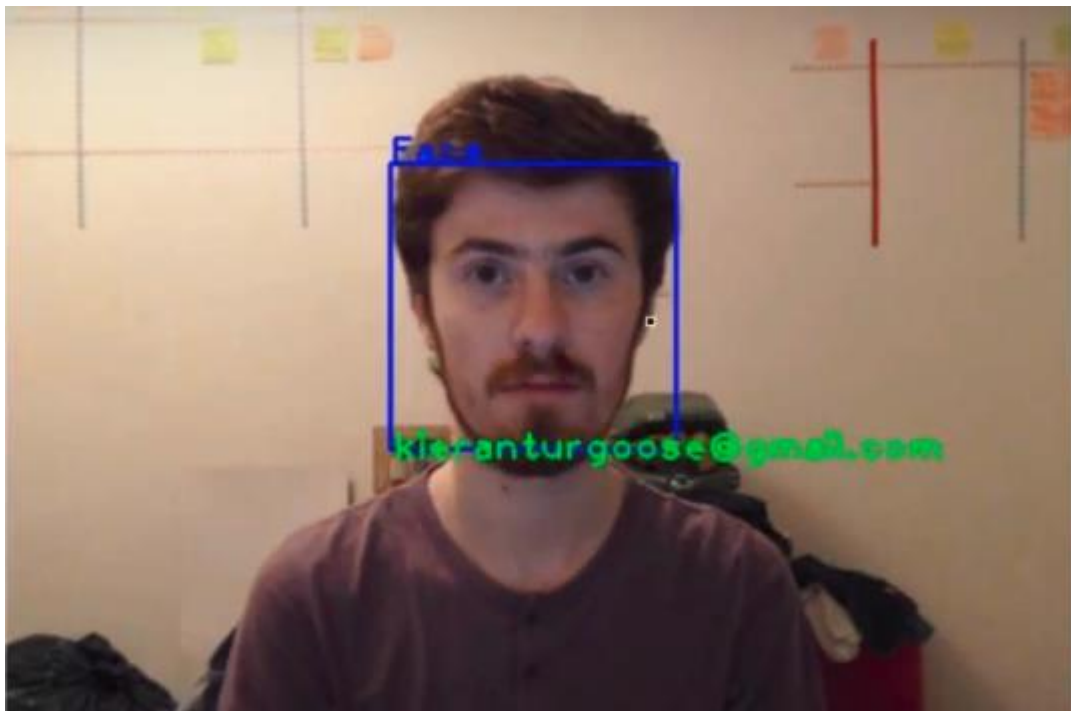
# Make predictions
def predict(img):
    global name, prevName, faceCnt, noFaceCnt, changeCnt
    face, box = predict_detect(img)

    if face is None:
        noFaceCnt += 1
        return None

    face_recognizer = cv2.face.LBPHFaceRecognizer_create()
    # Load trained model
    face_recognizer.read("face_model.xml")
    label, confidence = face_recognizer.predict(face)
    name = subjects[label]
    # Draw box around predicted face
    (x, y, w, h) = box

    if confidence < 100:
        - - -

```



Launching the mirror:

After the prediction has been made and the user identified, a series of counts decide what happens next. Two variables “name”, and “prevName”, are used to decipher if the person standing in front of the mirror is the same user who is currently logged in. For example, if User 1 had previously used the PiMirror and had since turned it off and left, then User 2 would be recognised as a different user. When User 2 has been correctly identified 10 times without fail, then the “logged in” user changes to User 2, and their PiMirror configuration will load. The variables “name” and “prevName” are related to User 2 and User 1 respectively in this scenario.

```

#If same person stood in front of mirror
if name == prevName:
    faceCnt += 1
    if faceCnt > 5:
        changeCnt = 0
else:
    #Increment change count if user is different to current "logged in" user
    changeCnt += 1

    #If person is different 10 consecutive times, turn off the mirror and change "logged in" user
    if changeCnt == 10:
        faceCnt = 0
        prevName = name
        changeCnt = 0
        try:
            thread = Thread(target = stop_mirror())
            thread.start()
            thread.join()
        except RuntimeError as e:
            print(e)

```

Starting the PiMirror consists of changing the configuration filepath that is being pointed to by the MagicMirror software, and then starting the script which executes the loading of the mirror interface.

```

def start_mirror():
    #Change the config/.js files to the correct user
    os.system("sed -i -e 's#config/config.*js#config/config' + name + ".js#g' /home/pi/MagicMirror/js/app.js")
    os.system("sed -i -e 's#config/config.*js#config/config' + name + ".js#g' /home/pi/MagicMirror/js/server.js")
    os.system('lxterminal -e sh ~/mm.sh')

```

If there is either no face detected in frame, or the face detected is not a recognised user, a “noFaceCnt” will increment. Once this reaches a count of 40 any PiMirror interface that is loaded will be terminated. The count in a real-life scenario would ideally be lower but this was optimal for demo purposes. This is to allow privacy from prying eyes if a user forgets to manually turn off their mirror, and also for ease-of-use, the user can simply walk away from the PiMirror without having to worry about turning anything off as it is automatically disabled for them.

```

elif noFaceCnt == 40:
    faceCnt = 0
    #Stop the mirror
    try:
        thread = Thread(target = stop_mirror())
        thread.start()
        thread.join()
    except RuntimeError as e:
        print(e)

```

4.2 MagicMirror Modules

PiMirror consists of six developed modules: Weather Forecast; Gmail; Google Calendar; Cryptocurrency; Dublin Bus; and News Headlines. Each of these is placed within the “modules” folder of the “RaspberryPi_MirrorCode” folder. The main file is named “’moduleName’.js”, and there are often “node_helper.js” files and any images to be used are stored in a “public” folder.

4.2.1 Core Module File: ModuleName.js:

The core of each moduleName.js file consists of the following essentials for each module to function.

Register Module:

```
Module.register("modulename",{});
```

Defaults:

Any properties defined in the defaults object are merged with the configuration for that particular module as defined in the user's configuration file. This is where the default properties for each module is listed, if their value is not edited by the user in the configuration file. Below we can see that for the cryptocurrency module, graphs are shown as default.

```
defaults: {  
  displayType: 'graphs',  
},
```

Start:

This method is called when all modules are loaded, the "dom" object has not yet been created at this point so nothing is displayed, but any additional module properties are defined here. Below we see an example from the Dublin Bus module, where the method that adds stops to be sent to the API is called, taking the stop information the user has provided from the configuration file.

```
start: function () {  
  Log.log("Starting module: " + this.name);  
  for (var s in this.config.stops) {  
    this.addStop(this.config.stops[s]);  
  }  
  
  this.stopData = {};  
  this.loaded = false;  
},
```

GetDom:

This function returns a DOM object. Whenever the MagicMirror needs to update information on screen – either from start, or on a timed refresh – the system calls this getDom() method. This method contains the formatting of the information that is to be displayed on screen. Below is the getDom method from the News module. The content's format, colour, brightness, size and actual content can all be defined here.


```

//Override dom generator
getDom: function() {
    var wrapper = document.createElement("div");

    //Reset the activeItem
    if(this.activeItem >= this.newsItems.length){
        this.activeItem = 0;
    }

    //If there's content to be shown
    if(this.newsItems.length > 0){

        //Get the news source and the time published
        var source = document.createElement("div");
        source.className = "light medium";
        source.innerHTML = this.newsItems[this.activeItem].sourceTitle + ", ";

        var time = document.createElement("span");
        time.className = "light small dimmed";
        time.innerHTML = moment(new Date(this.newsItems[this.activeItem].pubdate)).fromNow() + ":";
        source.appendChild(time);

        wrapper.appendChild(source);

        //Get the headline for the news story
        var headline = document.createElement("div");
        headline.className = "bright light";
        headline.innerHTML = this.newsItems[this.activeItem].title;
        wrapper.appendChild(headline);
    }
    else{
        wrapper.innerHTML = this.translate("There are no headlines available from your sources. Please check their documentation");
        wrapper.className = "small dimmed";
    }

    return wrapper;
},

```

Socket Notification Received:

When using a node_helper file to accompany your main file, the node_helper can help to send your module notifications. This method contains two arguments: the notification identifier, and the payload. Below is an example from the Dublin Bus module, where a notification has been received containing data on all the bus stops that have been returned from the Dublin Bus API. However, as shown, if the notification received was an error fetching the data then the error is logged accordingly. Once this data has been successfully received the Dom is updated, i.e. the screen is refreshed.

```

socketNotificationReceived: function (notification, payload) {
    if (notification === "RTPI_EVENTS") {
        this.stopData[payload.stopId] = payload.events;
        this.loaded = true;
    }
    else if (notification === "FETCH_ERROR") {
        Log.error("DublinRTPI Error. Could not fetch stop: " + payload.stopId);
    }
    else {
        Log.log("DublinRTPI received an unknown socket notification: " + notification);
    }

    this.updateDom(2000);
},

```

4.2.2 The Node Helper: node_helper.js:

The node helper is a Node.js script that is able to support the modules through backend tasks, this included collecting data from API requests. Only one node helper is required for each module, and some may not require one at all. Node helpers, like the main module.js file, can send and receive socket notifications.

Socket Notification Received & Sent:

The node helper can receive notifications from the modules, using the same two arguments as above, the notification identifier, and the payload. They can also send socket notifications too, with these same two arguments. An example of this is shown below from the cryptocurrency module.

Firstly, the crypto.js file sends a socket notification with the message “get_ticker”, and the API url as a payload.

```
getTicker: function() {  
    var url = 'https://api.coinmarketcap.com/v1/ticker/?convert=EUR&limit=100'  
    this.sendSocketNotification('get_ticker', url)  
},
```

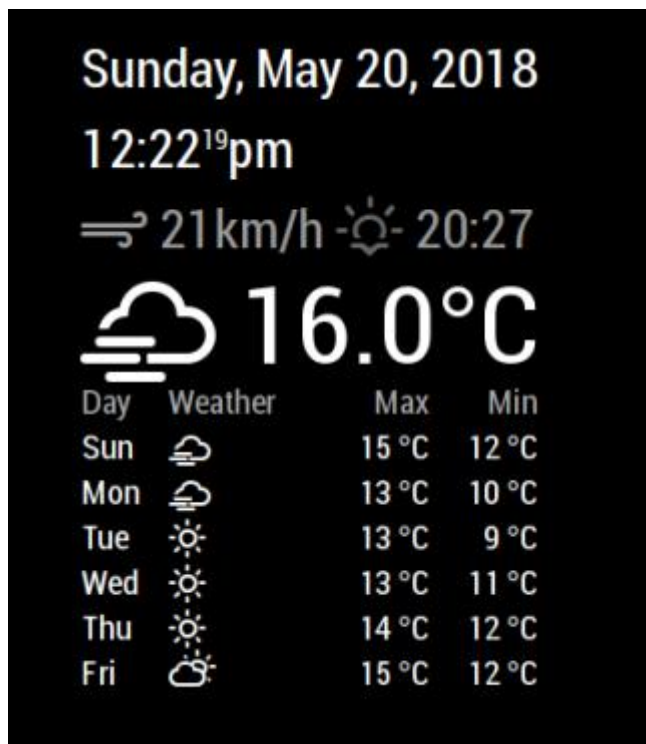
In the node_helper file, the notification is received and the “getTickers” method is called. This requests information from the API url and once received without error, sends a socket notification of its own with the message “got_result” and the payload being a JSON parse of the returned data.

```
socketNotificationReceived: function (notification, payload) {  
    if (notification === 'get_ticker') {  
        this.getTickers(payload)  
    }  
},  
  
getTickers: function (url) {  
    var self = this  
    request({url: url, method: 'GET'}, function (error, response, body) {  
        if (!error && response.statusCode == 200) {  
            self.sendSocketNotification('got_result', JSON.parse(body))  
        }  
    })  
}
```

Finally, back in the crypto.js file a socket notification is received with this “got_result” message. This allows the data returned from the API to be used and shown on screen.

```
socketNotificationReceived: function(notification, payload) {  
    if (notification === 'got_result') {  
        this.result = this.getCurrencies(this.config.currency, payload)  
        this.updateDom()  
    }  
},
```

4.2.3 Weather Forecast:



This module is designed to tell the user the current date and time, the current wind speed, and the time of sunrise/sunset; as well as supplying a five-day forecast for the coming days. The API that was used for receiving the weather information was from ["https://openweathermap.org/api"](https://openweathermap.org/api) and actually consisted of two separate APIs, one for the current weather and one for the five-day forecast. The one downside to this particular API is that a free account was required to access it, and therefore, the user also requires an account and an accompanying API key.

Due to the nature of this being two separate APIs, two separate requests for data were required. This is shown below.

```

var weatherRequest = new XMLHttpRequest();
weatherRequest.open("GET", currentUrl, true);
weatherRequest.onreadystatechange = function() {
    if (this.readyState === 4) {
        if (this.status === 200) {
            self.processCurrentWeather(JSON.parse(this.response));
        }
        else if (this.status === 401) {
            self.updateDom(self.config.animationSpeed);
            retry = true;
            Log.error("WeatherForecast: Incorrect APPID.");
        }
        else {
            Log.error("WeatherForecast: Could not load weather.");
        }

        if (retry) {
            self.scheduleUpdate((self.loaded) ? -1 : 2500);
        }
    }
};
weatherRequest.send();

//Request for 5 day forecast

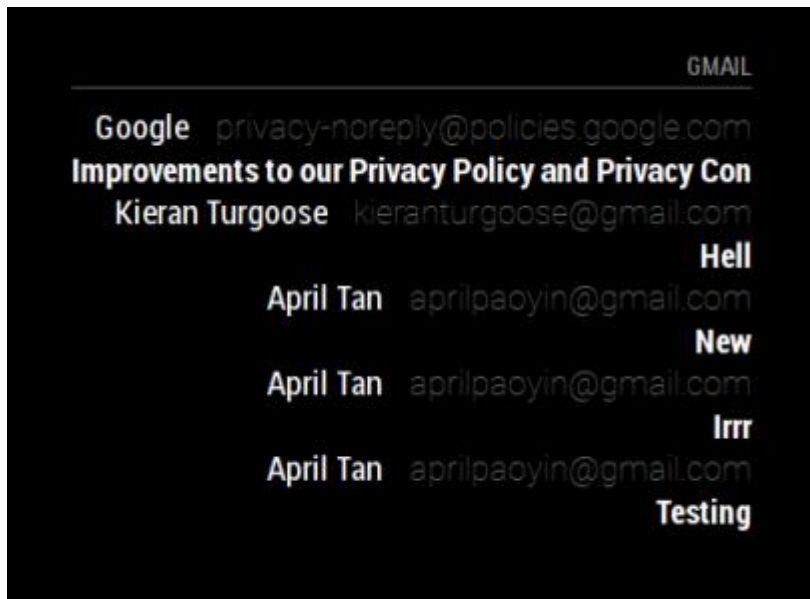
var forecastUrl = "https://api.openweathermap.org/data/2.5/forecast" + this.getParams();

var forecastRequest = new XMLHttpRequest();
forecastRequest.open("GET", forecastUrl, true);
forecastRequest.onreadystatechange = function() {
    if (this.readyState === 4) {
        if (this.status === 200) {
            self.processWeather(JSON.parse(this.response));
        }
        else if (this.status === 401) {
            self.updateDom(self.config.animationSpeed);
            retry = true;
            Log.error("WeatherForecast: Incorrect APPID.");
        }
        else {
            Log.error("WeatherForecast: Could not load weather.");
        }

        if (retry) {
            self.scheduleUpdate((self.loaded) ? -1 : 2500);
        }
    }
};
forecastRequest.send();

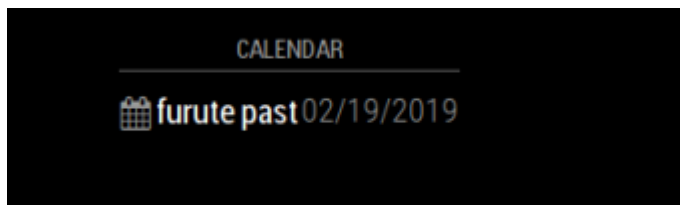
```

4.2.4 Gmail:



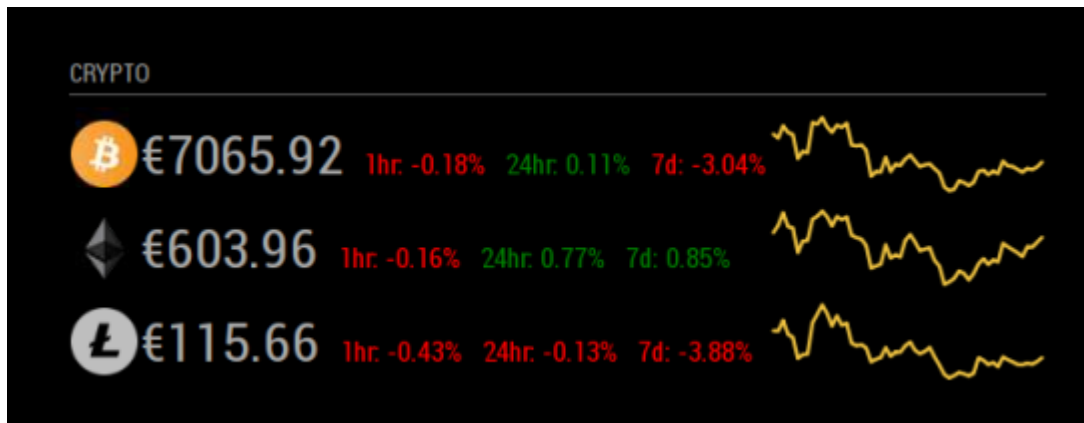
The Gmail module is designed to show the user their last five unread emails sent to the Gmail account connected to their PiMirror. To implement this, “mailparser” and “imap” are utilised to pull the emails’ details and then parse them into useable data.

4.2.5 Google Calendar:



The Google Calendar module is designed to show the user their next five events that are scheduled on their chosen calendar. In order to display the calendar, the “calendar.ics” filepath is needed from the user. However, if their profile has been made public, the “calendar.ics” filepath can be automatically generated and displayed for the account they use for the PiMirror. “ical.js” is used to help fetch the calendar information.

4.2.5 Cryptocurrency:



The Cryptocurrency module is designed to provide the user with up-to-date information regarding the fluctuations in market value of cryptocurrencies. There are ten available currencies that the user can track: Bitcoin; Bitcoin-cash; Cardano; Eos; Ethereum; Iota; Litecoin; Neo; Ripple; and Stellar. These were chosen as they were the current top ten cryptocurrencies with relation to market share as of creating this module as stated by <https://coinmarketcap.com/>.



The API used for this module was from <https://api.coinmarketcap.com>. The request for this API was shown above in the node helper section. Due to cryptocurrencies' fluctuations often being shown in graph form, the default view was made as above. However, if the user so wishes they can disable graphs from their app, which will instead display the module in a table format as shown below. This is particularly useful if the user wishes to keep track of a large number of currencies, but does not wish for the module to take up so much additional space because of this.

The screenshot shows a dark-themed interface with the word 'CRYPTO' at the top. Below it, a table displays the prices and percentage changes for Bitcoin, Ethereum, and Litecoin. The table has four columns: Currency, Price, 1Hour %, 24Hour %, and 7Days %.

Currency	Price	1Hour %	24Hour %	7Days %
Bitcoin	€7065.68	-0.18%	0.09%	-3.06%
Ethereum	€603.41	-0.25%	0.63%	0.69%
Litecoin	€115.67	-0.4%	-0.14%	-3.89%

4.2.6 Dublin Bus:



DUBLIN BUS		
	Stop Bus	Route Time
	195 140	IKEA 1 min
	195 83	Charlestown 8 min
	195 9	Charlestown 13 min
	195 140	IKEA 14 min
	195 4	Harristown 27 min
	195 9	Charlestown 29 min
	195 140	IKEA 32 min
	195 83A	Charlestown 34 min
	195 9	Charlestown 42 min
	195 140	IKEA 52 min

The Dublin Bus module is designed so that the user can enter up to three stop numbers from the app, and the corresponding stop information will be displayed in real-time on the PiMirror. A maximum of ten buses are shown so as to mitigate too much space being taken. The module shows the stop number, the bus route number, the end destination, and the time until arrival at that stop.

The API that is used for this module is from <https://data.dublinked.ie>. It allows for the same information as is used on the official Dublin Bus app, and at their stops, to be displayed. But with this added benefit of being able to track multiple stops at once.

4.2.7 News Headlines:



The News module is designed to allow the user to keep up-to-date with news headlines from around the world, with headlines being displayed on the screen. The three selected news sources were chosen with variety in mind, there is a local news outlet (RTE); a majority British news outlet (Sky News); and an overseas news outlet (New York Times).

The feeds for each of these outlets are from the following sources, respectively:

“<https://www.rte.ie/news/rss/news-headlines.xml>”,

“<http://feeds.skynews.com/feeds/rss/home.xml>”,

“<http://www.nytimes.com/services/xml/rss/nyt/HomePage.xml>”.

4.2.8 Configuration Files:

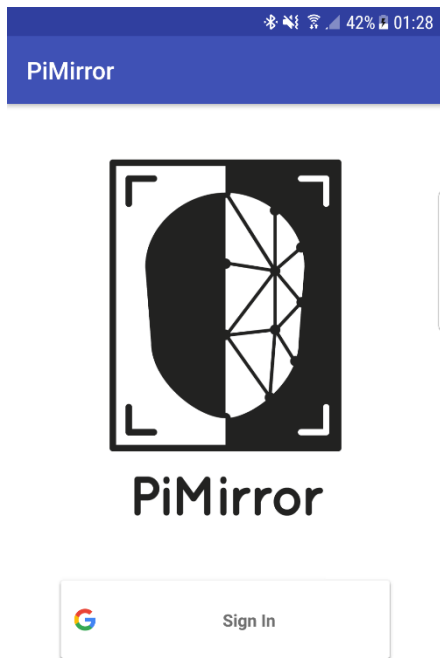
```
module: "WeatherForecast",
position: "top_left",
config: {
  location: "Dublin",
  appid: "35074db805265bffa7ff4288020f3b12d",
}
},
{
  module: "GoogleCalendar",
  position: "top_center",
  header: "Calendar",
  config: {
    url: "https://calendar.google.com/calendar/ical/fourthyearemail%40gmail.com/public/basic.ics",
  }
},
{
  module: "Gmail",
  position: "top_right",
  header: "Gmail",
  config: {
    user: "fourthyearemail@gmail.com",
    password: "Tadcaster1",
  }
},
{
  module: "Crypto",
  position: "bottom_left",
  header: "Crypto ",
  config: {
    currency: ["bitcoin", "ethereum", "litecoin"],
    displayType: "default"
  }
},
{
  module: "News",
  position: "bottom_bar",
  config: {
    feeds: [
      {
        source: "RTE",
        url: "https://www.rte.ie/news/rss/news-headlines.xml",
      }
    ]
  }
},
{
  module: "DublinBus",
  position: "bottom_right",
  header: "Dublin Bus",
  config: {
    stops: [
      { id: "195" },
    ]
  }
}
```

The configuration file is the location of the customisation that each user supplies from the app. As seen above, the list of each module and their respective positions are shown, and in the config section is any additional details that are needed for the module to function. This entire file is created from the app upon hitting the save button. Each user has their own configuration file that is named after the email account they have used to connect to PiMirror.

4.3 Android Application

Google Sign-In:

When the user first opens the application, they are met with the login screen, shown below. The app has Google sign-in functionality, this benefits the majority of users as most people would already own a Gmail account. The user can choose to sign-in with any account already logged in or add a new account from this screen.



Implementing this sign-in uses “GoogleSignInOptions” and requires the storage of the user’s email account as this will be used as their username for the system. It is required that the email address used is the same as supplied for the facial recognition. Signing in takes the user to the app’s home page.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

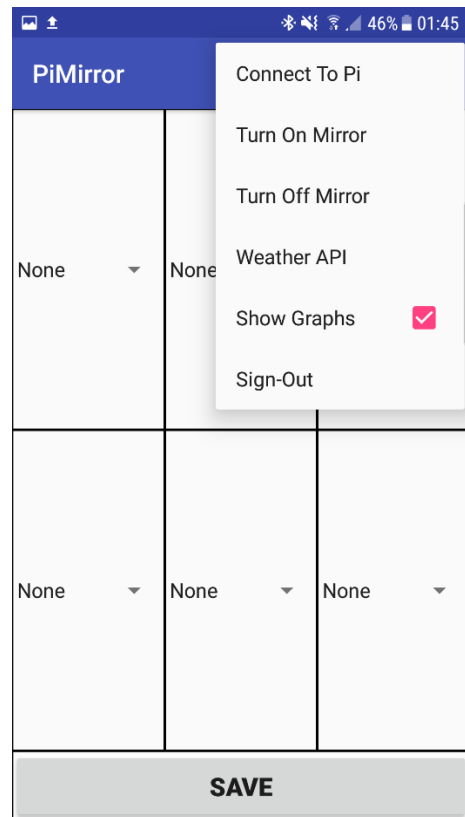
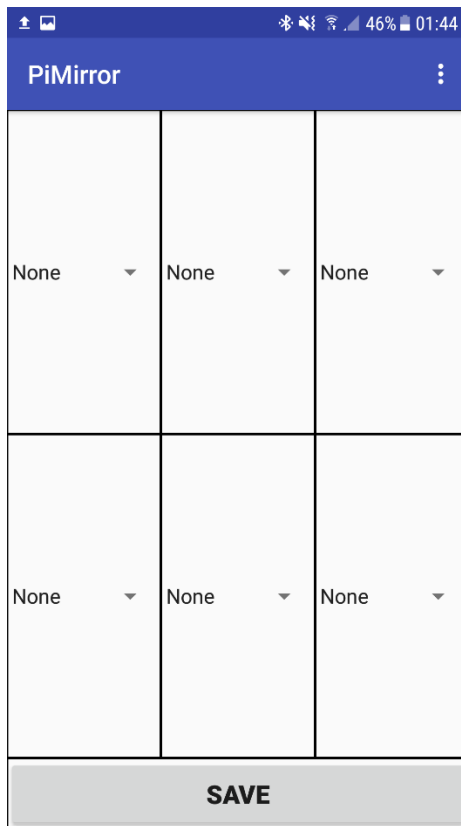
    signInButton = (com.google.android.gms.common.SignInButton) findViewById(R.id.signInButton);
    signInButton.setOnClickListener(this);

    // Configure sign-in to request the user's ID, email address, and basic profile. ID and basic
    GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestEmail()
        .build();
    // Build a GoogleSignInClient with the options specified by gso.
    mGoogleSignInClient = GoogleSignIn.getClient(this, gso);
}
```

If the app is closed and restarted without the user logging out, then the app will register them as still logged in and will therefore open straight onto the home page. This is done by using the “onStart” method and checking if the last signed in account was valid.

```
protected void onStart() {
    super.onStart();
    account = GoogleSignIn.getLastSignedInAccount(this);
    updateUI(account);
}
```


Home Screen:



The home screen of the application is where the user can access a variety of customisable options for their PiMirror. The options button in the top-right corner of the screen opens up a menu of the different settings and functions of the app. From here the user can enter their Raspberry Pi's login information; Turn on their configured PiMirror; Turn off the PiMirror; Enter their weather API key for the Weather Forecast module; Decide whether they want to display graphs for the cryptocurrency module; and Sign-Out from the app. This was done through the use of the Options Menu button type in Android, whereby the menu content is initialised, and also the functions that occur on click are called.

```

//option menu initiation
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.connect_to_pi, menu);
    inflater.inflate(R.menu.turn_on_mirror, menu);
    inflater.inflate(R.menu.turn_off_mirror, menu);
    inflater.inflate(R.menu.weather_api, menu);
    inflater.inflate(R.menu.show_graphs, menu);
    inflater.inflate(R.menu.signout, menu);
    MenuItem menuItem = (MenuItem) menu.findItem(R.id.showGraphs);
    if(moduleInfo.getBoolean("ShowGraphs", true) == true)
        menuItem.setChecked(true);
    else
        menuItem.setChecked(false);

    return super.onCreateOptionsMenu(menu);
}

```

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    SharedPreferences.Editor moduleEdit = moduleInfo.edit();
    switch (item.getItemId()) {
        case R.id.connectToPi:
            startActivity(new Intent(this, SetSshIdActivity.class));
            return true;

        case R.id.turnOnMirror:
            turnOnMirror();
            return true;

        case R.id.turnOffMirror:
            turnOffMirror();
            return true;

        case R.id.weatherAPI:
            startActivity(new Intent(this, SetWeatherAPIActivity.class));
            return true;

        case R.id.showGraphs:
            if(item.isChecked()){
                item.setChecked(false);
                moduleEdit.putBoolean("ShowGraphs", false);
                moduleEdit.putString("CryptoGraphs", "default");
                moduleEdit.apply();
            }
            else{
                item.setChecked(true);
                moduleEdit.putBoolean("ShowGraphs", true);
                moduleEdit.putString("CryptoGraphs", "graphs");
                moduleEdit.apply();
            }
            return true;

        case R.id.signOut: //sign out user after finishing current activity
            finish();
            signOut();
    }
}

```

Another key feature of the app is located within this home screen, that of the module selector itself. This layout, as shown above, is comprised of a 3x2 grid pattern, with each section relating to their corresponding location on the mirror interface. Each of these sections is a “Spinner”, which is essentially a dropdown menu, which allows the user to select their preferred module for each position.

```

topLeft = (Spinner) findViewById(R.id.topLeft);
topCenter = (Spinner) findViewById(R.id.topCenter);
topRight = (Spinner) findViewById(R.id.topRight);
bottomLeft = (Spinner) findViewById(R.id.bottomLeft);
bottomCenter = (Spinner) findViewById(R.id.bottomCenter);
bottomRight = (Spinner) findViewById(R.id.bottomRight);
Spinner[] spinArray = {topLeft, topCenter, topRight, bottomLeft, bottomCenter, bottomRight};

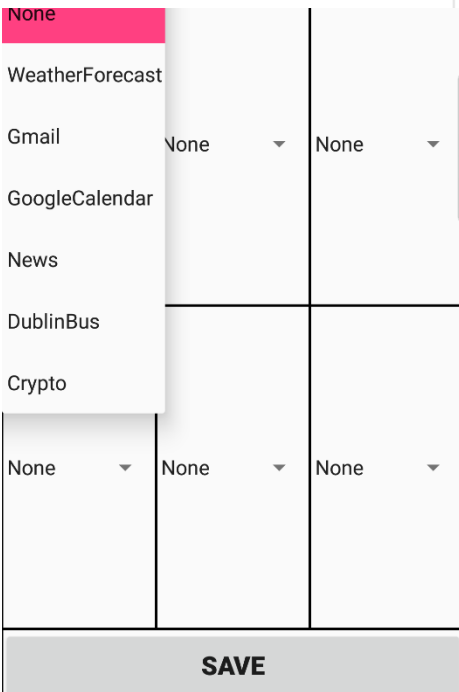
for (int i = 0; i < spinArray.length; i++) {
    createAdapters(spinArray[i], positions[i]);
}

```

```

android:layout_width="match_parent"
android:layout_height="0dp"
android:layout_weight="9"
app:columnCount="3"
app:orientation="horizontal"
app:rowCount="3">
<Spinner
    android:id="@+id/topLeft"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_rowWeight="1"
    app:layout_columnWeight="1"
    android:foreground="@drawable/border" />

```



None		
WeatherForecast		
Gmail	None ▼	None ▼
GoogleCalendar		
News		
DublinBus		
Crypto		
None ▼	None ▼	None ▼
SAVE		

The save button at the bottom of the screen provides the functionality of creating the configuration file that will be sent to the pi, if properly connected, and creating it on the pi via SSH.

```

private void saveModules(){
    gmailPassword = moduleInfo.getString("GmailPassword", "No password available");
    newsSource = moduleInfo.getInt("NewsSource", 0);
    weatherLocation = moduleInfo.getString("WeatherLocation", "No weather location available");
    stopNumbers[0] = moduleInfo.getString("StopNumber1", "");
    stopNumbers[1] = moduleInfo.getString("StopNumber2", "");
    stopNumbers[2] = moduleInfo.getString("StopNumber3", "");
    getCurrencies();
    final String fileName = "config"+userName;
    savedHashMap.clear();
    savedHashMap = positionModuleList;

    final String file = fileWriter(savedHashMap, userName);
    sshFileCreator(file, fileName);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.clear();
    for(String s : savedHashMap.keySet()){
        editor.putString(s, savedHashMap.get(s));
    }
    editor.apply();
    editor.commit();
}

```

Connecting to the pi:

Connecting the application to the Raspberry Pi is done via SSH. More specifically, an imported JAR file called “JSch”, that allows for SSH connection to be made. The user, on first opening the application, has to enter their Raspberry Pi’s IP address, and their personal password for SSH on the pi. Once this is done the user saves this information so that it can be used to send commands to the pi, such as saving configured modules, and turning the mirror on/off.

On clicking of the save button, the app attempts a connection to the pi to ensure that the details given have worked. If this is successful, a toast is shown stating “Connection Successful”, if the connection fails then a toast of “unsuccessful” is shown. This same method of opening a connection with the pi is also used for the other features of the app mentioned above.

```
public void tester(){
    try {
        JSch jsch = new JSch();
        Session session = jsch.getSession("pi", ssh.getString("IP", "NOT WORKED"), 22);
        session.setPassword(ssh.getString("Password", "Password not correct"));
        // Avoid asking for key confirmation
        Properties prop = new Properties();
        prop.put("StrictHostKeyChecking", "no");
        session.setConfig(prop);
        session.connect();
        Success = "Connection Successfully Saved";

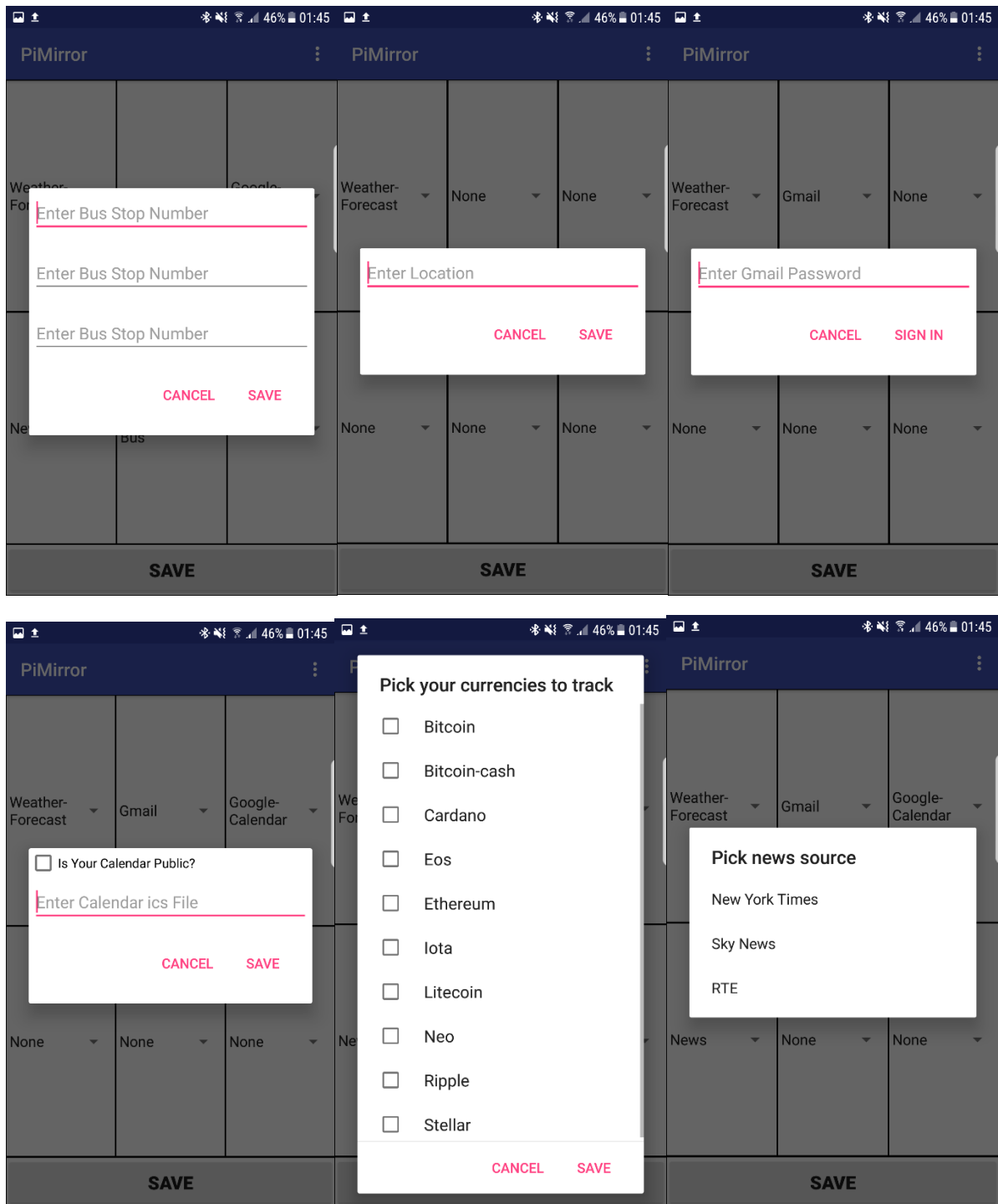
        Channel channelssh = session.openChannel("exec");
        channelssh.setInputStream(null);
        ((ChannelExec) channelssh).setErrStream(System.err);

        channelssh.connect();
        channelssh.disconnect();
        session.disconnect();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Selecting modules:

Selecting modules is done via the Spinners on the home page shown above. Any module can be chosen in any position, although for formatting reasons it is recommended to place the news module in either the top-centre or bottom-centre. Each of the six developed modules require additional information to be provided by the user, so that they can be customised and tailored to the user's needs. Additionally, the Weather Forecast module also requires a one-off API key to be entered via the options menu.

The Dublin Bus module requires the user to enter up to three chosen bus stops to be monitored. The Weather Forecast requires a location to be entered. The Gmail module requires the user to re-enter their Gmail password. The Google Calendar module requires the user to enter their calendar's .ics filepath, however if the user so wishes they can make their calendar public so that it is automatically generated. The Cryptocurrency module requires the user to select up to ten currencies to track, they also have the options to display graphs for the last week of market change. The news module requires the user to select one of three available news sources.



Each of these pop-ups are created via Dialog Fragment extensions, with their selected user inputs being saved and stored for future use. Each of these fragments are only shown when the user is selecting the particular module, if it is not currently saved. If it has been saved in another position already then the values for that module will remain the same. When the modules' configurations are saved, the names and positions of each module are stored in a HashMap so that they can be used to write the configuration file.

```

public class NewsHeadlinesDialogFragment extends DialogFragment{
    public static int newsSource;
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Pick news source")
            .setItems(R.array.news_array, (dialog, which) → {
                // The 'which' argument contains the index position
                // of the selected item
                SharedPreferences.Editor moduleEdit = MainActivity.moduleInfo.edit();
                moduleEdit.putInt("NewsSource", which);
                moduleEdit.apply();
            });
        return builder.create();
    }
}

name.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        if(((String) parent.getItemAtPosition(position)).equals("Gmail") && !map.containsKey("Gmail")) {
            confirmGmailModuleLogin();
        }
        else if(((String) parent.getItemAtPosition(position)).equals("GoogleCalendar") && !map.containsKey("GoogleCalendar")){
            confirmCalendar();
        }
        else if(((String) parent.getItemAtPosition(position)).equals("News") && !map.containsKey("News")){
            confirmNewsSource();
        }
        else if(((String) parent.getItemAtPosition(position)).equals("DublinBus") && !map.containsKey("DublinBus")){
            confirmDublinBusStop();
        }
        else if(((String) parent.getItemAtPosition(position)).equals("WeatherForecast") && !map.containsKey("WeatherForecast")){
            confirmWeatherLocation();
        }
        else if(((String) parent.getItemAtPosition(position)).equals("Crypto") && !map.containsKey("Crypto")){
            confirmCryptoCurrencies();
        }
        positionModuleList.put(modulePos, (String) parent.getItemAtPosition(position));
    }
}

```

Writing configuration files:

This functionality compiles the user's currently selected configuration of modules on the application, including all the additional information they provided, and creates a formatted configuration file from it that will be used to start the PiMirror. When the save button on the home page is clicked, the "fileWriter" function is called, which takes the user's username and the HashMap of saved modules and their positions, and calls specific writer functions depending on which module has been selected.

```

public String fileWriter(HashMap<String, String> positionModuleList, String userName){
    String fullFile = fileStarter();
    for(int i = 0; i < positions.length; i++){
        String moduleName = positionModuleList.get(positions[i]);
        if(positionModuleList.keySet().contains(positions[i])){
            if(moduleName.equals("Gmail")){
                String method = gmailWriter(positions[i], gmailPassword);
                fullFile = fullFile + '\n' + method;
            }

            else if(moduleName.equals("GoogleCalendar")){
                String method = calendarWriter(positions[i]);
                fullFile = fullFile + '\n' + method;
            }

            else if(moduleName.equals("News")){
                String method = newsWriter(positions[i], newsSource);
                fullFile = fullFile + '\n' + method;
            }

            //DublinBus
            else if(moduleName.equals("DublinBus")){
                String method = busWriter(positions[i], stopNumbers);
                fullFile = fullFile + '\n' + method;
            }

            //ForecastWeather
            else if(moduleName.equals("WeatherForecast")){
                String method = forecastWriter(positions[i], weatherLocation);
                fullFile = fullFile + '\n' + method;
            }

            //Crypto
            else if(moduleName.equals("Crypto")){
                String method = cryptoWriter(positions[i], currencies);
                fullFile = fullFile + '\n' + method;
            }
        }
    }
}

```

Since each module on PiMirror takes in different configurations in order for it to function, a standard module writer was not plausible and so a different one was created for each module. E.g. see below for the “newsWriter” method which takes the saved news source index from the user, and selects the correct RSS feed URL to be used on PiMirror, before formatting the file structure, and returning the string value to be added to the “full file” that will be sent to the pi.


```

public static String newsWriter(String position, int newsSource ){
    position = positionCheck(position);
    String title = newsSources[newsSource];
    String url = "";
    if(title.equals("New York Times")){
        url = "http://www.nytimes.com/services/xml/rss/nyt/HomePage.xml";
    }
    else if(title.equals("Sky News")){
        url = "http://feeds.skynews.com/feeds/rss/home.xml";
    }
    else if(title.equals("RTE")){
        url = "https://www.rte.ie/news/rss/news-headlines.xml";
    }
    String method =
        '\t' + "{" +
            '\n' + '\t' + '\t' + "module: \"News\"," +
            '\n' + '\t' + '\t' + "position: \"" + position + "\", " +
            '\n' + '\t' + '\t' + "config: {" +
            '\n' + '\t' + '\t' + '\t' + "feeds: [" +
            '\n' + '\t' + '\t' + '\t' + '\t' + "{" +
            '\n' + '\t' + '\t' + '\t' + '\t' + '\t' + "source: \"" + title + "\", " +
            '\n' + '\t' + '\t' + '\t' + '\t' + '\t' + "url: \"" + url + "\", " +
            '\n' + '\t' + '\t' + '\t' + '\t' + '\t' + "}," +
            '\n' + '\t' + '\t' + '\t' + "]" +
            '\n' + '\t' + '\t' + "}" +
        '\n' + '\t' + "},"
    ;

    return method;
}

```

Saving application state:

All the key values that are used and reused within this app must be stored so that the user does not have to start from scratch each time they open the app. To accomplish this, “SharedPreferences” were utilised. Considering the variety of different data that must be stored, three main types of SharedPreferences are created. “sharedPreferences” stores the user’s configuration of modules and their positions on PiMirror, the username currently logged in is used as the classifier so that it is user-dependant and unique to each user. “ssh” stores the Raspberry Pi’s IP address and password, this is not user-dependant and is standard across all users on the device as they would all be using the same PiMirror. Finally, “moduleInfo” stores all the additional information that the user provides, from the module pop-ups, to the weather API key, this is also user-dependant so that their PiMirror is customised specifically for them.

```

sharedPreferences = getSharedPreferences(userName, Context.MODE_PRIVATE);
ssh = getSharedPreferences("SSHPREFS", Context.MODE_PRIVATE);
moduleInfo = getSharedPreferences(userName+"Modules", Context.MODE_PRIVATE);

```

When the app is reloaded after a closure, the user’s last saved configuration of modules is displayed on the home page, just as the user left it. This is done by creating a HashMap of the “sharedPreferences” as the home page is created, and using this map to initialise the variables shown in the Spinners. If no module is configured for a specific position, “None” is displayed instead.

```

if(sharedPreferences != null) {
    map = (HashMap<String, String>) sharedPreferences.getAll();
    for (String s : map.keySet()) {
        String value = map.get(s);
    }
}

public void createAdapters(Spinner name, final String modulePos){
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.spinner, items);
    name.setAdapter(adapter);
    String selected = map.get(modulePos);
    if(map.get(modulePos)!=null) {
        for (int i = 0; i < items.length; i++) {
            if (selected.equals(items[i])) {
                System.out.println("items[i] = " + items[i]);
                name.setSelection(i);
            }
        }
    }
    else {
        name.setSelection(0);
    }
}

```

4.4 Component Installation

If this project were to go to market as a product, it would be sold with all software installed. Considering this, this is a list of all the major installations and configurations that were made to create this project.

Raspberry Pi Setup:

First Boot:

1. Flash Raspbian onto a MicroSD card
2. Insert SD card then plug in power
3. Boot Pi

Expand Filesystem:

1. `sudo raspi-config`
2. Select "Advanced Options", select "Expand File System", hit "Enter", select "Finish"
3. Select "Yes" when prompted to reboot the pi, or use the command "`sudo reboot`"

Install Dependencies:

1. Updates and upgrades existing packages
 - `sudo apt-get install build-essential cmake pkg-config`
2. Developer tool to configure OpenCV
 - `sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev`
3. Image I/O packages to load image formats
 - `sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev`
 - `sudo apt-get install libxvidcore-dev libx264-dev`
4. Video I/O to read video formats and allow vide streams
 - `sudo apt-get install libgtk2.0-dev`

5. Enables use of OpenCV submodule “highgui” to allow images to be displayed on screen
 - `sudo apt-get install libatlas-base-dev gfortran`
6. Optimises certain OpenCV operations
 - `sudo apt-get install python3.5-dev`

Change Default Python Version:

1. Check to see if Python 3 is available (Should be available)
 - `ls /usr/bin/python*`
2. Check the version (Should be 2.7)
 - `python --version`
3. Change the version on a user basis
 - `alias python='/usr/bin/python3.5'`
4. Source .bash file
 - `. ~/.bashrc`
5. Check version again (Should 3.5)
 - `python --version`

Download OpenCV:

1. Change to home directory
 - `cd ~`
2. Download latest OpenCV
 - `git clone https://github.com/opencv/opencv.git`
3. Download latest OpenCV extra modules
 - `git clone https://github.com/opencv/opencv_contrib.git`
4. Create Build directory
 - `cd ~/opencv`
 - `mkdir build`
 - `cd build`
5. Configure OpenCV
 - `cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local ..`
6. Configure OpenCV Contrib
 - `cmake -DOPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules ~/opencv`
7. Build
 - `make -j4`

Check Installation:

1. Should return Python 3.5
 - `python`
2. Should return CV 3.2.0
 - `import cv2`
 - `cv2.__version__`

MagicMirror:

In the directory of choice enter the following command:

- `bash -c "$(curl -sL https://raw.githubusercontent.com/MichMich/MagicMirror/master/installers/raspberry.sh)"`

5. Problems & Resolutions

OpenCV Installation:

The first problem was encountered when building OpenCV with extra modules from OpenCV_Contrib, this was needed to use the facial recognition aspects of OpenCV. While setting up the libraries needed for computer vision, I cloned the OpenCV_Contrib Git repository into the OpenCV directory which is incorrect. Due to poor documentation, it was difficult to understand the issue and I spent almost a month trying to fix it as it was a crucial part of the project. After a lot of trial-and-error, I found that the issue is fixed by cloning the repository into the same level as the OpenCV directory; meaning OpenCV and OpenCV_Contrib should be folders on the same level. OpenCV can then be built with the extra modules from the Contrib directory. It was a straightforward solution to a poorly documented problem.

Updating Python:

As I am using the latest version of Python for this project, it was required to change the Python version on the Pi to Python 3. Python 3 was needed for certain changes to the face module in OpenCV that required Python 3. The challenge faced here was that the default version for Python is Python 2. This means that each time the Pi is booted, the version will default to 2. The solution was to manually change the .bashrc file. This was done by adding the following line to that file: `alias python='/usr/bin/python3.5'`. I then sourced the bash file to permanently change the version used when python is called.

Facial Recognition:

An important factor of this project was ensuring that the facial recognition was accurate and stable. It took quite a while to fully implement due to the nature of the pictures that were being used. Two of the users that have been trained are from "real-life" (myself and a friend), while the other two were famous people (Elvis Presley and Ramiz Raja). I found it difficult to find reliable pictures that properly trained the data. This was rectified by including larger amounts of photos for the two real-life users that were taken with the same webcam that was to be used for predicting faces. The similar framing, and lighting helped to improve the accuracy of the facial recognition greatly.

Learning various new languages:

Being a largely Java taught student over the past four years, the variety of different languages I had to cover for this project was certainly a challenge. Having only basic knowledge of Python, zero experience using Android Studio – albeit that being Java based, and no experience using Javascript or Node.js, I was consistently learning throughout the course of this project. The solution to this large learning task was to spend time early on in semester one learning the basic of each language through online courses, and watching more specific and detailed YouTube tutorials. Designing in android studio was significantly less of a problem due to the familiarity with the language.

API calling:

Despite having knowledge in quite a variety of different programming aspects, I had not previously had to work with requesting and parsing data from an external API. It took me quite a while to get a handle on how to properly request and parse data from the Node.js code, whilst also linking it correctly to the display of the PiMirror. The solution to this problem was really more of a trial-and-error situation. Although, the MagicMirror developer documentation and the APIs' documentations were quite helpful, they certainly didn't hold your hand. It took many attempts to truly get a grip on developing this functionality.

6. Results

Here I will describe a brief overview of the tests that I carried out and the reasoning behind choosing these types of testing. The full testing results and screenshots are shown in the accompanying testing documentation, located in the documentation folder. Please view this document to see the full results.

6.1 Facial Recognition Accuracy Tests

In order to test the accuracy of the facial recognition code I used, I conducted some tests using the pictures I was using for my two "users", and also some celebrity faces that I found online. These tests were run using the python file "testingFR.py". I trained the images the same way as I had done for the implementation of the project, but instead of using the live webcam feed as the image source to be predicted, I used saved images of each of the users. The people to be tested on the system were: myself, a friend of mine April Tan, Elvis Presley, and Ramiz Raja.

The pictures to be tested were images that were not included in the training set so as to be more concrete in the results. The python code used the trained images to predict the names of each of the given test pictures. I also included some pictures of people who were not trained to ensure that the software did not recognize people falsely who were not intended as users, i.e. Give false positive results.

The results were 100% accurate on the tests carried out and show that the facial recognition code works efficiently and to a pretty good quality on still images. However, the real test is if can run as accurately when running in real-time using the webcam. The only way this can be tested is through checking the output on the screen when the user is stood in front of the mirror and ensuring it is the correct user labelled. Therefore, it is impossible to provide a solid metric for this scenario, but from experience running the software multiple times, it is very solid especially in good lighting.

6.2 Climate/Scenario-based Testing for Facial Recognition Python Code **Climate-Based:**

This set of tests focused on testing different environment conditions when running the facial recognition in real-time, and to see how robust the facial recognition could be when faced with this.

I designed tests to:

1. Recognise Face in a well-lit room.
2. Recognise Face in a dimly-lit room.
3. Recognise two separate users correctly.
4. Ensure untrained users are not recognised.

All of these tests passed and so I am happy with the robustness with which the facial recognition can handle differing conditions and still correctly identify users and non-users.

Scenario-Based:

This set of tests focused on testing the Python code that is running with the facial recognition. These tests were aimed at producing a complete set of possible scenarios that may occur, and to ensure that the system produced the correct output for these scenarios. E.g. If a trained user were to stand in-front of the mirror, does the system correctly identify them and also start the interface correctly with that particular user's configuration?

The designed tests were:

1. Trained user stands in-front of mirror.
2. Untrained user stands in-front of mirror.
3. Trained second user stands in-front of the mirror, with first user's interface loaded.
4. Mirror turns off automatically when no user present.
5. User attempts to restart mirror, after automatic turn off.
6. Second user stands adjacent to first user.

All of these tests passed, and so I am happy that each of the potential scenarios in which this system could be used have been tested and found successful. This proves that the system is well-designed and well implemented to a high-standard, so that any scenario will be handled comfortably without the system failing or producing a drastically wrong outcome.

6.3 Scenario-based Testing for Android Code

This set of tests focused on a similar role to the above Python testing, but instead focused on the Android application functionality. All functionalities that the app is capable of producing was tested, every page was visited, and every button was pressed, to ensure that the correct outcome was produced. A few purposeful errors were made to ensure that they were handled correctly and did not break the application.

The designed tests were:

1. Sign In
2. Connect To Pi – Successfully and Unsuccessfully
3. Turn on Mirror interface with no modules selected
4. Turn off mirror
5. Save the weather API
6. Save modules in each of the six possible positions
7. Sign Out
8. Show previously saved configuration on next Login
9. Reload mirror with a different configuration
10. Turn Mirror On when already in-use
11. Enable/Disable graphs for the cryptocurrency module.

All of these tests passed, and so I am happy that the Android code is well designed and robust enough to handle all scenarios that may occur. To add another level of testing, I installed the app on another phone with a different version of Android, and re-tested all of these tests. This is shown in the testing documentation as there are two results columns, one for Samsung and one for the Google Pixel.

7. Conclusion

7.1 Future Work

The scope for improving this project even further is quite large, considering its nature. However, to greatly improve and evolve, some things would need to change.

Additional Modules:

The no-brainer for future work is the addition of more modules. Some would be brand new, others, slight variations on already existing modules. For example, a module similar to Cryptocurrency could be developed for actual stocks; news updates could be changed to traffic updates, etc. In terms of brand new modules, some ideas that came to mind were: GoogleMaps integration, social media posts, and a music/media player.

I would also like to improve the already existing modules. Add an indefinite list of currencies for the cryptocurrency module rather than the top ten; adding luas and Irish rail information to the Dublin Bus module, as they all run from the same API.

More customisability:

I would like to develop a way for the user to have more freedom with how their PiMirror interface looks. They should be able to perhaps move the modules around more freely than just six grid positions, and choose colour schemes and sizing options too.

Better Security:

Due to it being outside the scope of what I planned for this project, I did not consider the security side of PiMirror. Sending passwords and API keys unencrypted over SSH, and having them displayed in free text in the configuration files is far from ideal. This is something that would certainly be looked into immediately if this project were to go any further into development.

Restful Architecture:

Instead of utilising SSH to create and change the configuration files, and to supply commands to the PiMirror, I would like to develop some sort of architecture where I could properly arm & disarm the system, and edit its features safely and securely.

More Informative Error Messages:

If an error occurs on the PiMirror interface, a specific reason for this is not supplied as of yet (in most cases). I would like to properly manage errors so that a message describing the issue to the user was displayed in the modules place on the interface.

7.2 Summary

On the whole, I am very pleased with the outcome of this project, and I believe that it was a successful endeavour. From a technical standpoint, the intended functionality was completed, and the quality of the facial recognition and module development was more than I had originally thought capable considering the timespan and hardware limitations.

The aspects that were new to me were tackled well and achieved to a satisfactory level. While the more familiar aspects of this project, i.e. The java implementation as part of the android application, were genuinely quite fun to complete, even at the more stressful times of the year.

This project, encompasses a lot of what this Computer Applications course is all about, and so to deliver on such a project is greatly fulfilling. Mistakes were made and lessons learned, and I believe this will lead me in great stead in my future work.