



**DUBLIN INSTITUTE
of TECHNOLOGY**

Institiúid Teicneolaíochta Bhaile Átha Cliath

Complex Weather App Final Year Project Report

**DT211C
BSc in Computer Science (Infrastructure)**

Adam Noonan – C12492572

Supervisor: Mark Foley

School of Computing

Dublin Institute of Technology

6/04/2016



Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Adam Noonan

Abstract

The weather has been around longer than humans have and ever since we have been trying to predict the weather. This has evolved from basic understandings in ancient Greece to the highly sophisticated weather satellites in orbit above use right now. Accurate weather prediction and display in meaningful formats has been on the rise in recent years. This is due to the increase in accuracy of the forecasts and the rapid development of powerful and extremely mobile technology such as smartphones and tablets. Weather forecasting is very important to most people in their everyday lives and to major corporations and military operations. The average person simply wants to know what the general conditions are so they can prepare how to leave their home. Others have a vested interest in certain hobbies that may rely heavily on the day's weather conditions. These may include farming, surfing or sky diving. Major corporations need to know the yearly weather conditions of entire regions to plan their operations. This could include a major telecommunications organization building massive data centers. They need to know if the region is prone to extended heatwaves as they must keep the servers cool which will increase their spending on the data center.

Point Break Weather intends to tap into the average user's daily routine. Point Break Weather is a user friendly web application that allows people to plan their day or days based on the weather predictions and interactive maps that it provides. A user can sign up to the website and avail of its features. These include contact to the admin of the site if they have any questions, find the best surfing and historical sites in Ireland, store their personal preferences in a database for extended use, loading of their own files to aid them in their efforts further and finally through use of an Inverse Distance algorithm the user can see how the following weather conditions will pan out.

This is all achieved through the language of python. The language allows for extensive use of other more powerful features. This includes using GDAL, numpy, scipy and matplotlib libraries to achieve visual excellence and data accuracy. The User Interface is a friendly and easy to use system that is made easier through the python MVC framework of Django and the visual use of Bootstrap3. The storage of the user's data and weather data is achieved using the very powerful PostgreSQL. This database allows for GIS extensions that will allow large amounts of geospatial data to be stored there for use in the web application.

KEYWORDS: python, Django, MVC, PostgreSQL, Bootstrap3, GIS, GDAL

Acknowledgments

I would first like to thank Dublin Institute of technology. They have allowed me to participate in this course which has allowed me to meet lifelong friends and obtain a set of skills that is highly sought after in the workplace. I would also like to thank my supervisor Mark Foley. His previous knowledge in this field allowed him to aid me when it was needed. His patience throughout the project was also greatly appreciated. Thanks Mark.

Table of Contents

1. Project statement.....	6
2. Project Research.....	6
Background research.....	6
Alternative existing solutions to the problem	8
2.1 Radio	8
2.2 Television.....	8
2.3 Websites.....	8
2.4 Apps	9
2.5 Raw Data.....	9
Technologies researched	10
2.5 Selection criteria	10
2.6 Java for Front End.....	10
2.7 PHP for Front End	11
2.7 Weather API.....	12
2.8 REST API	14
2.9 Storage of API data.....	14
2.10 Python Algorithm for Prediction.....	14
2.11 Django MVC Framework in Pycharm	15
2.12 Inverse Distance Weighting ^[14]	16
2.13 GDAL	17
2.14 Docker for local testing.....	18
2.15 Cron scheduling	18
Resultant findings/requirements	19
Technologies Selected for Project.....	19
Survey	20

3. Design and Methodology	23
Introduction.....	23
3.1 Functional and Non-Functional Requirements	23
3.2 Methodology	24
3.3 Platform & Design Layout.....	25
3.4 User Interface.....	25
3.5 Database	29
3.6 Views & Forms	31
3.7 Emails	34
3.8 Use-case Diagrams	35
3.9 Activity Diagrams	36
Conclusions.....	39
4. Architecture and Development	39
Introduction.....	40
4.1 System Overview	40
4.2 System Walkthrough (Implementation).....	41
4.2.1 Register	41
4.2.2 Login	42
4.2.3 Password Reset	43
4.2.4 Logout	44
4.2.5 List of Features	44
4.2.5 Navigation Bar Features.....	45
4.2.6 Preferences	48
4.2.7 Location and File Opener *.....	50
4.2.8 Weather Forecast *	52
4.2.9 Current Weather *	54
4.3 Project Difficulties	54
Conclusions.....	56
5. System Validation and Demonstration	57
5.1 White-box Testing	57
5.2 Black-box Testing.....	58
5.3 Automated Testing (Considered using)	60
5.4 Test cases	61
5.5 Demonstration overview	63

5.6 Demonstration walkthrough (Screenshots)	64
6. Project Plan	68
6.1 Pre interim report project plan	68
6.2 Post interim report observations & changes	70
6.3 Final project plan	71
6.5 Future work and Advice.....	72
Conclusion	73
Bibliography (research sources).....	75

1. Project statement

This project is a complex weather application. It will allow users to find useful resorts and locations to further their hobbies or holidays if they are staying in the country. The users can register to the site and avail of a list of features. These features include storing their preferences for further use and reference. Loading a file that may help them further understand the weathers conditions. Find the current weather conditions in Ireland. They can finally request an animation that will show the next day's weather conditions unfold.

2. Project Research

Background research

Weather forecasting has been an integral part of human history and human advancement. It has influenced cultures and defining moments, possibly changing important events (i.e. D-Day June 6th 1944). However when we think of forecasting many of us think that it is very recent when in fact it began millennia ago. The earliest known civilisation to attempt weather forecasting were the Babylonians. They attempted to predict short-term weather by studying clouds and other optical phenomena. Aristotle published his book *Meteorologica* in 340 B.C. In this book he included his theories on cloud, rain and other weather formations. He also included other important topics such as chemistry and geography. The Renaissance saw an

increase in logical observations of the weather, rather than philosophical observations. Galileo created an early form of the thermometer in 1592 and Evangelista Torricelli invented the barometer in 1643 for measuring atmospheric pressure. While these types of technology could make accurate readings the information could not be transmitted to the masses until the mid-nineteenth century with the invention of the telegraph. Weather observation stations began to appear around the globe and the first weather maps were drawn. They would show wind patterns and possible routes of major storms. In the 1860s synoptic weather forecasting began. This was the analysis and compilation of these many observatories simultaneously. By the 1920s the radiosonde was invented. This was a small box that was lifted into the atmosphere (roughly 30km) by a hydrogen or helium balloon. While the balloon was ascending it would transmit moisture, pressure and temperature. This technique is still used today with radiosondes launched “every 12 hours from hundreds of ground stations all over the world” ^[1]. Numerical prediction was also used to predict the weather.

The use of mathematical equations to predict weather patterns was first formulated by Wilhelm Bjerknes and Lewis Fry Richardson in 1904. However it would take Richardson six months to make a six hour prediction that turned out to be inaccurate. However Richardson published his reports in his book *Weather Prediction by Numerical Process* in 1922. He proposed a group of people working on individual equations. However he would have needed 64,000 human calculators in one very large room. By the late 1940s early computers made numerical prediction more feasible. Mathematicians at Princeton University made a huge leap in the correct direction. John von Neumann constructed the computer and Jules Charny lead the science team. Charny determined that removing some of Richardson’s variables such as sound waves would make predictions more accurate and easier to calculate ^[2]. In April 1950 Charny’s group made successful 24 hour predictions over the continental United States. By the mid-1950s “numerical forecasts were being made on a regular basis” ^[3].

Modern computers and satellite technology has given use faster and more accurate data for weather prediction. The TIROS 1 (Television and Infrared Observation Satellites) satellite gave the first orbital cloud cover information on April 1st 1960. Although the picture was primitive compared to modern standards, it paved the way for more investment in weather satellites. This has allowed satellites to provide more than just visual information. They can provide atmospheric temperature and moisture using atmospheric sounders. The data that is received from satellites is similar to radiosondes but with the “major advantage that the satellite data are more compelling spatially” ^[4]. This aids in filling in the gaps that weather stations that are in other nations or continents have.

Benefits from weather forecasting can include an airline knowing of a major storm front along a flight path. They can reroute the aircraft alleviating airline costs. Better wind information helps forecast possible hurricane paths and duration. As previously stated weather forecasting has a significant impact on military operations. The D-Day landings were scheduled for June 5th 1944. However extreme weather conditions meant that it was delayed by a day to June 6th 1944.

Alternative existing solutions to the problem

2.1 Radio

A user can listen to various radio stations to get weather forecasts. This is the most basic and unreliable source of weather forecasting. The radio does not allow the listener any input. Depending on the radio station it may be very much localised or it may be national or in rarer cases regional. This further complicates things for the user, which station is most accurate? , Which will give national forecasts not just by city or county. There is very little functional requirements for radio broadcasts and there is no user input. This form of forecasting would be the lowest in terms of user input and accuracy. Not much can be gained for the proposed project by using the functionality of radio.

2.2 Television

TV would be similar to radio in terms of the actual information, however TV can display it information visually making the interpretation of the data much easier and faster for users. TV also allows the broadcaster the ability to show weather forecasts over much wider areas without increasing complexity of the data. The visual aspect of TV also allows the display of information such as wind direction, isobars and cold/hot fronts to name a few. Just like radio there is no user input available. The displayed weather forecast is simply to understand and the users watching are not “bombarded” with information considering the short amount of time that weather forecasting is shown on TV. Examples of TV stations that regularly show weather reports would be Sky news.

2.3 Websites

Websites are very practical and would be the most accurate of all forms of weather reports. There are many types of weather sites that will vary in their display of the information, accuracy of the reports and who actually supplies the information. Websites will most of the time just have weather data visually displayed to users. There are not many websites that would have the option to convey the weather forecasts through audio. The greatest appeal of websites is the accuracy of the data. Most websites will be run by major broadcasting corporations or government agencies. There are also websites that will be run by ordinary people who simply have an interest in weather. An example of government website would be the National Oceanic and Atmospheric Administration (NOAA) ^[5]. This site is run by various departments of the U.S. government. This means the technology and algorithms being used to forecast the weather data will be of the highest quality. The best example for ordinary people that may have no qualifications in weather would be Weather Underground ^[6]. This site may not be as accurate as NOAA but it is not limited to the United States and the amount of contributors makes up for the lack of resources. All websites will have user input available which gives them much more appeal than TV or Radio.

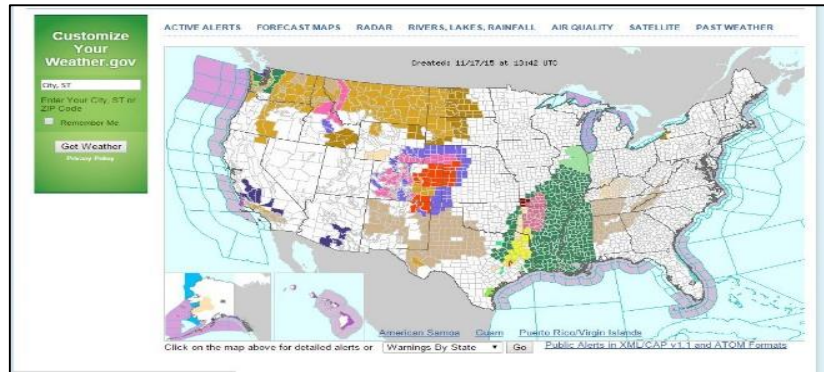


Figure 2.1. NOAA forecast map of the United States

With colour coded in information.

2.4 Apps

Apps on smart phones or tablets are the pinnacle of weather forecasting. They encompass all of the advantages of the previously stated alternatives to weather forecasting with one major advantage that none of the others have, convenience. Apps can be accessed at anytime, anywhere. Even when a user is not using it the app will run timed updates so that if the user does not have an internet connection they will have a forecast that will be updated within the hour. The probability of the weather changing drastically within an hour is very unlikely meaning this feature that most apps have is very useful. Most people do not have access to a computer at any given time of day, however the probability of a person having a smart phone or tablet is very likely in modern times. It is estimated that by 2017 69% of the global population will use some form of mobile phone ^[7]. Although apps may not have the sophistication of websites like NOAA or TV stations like Sky news, the accessibility of these apps and there ease of use is the main reason for their sharp rise in recent years.

2.5 Raw Data

Using regular get requests can give a user raw data from various sources. A simple get request in python to a weather API will return various bits of information in various formats. JSON is usually the default type of the returned data but in most cases it can be specifically stated. Other calls will give the data in an even more basic comma separated format. This can be obtained from places such as airports.

```
import requests
r = requests.get("http://api.openweathermap.org/data/2.5/weather?q=London&APPID=49153ef926a23fa4514a045f1ea0fe87")
print (r.content)
```

Figure 2.2 – Python get request to Open Weather API

```
b'{"coord":{"lon":-0.13,"lat":51.51},"weather":  
[{"id":801,"main":"Clouds","description":"few  
clouds","icon":"02d"}],"base":"cmc stations","main":  
{"temp":282.729,"pressure":1033.77,"humidity":84,"temp_min":282.729,"  
temp_max":282.729,"sea_level":1044,"grnd_level":1033.77},"wind":  
{"speed":2.01,"deg":154.501},"clouds":  
{"all":24},"dt":1457789814,"sys":  
{"message":0.0047,"country":"GB","sunrise":1457763557,"sunset":145780  
5696},"id":2643743,"name":"London","cod":200}'
```

Figure 2.3 – Get request result in JSON format.

The above figures show how quick and simple it is to get weather data. However this data on its own is confusing and useless to the average user. For this raw data to be of any use it must be formatted in a way that makes sense. This could be achieved by placing this data into a database and pulling it down to an html template.

Technologies researched

2.5 Selection criteria

There were several factors that influenced the technologies that were selected for this project. Do I have experience with the researched technology? I decided to research technologies or environments that I have previously used. If the technology or environment is sufficient then implementing the code will be much shorter than something that is foreign to me. Another aspect used in research and selection was if I have not used this technology, how long will it take to familiarise with it? Some technologies had aspects that I knew and others that I have never used or implemented before. Some technologies and environments had only small extensions that had never been seen and these could be learnt within a few hours of research and practice. Others were completely new and would have taken much longer to understand which would have affected other stages of the project such as testing. Would the technologies being researched do most of the work for me? I could not choose a piece of software or environment that would do all the proposed work for me. However some technologies would be used early in development to see if a certain aspect of the project is in fact feasible. The most important selection criteria was the technologies and environments compatibility with REST API architecture. The proposed project will be following the REST API concept. REST is an industry standard used in the development of many apps or websites. The technologies and environments would have to be able to transfer data/communicate over the common medium that rest uses (HTTP).

2.6 Java for Front End

The front end of the weather app could be coded in Java. The “technologies” that were researched were environments that the front end would be coded in. Eclipse was the first

environment that was researched. Early on in testing of the environment it was already quite “buggy”. The main issue that arose was that of R not being assigned to a variable. After many attempts to fix the problem it was clear this was an Eclipse issue and not a java issue. Numerous searches for solutions to the problem confirmed this theory as many people reported this only occurring in Eclipse. The next environment that was tested was android studio. This environment was very similar to Eclipse which I have experience in. The transition was not difficult and only took a few minutes to understand all the necessary commands and file structure. This environment had significant advantages to Eclipse. There is a multi-window app developer making the creation of apps for multiple devices much easier. There are google APIs incorporated into the environment on download. Faster virtual devices for quick testing. There is built-in support for google cloud platform which would allow incorporation of google cloud messaging into the app engine. Android studio was chosen for several reasons. I have experience in android app development. This environment seems optimized for android app coding and development. The amount of time to set up the environment and coding seems far less than that of Eclipse. A basic front end was created within an hour with xml calls to a website containing weather data. The fact it took such a short amount of time to get a basic prototype up and running is the main factor for choosing this environment for the front end or “View” of the proposed weather app.

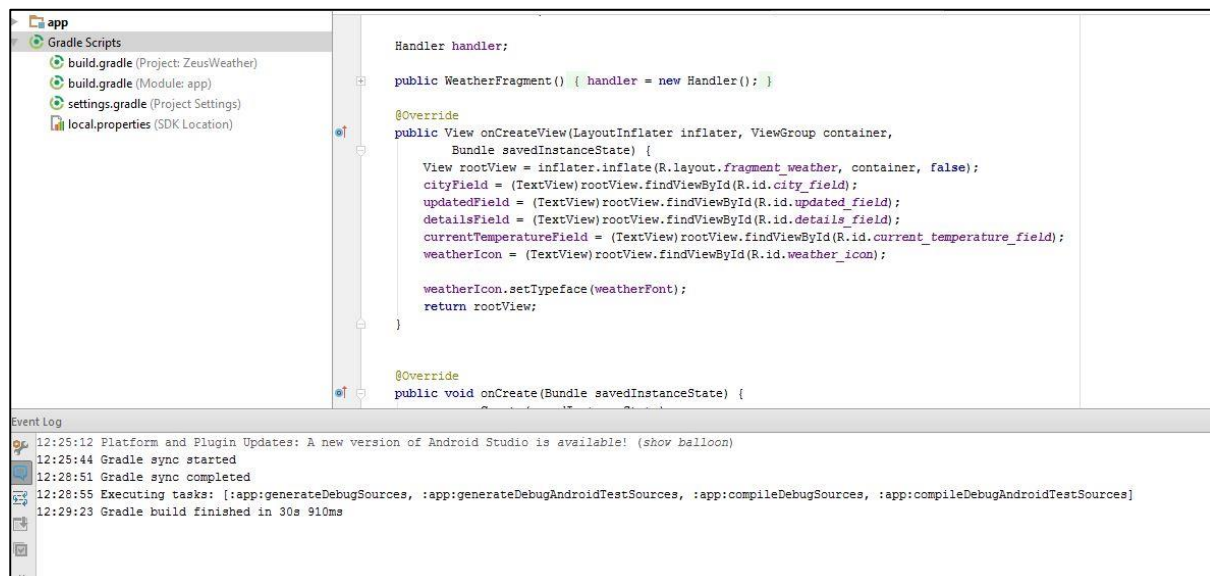


Figure 2.4 – Java code snippet from Android Studio.

2.7 PHP for Front End

Another solution that has been researched is using PHP to create a webpage for the front end of the app. Creating a webpage for the view would be less time consuming and would be much easier to implement in the MVC model. Possible environments to create this webpage were researched. The first was Notepad ++. This is a very good editor for programmers that incorporates almost every language. There is very little help given to a user in this

environment. However the main benefit and major factor to consider is the ease of use of this editor. This editor's main flaw could also be attributed to its main strength. The fact that there is very little clutter unless the user selects one of the dropdown menus will contribute in reducing confusion during the coding of the webpage. Context is another editor that is very similar to Notepad ++. It has the same features as Notepad ++ with a few extra features such as the ability to record macros. There are far more powerful editors out there such as Espresso, Sublime text and Edit Plus. However these require a fee to use with only a 15 to 30 day free trial. This may be enough time to code the front end but that kind of restriction cannot be allowed in this project as it will concentrate focus on one particular area and will take time from other aspects such as testing and the coding of the back end of the project.

2.7 Weather API

A weather API was needed for this app. There are several websites that will provide these APIs with varying degrees of accuracy and the amount of calls that can be made per hour or day. The first website that was tested was openweathermap.org (<http://openweathermap.org/>). This site provided an API for free but although I signed up for it I did not seem to get an API key. The layout of the site was basic put some of the maps were confusing and some weather information would obscure the map or even other weather data such as isobars.

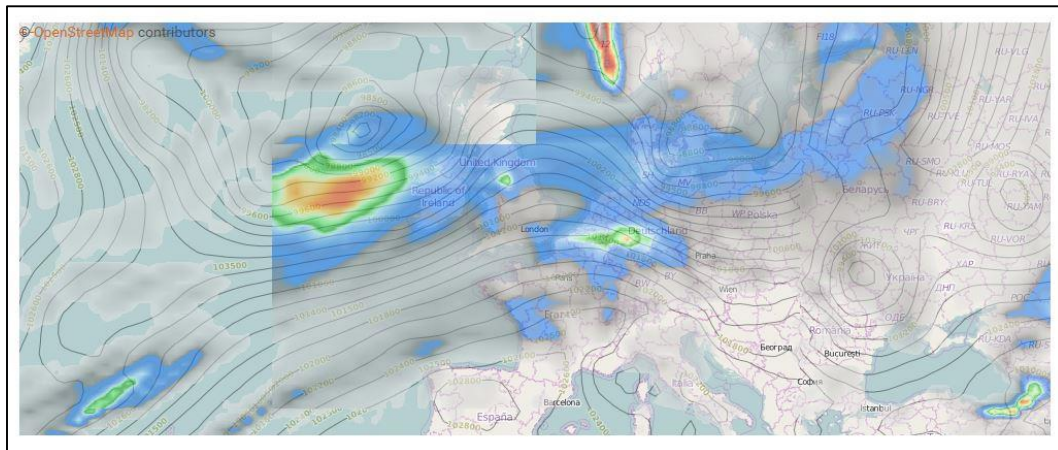


Figure 2.5. Openweathermap.org interactive map.

Figure 2.2 shows how the map is somewhat confusing although not impossible to understand. The next site that was investigated was the National Oceanic and Atmospheric Administration or NOAA (<http://www.noaa.gov/>). This website was an official United States government website with the .gov domain. This site was the most accurate of all but the fact that it was nearly all North American weather data it would not suit an app that at the least is designed to show weather data for Ireland. This site does transmit weather data from boats in the North Atlantic that could be used to predict weather over Ireland or Western Europe but it would simply eat into time that could be spent elsewhere in the project. Another site that was investigated was the Norwegian met service (<http://www.yr.no/place/Ireland/>) specifically over Ireland. This site is basic in its display of the data. It simply gives an average

temperature for the entire day based on major cities such as cork and Dublin or provinces in Ireland like Leinster or ulster. The user can select the hour by hour feature for each day to get a more accurate reading. The map that is provided is simple in its display but there is advantages to this that may be incorporated into the app.

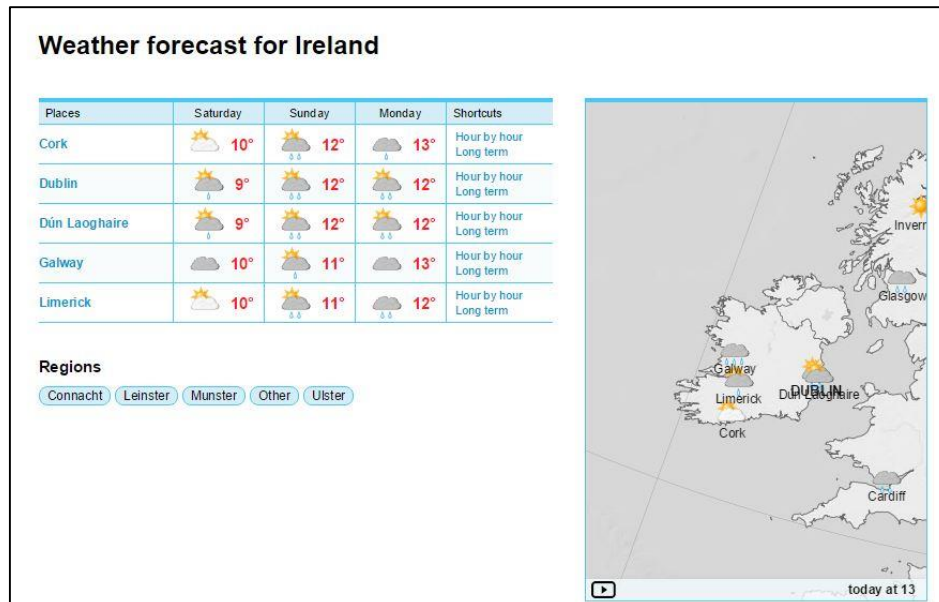


Figure 2.6 – Norwegian site yr.no forecast of Ireland

The last website investigated was weather underground (<http://www.wunderground.com/>). This website was the most user friendly in terms of layout, the way the data was presented and even the use of colour on the site. This sites API was the most thorough in terms of what was offered for free users. Everyone can get the same information all the way up to hurricane and earthquake reports. The only advantage to the paid API is the increase in the number of calls that can be made per day. This website is the one that will be used over most others. Other readings such as oceanic conditions will be taken from NOAA as they have very accurate weather bois in the north Atlantic. The fact that more languages have been incorporated into the API such as JSOC and XML was a major factor in choosing this as the primary API.

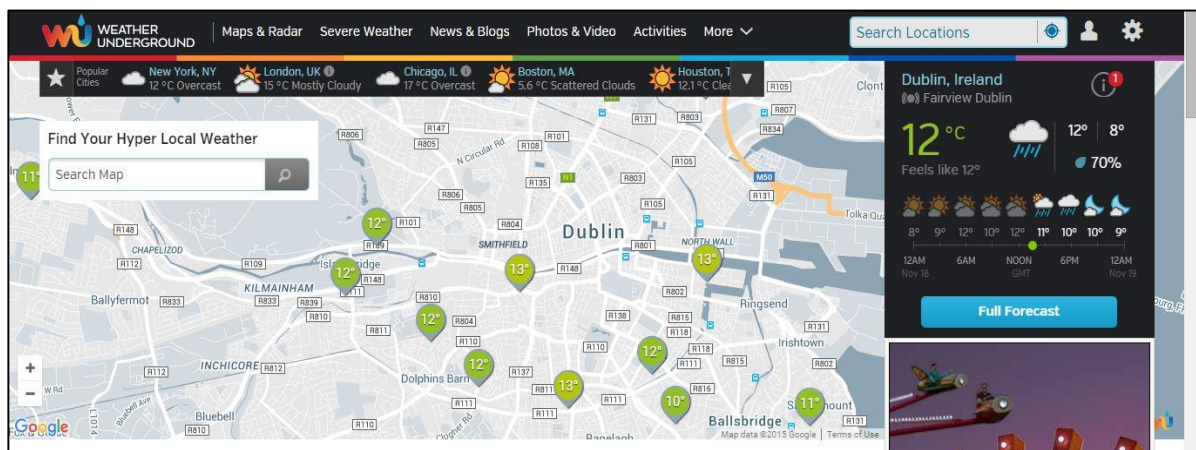


Figure 2.7. Weather underground home screen.

2.8 REST API

All components of the proposed app will be following REST API. REST is an architectural model that uses HTTP for inter-machine communication. Most forms of communication between different software over different networks or even different machines are quite complex such as SOAP, COBRA and RPC. Using REST means that all communication will be simple message-based using the HTTP standard. There are only four request that a user can use when using REST. These are GET, POST, PUT and DELETE. REST also allows the data format to be either JSON or XML and it can be switched at any time. I have no experience in REST but there are very useful tutorials on this topic. The fact that I will have many different pieces of software communicating to each other is a major factor in choosing REST, as previously stated the use of HTTP makes communication and data retrieval very simple.

2.9 Storage of API data

For the app to be practical the weather data would have to be stored on a database that is not located on my laptop or local machine. The easiest and most logical storage was using PostGIS which is a special database extender to PostgreSQL that adds the ability to store geographic objects. PostGIS allows for the storing of polygons, raster data and spatial operators and predicates. The database will be used for the prediction feature of this system. When a call is made by the user from the front end for current weather it will get that info directly from the weather API. The database will be bypassed in that particular scenario. However when the user wants a prediction the data that is being stored will be supplied to an algorithm separate from the weather API that will then make the prediction. I have experience with SQL but not in relation to geographic objects. An Ubuntu sever has been supplied for this project. It is being run by Microsoft azure and has a PostgreSQL database extension. In order to allow PostGIS a simple SQL command of extend PostGIS will spatial reference the database and allow geospatial data to be stored. The data will be retrieved using cron jobs on the Microsoft azure sever that has been supplied to me. I have chosen GIS as my optional module for first semester. This will aid me greatly in enhancing my knowledge of PostGIS.

2.10 Python Algorithm for Prediction

Python will be used as the language for making the weather prediction. It will achieve this by pulling data from the PostGIS database using SQL queries transferred through REST API calls. Once the algorithm has this data it will make its prediction. There are several ways the

prediction could be made. There is a python programme sklearn or scikit. This python programme allows you to feed numerical data into it by using an estimator which will be a python object that will implement the methods fit (x, y) and predict (t). The estimator will normally be called clf as it is a classifier. This will then predict the data it is given and represent it through an image.

```
>>> clf.fit(digits.data[:-1], digits.target[:-1])
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape=None, degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Figure 2.8 – Scikit code example.

However although this python programme does predict certain data it may not be compatible or suitable with the data that I will be providing. Another possible way of predicting the weather could be to use the sliding window algorithm. It is estimated that when this is applied to weather predictions on a wide scale (roughly three years) it has an accuracy of over 92%. It is unclear if this could be applied to my system which will be collecting the previous few day's data and applying that to the algorithm. Either way python will most certainly be used as it is a familiar language and is very useful when applied to weather or geographic data.

2.11 Django MVC Framework in Pycharm

Django is a possible candidate for my MVC framework. Django uses python instead of PHP when creating rich web applications ^[9]. Django was specifically created for complex database driven web applications. The fact that PostgreSQL will be used in this project and the fact it was used when creating Django is another major factor in its possible use. The MVC framework in Django is achieved by a mapper between Python classes and a relational database (Model), processing HTTP requests (View) and a URL dispatcher (Controller). Another extremely useful “extension” of Django is the fact that third party software can be plugged in to a project. However it must follow the reusable app conventions ^[10]. There are estimated 2500 packages available for extension of a project ^[11]. If Django is selected for the construction of the front end and the integration of both back and front end then it will be coded in Pycharm. Pycharm is an Integrated Development Environment (IDE) that is used when coding in python. It has two versions community and professional. The professional edition is far more extensive and this version will be sought after. Pycharm has many useful features such as a debugger, code analyser, and integrated unit tester and most importantly it supports web development with Django. Pycharm is developed by a Czech company called JetBrains ^[13]. Pycharm professional addition has been licensed under the apache license. This means that Apache servers have been integrated within the Django framework making testing and distribution much easier for developers.

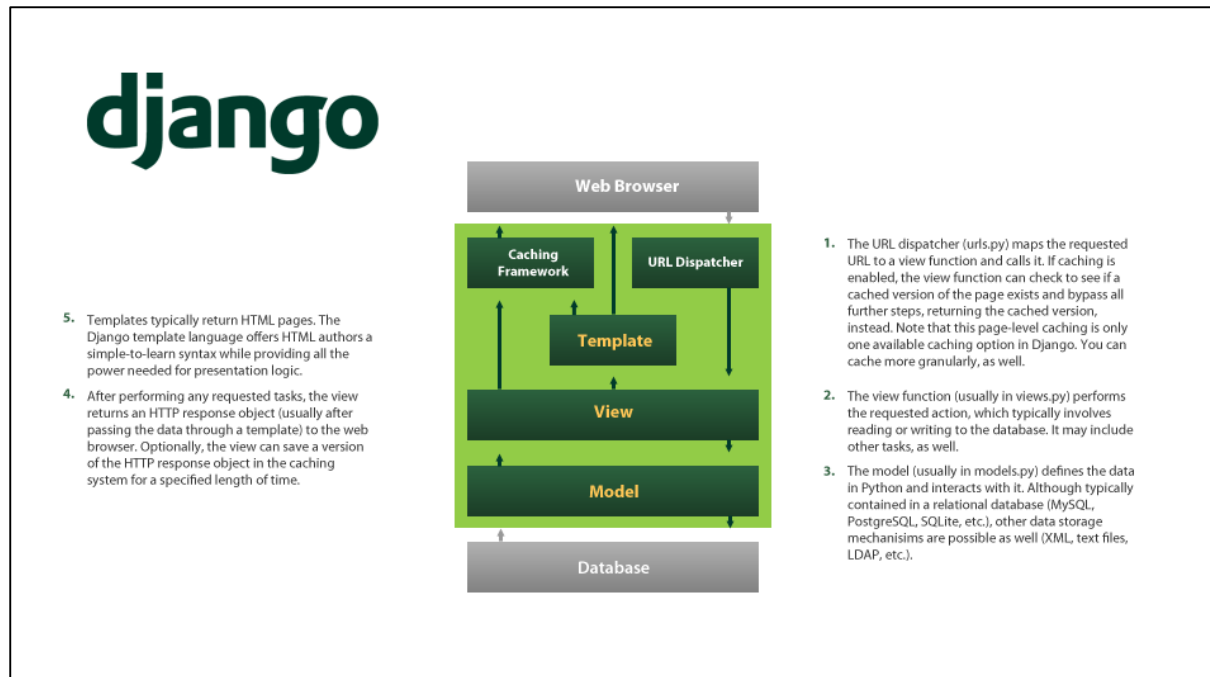


Figure 2.9 – Django architecture with description of each component

<https://mytardis.readthedocs.org/en/latest/architecture.html>

2.12 Inverse Distance Weighting^[14]

Inverse Distance Weighting (IDW) is a deterministic method of interpolation when a group of scattered points are known. Each point has a value attached to it and a weighted average of those values is calculated. IDW also implements the assumption that points that are close together are more alike than point that are farther apart. IDW gives greater weights to points that are closer to the prediction area than those farther away hence the name “Inverse Distance”. During the calculation a very useful value is used called P or power value. The weights of the data points are raised to the power P. As the distance increases so too does the weight (the importance of that point and its values). This increase or decrease in the weight of each point is dependent on the value P. if $p = 0$ then there is no decrease in the distance and the prediction will be the mean of the values at the various points. Consequently if p increases then only the points in the surrounding area will be used. Smoothing is used during the IDW calculation to give an even greater visual representation of the data values at the points.

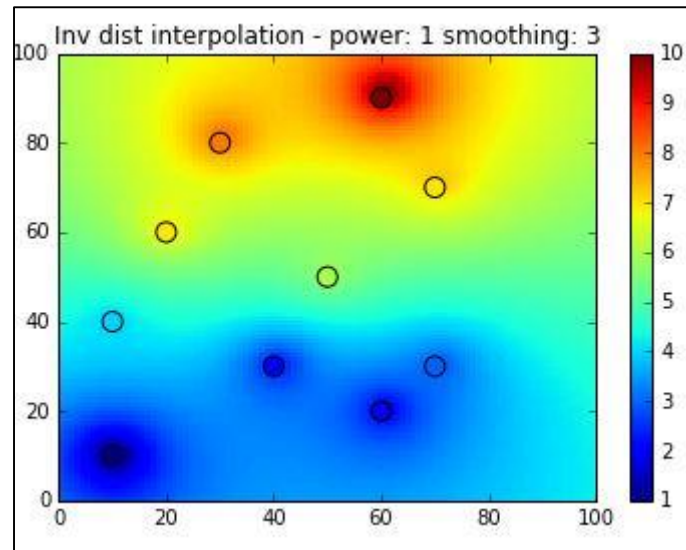


Figure 2.1.1 – IDW with smooth of 3

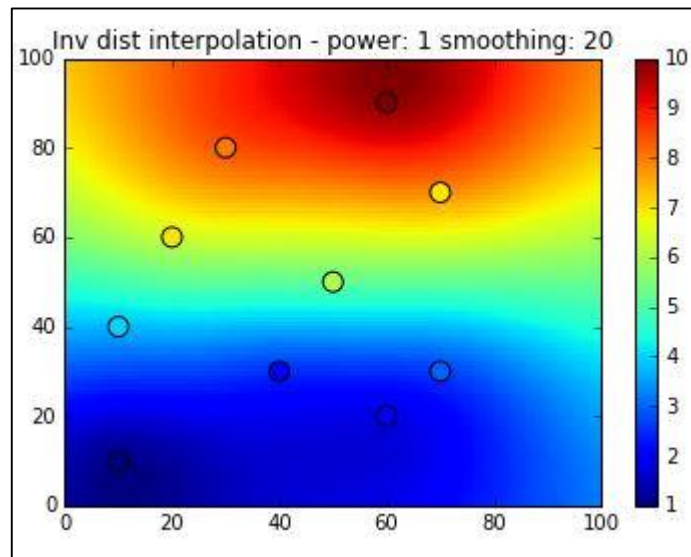


Figure 2.1.2 – IDW with smooth of 20

The grids can be specified not just with regular numbers but with Cartesian co-ordinates. This would allow IDW to take in latitude and longitude values together as single points. The data values at the co-ordinates could then be used to make various calculations such as temperature and air pressure over a certain region or country. If these values could then be obtained on a regular basis they could be overlapped to create an animation. This would then give a clear and fluid animation of various weather features and events.

2.13 GDAL

The Geospatial Data Abstraction Library (GDAL) is a software library that reads and writes vector and raster geospatial data formats. As GDAL is a library it presents an abstract data model to the calling app. Although GDAL is a free piece of software it is considered very expensive due to its extensive capabilities and its widespread use throughout the Geographic Information community ^[15]. GDAL can access an enormous amount of information in order to create raster's and vectors which can then be used for a number of various tasks. This includes satellite data which is extremely accurate and can cover a very large part of the earth's surface. The current version of GDAL supports up to 120 raster formats ^[16] and up to 200 drivers ^[17]. GDAL was originally written in C++ but there are a number of binary packages that can be downloaded to ensure it is compatible with other languages. This includes a python package that will be used in this project. Installing GDAL on a Linux system would be an easy task as the `sudo apt-get install GDAL` command will download the correct version of GDAL. It is possible to download and use GDAL on windows but there may be complications in the future during the installation of GDAL. Various pieces of software can be used to extend GDAL on your system. These include OSGeo4W, ArcGIS and QGIS to name a few. These programmes can perform various tasks such as raster creation and Inverse Distance Weighting to name a few. GDAL will be an invaluable part of this project as it will allow the creation of complex and fluid animations for weather conditions.

2.14 Docker for local testing

Docker is a relatively new piece of software first released on 13th march 2013. It is an open source piece of software written in Linux that allows software to be developed and tested within software containers ^[18]. The fact it is written in Linux means it can make use of the kernel to allow multiple containers to run in a single environment without using multiple virtual environments ^[19]. Docker creates images that work as the software containers. This means that when a change is made to the software inside the image or the image itself only the layer updates need to be propagated as opposed to the entire virtual machine. Docker will allow the testing of the proposed system to be run locally inside an image on a local VM. This ensures that a major failure will not affect the server running the system or any other integrated pieces of software.

2.15 Cron scheduling

Cron is a program that allows users of UNIX systems to execute commands or scripts on a regular basis at specified times and dates. This will be very useful for the proposed system as regular pieces of weather data such as temperatures, wave heights and most importantly

geospatial coordinates must be taken and placed in a database (PostgreSQL for this system) on a regular/daily basis. Cron is a daemon, meaning it will be called once and lay dormant until it carries out the specified tasks. In the proposed system Cron will be used with the wget command to call weather site APIs and then place this data (in JSON format) into the database. This may be accomplished by calling in the same wget command a PHP script that will take the JSON data and place in the database. Collecting weather data very regular (every 2 hours) in order to make a forecast can place strain on the database and the entire system. Cron can also solve this problem. A Cron job can also be executed that will remove data that is more than five days old. This could be achieved by calling an SQL statement once a day to free up space. Cron in terms of code and complexity although not impressive or very large will still play a very important role in the proposed system.

Resultant findings/requirements

Based on my research there are several requirements that must be met if I am to use the technologies and environments.

- Inter-machine communication must be simple and return the data every time it is requested
- The environment that the front end or “View” will be created in must be efficient and not give errors or problems that are related to the environment itself as opposed to a syntax error.
- The front end must be clear and concise. The data must be easy to understand and displayed with various graphics to reinforce that text being displayed. This would aid older users of the system.
- The storage must be able to hold geographic data such as polygons and raster’s that will be used later in the animation or prediction of the weather data.

The weather API or software must provide clear and concise data but it must not do all of the work for me. Later in the development of the project the use of these APIs may be lessened in favour of data being taken directly from the back end database.

Technologies Selected for Project

The following is the complete list of technologies and app/packages that will be used over the development, implementation, testing and final release of the project. These were selected based on the above findings.

- Python
- Django
- Pycharm
- Cron
- Inverse Distance Weighing
- PostgreSQL/PostGIS
- OpenWeatherMap API
- GDAL

The language that most of this project will be coded in will be python. The reason for this is an underlying knowledge of python and a lot of GIS systems use python so there is a greater chance of support and tutorials if they will be needed.

The MVC framework that will be used will be Django. Django is written in python and is designed to speed up the process of creating a basic but sound skeleton for a web application. Pycharm will be used as the main tool for coding this project.

Pycharm makes the downloading of python and Django packages much easier and allows for the creation of virtual environments which will separate my project and base machine meaning they will not interfere with each other.

Cron will be used to schedule the removal of data from the database which will in turn ensure that the system runs much faster and smoother to the user.

PostgreSQL will be used as the backend storage of the API data. PostgreSQL is a very powerful back end database especially when it comes to spatial data and extremely large amounts of complex data. It also means that the PostGIS extension can be used which will then make the creation of the database in Django much easier.

OpenWeatherMap will be the main API from where the weather data will be called from. After the interim presentation I decided to narrow down the idea of the web application to that of surfing data. The best API for this would have been magic seaweed. However their API was still in the beta stage and an email was sent requesting an API key but no response was ever received back. This is why OpenWeatherMap has been chosen.

GDAL and Inverse Distance Weighing has been chosen to perform the necessary calculations and spatial references that will be needed for data retrieval and map display and animation.

Survey

During the research of the various technologies and existing solutions a survey was conducted to ascertain what a regular user would want and expect from a weather app or webpage. The survey was created on the site Survey Monkey. There were eight questions in total which covered various topics such as frequency of use, types of data displayed,

prediction feature and other features to make interpretation of data easier and more accurate. This was sent out to numerous random people who do not have any experience in computer systems or weather. There were a total of 11 responses over the course of two days after which the data was not recorded. This survey will help shape the app all the way down to what data is collected and interpreted. Many people said they would use a weather app daily and over 70% said they would. Next they were asked what type of weather data was most important to them such as weather conditions, temperature etc. most people believed weather temperature and conditions were the most important conditions and that these should be a priority when creating the system.

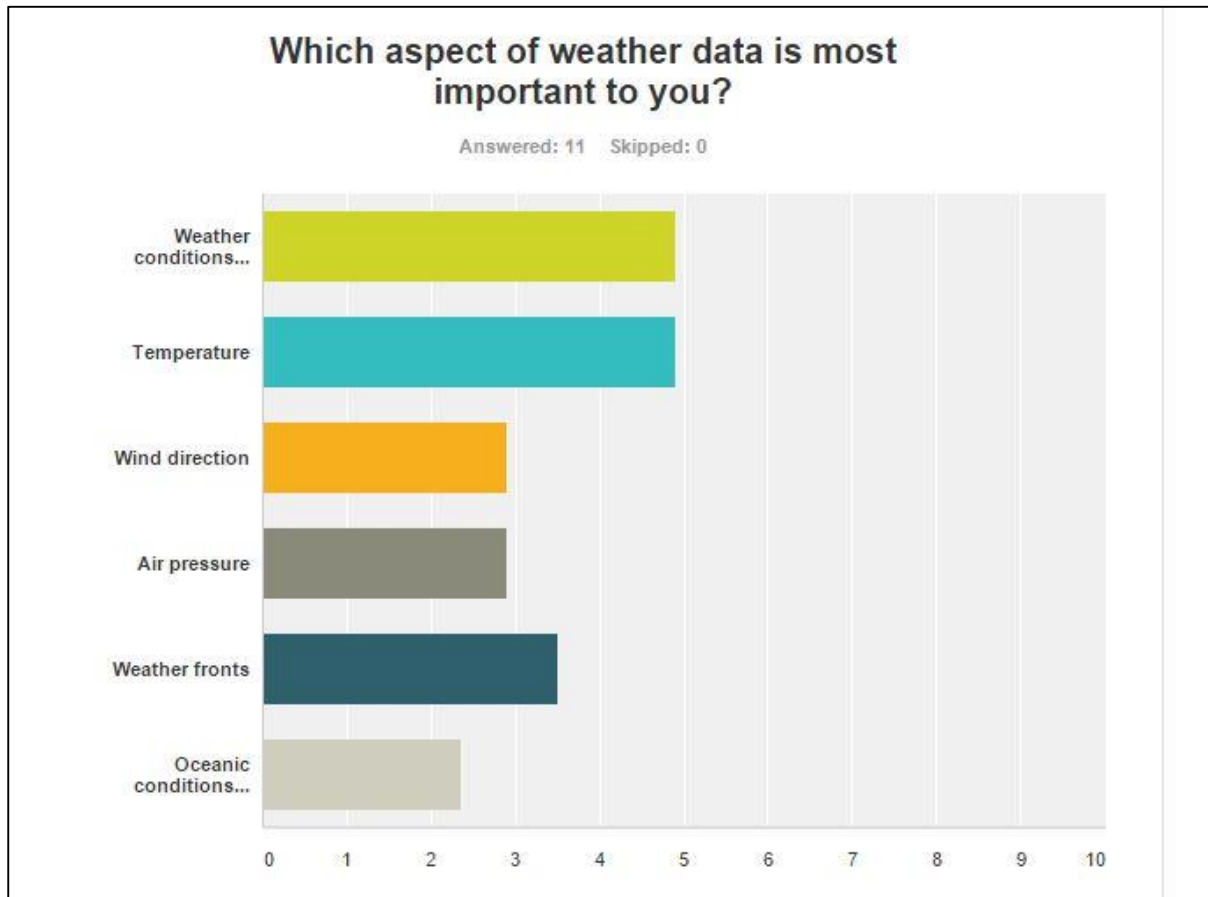


Figure 2.1.1 – Survey question asking what data should be given priority.

The next question asked would you like a weather prediction service. There was a 100% yes response on this question. What type of weather conditions should be predicted was asked next and the majority wanted just weather conditions such as will it rain at 14:00 tomorrow, while some wanted all weather conditions predicted. This part of the survey allows the creation of an app with less prediction features but will not disappoint the majority of users. This will help later in the development and testing stages as time constraints may become an issue. Next question was how the predictions should be presented to the user. The majority (over 60%) wanted simple graphics such as a picture of the sun over a city if the weather is to be sunny or a cloud with rain if rain is most likely.

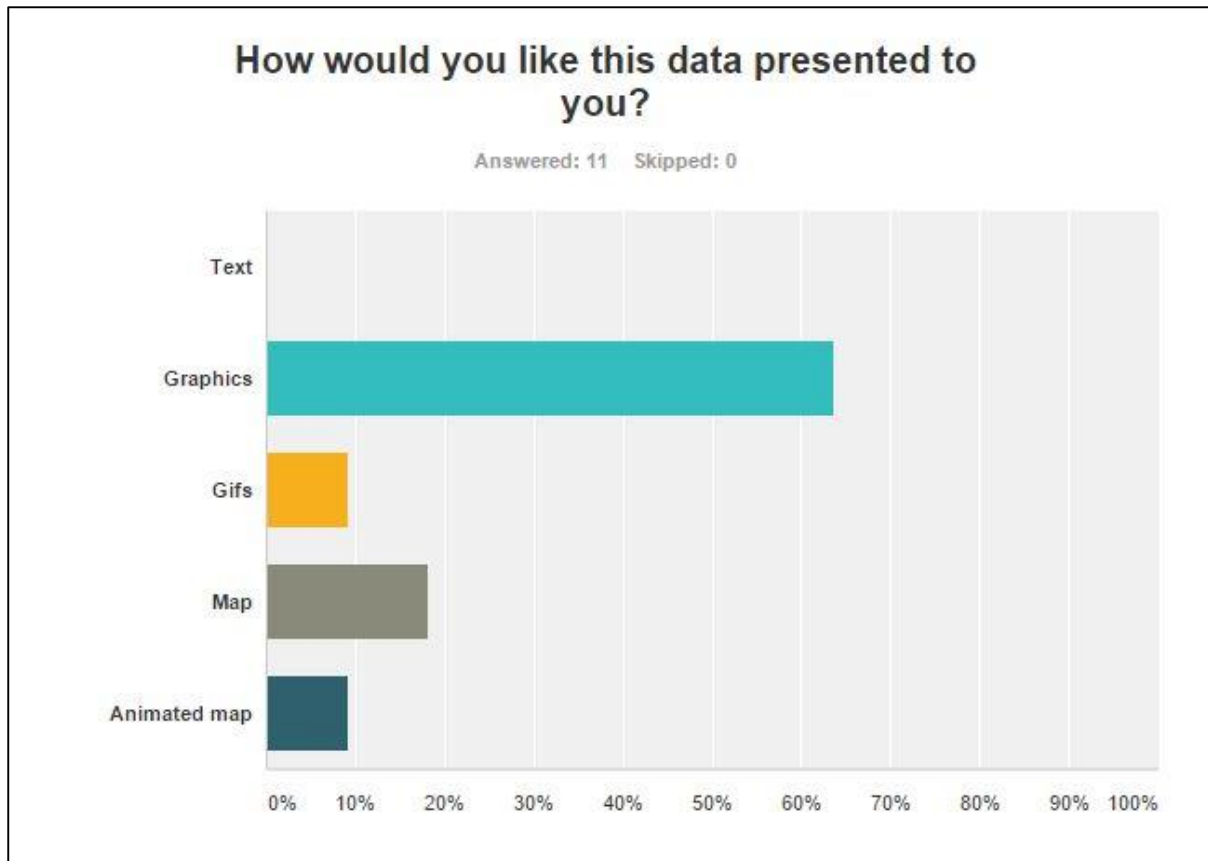


Figure 2.1.2 – How the predictions should be presented.

The next questions was important in the overall approach to the data collection and prediction of the project. The question asks would you prefer small scale and more accurate or a large scale and less accurate. 100% responded with small scale. In terms of this project that would mean just Ireland. This means that data collection will not be as difficult meaning more data in a shorter period of time can be collected for Ireland. This in turn will mean the prediction can be more accurate. Finally would a natural disaster notify be relevant if the app was small scale. Most were undecided with an equal amount saying yes and no. This feature should be left till the end as most users seemed uninterested in said feature.

3. Design and Methodology

Introduction

In this section of the report I will be discussing the design of the web app. This will include the design layout, the platform selected, and language chosen, file layout and file function. Next there will be various diagrams such as use-case and activity diagrams that will give a visual description of the systems functions and also specific functions of the web app and how they function individually. Finally a conclusion will wrap up the chapter.

3.1 Functional and Non-Functional Requirements

There are several requirements that this system must do in order to function correctly while also keeping the interaction between the user and the system simple and efficient.

Functional Requirements:

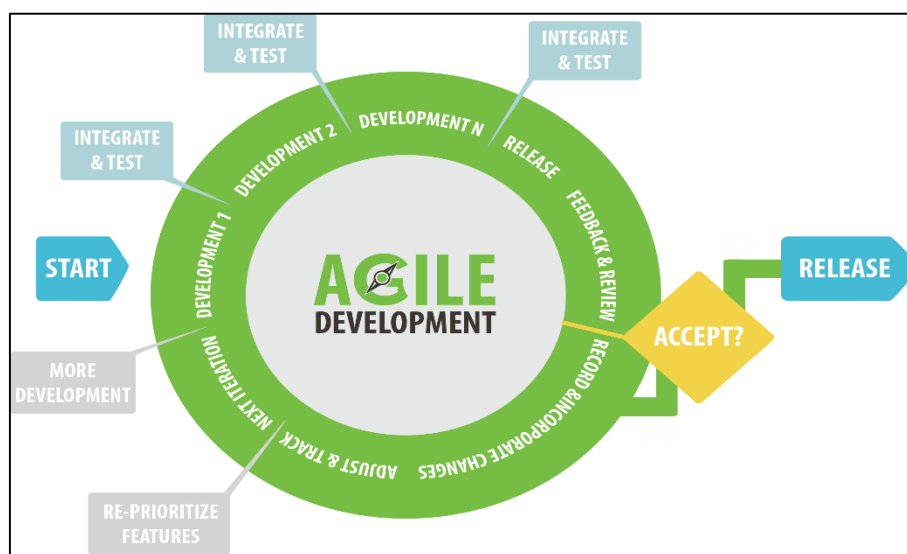
- A user must sign up for the weather app
- User can login and out of the system
- E-mail will be sent for account activation and password reset
- User can enter any location for weather data
- User can ask for weather prediction
- Weather prediction must be easy to understand
- Database will still collect and delete data even when not being used.

Non-Functional Requirements:

- Accessibility
- Usability
- Documentation
- Reliability
- Scalability
- Portability
- Maintainability

3.2 Methodology

The methodology chosen for this web app is the agile software development methodology. The main reason for choosing this methodology is the flexibility it can provide and its emphasis on prototype creation and regular testing. This web app has various pieces that are not directly connected meaning they must communicate over a certain medium. That may be HTTP for API requests or SMTP for emails with activation codes for registration to the site. This is why agile development is the best choice as its emphasis on testing of individual components on a regular basis ensures errors are discovered quickly. The traditional waterfall model was considered as the choice of methodology but there was one major reason that it was not chosen. The waterfall model requires that the system be completed and assembled (build phase) in some way before testing. The fact that this system is receiving data from an outside source means that regular testing of this data and the functions within the web app that will process and display this data are also tested regularly. Regular testing of individual components in the system such as login feature ensure detection of errors are relatively simple using agile development methodology.



3.2.1 Agile Software Development

3.3 Platform & Design Layout

The majority of this web app has been coded in python, specifically Django. Django is an MVC framework that is used to create complex web apps with most of the basic coding and features shortened in order to speed up the process of development which in turn will speed up testing and deployment. The web app follows this Django MVC framework. There are certain files that are generated during the start-up of a Django project that are essential to this speedy process. These are the Models.py, Views.py and URLS.py. These three files handle a lot of the functionality of this project. Models.py holds all the models that will be needed for the web apps database. Models are translated from Django into the relevant SQL code to create, insert, delete or update you database tables. Views.py handles most of the specific parameters for the web app. These would include granting permissions if a user is logged in or posting data to a database. The URLS.py holds the necessary URLs that will be used to redirect the user when they select a link on the web app. Another important feature in this web app and Django in general is the use of templates. Templates are HTML files that will specify how the front end or User Interface (UI) will look and how it will act. Django allows extensions of other HTML files. This means you do not need to continually write the same code to have a fluid and uniform web app. smaller parts of the project include Cron that would be used to schedule jobs such as receiving the necessary data and then posting to the database. Django-extensions can also be used to schedule these tasks which will make it easier to keep track of the data collection and transfer.

3.4 User Interface

The User Interface for this web app is written in html and a mix of Django specific fields and extensions. There are also several extra styles and forms that make the User Interface easier to use and navigate as well as giving it a complete, professional look and feel. The base html code is based on bootstrap. The layout is a simple navigation bar on the top of the screen with the various other forms of content under this. The navigation bar holds the web app name a home button an about section and a contact page. On the right side of the navigation bar there is a login/out and register feature. The web app requires you to register or login to access certain features of the web app such as current weather, weather forecast, user location and adding of files and a useful help section. The majority of the User Interface is written as html files in the templates section of the Django folder structure. This allows for extensions and ease of access for Django and for future developers.

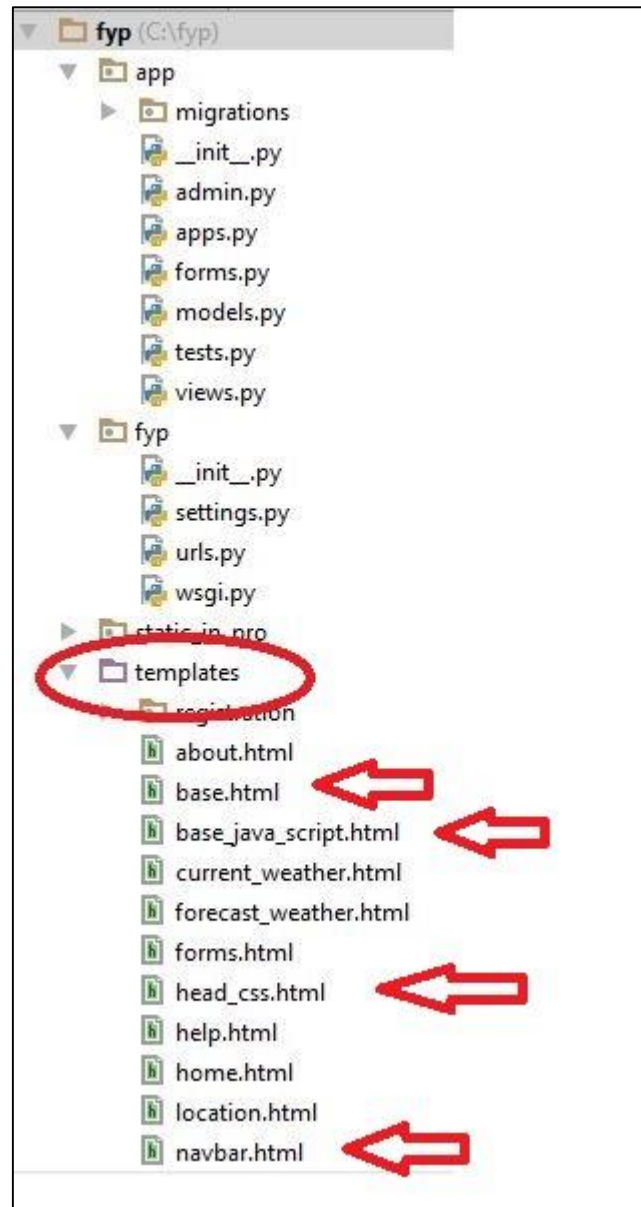


Figure 3.4.1 Django directory structure

The above figure shows the template directory in the fyp directory structure. The User Interface was designed using as efficient use of Django features as possible. This meant creating some “Base” html files that would then be extended or inherited by every other html file. The above figure also shows the base html files with arrows beside them. These are base.html, navbar.html, head_css.html and base_java_script.html.

```

<body style="padding-bottom: 40px">
{% include 'navbar.html' %}
<div class="container">

    {% block jumbo %}

    {% endblock %}

    {% block content %}

    {% endblock %}

</div>
{% include 'base_java_script.html' %}
</body>
/html>

```

*Figure 3.4.2 the base html file that will be
Extended throughout the web apps usage.*

The base.html file holds how the structure of the body will look and behave. The body has a bottom padding of 40 pixels. This design is used because otherwise any content would be at the very bottom of the page making it confusing and unprofessional. Next is the Django includes. This specifies what file should be included in the file which happens to be the html file that holds the navigation bar. This is the first example of a reusable design in the web app. The body of the html file contains a div. The div is classified as a container meaning everything inside of this div will not extend beyond its boundaries. Inside this div are two content blocks. Content blocks in Django separate parts of the containing structure which in this case is a div into customisable blocks that will not interfere with each other. Blocks can also be customised in a separate CSS file or in the head of the html document. Then below the containing div is another include that specifies to include another html file that holds as the name suggests JavaScript. Later in this section of the project you will see this base.html file being used a lot in all other html files.

Navbar.html holds the code that specifies what is in the navigation bar. The navigation bar holds the home, about, contact, login/logout and register features. Each one of these redirects the user based on the {% url 'name' %} in the form of a href. These URL names are linked back to the urls.py file which handles how forms and links should act and what the URL should display.



Figure 3.4.3 navigation bar when logged out

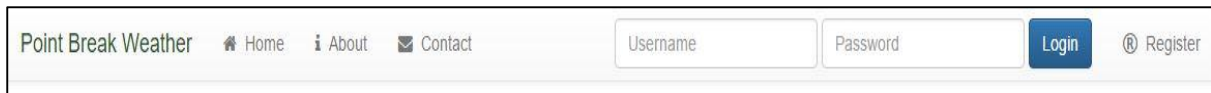


Figure 3.4.4 navigation bar logged in

When a user is logged out there will not have access to the core of the site. There will be a large message asking you to login to gain access to these features. Once logged in you may use said features.

```
<!--[if lt IE 9]>
  <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
  <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
<![endif]-->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" integrity="sha384-1q8mTJQq"
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css" integrity="sha384-f
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css">
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.css" />
<link rel="stylesheet" href="http://makinacorp.us.github.io/Leaflet.FileLayer/Font-Awesome/css/font-awesome.min.css"/>
<link rel="stylesheet" href="http://k4r573n.github.io/leaflet-control-osm-geocoder/Control.OSMGeocoder.css"/>
<link rel="stylesheet" type="text/css" href="leaflet-openweathermap.css" />
<script src="http://makinacorp.us.github.io/Leaflet.FileLayer/leaflet.filelayer.js"></script>
<script src="http://makinacorp.us.github.io/Leaflet.FileLayer/togeojson/togeojson.js"></script>
<script src="http://leaflet.github.io/Leaflet.heat/dist/leaflet-heat.js"></script>
<script src="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.js"></script>
<script src="/js/leaflet-0.7.2/leaflet.ajax.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js" integrity="sha384-0mSbJDEHialfmuBBQP6A4Qrprc
```

Figure 3.4.5 code snippet from head_css.html

The head_css.html file holds the necessary link stylesheets and scripts that allow the various styles and actions occur. The links hold stylesheets for both bootstrap and leaflet. Leaflet is similar to google maps in it allows you to create heat maps, popups, polygons, and polylines on an interactive map. The scripts then hold JavaScript files that allow certain actions to perform such as opening a GeoJson file or search a location anywhere on earth.

```
<!-- Bootstrap core JavaScript
===== -->
<!-- Placed at the end of the document so the pages load faster -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="...'./assets/js/vendor/jquery.min.js"></script>')</script>
<script src="...'./dist/js/bootstrap.min.js"></script>
<!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
<script src="...'./assets/js/ie10-viewport-bug-workaround.js"></script>
```

Figure 3.4.6 html JavaScript file for extension

The final core html file is base_java_script.html. This file holds only four scripts but are very important. This file is placed at the end of every page as it will make them load faster. It also holds scripts that allows jQuery and Ajax to be performed if necessary.

The rest of the files for the user interface are also html code with Django. These include home.html, current_weather.html and so on. The files current_weather.html, forecast_weather.html and location.html all contain JavaScript for performing tasks related to map display in leaflet. An example would be on the location.html file. The following example loads a base layer that any other additions will be added to. The first line sets a variable called “map” where the centre of the screen will be when the page is first loaded and how far out it will be. All other features will then be added to the variable map. The last section of code specifies a folder button which will allow the user to load a GeoJson file to the map.

```
var map = L.map('map').setView([53.2738, -9.0518], 6);

// load a tile layer
var osm = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png',
{
    maxZoom: 16,
    minZoom: 4
}).addTo(map);

var style = {color:'red', opacity: 1.0, fillOpacity: 1.0, weight: 2, clickable: false};
L.Control.FileLayerLoad.LABEL = '<i class="fa fa-folder-open"></i>';L.Control.fileLayerLoad({
    fitBounds: true,
    layerOptions: {style: style,
        pointToLayer: function (data, latlng) {
            return L.circleMarker(latlng, {style: style});
        }
    },
}).addTo(map);
```

3.4.7 JavaScript code example in location.html

3.5 Database

The database that was used for this web application was PostgreSQL. The reason for this is PostgreSQL is a very powerful database when it comes to large amounts of data and is very useful for GIS data which is the main form of data that would be stored in the database for this web app. During the development of this web app most of the coding and testing were run on a local machine meaning there was no worry about networking. However the database from the very beginning was being run on an offsite Linux server, specifically Ubuntu. In most cases this would frustrate the development and testing of the code, however Django makes connecting to the database very simple. There is a file called settings.py that handles various things. These include specifying a session id, database details, sending emails and OS path. When a new project is created in Django a database connection is automatically created.

The database is SQLite. This was not suitable for this web app as stated above due to the nature of the data. As a result new connection parameters were added to connect to the PostgreSQL database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'Test3',
        'USER': 'postgres',
        'PASSWORD': 'network',
        'HOST': 'pointbreak.cloudapp.net',
        'PORT': '',
    }
}
```

3.5.1 Database connection details

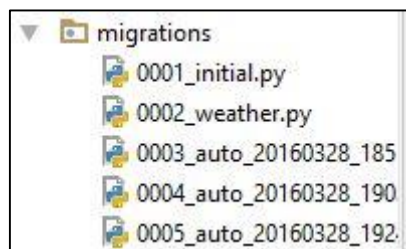
However extra packages are needed in order to connect to the database and create tables, exchange data, etc. This package is psycopg2. Psycopg2 is a DB API from PostgreSQL that allows the integration of Python with PostgreSQL. Once this module is installed you can connect to the database. In Django the modules.py file makes the creation, deleting and updating of tables very simple. Django models are written in a way that Django can understand but Django will then translate this into the relevant SQL code so PostgreSQL can perform the required tasks. Django uses special “fields” for the specification of the data. An example would be entering a name into the database. This parameter would be specified as a Charfield. Inside fields there are extra properties that can be added such as max length and null = True. Below is an example of a table created in this web app. The table will hold weather data that can then be queried later.

```
class WeatherDetails(models.Model):
    location_name = models.CharField(max_length=30, null=True)
    lat = models.DecimalField(max_digits=9, decimal_places=6, null=True)
    long = models.DecimalField(max_digits=9, decimal_places=6, null=True)
    clo = models.IntegerField(null=True)
    temp = models.IntegerField(null=True)
    temp_min = models.IntegerField(null=True)
    temp_max = models.IntegerField(null=True)
    pressure = models.IntegerField(null=True)
    humidity = models.IntegerField(null=True)
    weather_details = models.CharField(max_length=40, null=True)
    description = models.CharField(max_length=100, null=True)
    wind_speed = models.FloatField(null=True)
    wind_degrees = models.FloatField(null=True)
    date_time = models.IntegerField(null=True)

    def __unicode__(self):
        return self.location_name
```

3.5.2 Table that will hold weather information

The `null = True` parameter specifies that there can be nothing in this table. This makes the creation of the table simple and means less errors when placing the weather data into the database. Once this model has been created Django's functionality is used once again and makes the process quick and simple. The command `python manage.py makemigrations` is run. This command checks the `models.py` file and will detect any changes to the models based on the previous "migration" to the database. This could include a new model, an update on an existing model or deleting part of a model. Once Django detects any changes it places the migration in the migration directory and creates a file stamped with the date and time of the detection. Then the command `python manage.py migrate` will communicate with PostgreSQL and make the changes in SQL on the database side. This web app follows this procedure to ensure a secure and up to date database.



3.5.3 Django migrations to database

3.6 Views & Forms

The `views.py` file is the backbone of this web app and any other Django application. This file is generated automatically when the project is started. The `view.py` file is used in this project to display web content, post and get data from an API or the PostgreSQL database, display content based on the user and so on. Django views are basically python functions that will perform said tasks. These functions or any code that will perform the above tasks can be placed anywhere on the python path but Django gives us a pre-generated file that makes it much easier us and Django to find and execute. In this web app all of the necessary functions for performing the various tasks that will be needed are placed in `views.py`. Most of the time a view will simply return a render to the linked html file for the User Interface.

```
return render(request, 'home.html')
```

3.6.1 An example of a view function rendering

The home html file

However most of this web app requires sending and receiving of information meaning query sets will be placed in these functions or in the case of sending correspondence a form will be “cleaned” and saved in the function. Forms will be covered in more detail later.

The web app uses query sets a lot as they retrieve or send objects to your database. As this is a weather web app the user cannot make their own weather so the query sets will only retrieve information in the form of objects from the database. Basically a query set is like a “select” statement in SQL and the filters used are limiting clauses such as a “where” clause in SQL. The main query sets used in this web app are retrieving of geographic co-ordinates specifically latitude and longitude and weather details such as temperature and air pressure to name a few. The API that is being used to get this information is OpenWeatherMap. The information that is received is all relevant to most users and can also help others who want specific information. As a result the information that is returned in the query sets in the web app are only limited by a latitude/longitude co-ordinate or a range of co-ordinates. An example of this is retrieving current weather for Dublin.

```
WeatherDetails.objects.filter(lat='53.350140', long='-6.266155')
```

3.6.2 Weather query

Weather Details is the model that holds the data, objects specifies retrieve all objects in the table and filter basically filters the results in this case the co-ordinates for Dublin.

Earlier I spoke about views holding forms. Forms are ways of sending user input to a database or retrieving input to send via email or another method. There are two ways to create a form one is to inherit from a model which means the inputted data will be saved to the database or create a form on its own and take and clean the data in the view. The below example is a signup form for the web app.


```
def home(request):
    title = "Welcome"
    form = SignUpForm(request.POST or None)
    context = {
        "template_title": title,
        "form": form
    }

    if form.is_valid():
        instance = form.save(commit=True)
        instance.save
        context = {
            "template_title": "Thanks"
        }

    return render(request, 'home.html')
```

3.6.3 A model form

The example is specifying that the form is SignUpForm in the forms.py file. SignUpForm directly inherits from the model and is specified in the context variable which will then be passed to the return and can then be used in the home.html file. The next example is the contact form. This form is specified without the use of a database in forms.py

```
class ContactForm(forms.Form):
    full_name = forms.CharField(required=False)
    email = forms.EmailField()
    message = forms.CharField(widget= forms.Textarea)

class SignUpForm(forms.ModelForm):
    class Meta:
        model = SignUp
        fields = ['fullname', 'email']

    def clean_email(self):
        email = self.cleaned_data.get('email')
        email_base, provider = email.split("@")
        domain, extension = provider.split(".")
        #if not extension == ".com":
            #raise forms.ValidationError("Please enter an email with .com ")
        return email

    def clean_full_name(self):
        full_name = self.cleaned_data.get("full_name")
        return full_name
```

3.6.4 Example of form models and forms

The contact forms objects are specified in the forms.py folder unlike SignUpForm which inherits from models.py. This means it will not be saved in the database. When a form is inheriting from a model the cleaning of the data and error checking is done in the forms.py file but if it is just a form this is done in views.py.

```
def contact(request):
    title = 'Contact Us'
    form = ContactForm(request.POST or None)
    if form.is_valid():
        form_email = form.cleaned_data.get('email')
        form_message = form.cleaned_data.get('message')
        form_full_name = form.cleaned_data.get('fullname')

        subject = 'Site contact form'
        from_email = settings.EMAIL_HOST_USER
        to_email = ['adamnoonan93@gmail.com']
        contact_message = "%s: %s via %s"%(
            form_full_name,
            form_message,
            form_email)

        send_mail(subject, contact_message, from_email, [to_email], fail_silently=False)
    context = {
        "form": form,
        "title": title,
    }
    return render(request, "forms.html", context)
```

3.6.5 Cleaning form data in view

3.7 Emails

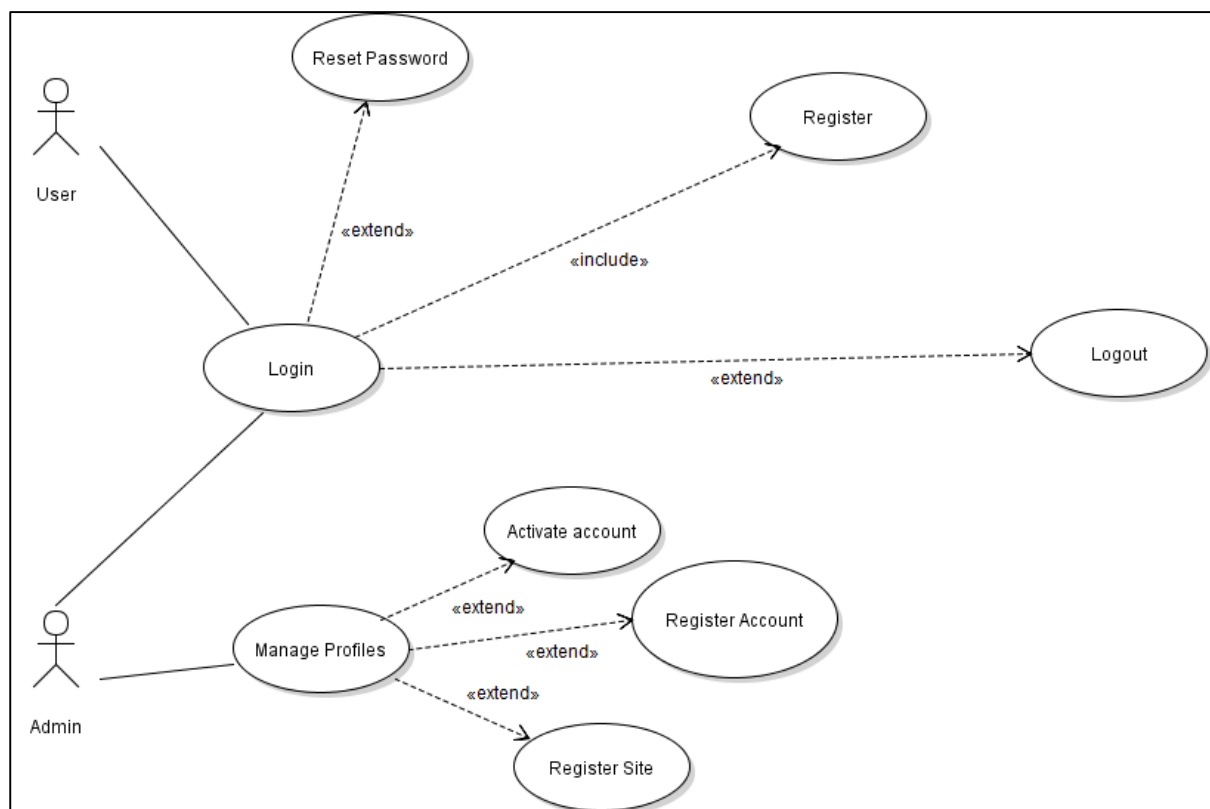
Sending emails to the admin of this site seemed like a necessary feature. This web app is open source and anyone may use any part of the project freely. In order to keep up regular contact with users of the site a contact form was created as previously stated. This will then send emails to the specified email address. In order for this to happen some configurations to the settings.py file was needed. This meant specifying what email backend the site would use. Gmail SMTP was selected and the host, password, port and other information was specified.

```
ALLOWED_HOSTS = []
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'adamnoonan93@gmail.com'
EMAIL_HOST_PASSWORD = '*****'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
```

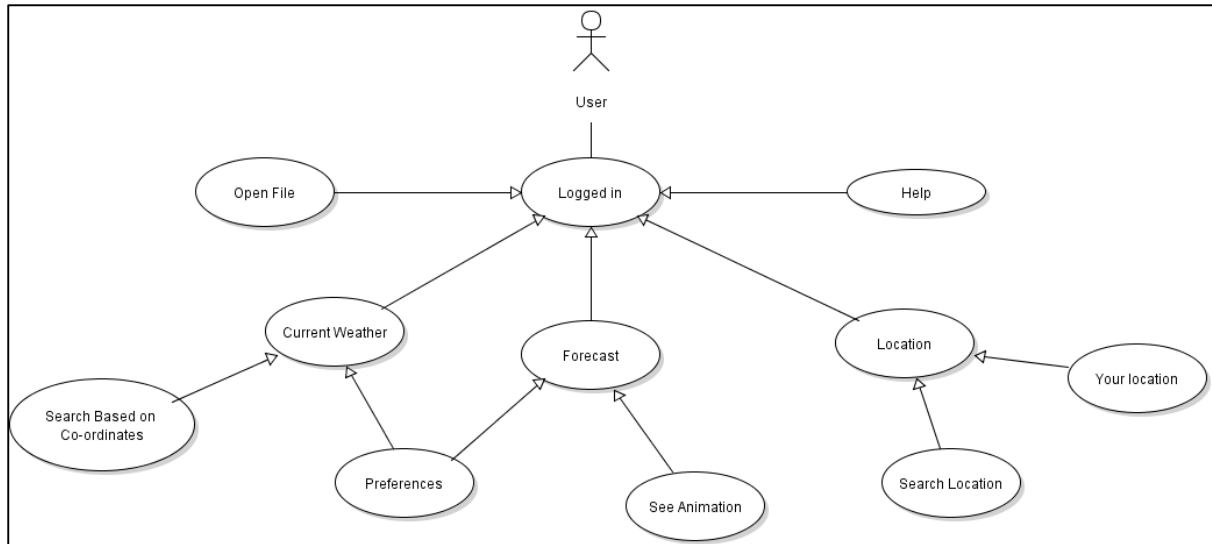
3.7.1 Enabling Email usage

This allows for more professional use of the site. Registering to the site, changing password etc. will now send an email to the email address specified in the profile with an activation code. In development on the localhost this would not work as the domain was set to example.com thus redirecting to a publicly owned domain for educational purposes.

3.8 Use-case Diagrams

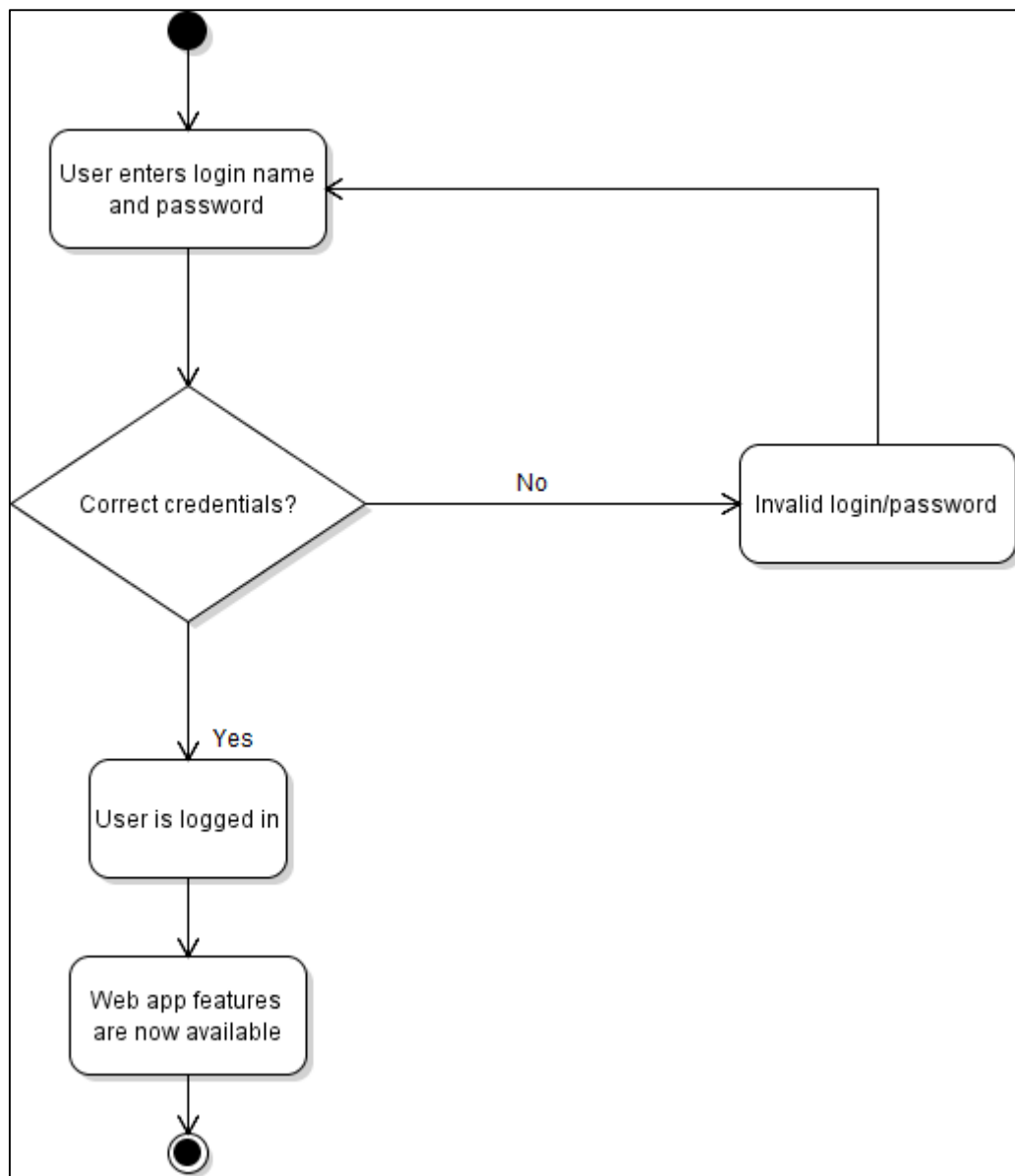


3.8.1 Login/Register use-case of the web app

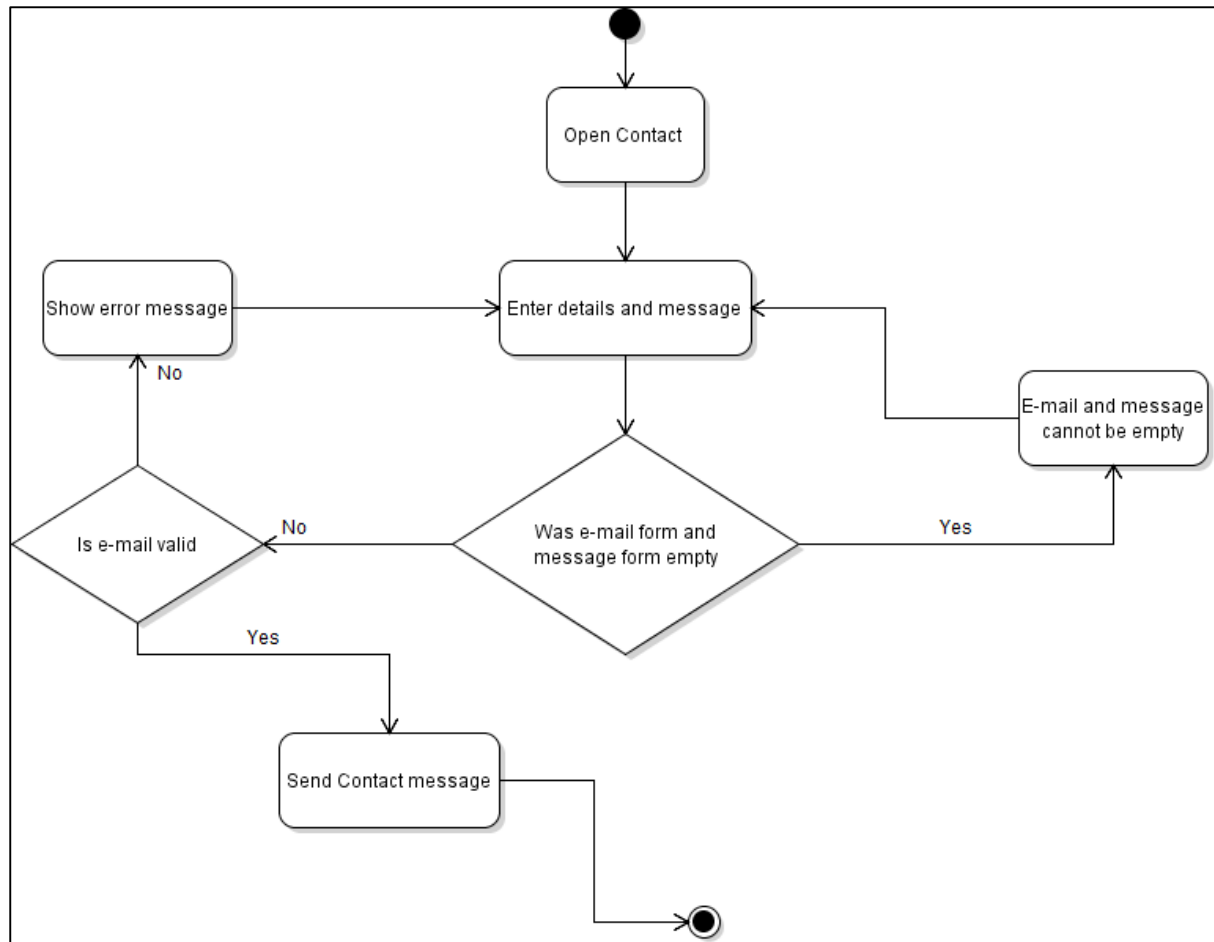


3.8.2 Use-case describing what a user can do once they have logged in

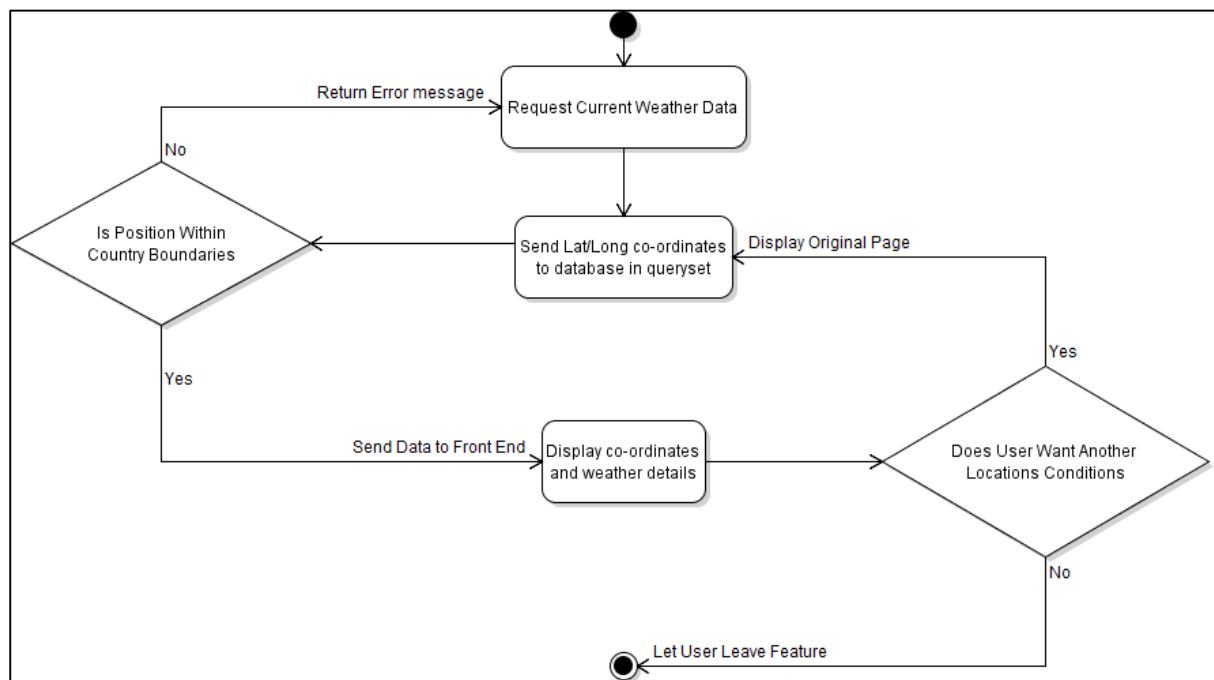
3.9 Activity Diagrams



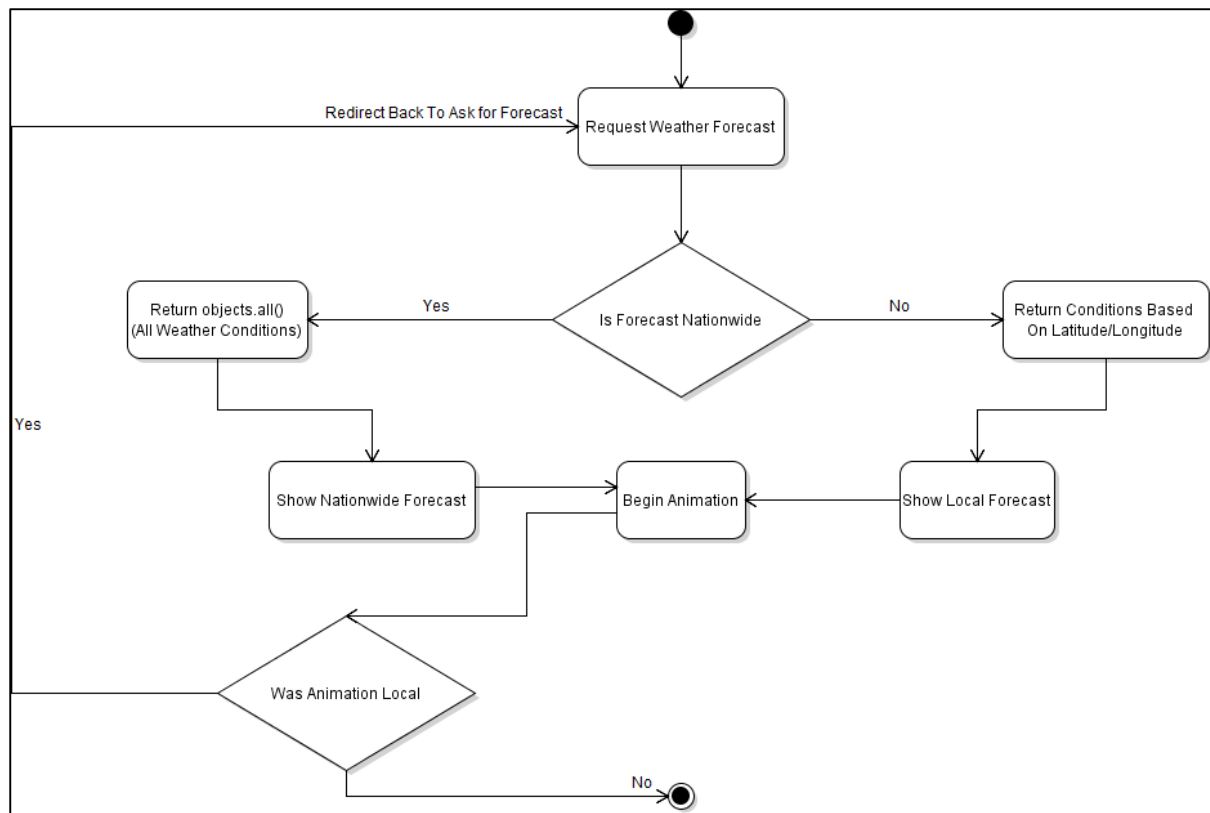
3.9.1 Activity Diagram for login



3.9.2 Contact form Activity Diagram



3.9.3 Activity Diagram for current weather conditions



3.9.4 Activity Diagram for weather forecast

Conclusions

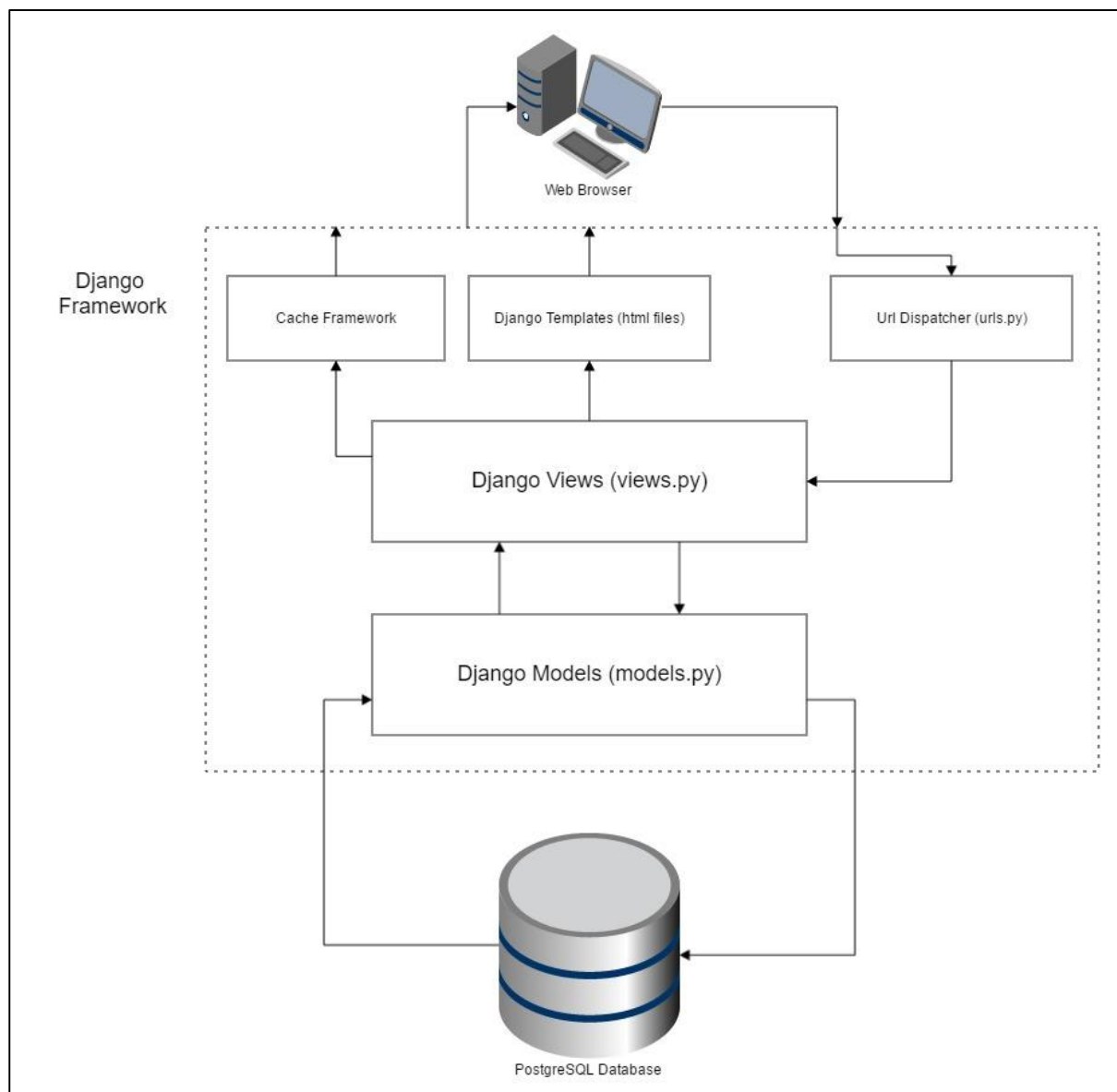
This section of the document discussed the various sections of the app, their design, how they function and in some cases how they interact with each other. These included the User Interface, Django models, Django views and forms and sending emails over SMTP. The various use-cases, activity diagrams, design architecture and class diagrams were also shown with further descriptions of how the web app functions.

4. Architecture and Development

Introduction

This section of the document will discuss the architecture of the web app. There will be a system walkthrough that will serve as a user guide to the system, a brief description of each functions capabilities and how the inner code works. The last part of this section will discuss problems or challenges faced over the course of the project and how they were overcome.

4.1 System Overview



4.1.1 Architectural Diagram of how this system functions

The above diagram shows how the web app architecture looks and how each component functions and communicates with other parts or “layers” of the system. The web browser can

be anything from google chrome to Firefox it doesn't matter to Django or this web app. The user will see like any other web app an html page. This page is the home page with various URLs that link this page to all of the other parts or features of the web app. When a user clicks on one of these the URL sends the request to the file Views.py. This file is the "Controller" of the Django framework. This file takes the URL in question and performs certain tasks before loading the corresponding html file. These tasks could be cleaning form data, requesting data from a database or displaying error messages or redirection if there is a network error. If the function in Views.py is simply a rendering of an html file it then sends this file from the templates section of the Django framework. Templates are html files that constitute the "View" of the framework. They can consist of plain html or can have extra parameters or arguments specific to Django that will allow certain things such as displaying URLs or content if a user is logged in or denying a URL if the user is logged out. Going back to what was said in the Views.py file if the function is requesting data then it sends this request to the Models.py file. Models.py constitutes the "Model" section of the Django framework. It communicates with the database. This "communication" could be posting or "getting" data from the database, creating, deleting or updating tables in the database or the objects within those tables. Django allows you to perform the creation, deleting and so on of tables and objects using certain fields such as Charfield or Decimalfield. Django then turns this into the corresponding SQL code so the database can perform the tasks. Once the data has been sent to or received from the database it then tells the Views.py file that the task is complete which then in turn renders this in the html template file for the user to see. Due to Django's dynamic nature the cache framework will store previously rendered template files with the corresponding data in case the current one cannot be reached. Next there will be a detailed walkthrough of the system by an average user. This will serve as a guide on the web app and will also allow more in depth detail of the system architecture and code.

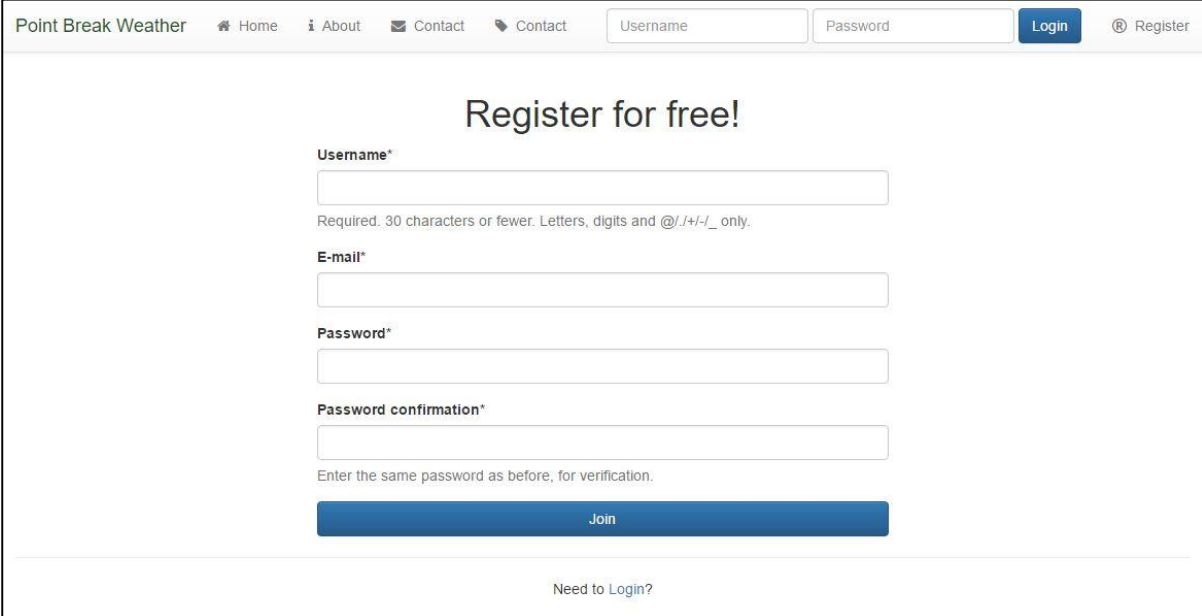
4.2 System Walkthrough (Implementation)

IMPORTANT: Some of the features described in this system walkthrough are not fully implemented or will only work on my local machine as I was unable or simply ran out of time to incorporate it into the web app. There will be a red mark besides the heading that fits this category (*).

4.2.1 Register

The first step that a user of this site must do is register to the site. The core features of the site will not be available unless the user is registered and logged in to the site. On the right side of the navigation bar there will be a login section and a register section. The user will click on the register URL and will be directed to the registration page. Once they are at the registration page they will be

asked to enter a username, a valid email address and a password that will be entered twice to ensure it is correct. All of the fields must be entered if not the field(s) will highlight in red and say this field is required. Once the user enters the credentials an activation email will be sent to the email address specified. The user will click on the link and they will then be able to log in to the site and use the various features.



The screenshot shows the registration page of a website titled "Point Break Weather". The navigation bar at the top includes links for Home, About, and Contact, along with input fields for Username and Password, and buttons for Login and Register. The main heading is "Register for free!". Below this, there are four required input fields: Username, E-mail, Password, and Password confirmation. Each field has a red asterisk indicating it is required. The Username field has a hint: "Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only." The Password confirmation field has a hint: "Enter the same password as before, for verification." A blue "Join" button is positioned below the fields. At the bottom, there is a link that says "Need to Login?".

4.2.1 Registration page

4.2.2 Login

Once the user is registered they can then login to the site. This can be done at the bottom of the register page or by simply entering their username and password into the two fields on the right side of the navigation bar. The principle is the same as the register page both fields are required to login and if the credentials are wrong an error message will appear.

Login

- Please enter a correct username and password. Note that both fields may be case-sensitive.

Username*

Password*

Submit

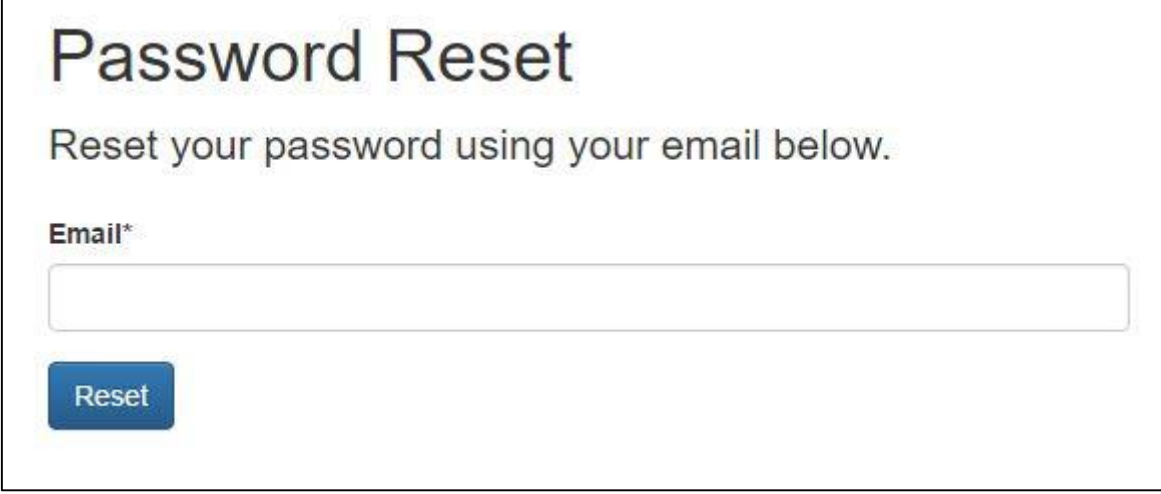
Forgot password? [Reset it!](#)

Not member? [Register!](#)

4.2.2 Incorrect login details

4.2.3 Password Reset

At the bottom of the login page a user can reset their password if they have forgotten it. They will be redirected to a new page where they will enter their email address and an email will be sent to click a link that will allow them to reset or change their password.

A screenshot of a web form titled "Password Reset". Below the title is the instruction "Reset your password using your email below." There is a text input field labeled "Email*" and a blue button labeled "Reset".

Password Reset

Reset your password using your email below.


Email*

Reset

4.2.3 Reset password

4.2.4 Logout

Once the user is logged in they can logout by simply clicking the logout button which will now replace the register and login links on the right of the navigation bar. Once it is clicked it will redirect them to a page confirming the logout was successful.

A screenshot of a web page showing a success message "You have successfully logged out." with a blue "Login" link below it.

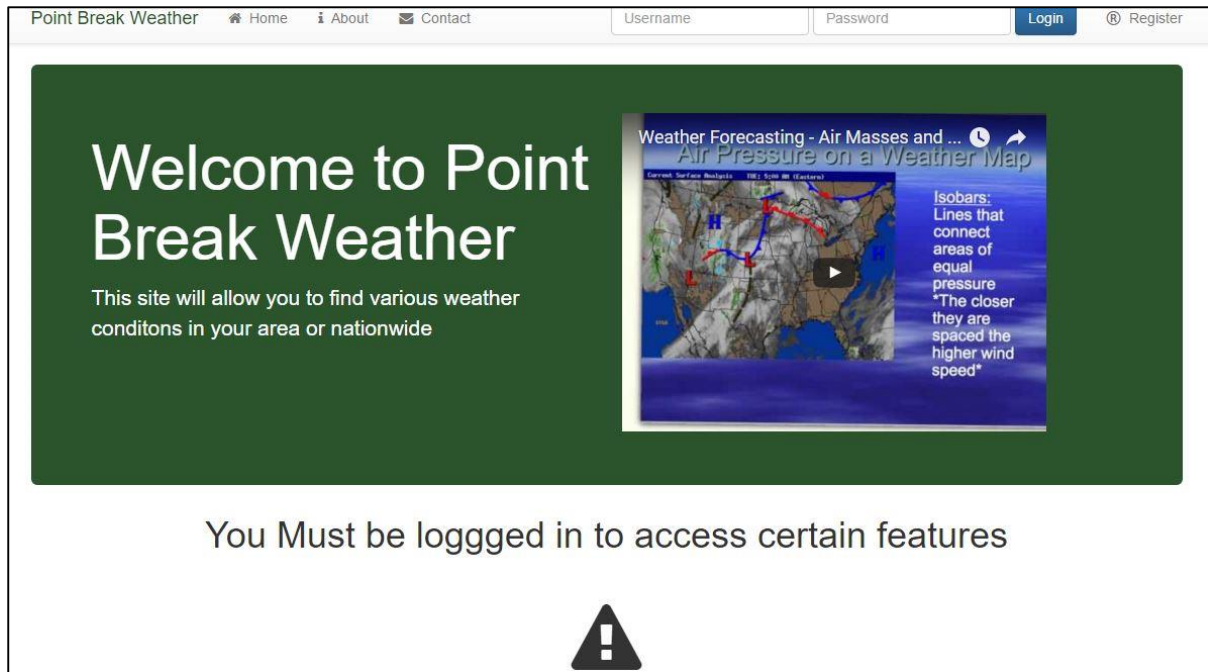
You have successfully logged out.

Login

4.2.4 Logged out success page

4.2.5 List of Features

Now that the user is logged in they can now avail of the web apps various features. If the user was logged out there will be a notice instructing the user to login in order to see and use these features.



4.2.5 Home page when user is logged out

However once they login they can now see and use the features. These include getting the current weather, seeing a weather forecast for the next day, opening a local GeoJson file and finding a location if needed and finally adding places to your preferences. These features will be explained in detail later in this section.

4.2.5 Navigation Bar Features

On the navigation bar there are four features that can be used regardless of being logged in or out of the system. These are the home feature, about the site, a form of contact with the sites administrators and a help section.



4.2.6 Navigation bar

The home feature simply redirects the user to the home page wherever they are in the web app. The about section of the web gives a brief description of the site and a link to the contact feature if they have any questions that they wish to clarify. The help section has an interactive map that shows the best surfing locations in Ireland as well as useful tourist attractions

around the country. This feature allows the users to plan their day based on weather and where they want to see during their stay in the country or if they simply wish to surf.



4.2.7 Help section of navigation bar

The user can select the icons and a description will appear explaining what the location is and the best conditions to visit such as wave direction for surfing. Finally the contact section is a way for users of the site or users interested in using the site to send contact details to the admins of the site.

Contact Us

Full name

Email*

Message*

Submit

4.2.8 Contact Page

At the moment the only address that the user can send to is the email address specified in the Django views.py file under the function contact which will be described in further detail next.


```
def contact(request):
    title = 'Contact Us'
    form = ContactForm(request.POST or None)
    if form.is_valid():
        form_email = form.cleaned_data.get('email')
        form_message = form.cleaned_data.get('message')
        form_full_name = form.cleaned_data.get('fullname')

        subject = 'Site contact form'
        from_email = settings.EMAIL_HOST_USER
        to_email = ['adamnoonan93@gmail.com']
        contact_message = "%s: %s via %s"%(
            form_full_name,
            form_message,
            form_email)

        send_mail(subject, contact_message, from_email, [to_email], fail_silently=False)
    context = {
        "form": form,
        "title": title,
    }
    return render(request, "forms.html", context)
```

4.2.9 Contact Function

The function inherits the layout of the form from the (Contact Form) function that is specified in the forms.py file. It is then placed into the form variable. Next the function uses if statement to check if the form is valid. If the form is a valid one then the data in each field is cleaned by taking the value specified in single quotes. Next to send the actual email we must use Django send_mail. This holds certain parameters that must have content or be specified. These are the email subjects which is a string, the email being sent from which is obtained from the settings.py file, where the email will be sent and the contact message which will be the input obtained in the cleaned data section of the if statement. Finally the fail silently parameter is set to false which means if the email is unable to send display an error. This is how the contact portion of the web app functions.

4.2.6 Preferences

The preferences function is located on the right of the main features section on the home page. Simply click on this and the user will be redirected to the page. The user will see a map and two fields to fill in. All fields are required in order to proceed which includes clicking on the map to get the latitude and longitude points. The function that handles the preferences tasks is the preference view in views.py.

```

def preference(request):
    if request.method == 'Post':
        form = PreferencesForm(request.POST)
        if form.is_valid():
            new_point = Preferences()
            cd = form.cleaned_data
            coordinates = cd['coordinates'].split(',')
            new_point.geom = Point(float(coordinates[0]), float(coordinates[1]))
            new_point.place_name = cd['place_name']
            new_point.description = cd['description']

            new_point.save()
            return HttpResponseRedirect('/preference/success')
        else:
            return HttpResponseRedirect('/preference/error')

    else:
        form = PreferencesForm()

    args = {}
    args.update(csrf(request))
    args['form'] = PreferencesForm()

    return render(request, 'preferences.html', args)

```

4.2.10 Preferences View Function

The initial if statement basically says that if the request is a post which means you are going to post data to the database then continue with this function and place the inherited form into the value form. The only purpose for the preferences function is posting of data but this is required nonetheless. The next if statement continues the function if the form is valid. If the form is valid the model to which the data will be posted is placed in the new_point value. The cd value is used as a variable for the form clean function. The next three steps take in the data from the form, clean the data and place the form data into the corresponding objects within the Preferences table. Finally the form data is saved so it may be posted. The final line before the return basically places the variable inherited form from forms.py and places into args as a dictionary. The next diagram shows the JavaScript used for the map.

```

<script>
  var map = L.map('map').setView([53.2738, -9.0518], 6);

  var osm = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png',
    {
      maxZoom: 16,
      minZoom: 4
    }).addTo(map);

  function onMapClick(e) {
    var lat = e.latlng.lat;
    var lng = e.latlng.lng;

    if (typeof marker != 'undefined') {
      map.removeLayer(marker); // delete previous marker
      marker = L.marker([lat, lng]).addTo(map); // add new marker
    }
    else {
      marker = L.marker([lat, lng]).addTo(map); // add new marker
    }

    $('#coordinates').val(lng + ', ' + lat)
  }

  // call the onMapClick function when user click on map
  map.on('click', onMapClick);
</script>

```

4.2.11 Preferences JavaScript

The first line of the script specifies where the map will be centred when it is loaded and how far in it should be zoomed. The variable osm loads an OpenStreetMap picture as the base layer of the map. It then specifies how far in and out a user can zoom and adds it to the base layer. The function allows the user to click on the map and the latitude and longitude will be stored so it can be handled by the view function above. The statement deletes an old marker if the user clicks on the map again. This is an overview of how the preferences section of the web app functions and how a user can navigate through it.

4.2.7 Location and File Opener *

This feature allows a user to find any location on the earth by searching the name in the search bar. The can also load a GeoJson file from their local machine which will load it on the interactive map. This feature is entirely JavaScript and utilises the Leaflet plugin which allows for the creation of fluid and interactive maps. The map is initially set over Ireland. The file opener feature is located just under the zoom options and will allow the user to open a

GeoJson file to render on the map. The user can then in the bottom left corner of the screen type in any location and the map will immediately go to that location.



4.2.12 File opener and location page

The file opener will open a dialogue box. The user can then select the file to load. The map will immediately zoom to the location and display the information. This could be polylines, points or heat maps. This feature is still untested as a GeoJson file on my local machine does not render the points that are contained in the file. This will be described later in the development section. The search option functions perfectly and will find anywhere as long as the name is correct. The search box is hard to see and the input is not visible. However any change to the style breaks the map so it must be kept the way it is. Below is the JavaScript that allows the two functions to work.

```

var map = L.map('map').setView([53.2738, -9.0518], 6);

// load a tile layer
var osm = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png',
{
    maxZoom: 16,
    minZoom: 4
}).addTo(map);

var style = {color:'red', opacity: 1.0, fillOpacity: 1.0, weight: 2, clickable: false};
L.Control.FileLayerLoad.LABEL = '<i class="fa fa-folder-open"></i>'; L.Control.fileLayerLoad({
    fitBounds: true,
    layerOptions: {style: style,
        pointToLayer: function (data, latlng) {
            return L.circleMarker(latlng, {style: style});
        }
    }
}).addTo(map);

var osmGeocoder = new L.Control.OSMGeocoder({
    collapsed: false,
    position: 'bottomleft',
    text: 'Locate'
});
map.addControl(osmGeocoder);

```

4.2.13 File opener and location JavaScript

4.2.8 Weather Forecast *

This section is the main section of the web app. The user will be able to select a forecast for the next day. They can display certain weather features to display and animate. This will include temperature, wind speed, wind direction, air pressure and so on. Once the user selects the feature they want the web app will take the forecast from the database or an external API if the data is not available and animate it. The information will then be run through the Inverse Distance Weighting function. The method of calculation is shepherds method. This will take in the values x, y and z. the x value will be the latitude on the map and y will be the longitude. The z value will consist of the weather condition that will be displayed on the map. This can be temperature, air pressure and so on. Once the calculation is finished matplotlib will be used to display the calculation on a graph. The matplotlib section will be inside for loop. The reason for this is that each iteration of the loop will represent a point of time in the day when the data forecast is accurate. In this case each iteration will represent an interval of two hours over the course of the next day. This will produce a png file for every iteration of the loop. The reason for a png file is that png is easier for web development purposes. The png files will then be placed together to form a gif or another form of animation such as an mp4 file. This will then be overlaid on the leaflet interactive map. The best way to show this would be to set the opacity of the animation so you can see the background map as well as the information being shown.


```
# -*- coding: utf-8 -*-

from __future__ import division
import numpy as np
import matplotlib.pyplot as plt

def iwd(x,y,z,grid,power):
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            distance = np.sqrt((x-i)**2+(y-j)**2)
            if (distance**power).min()==0:
                grid[i,j] = z[(distance**power).argmin()]
            else:
                total = np.sum(1/(distance**power))
                grid[i,j] = np.sum(z/(distance**power)/total)
    return grid

#np.random.seed() # GIVING A SEED NUMBER FOR THE EXPERIENCE TO BE REPRODUCIBLE
for i in range(10):
    #with io.open("file_" + str(i) + ".png", 'w', encoding='utf-8') as f:
    grid = np.zeros((100,100),dtype='float32') # float32 gives a lot precision
    x,y = np.random.randint(0,100,10),np.random.randint(0,100,10) # CREATE POINT SET.
    z = np.random.randint(0,10,10) # VARIABLE

    grid = iwd(x,y,z,grid,2)
    plt.imshow(grid.T,origin='lower',interpolation='nearest',cmap='jet')
    plt.scatter(x,y,c=z,cmap='jet',s=0)
    plt.xlim(0,grid.shape[0])
    plt.ylim(0,grid.shape[1])
    plt.grid()
    #plt.show()
    plt.savefig('test ' + str(i) + '.png')
```

4.2.14 Inverse Distance Calculation and File Creation

This section of the project was not fully completed. The function does not read in data from my database but randomly selects the positions and values. The reason for this is to simply prove that the calculation works and would also work if the data was read from a database. The code is not in the Django framework it is running on the local machine that the programme is saved on. The png files are created correctly and will overwrite previous files if the programme is run again. The package ffmpeg was installed to handle the animation. However this only seems to work if the command is entered into the terminal of the local machine which is unsuitable for the purpose of this section of the web app as it will eventually be run on a Linux server. Here is an example of the command to run on windows.

```
ffmpeg -f image2 -r 1/5 -i image%05d.png -vcodec mpeg4 -y movie.mp4
```

4.2.15 Ffmpeg command to create animation.

4.2.9 Current Weather *

The current weather feature will have an interactive map of Ireland with a table containing today's weather conditions and a map with various graphics to represent this information. There will be a table with the various weather conditions and their values associated with each of the main cities around the country. This will make the scope of the information much greater and will allow users seeing the data to make their own conclusions of how the weather will be much easier if they so desire. The map will have graphics such as sun, clouds, rain, snow, etc. based on the information in the corresponding table. The weather information will be taken just like the weather forecast feature, from the database or the external weather API if the database is empty or data retrieval from the database cannot be performed. Currently this feature only has a skeleton framework. This means that there is an interactive map with weather image example over the main cities in Ireland. These would change based on the information in the database or API. There is also a table that would display the city names and weather conditions. This would use Django's functionality to pass the data in from the function in views.py that is retrieving the data from the database or API.

The next section of this chapter in the document will discuss how the web app changed, what problems were encountered and eventually overcome and how this web app could progress in the future.

4.3 Project Difficulties

Over the course of this project I ran into several problems or 'snags' that hindered my overall progress and did not allow me to fully implement what I stated and intended to accomplish. Some of these problems were research based. These included insufficient data for certain types of programmes or software that I was considering for use. User friendliness in terms of the front end of the app. Tutorials on how to perform certain tasks such as inverse distance weighting for raster creation which would then lead into fluid animations. Other problems were also software based. These included language problems, missing libraries or packages and OS compatibility problems. Out of all these problems I believe the area that impeded me the most was definitely the environment that I chose to create my project in which is windows which I will explain in more detail later.

Trying to install Django on windows was the first real issue I encountered during this project. The official Django documentation describes how to install Django on a Linux system. I

spent quite some time trying to get around this issue just after the Christmas exams were over but with no success. I then notified my supervisor that I wanted to create this web app through Django but was unable to install it on my machine. He then suggested I download Pycharm and gave me an activation code that allowed me to download the professional version. Thanks to my supervisor I was able to code this web app in Django which I believe was the best course of action considering the use of geographic data and python functions and packages such as GDAL and GeoDjango.

For a time I could not access my database through Django. As previously stated the `models.py` file in Django is where you write the various tables and their objects that will be used in the project. Once this is done you execute the `makemigrations` command which creates a python file. This file will then communicate with the database and translate the Django code into the corresponding SQL. I created a table that would hold all of the Json data from the OpenWeatherMap API. However the python package that allows Django to connect to the database `psycopg2` was up to date but PostgreSQL was not. I continually received a `jsonb` error. I later discovered this was because although `psycopg2` was up to date, the version of PostgreSQL I was running was 9.3 but 9.4 is needed to allow a `json` field in `models.py`. To fix this the logical course of action would have been to upgrade PostgreSQL. This would require me to dump all data to another storage space. This would have taken more time than I wanted to spend on the problem. Fortunately by simply deleting the migration file I could then continue regular use of my database through Django. Although this problem persisted for nearly a week it forced me to spend time on other aspects of the project such as User Interface and this document. This was most certainly a challenge but it was not as severe as others that I faced.

Several important packages were needed in order to achieve some of the systems goals such as animations and various geographic calculations. Some of the packages needed were `matplotlib`, `numpy`, `GDAL`, `scikit learn`, `ffmpeg` and many others. The installation of these packages was not easy due to the fact that I was using a machine with windows and the paths to these files and packages had to be manually imputed. In the end these installations were eventually achieved after many attempts. However testing the various programmes and algorithms that were needed for the above features highlighted another issue. For most of the python programming I was using SPYDER which is supplied with Anaconda. SPYDER is the same as regular IDLE but with an ipython console. Anaconda already comes with various packages such as `numpy` and `matplotlib`. However in order to create animations of the weather obtained from the Inverse Distance weighting algorithm I needed FFMPEG. This required another manual install but after the installation was completed I discovered that I could not use it in Anaconda. I tested the same programme in Python 3.4 and created a small `mp4` file. I am still unsure why Anaconda cannot access FFMPEG but it may require manual path manipulation similar to a regular install on a windows system.

As I previously stated the largest issue over the implementation of this project was the operating system which was windows and the integration of various plugins. Most of the problems that I have already explained were simply a lack of information or not having a complete understanding of the code or environment that I was coding or testing in but these

could be solved with practice or wiki/tutorials for guidance. However the problems with windows mainly stem from packages that were needed for certain features of the project mainly GDAL. GDAL was originally written in C++ but I was coding through in python using the Django framework to create my application. As a result I needed to install GDAL on pycharm. However I could not as GDAL is not a native python library. I needed to install GDAL to my laptop but this is much more difficult on windows than say Linux which is a simple `sudo apt-get install`. I had to find the specific version that would be compatible with my version of python which is 3.4. Then I needed to install python extension that would mean GDAL was compatible. Then the paths needed to be edited manually. The main PATH was edited with the line (`;C:\Program Files\GDAL`) being added at the end. Next two more variables were added called GDAL_DATA (`C:\Program Files\GDAL\gdal-data`) and GDAL_DRIVER_PATH (`C:\Program Files\GDAL\gdalplugins`). This process of integrating GDAL into windows took a very long time as most tutorials suggested getting OSGeo4W instead and there were not many tutorials suggesting just GDAL. However downloading OSGeo4W would not work and was later discovered that the path was duplicated five times. This cost me valuable time overall and I believe was the biggest reason for not fully completing my original goals. There is extensive documentation for Django but most of these walkthroughs or tutorials are done in a Linux environment. This meant that testing the code after reading the walkthrough was difficult or not possible. Most of the tutorials that showed how to manually input data for testing was done in python manage.py shell. This would then bring up a python shell that you could perform manual input. However this shell would never work and basic python would throw errors. This lack of documentation for Django on windows really frustrated my progress and I felt I was spending hours coding only to start over or abandon the code which further hindered my progress.

Geodjango was the component of this project that would allow me to perform various geographic tasks such as vector overlay and the creation of heat maps from the data that would be stored in my database. However the documentation for this was very brief. Most tutorials would only go so far as creating the models needed in the database. I could not find any in depth tutorials that would allow me to perform calculations or any form of meaningful creation of the data. This component of the project was the main section that I wanted to create but found it very difficult and spent a lot of time trying to find tutorials that could aid me but found very little. I have never used Geodjango or JavaScript before taking up this project and I feel I was at a disadvantage from the beginning.

Conclusions

This section of the document discussed several key pieces of the web app. There was an architectural diagram and description describing an overall view and functionality of the system. Next there was a discussion and more in depth detail of how a user can navigate the system and how the code functions in the respective sections and functions of the web app.

After the architecture of the web app the challenges faced throughout the project were listed and how they affected the projects progress and how they were finally overcome.

5. System Validation and Demonstration

5.1 White-box Testing

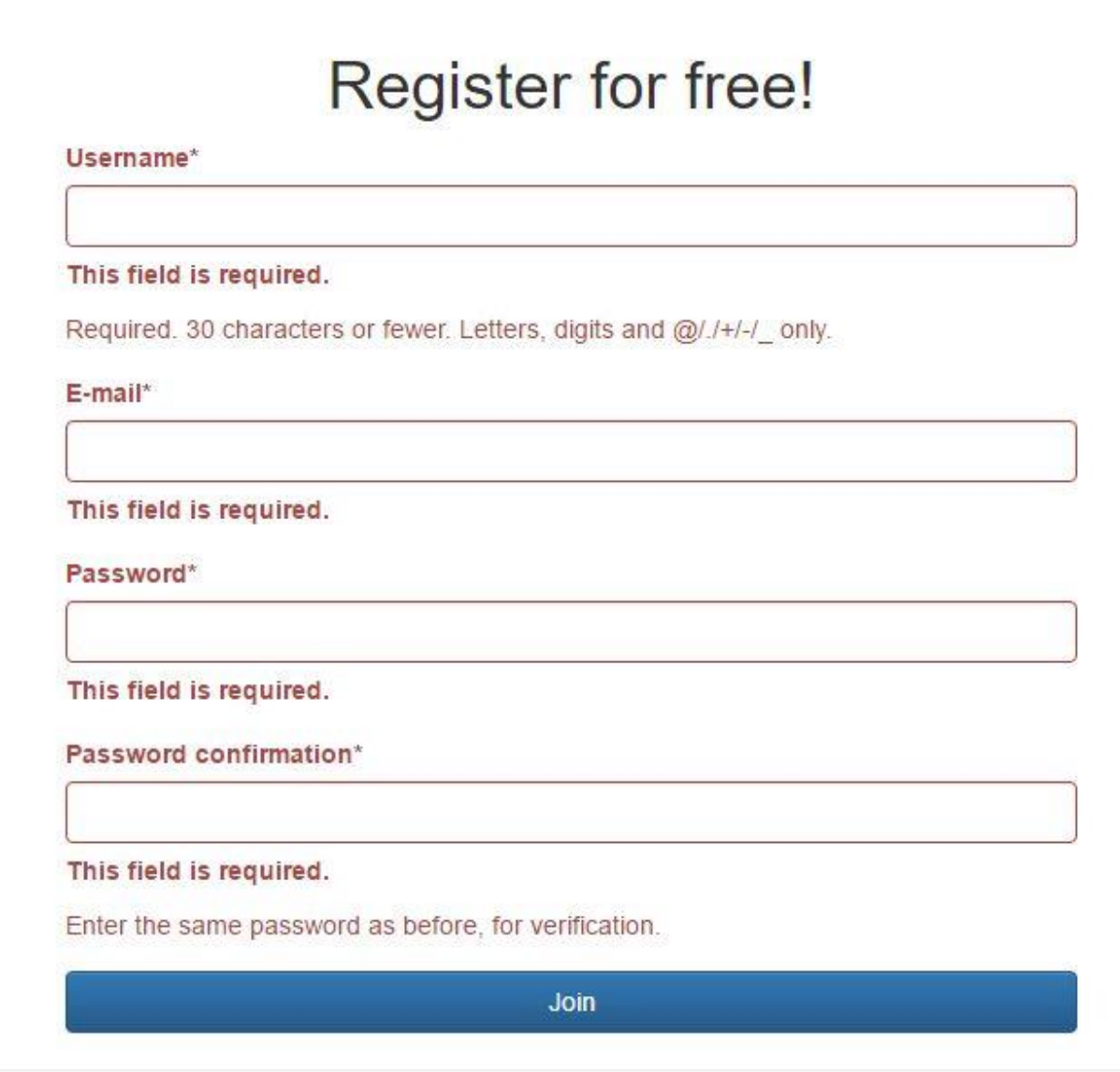
White-box testing is a method of testing the inner workings of a system as opposed to the systems functionality which is black-box testing. White box testing can be applied at different levels of the system. These include unit testing, system testing and integration testing. White-box testing will prevent hidden errors from developing later on in the development of the system and give a greater understanding of each line of code ^[20]. In the case of this system this will be the code of the whole weather app. The various techniques that will be implied will test the source code and prevent any errors in the overall app later on by showing any flaws that may not be obvious by simply looking at the code even if there are zero syntax errors. White-box testing is very important for testing the code of the web app as there are no hardware components. The coding of this web app was conducted in Pycharm. Pycharm is a powerful IDE that has many useful features including a debugger that will cover most if not all of the white-box testing. The Pycharm debugger allows you to set stop or break points in the code. This means that when the code is run you may see what the code I doing. This would include what variables are being passed through a certain function. The main white-box testing that was conducted for the web app was posting of data to the database. This was the first piece of functionality implemented in the system. The data that was being posted was user input such as a profile or contact message. The Pycharm debugger ran through these sections of code and showed no errors and the values being posted to the database. The next step in this testing was checking the database and emails to see if the information was transferred. For the user input and email sections of the web app both of these were successes. This is the only real white-box testing that could be done as the main section of the web app was being run on the local machine and not Django in Pycharm. The algorithm was written to take in random values to prove that it functioned properly. Connecting to the database from the local machine to test the algorithm was not feasible in the time frame that was left at this stage of the project. White-box testing requires a high level of understanding of the language and methods used within it. Most of the methods and languages were completely unknown before this project. This means some sections of the web app could not be completed and in turn white-box testing could not occur on said sections.



Figure 5.1 Email sent with all details

5.2 Black-box Testing

Black-box testing is a method of testing a system with no knowledge of the inner workings of the system. Black-box examines the functionality of the system not the inner workings of the code. Black-box testing can be applied to all forms of testing be it unit, integration or system testing to name a few. A person testing the functionality of the system knows that an input will produce some form of output but does not know how this is achieved ^[21]. Most of the testing conducted on this web app was black-box testing. Most of the functionality of the current state of the system is that of User Interface design and complexity. This includes error checking on forms, sending emails with correct layout, interactive maps displaying meaningful data and changing when needed. The first section that was tested was the register section of the web app. The first test was simply clicking the join button with no information entered. The error checking on the forms functioned correctly and displayed all fields are required. This occurs also when partial information or only certain fields are entered with data.



The image shows a registration form titled "Register for free!". It contains four input fields: "Username*", "E-mail*", "Password*", and "Password confirmation*". Each field has a red border and a red error message below it: "This field is required.". The "Password confirmation*" field also has a red error message: "Enter the same password as before, for verification.". At the bottom of the form is a blue "Join" button.

Register for free!

Username*

This field is required.

Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

E-mail*

This field is required.

Password*

This field is required.

Password confirmation*

This field is required.

Enter the same password as before, for verification.

Join

Figure 5.2 registration error messages on forms

This same error checking on fields is the same for all other sections of the app that required user input. This was achieved by specifying the objects in the tables in the models.py file. There is also a message sent to the email of the admin of the site if a user wishes to change their password. If the email does not exist then it will not be sent and an error message specifying that the email trying to reset the password is invalid or does not exist.

The interactive maps were also tested. This involved changing the layers that the map could be viewed on and the data being displayed. The location and file opener section of the web app was tested first. Random locations were entered into the search bar to see if they existed. If they did not exist a red line indicating that they were not a valid location will appear under the word. Unfortunately the words that are entered cannot be seen unless the word is highlighted. This error was discussed earlier in the document explaining why it was left this way. Next the file opener section was tested. When this is clicked a window will open on the local machines storage space allowing the user to navigate their file structure to find the file

they wish to display on the interactive map. Leaflet will then read in the various points and display them as markers on the map.

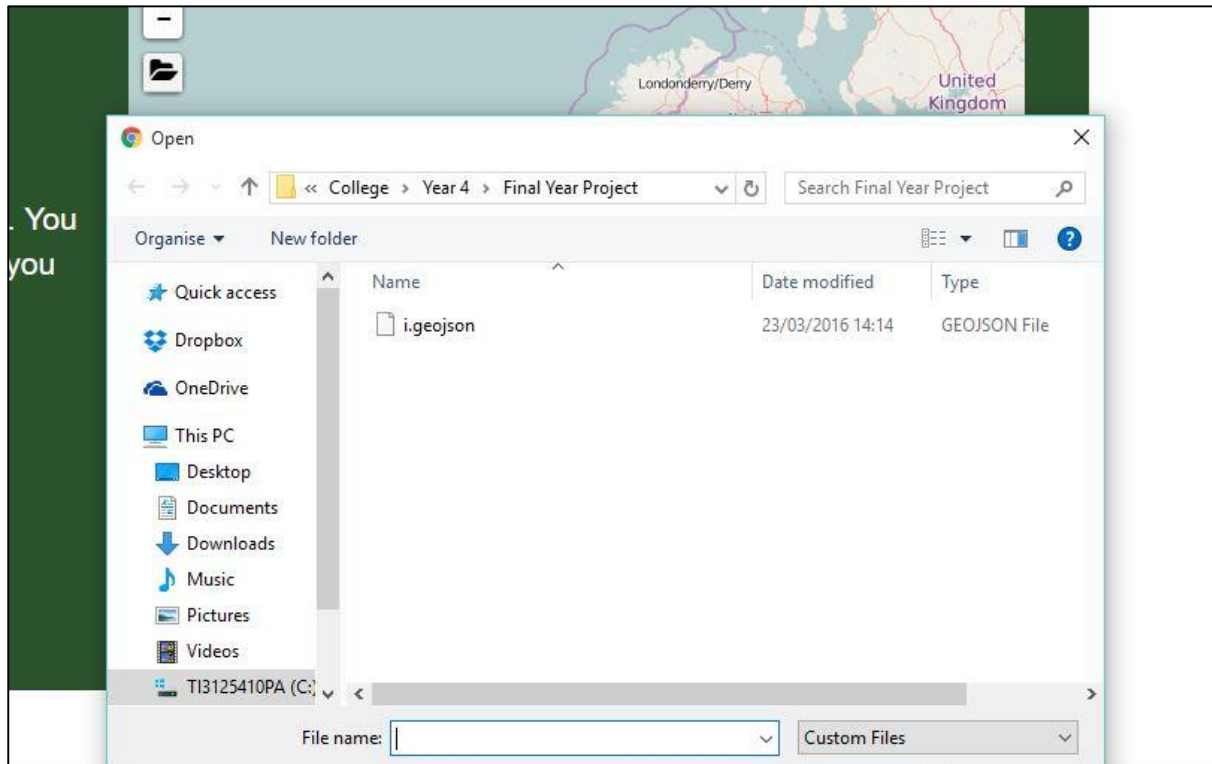


Figure 5.3 File opens correctly

Lastly the various sections or URL redirects were all tested. This simply involved clicking on every link and seeing if it redirected the user to the correct web page. All of the URL redirects function properly including those that redirect after successful logins or form submissions.

5.3 Automated Testing (Considered using)

Automated testing is the use of software separate from the software in the system that is being tested that will control the execution of said tests and compare the outcomes to the predicted outcomes to give an accurate result. This form of testing would suit the methodology and approach chosen for this app. Automated testing is very useful for continuous testing which is an essential part of agile development where it is called test-driven development (TDD). The main area that automated testing would handle would be white-box testing, specifically unit testing. The unit testing software would be written before the code is begun but it would have to change as the code progresses such as code refactoring [22]. This form of testing requires extra coding which may use up valuable time that could be used in other areas of the project such as implementing more usability in the front end of the app. Automated testing is also more practical in a business environment where there are more significant risks that are being taken. This form of testing would cover white and black-box

testing but would take the value of using real users in the black-box testing away and this seems like a less accurate alternative. The line between white-box and black-box testing in recent times has become very blurred meaning automated testing could be very convenient and more accurate. This section was considered but was ultimately not chosen due to time constraints. As described above extra code would have to have been written to automate the testing. This would have meant updating this extra piece of code constantly throughout the course of the project. This fact was the main reason for not choosing automated testing.

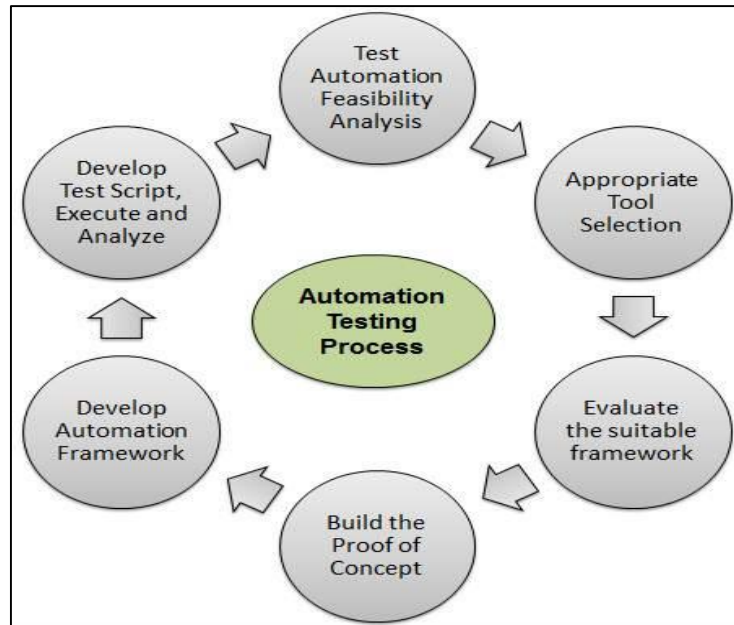


Figure 5.3.1 – Test automation process.

5.4 Test cases

ID	1
Title	Sign up
Pre-conditions	None
Test Steps	<ol style="list-style-type: none"> 1. Open browser 2. Enter webpage URL 3. Click sign up 4. Enter details 5. Tick send e-mail notifications if you wish

	6. Click Done
Expected Results	A message will appear saying e-mail has been sent to the specified address. Click on the link to confirm your sign up details.

ID	2
Title	Enter Location
Pre-conditions	Must be logged in
Test Steps	<ol style="list-style-type: none"> 1. Click on enter location 2. Enter the desired location 3. Click "Search"
Expected Results	The entered location will appear with all the basic weather details such as temperature, humidity and air pressure.

ID	3
Title	Request Forecast
Pre-conditions	Must be logged in

Title Steps	<ol style="list-style-type: none"> 1. Click on Request Forecast 2. Select location or full forecast 3. Click done
Expected results	The map of the specified area or just Ireland should appear. The main areas of the country will be labelled with graphics and weather data such as temperature

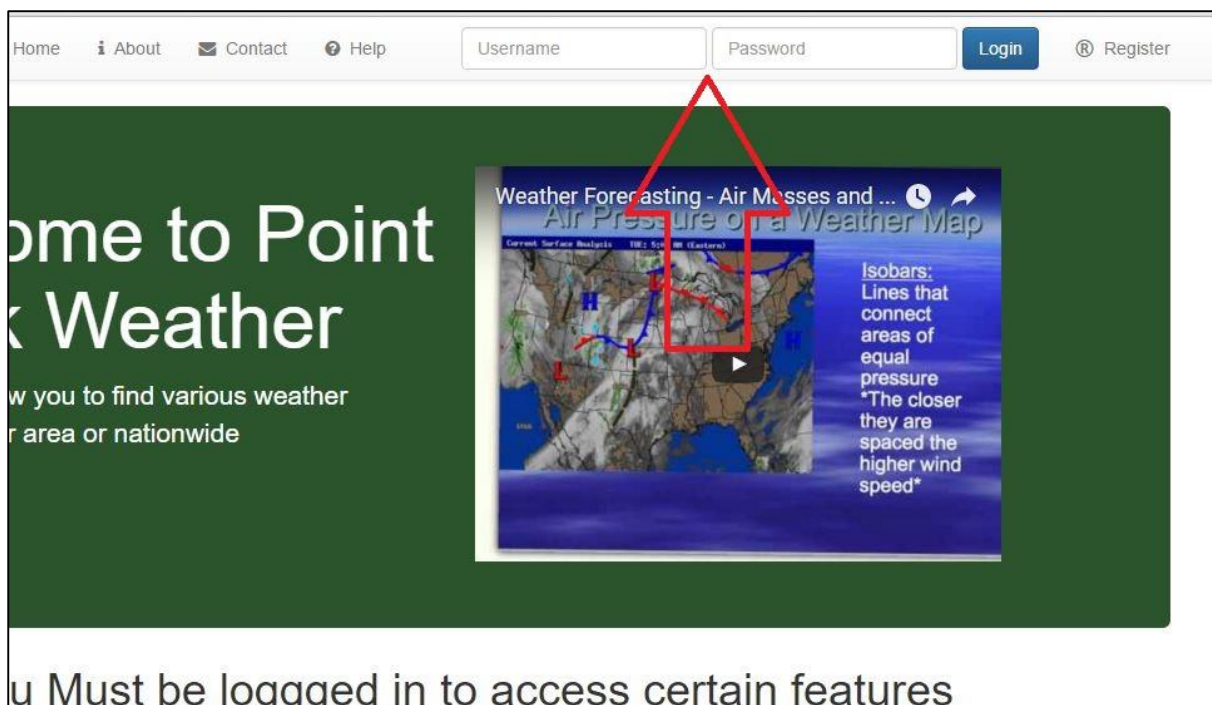
5.5 Demonstration overview

This section will describe what can be demonstrated if a person requested one. The main aspects of the user interface can be demonstrated. The registration feature can be demonstrated with error checking and a successful email with an activation code. The login feature can be demonstrated showing error checking and a successful login. A successful login will redirect the user to the home page with the login and registration feature now omitted and the main features of the web app now visible. The logout feature will redirect the user to a successful logout page and the features that were omitted such as the login and register feature will now be visible again and the main features will again be unavailable. The contact form can also be successfully demonstrated. This will send an email to the admin of the site which is specified in the settings.py file. The help section contains a map with pop-up markers displaying the best surfing locations in Ireland with information included in the pop-up. The location finder can be demonstrated and it will locate anywhere on the planet. The open file section will also work however loading the file can be tricky as certain files will not load at all. The preferences section of the web app can also be demonstrated. This section allows a user to click anywhere on a map and the location will be recorded. The user will then enter the name of the location with a brief description of that place. The information will then be posted to the database. The forecast function can be partially demonstrated. Will only work on my local machine. It will perform the Inverse Distance calculation perfectly using shepherds method and will create the necessary files associated with each calculation. Each file will represent a certain period in the day similar to two hour intervals throughout the next day. All of these features can be demonstrated fully or partially with the only section that is not functional is the current weather section which should display today's weather conditions in a table and based on the weather conditions in the table display graphics on the map.

5.6 Demonstration walkthrough (Screenshots)

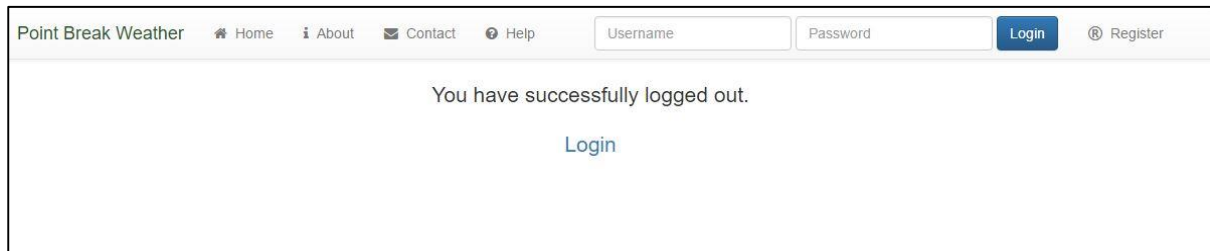


Select the register button on the top right of the screen. This will redirect you to the registration page. All fields must be entered in order to register to the site. If the credentials are correct select join and you will receive an email with an activation key. Click the link to activate your account.



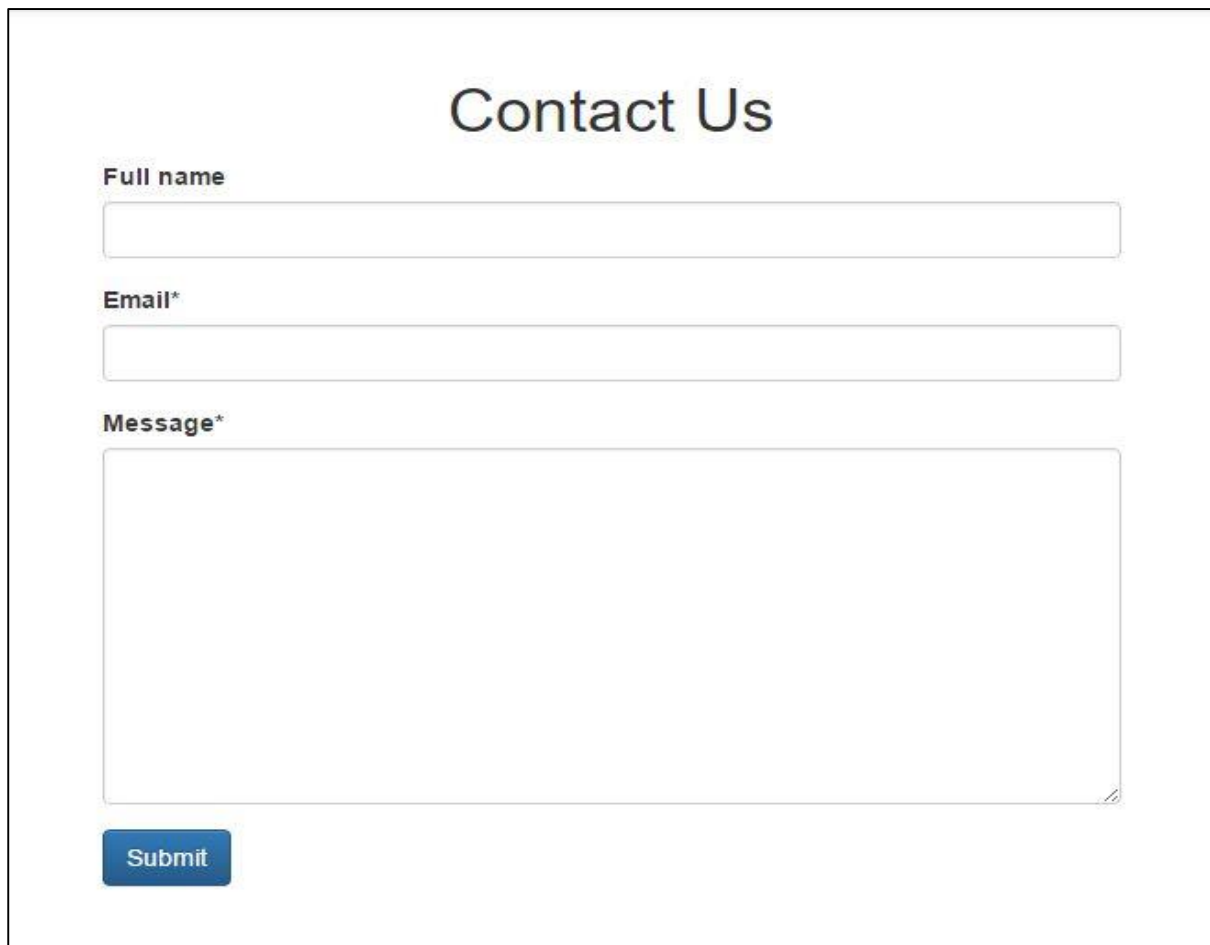
To login simply enter in the username and password you specified during the registration process. If the credentials are correct you will then see the features at the bottom of the page appear and the logout button will now replace the login form and register button on the top right of the screen.

To logout of the system simply click on the logout button that is now in the upper right hand side of the screen. This will redirect you to the logout page which confirms that the logout has been successful.



The screenshot shows the top navigation bar of the 'Point Break Weather' website. The navigation bar includes links for Home, About, Contact, and Help. On the right side, there are input fields for 'Username' and 'Password', a 'Login' button, and a 'Register' link. Below the navigation bar, a message states 'You have successfully logged out.' with a 'Login' link centered underneath.

To use the contact feature simply click on the contact button in the navigation bar. This will redirect you to the contact form to fill out. This contact form is used to ask the administrators of the site questions. The email and the message are required in order to send the email. You do not need to give your name if you do not want to.




The screenshot displays the 'Contact Us' form. At the top, the title 'Contact Us' is centered. Below the title, there are three input fields: 'Full name', 'Email*', and 'Message*'. The 'Message*' field is a larger text area. At the bottom left of the form, there is a blue 'Submit' button.

Beside the contact feature there is a help feature. To use this feature click on it and you will be redirected to the help page. On this page the user may select different layers to add to the map. These include the best surfing location in Ireland and some attractions that are in the vicinity of those surfing areas or nationwide. This section of the web app allows users to plan their day by knowing what the weather will be like and seeing that in relation to the locations. The user can select both options to display at the same time so they can have a better idea of the locations in relation to each other.



The preferences feature of the web app is located at the bottom right of the page. This will redirect the user to the preferences page. To use the page the user can click anywhere on the map and that will be recorded. Next the user must enter the name or area that the marker has been placed with a brief description of said place or area. The user will then be able to submit this to the database for future use.



The map shows Ireland, Northern Ireland, and parts of the United Kingdom, including London, the Isle of Man, and Wales. A red location pin is placed on the east coast of Ireland, indicating the location of Rosslare. The map includes zoom controls (+ and -) in the top left corner.

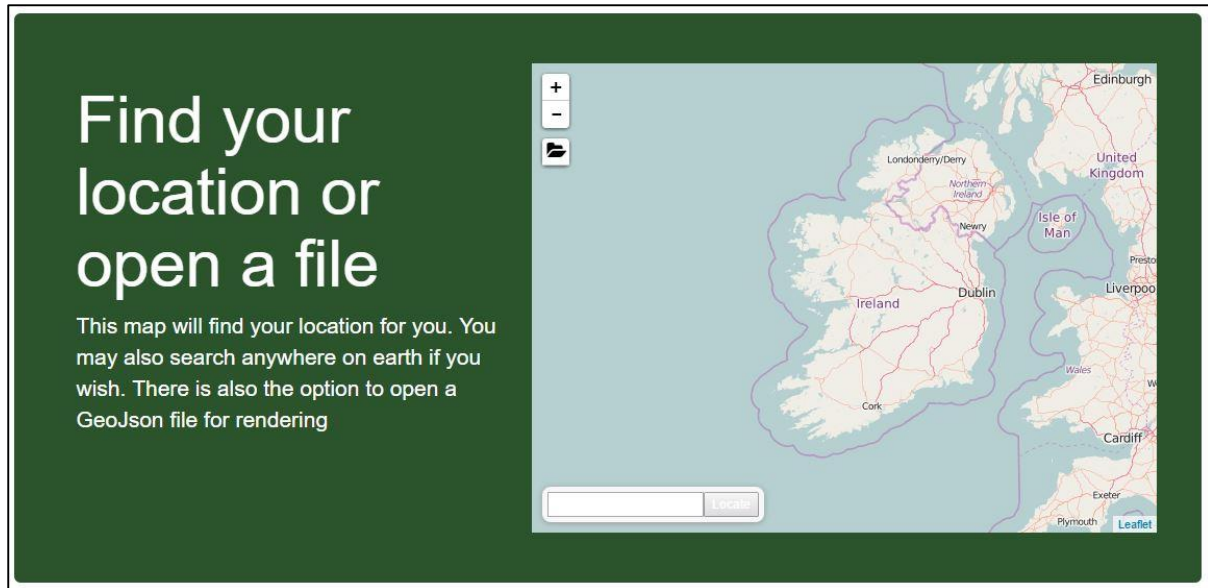
Place name*

Rosslare

Description*

European Port

The location and file opener feature of the app are located beside the preferences feature under the find location heading. The user will be redirected to the page where they can use said features. The user simply enters the location into the search bar and it will be found if it is valid. The file opener is a button located just under the zoom options. The user clicks this and it will prompt the local machine to open its respective storage space so the user can search for the file to open and render on the map



The forecast section of the web app would have a map with a button asking the user to start the forecast. This will then begin an animation of weather conditions over the map. The main algorithm that performs the Inverse Distance calculation is written in python outside of the web app. To run the programme simply click f5 and it will run. It will produce png files. Each file represents a certain point in the day. These can then be turned into a gif if the user desires.

Conclusions

This chapter in the document described the various forms of testing that was performed on the system. This included white-box testing on the inner workings of the code using the Pycharm debugger. Black-box testing was also conducted to test the functionality of the system. This meant testing the various forms and maps that were in the web app and seeing the results. Automated testing was also considered but time constraints meant it had to be scrapped. Finally what aspects and functions of the web app that could be demonstrated.

6. Project Plan

6.1 Pre interim report project plan

Before the interim report and presentation the original idea for the project was that of an android app. The app would display weather details without a map just the place name. The forecast was also just the weather details on a table with place name and possible graphic to further the users understanding of the weather conditions the next day. The initial coding would have been done in Java on Eclipse. The fact that it would be an android app conflicted with the original user base. The user base was intended to be anyone but the fact that it was coded as an android app reduced the user base. The use of Java was difficult throughout the pre interim report duration of the project. The main reason for this was the fact the coding was being done in Eclipse. Eclipse continually threw an R error. The main reason for this was suggested that the R.java file was missing or not rendering correctly. This was not the case and the decision to try a different editor began. Android studio was then chosen as the replacement. This change in code editor immediately solved the R.java file error. The only prototype that could be completed was a basic front end with changing symbols based on user input. However when placed on an android phone it would immediately crash even though there were no syntax or semantic errors visible in the code.

```
private void setWeatherIcon(int actualId, long sunrise, long sunset){
    int id = actualId / 100;
    String icon = "";
    if(actualId == 800){
        long currentTime = new Date().getTime();
        if(currentTime>=sunrise && currentTime<sunset) {
            icon = "☐";
        } else {
            icon = "☁";
        }
    }
}
```

Figure 6.1 Original java code snippet for emblem change

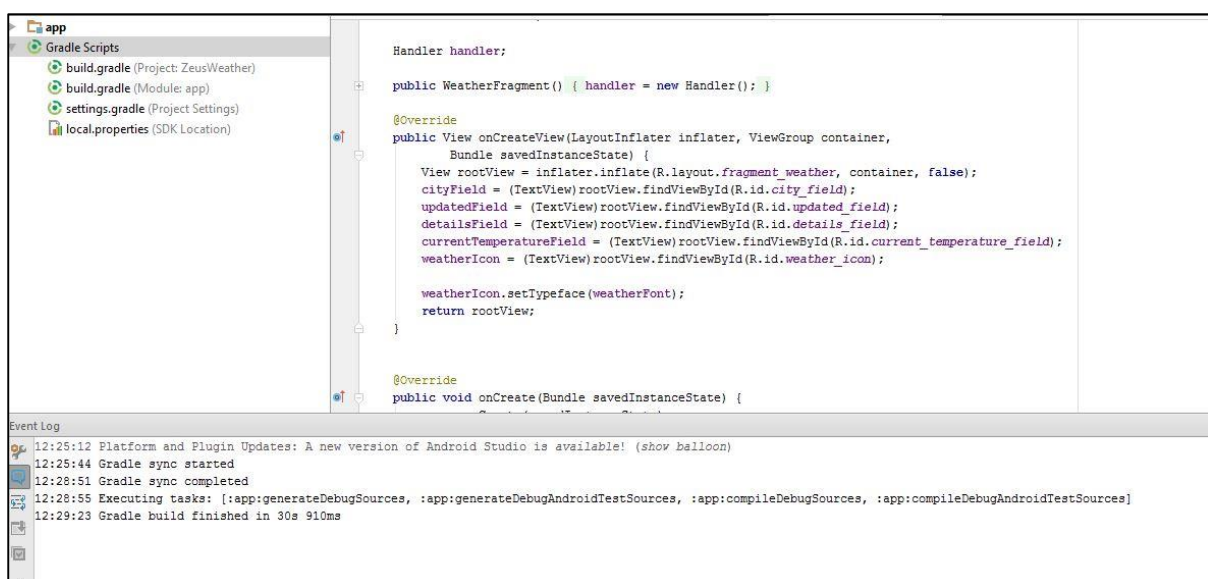


Figure 6.2 Android studio solved R error

There was no clear structure for how the weather data would be collected and stored. Using a cloud service was one solution to store the data offsite as the size and scale of the data would not be feasible for the mobile phones storage capabilities. The collecting of the data was another issue that was not fully researched or chosen before the interim report. Several APIs were researched and considered. Some of these include NOAA, Wunderground and OpenWeatherMap APIs. These all allowed for the correct weather data in the correct formats such as Json or xml. But some of the information was irrelevant and would simply take up space in the database or extra code would have been needed in order to remove these specific details. This would have consumed for time that could be spent coding other areas of the project. The project plan submitted for the interim report was an android app that more than likely would change to a web app. This would remove the restrictions on the users of the application. The platform would not matter as any browser would work. Using a web app would also allow for a more secure and easier running of the application on a server which would then improve the database that would store the data.

6.2 Post interim report observations & changes

During the submission and presentation of the interim report in December several issues were raised about the proposed system. The main issue was the user base. The users that this system would cater to was not specified and was far too broad. This would have made the project too narrow and may have been too simple or too complex for certain users. As a result of this observation the user base has been narrowed down. The system will now be changed from a general weather site to a surfing weather forecast site. The site will now gather and display information that will be useful for surfers. Information that would concern surfers in general would be wave height, wave break, gusts, and wind speed and wind direction to name a few. Surfers will also need an animated source of information on a map to reinforce this information. The animations will also make it easier to understand and will reduce the time the user spends trying to make a decision based on said information.

Another issue raised was the uncertainty of how the front end would look. Originally it was proposed as an android app. When the interim report was submitted and presented the system had shifted to a web app. Even then it was unclear how this web app would be implemented. PHP and python were proposed but was still not “set in stone” which would be chosen. Since then an apache server has been created through Microsoft azure that will run this web service. The main component is the weather forecast animations using Inverse Distance Weighting and how the front end will look. This will now be coded in python as the Inverse Distance algorithm will be coded in python. This will make the front end and back end easier to integrate. Django will be used to code all of the python code. Django is an extremely useful way of making a web system in an extremely refined manner. It is similar to PHP MVC methods but have found it to be even more useful and more reliable.

Another issue raised was the forecast algorithm. At the time of the interim report a sliding door algorithm or sklearn were proposed but very little was researched about them. After the interim presentation more research was conducted and found the sliding door algorithm too complex for the time frame of this project and the infrastructure being used. The sliding door algorithm would need information fifty weeks before the current date of the forecast request. This cannot be achieved in the time frame that has been given for this project due in April. Sklearn could still be used but is still unclear if it will be accurate enough or can hold the type of data that will be collected. During the second semester of the year I took the Artificial Intelligence 2 module. After taking this module it seems that python's Panda extension may be used for the forecast algorithm. Panda is a machine learning library for python. Over the course of the second semester the forecast algorithm will be improved as the AI 2 module advances. A small but effective change will be made to the forecast algorithm. The algorithm will be pre-cooked in the background. This will mean the user does not have to wait several minutes for the algorithm to finish.

Graphics and geospatial data was another issue that needed some small amount of change. How the animations or graphics would be created was not specified during the interim presentation or the report. After the report more research was conducted and will now be completed from the data collected and run through some GIS software such as QGIS. This software will create raster images that can then be overlaid to create a gif on a map. This is a simple method but very effective in conveying the necessary information or weather forecast. Like the forecast algorithm these raster images will be pre-cooked before a user asks for the gifs. This puts less strain on the user's computer and reduces waiting time.

6.3 Final project plan

The final project plan as taken most of the observations made during the feedback of the interim report on board and as implemented almost all of the features to some degree. Extra features were added to the system to give it a more professional look as well as increase the web apps usability for others making a similar system. This included a registration feature. The registration feature gives a level of security to the web app. Although displaying weather data may not be sensitive information someone may use this sites features to redirect a user to a fake web app that may harm their system. The registration feature allows for login and logout of the web app which in turn allows for the use of the main features of the web app. There were no interactive maps in the original template but the final web app utilises several interactive maps. This is achieved through leaflet using JavaScript. This improves the user friendliness of the web app a great deal. The leaflet interactive maps allow for easy switching of displays and information layers. This removes the need for complex tables or redundant weather information that the user will not need. Originally the web app would perform the weather forecast but this is not the case in the final web app template. On the advice of the project supervisor the decision to perform weather display using Inverse Distance Weighting

was implemented. This was achieved using shepherds method and would create pictures that can be transformed into a gif if necessary and then overlaid onto the leaflet base maps. This will then show the progression of the following day's weather conditions.

6.5 Future work and Advice

This section of the chapter will discuss where the current web app can go and how this can be achieved by myself or another user who wishes to create a similar system. At the moment the current web app has

- User registration
- User login/logout
- Admin contact via email over SMTP
- Interactive maps using Leaflet
- Storing of users preferences
- File opener
- Location finder
- Inverse Distance algorithm that will create heat map graphs for overlay and gif creation

The functions that were not fully implemented were the weather forecast and current weather. The reason for this was data retrieval. Transforming the data from a byte string to Json format and then placing in a python object or dictionary for posting was tricky. This meant that the current weather feature could not be implemented as there was no data to display. A basic map demonstrating how it should look is included but is based off of hard coded data. The forecast section is partially complete. It can be demonstrated outside of the web app as described earlier in the document. There is no interactive map for it at the moment but the calculation and images will work if this function is completed in the future.

This is a list of recommendations for future developers of this system or something that closely resembles this one. The first piece of advice is do not spend most of your time researching the environments and algorithms. Throughout the development of this web app a significant amount of time was spent researching Django, JavaScript, jQuery, Rest API and the Inverse Distance algorithm. This extensive amount of research meant less time was spent coding the actual web app. This then developed into not understanding the errors that were given throughout the development of the web app. Although the research gave a better understanding of the languages and how they connect, when an error appeared more research was needed again to discover what the error was and how to resolve it. If more time was spent simply coding and learning the languages in this way more time could have been used researching solutions to the errors or alternative methods. A fitting comparison to this project

would be that of learning the guitar or any other musical instrument. Unless you start practicing with the instrument which would be the languages in the case of the web app you will never fully understand it. The next piece of advice would be choose your platform very carefully and stick by it. After the interim report the project shifted from a Java android app to a Django web app. This again meant more time researching the new framework that is Django. The operating system of the machine that the web app is being coded on is also very important. Most if not all of the Django documentation and tutorials are aimed towards Linux systems. This required that I spend time finding an alternative way to install Django. This was achieved by using Pycharm. The OS also plays a major role in installation of the various packages needed for the Inverse Distance Weighting calculation. On a Linux system you simply open the terminal and type `Sudo apt-get install [package name]`. On a windows machine this is not so simple. You cannot just download through Pycharm the packages GDAL, numpy and scipy. These must be installed on the base machine with python wrappers so the interpreter you are using can understand it. You must manually edit the PATH on the windows machine. This again wasted a significant amount of time during the development of the web app and ate into testing and implementation. These are the main pieces of advice that should be adhered to for any future users. The last piece of advice would be always attend your supervisor meetings. I attended every meeting throughout the year and I believe it helped greatly. The meetings allow for discussion of how the project should be undertaken, what direction to go, the scope of the project and any errors that you experience.

Recap of advice:

- Choose platform and code carefully
- Spend less time researching and more coding
- Be aware of the OS you are using
- Know what packages are needed before coding
- Go to all supervisor meetings

Conclusion

This chapter discussed how a user could improve their efforts if they were undertaking a similar system. This included advice on OS and platforms and supervisor meetings every week.

7. Conclusions

This document has so far discussed the background of weather forecasts. It then described the various technologies that were researched and evaluated for selection. Next the document discussed the methodology and design that was used. Next the architecture and development of the system was discussed. System validation and demonstration was discussed next. Finally advice and future work was discussed. This final section of the document will give a recap of the previous sections of the document in more detail

In chapter 2 a discussion of the technologies that already exist and alternative solutions began. What technologies such as languages, frameworks and libraries were then discussed and reviewed for possible use in the development of the web app. These included Django, GDAL, cron, Inverse Distance Weighting and python were discussed. Finally the languages and so on that were chosen were then validated.

In chapter 3 a discussion on the design and methodology that would be used in the development of the web app. The methodology selected was the agile method. This is because of its constant testing which suited this web apps development cycle. The design discussed which files performed the various tasks within the system. This included view.py that held the main functions of the web app.

In chapter 4 the architecture and development of the web app were discussed. This included a high level view of the system as well as a detailed walkthrough of the web app that discussed the inner working code of the web app. Difficulties throughout the project were highlighted and how they were overcome in the end.

Chapter 5 discussed the testing and demonstration of the system. The testing included white-box and black-box testing. This included using Pycharm debugger for white-box testing and using the user interface extensively to test the black-box section which essentially test the web apps functionality. Automated testing was considered and a description of it was included. Finally the features of the web app that could be demonstrated were discussed.

Chapter 6 discussed how I or another user could go about making a similar system or continuing on and making for functions and features available. Various pieces of advice on how to perfect the development of the web app in the future were also discussed.

To finish this project has been the most challenging of my academic career. Most of the languages were new to me and I had to learn some of them from scratch. I did not fully implement the system which I am very disappointed about. However I feel that I worked hard to achieve what I am producing at the end of this report and project. Looking back on my decision to build a web app I feel I made the right choice. Most of the technologies used in this web app will be invaluable in my career later in life. I feel that I made the right decision and do not regret undertaking this task.

Bibliography (research sources)

- [1] Frisinger, H. Howard, 1977: *The History of Meteorology: to 1800*, Science History Publications, New York, 148 pp.
- [2] Goldstine, Herman H., 1980: *The Computer from Pascal to von Neumann*, Princeton University Press, Princeton, New Jersey, 378 pp.
- [3] Shuman, Frederick G., 1989: History of numerical weather prediction at the National Meteorological Center, *Weather and Forecasting*, vol. 4, pp. 286-296
- [4] Moran, Joseph M., and Michael D. Morgan, 1994: Weather analysis and forecasting. In *Meteorology: The Atmosphere and Science of Weather*, fifth edition, Prentice Hall, Upper Saddle River, New Jersey, pp. 374-401.
- [5] <http://www.noaa.gov/>
- [6] <http://www.wunderground.com/>
- [7] <http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536>
- [8] *Learning Test-Driven Development by Counting Lines*; Bas Vodde & Lasse Koskela; IEEE Software Vol. 24, Issue 3
- [9] <https://docs.djangoproject.com/en/1.9/faq/general/>
- [10] <https://django-reusable-app-docs.readthedocs.org/en/latest/>
- [11] <https://www.djangopackages.com/>
- [12] Cutting, Thomas Deliverable-based Project Schedules: Part 1, PM Hut
- [13] "JetBrains Strikes Python Developers with PyCharm 1.0 IDE" <http://www.eweek.com/c/a/Application-Development/JetBrains-Strikes-Python-Developers-with-PyCharm-10-IDE-304127>
- [14] Shepard, Donald (1968). "A two-dimensional interpolation function for irregularly-spaced data".
- [15] Neteler M., Raghavan V. (2006). "Advances in Free Software Geographic Information Systems"
- [16] http://www.gdal.org/formats_list.html GDAL - Geospatial Data Abstraction Library
- [17] <http://erouault.blogspot.ie/2012/01/welcome-to-200th-gdalogr-driver.html>
- [18] https://en.wikipedia.org/wiki/Software_container
- [19] "Docker Documentation: Kernel Requirements". *docker.readthedocs.org*.
- [20] Mohd. Ehmer Khan http://www.sersc.org/journals/IJSEIA/vol5_no3_2011/1.pdf
- [21] Ron, Patton. *Software Testing*.

[22] *Learning Test-Driven Development by Counting Lines*; Bas Vodde & Lasse Koskela; IEEE Software
Vol. 24, Issue 3