Sky Scraper - A Flight Comparison App for Android Smartphones

Klaudija Medeksaite - G00353761

B.Sc.(Hons) in Software Development

May 10, 2021

Final Year Project

Advised by: Dr. Dominic Carr
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Contents

About this project

Abstract I have developed the Sky Scraper app as I enjoy travelling and I know first hand how difficult it can be to plan journeys sometimes. This project is an attempt at easing the burden of deciding which flight to book for a trip, by providing the everyday user with a simple app that compares the various options available. The app relies on a database that is updated every fifteen minutes with live information gathered by scraping a website. This information is used to create the statistics for the flight that are then presented to the user in the Sky Scraper app, allowing them to make an informed decision. Sky Scraper is created using multiple technologies, including Python and Dart programming languages, Scrapy Cloud as the server where the back end code runs and updates the database on Google Cloud, and Android studio and Visual Studio Code as my main development environments. Along with all of these technologies, I also used some interfaces such as the Scrapy API to further help me in the development of my app.

Authors Klaudija Medeksaite - final year software development student in Galway-Mayo Institute of Technology, sole developer of Sky Scraper.

Chapter 1

Introduction

Sky Scraper was inspired by my passion for travelling as well as planning trips. During the Covid-19 pandemic, travelling has been very limited due to lock-downs and quarantining, so I haven't gotten to do it as much as I would like.
Every time I would plan a trip it was always really overwhelming due to the sheer number of choices available and this was essentially the motivation for developing this project. My app aims to gather information, including when flights take off, when they land, if they are cancelled or delayed, how often they're on time, etc. and then present it to the user in a convenient and comprehensible manner. In order to gather this information, I two primary options, I could have developed my own web scraper, or I could have used an API, otherwise known as an Application Programming Interface. There are multiple reasons why I chose to develop my own web scraper instead of relying on an API, including costs and difficulty finding one, outlined in the Technology Review chapter.

1.1 Web Scraping

Web scraping is when a piece of code is used to extract data from a website often for data mining and price comparison, amongst other uses [1]. I looked through a few websites, and even tried scraping some, before landing on the website I chose which is flight-status.info. The websites I excluded either didn't have the kind of data I needed, they were seldom updated or else there were few airlines or flights available. Some of these websites even charge subscription costs to see all of their data and others block web scrapers.
The website I chose has live information for over 500 airlines, it updates the information for each flight in real time, and the best part is they don't block

web scrapers, which I found even most airlines do on their own websites. Once I decided on the website to scrape, I had to compare my options on what software to use, and this is actually described further in the Technology Review chapter. I ended up using a Python script to actually extract this information and return it to me, which actually takes quite a long time as it is working on fifteen pages in one go. Because it takes so long (around seven minutes), I had to find a way to run it in the background somehow so the user wouldn't have to wait for the information to load for a long time. The tool I used to scrape the information is called Scrapy, which is a Python framework. The company who designed it, an Irish company called Zyte, actually have their own cloud host website built specifically for web scrapers called Scrapy Cloud. I deployed my scraping script on Scrapy Cloud and set it up so that it would run every fifteen minutes to update my database that lives on Google Cloud.

1.2 The User Interface

Once that was up and running, I started working on the front end of the application. I decided to do a mobile app for my project as such a huge majority of people own a smartphone, 3.6 billion people as of 2020 to be more accurate[2]. The choice to create it as an Android app was made quite easily as it is the most popular operating system for smartphones at this time, and it also happens to be the only type of smartphone I own. Having decided to create an Android app, I had a few options to choose from when it came to the development kit of my user interface. There are lots of options out there, including some really well developed choices such as React Native, but having seen how quickly Flutter was gathering popularity and with it still being quite novel, it seemed like the most exciting option for me. Dart is the programming language used to code in Flutter, so this was another new technology I was learning. For this project I used quite a few technologies I had little to no experience with, including Python, Flutter and Dart, Google Cloud, Scrapy Cloud and Android Studio.

In the Methodology chapter I will be describing the different planning and testing tools I used and how they all worked together in helping me complete this project. The following chapter, The Technology Review, will include more in-depth information and research about the technologies I used, why I decided to use them, as well as my own findings. The System Design chapter will elaborate exactly how my System works as well as all of the components that went into producing the final product. The System Evaluation chapter will go through the testing of the app against the objectives set out, future scalability, as well as the limitations of the system. Finally, the Appendix includes my GitHub repository link as well as summary of what is included there.

1.3 Context

The reason I set out to make the Sky Scraper app was to allow the user to access what I deem to be crucial information before spending money on an airline ticket. I wanted the user to be able to have this important information gathered in one place, in one app, so they could choose where they are flying from and a date and see all the possible destinations from different airlines. They would then be able to select options and compare them, which would make it easier to come to a decision. This could be the first step when planning a journey, and instead of booking flights and being disappointed when they have to wait hours for delayed flights, now users could avoid such flights and plan a journey with less bumps along the way.

1.3.1 The objectives

The main purpose of creating this app was to provide the user with an easy way of comparing flights so they could make a more informed decision when planning a trip. My own main objective as a developer, was to create a fully functioning system, including the back-end and user interface, while using multiple languages, and see how they can all work together. I wanted to challenge myself and do something bigger than I had ever done before while using and learning brand new technologies.

Chapter 2

Methodology

As a sole developer the methodology for developing my project was quite different to what it would have been within a team. When it came to planning and meetings, I didn't have to take anybody else's schedules into account. It also meant that I wouldn't need any kind of collaboration software, although my version control methodology would have probably worked quite well for this.

2.1 Planning

The first step of starting any new project, in my opinion, should always involve devising a plan. First, I created a list of different ideas that I had. I went through my list and thought about what each one would look like, what technologies I would need and how I could implement it. I had to decide based on a number of factors, such as the size of the project, how relevant it was in the current climate, who my target audience would be, and which technologies were available to me. I started researching the different technologies and languages, trying to figure out which option would make the most sense for me and what I was doing. I decided to create an application for tracking flight statuses for a number of reasons, and in a time where travelling isn't as easy to do any more, I wanted to make it simpler. I rely heavily on visual representations so I drew out what I wanted my app to look like, as well as the schema for my databases, before starting to design them. Once I knew what I wanted to create and the technologies it would require, I devised a proposal for my supervisor. As soon as they reviewed the proposal, and we were both satisfied with my idea, I created a checklist of what I needed to do to complete it. Then, when I had my checklist, I turned each item into a milestone and set myself a date by which I wanted it to be completed, I created a timeline. Planning my project this way made things easier and less overwhelming, I could see everything that had to be done, but it didn't seem like one big thing, instead it was broken down into smaller, more manageable parts.

2.2 Meetings

Throughout the duration of developing my project I had weekly meetings with my supervisor. In these meetings we discussed progress, and my supervisor would help me find resources when I was stuck. I found having these weekly meetings incredibly useful as they helped to keep me accountable - I hated showing up empty handed to a meeting. Even in a busy week when I didn't have much time to spend on my project as I had other, more pressing, priorities, I found myself spending at least some time trying to get something new to work. These weekly meetings really pushed me to stay on track with my timeline.
The meetings were also useful when I was facing any issues as my supervisor was very helpful in such cases. When developing a project on your own it's easy to feel like you are alone, and at times I struggled because I felt like nobody could help me and there was no one to turn to when I was stuck. Then I would talk through the problem I was having with my supervisor and, even though my project wasn't something they had worked on before, or knew much more about than me, they would often suggest different perspectives, which were incredibly helpful. It's true what they say, two heads are better than one, and if this were a team project I would have had someone else to discuss such things with, other than my supervisor.

2.3 Testing

I started with unit testing for the Python scripts, as well as the Dart files. This meant that I would test as soon as there was a new feature, to make sure it was working. For the Python scripts, I would do this by running the code, and this involved printing out values to the console, which would assure me that I was doing the correct calculations and everything was executing as it should. I created test tables on my database to test out any SQL statements I wanted to use, to make sure they were correct before using them in my code, as well as to test that my Python scripts were connecting successfully to the database. I devised an excel spreadsheet for testing different features of my dart app, and marked them off when the feature was working as expected.

I also had to do some regression testing with every new feature I implemented, to make sure that the older features were still working and I hadn't broken anything. Every time I would implement something new in my Python Script, before deploying it to the Scrapy Cloud, I would run it locally without connecting to the database. This ensured that if there was something broken I would catch it without messing the existing data up. Next, I would connect it to my test database table and make sure the connection was still working, and once it had passed my tests I would deploy it to the Scrapy Cloud.
The Flutter side of the project was quite easy to test as the application itself is quite simple and doesn't have too many features. For the unit testing I would just use the app to make sure the new feature was displaying what I expected it to, and due to Android Studio allowing hot reloading, it was made easier and faster. Once I had a new feature working and it was passing my unit test, I then did some regression testing to make sure I didn't ruin something else. This would involve going through the process of quitting the app, starting it over, then going through all the screens until I got to the result page. That way I could make sure all my features worked together and nothing was broken.
Once I thought everything was finished I worked through the acceptance testing using the proposal I had written before I began the project. For the acceptance testing, I went through my app and made sure every feature I had outlined in the proposal was implemented and worked as expected.

2.4 Version Control

I used GitHub as the version controlling software for a number of reasons. Firstly, I already knew how to use it, and when I was trying so many new things as it was, I didn't want to have to spend too long troubleshooting my version control software. I also had to use it for the submission of the project anyway, so if I decided to use different version control software I would have had to use two technologies instead of one. GitHub also proved to be a good choice because Android Studio projects automatically generate their own .gitignore files, which means the temporary or auto-generated files that don't need to be downloaded to run the project weren't being uploaded by default.
One thing that I have learned time and time again, is how important version control software is. It actually came in useful more than once throughout development, once when I had accidentally deleted a file that was then preventing me from deploying my scraper to Scrapy Cloud, and the other time when I had over written a file with something that didn't work and I couldn't

undo it.

Chapter 3

Technology Review

When I set out to develop this app, the idea was quite different to what it ended up being. In the beginning, I had no idea what technologies to use in order to achieve the ideas I had, and there are so many options when it comes to software these days. I had to research different technologies to be able to come to a decision for each component of my project, and even having done that, in hindsight I might have chosen some differently anyway.

3.1 Web Scraping

There are many different terms for web scraping, such as web harvesting or screen scraping, but it can be simply described as a program that extracts desired data from a chosen website[3]. Web scrapers are often automated and are usually designed to run periodically, for example every day at noon, or every fifteen minutes. Scrapers can be incredibly useful if a website updates often and you rely on it for information. It would be difficult and not efficient to have a person manually check the website every half an hour for changes and update the database accordingly, especially if there are dozens of pages with tens of rows to look through. They are especially useful when the website is consistent in its design and layout because then your web scraper can just look for every, for example, row in a table with id of ”flight number” to find all the flight numbers.

3.1.1 Why not an API?

What is an API?

API stands for Application Programming Interface, it is something, like another program, that you can make use of when designing your app which

allows it to access certain information generated or retrieved by another program. For example, if you were to develop a program and make use of a weather API, your application would be able to access weather information. There are thousands of APIs out there and a majority of them are free. Some, however, require an API key which can have costs associated when generating it.
Most of the time it would make more sense to use an API if the option is there. However, there are several reasons why one would choose to create a web scraper instead of using an API.

- If an API does not exist for what you need. Sometimes what you need to be able to access is so niche that there is not an existing API readily available. In my case, there were actually quite a few options available, such as FlightAware API.
- If the information you are trying to access doesn't warrant an API. Sometimes you need just one piece of information from a certain website. That information may or may not be available in an API but it can simply be quicker and easier to scrape this information than to spend time searching for the right API.
- Web scrapers do not have rate limitations. APIs almost always have usage policies that apply limitations on how many requests you can make per day, and if you would like to make more you could find yourself paying for a premium version. Such limitations do not apply to web scrapers, the only limitations you should conform to (it's not even mandatory) are what is set out in a website's robots.txt file.
- Any web scraper you design yourself will be free, the same cannot be said for an API. Some APIs will be free, and most of the time you may not need to go over their limits. However, with a web scraper that you design yourself there won't be a cost associated with running it as many times as you like, and you can modify it as well.

What is a robots.txt file?

A robots.txt file is a text file that a website could have specifying what is and what isn't allowed to be scraped from it [4]. Most smaller and less important websites will not have a robots.txt file as it isn't overly important to them if the information is gathered by a third party, however bigger companies, such as Ryanair, will make use of this file. Ryanair's robots.txt file actually prevents scrapers only from accessing flight information, but everything

else, including 'about' information and airport lists, is not blocked from web scrapers. Such files are actually only instructions and can be set to be ignored by a scraper, they are actually not legally binding.
The reason I chose to create my own web scraper instead of using an API includes most of the reasons mentioned above. I wanted something that I could run often and not incur any penalty for it, such as having to pay money. Although there are APIs out there for my purposes, a lot of them are U.S. focused and wouldn't include airlines like Aer Lingus or Ryanair, which are really the more revelant airlines to me as a resident in Ireland. Another reason for coding my own scraper was the experience itself - I was interested in seeing how it works and incorporating custom data from a website into my project.

3.1.2 Python

The language

Similarly to a lot of popular programming languages, Python has been around since the previous century - the first version was released in 1991. Unlike many other languages it was actually created by one person, Guido van Rossum, however it is open source and he did not develop every single aspect of the language by himself[5]. It is intended to be a simple and intuitive language and "as understandable as plain English". Due to the fact that Python is an open source programming language, there is an abundance of different libraries and packages available for Python making it that much simpler and faster to program something using this language. Python is a widely popular language, some sources placing it as third or fifth on their list of languages, IEEE has put it first on their list of the most popular language in 2020[6], for the fourth year in a row.

Python in Web scraping

When it comes to scraping the web, it is possible to do it in a wide variety of programming languages, however it is widely insinuated that if there is a web scraper it's likely written in Python. The top results for web scraping in Google and Google Scholar all use Python, it seems to be the de facto web scraping language. There are a few reasons why Python is so popular for this specific activity.

- Python allows the programmer to avail of multiple scraping frameworks and parsing libraries, such as Beautiful Soup and Scrapy. Without one of these tools it would be quite difficult to create a web scraper, they are an enormous help.

- Python also supports XPaths (language used to select elements in XML or HTML files) which are very useful when you need to select certain parts of a website. You can simply specify what the XPath for the element is and Python can find it. When you are scraping many pages of a website and looking for the same data in the same place, this is the easiest way to find the information.
- The large amount of documentation and examples of Python web scrapers online is a tremendous advantage of using the language. For somebody who has never developed a web scraper before, and would have no idea on where to begin, the examples available online made it a lot easier to get my head around.

Python is also a language I had very little exposure to before beginning this project, but from what little experience I did have, there was a desire to learn more, which is another reason I decided to use it for my project. Python in my opinion is a beginner friendly language, and having programmed in Java and C before, I found it quite easy and pleasant to pick up.

3.1.3 Choosing the Scraper

Having chosen Python as my programming language for the web scraper I then had to choose a scraping framework or library. Luckily, as mentioned above, Python has multiple options. When trying to figure out how to scrape from a website using Python, I had no idea how to actually go about it, what was involved, or where to begin - it was a whole new experience for me. I didn't know that I would necessarily need a scraping framework or library, but every resource that explained how to do it either used Beautiful Soup or Scrapy in their examples. Although they are not the only options for web scraping, they are definitely two of the biggest contenders.

What is the difference between a package, a library and a framework in Python?

A package is a collection of modules, and each module is a .py file or a script. A library is then a collection of packages, some popular ones include Numpy and TensorFlow. A framework provides basic functionality that can be adapted to your own program to suit your needs and can also be described as a collection of packages.

The difference between a framework and a library is mainly the complexity, libraries are less complex than frameworks. The Scrapy framework is a web-crawling framework which allows the user to get data from website pages and even extract data from APIs. Beautiful Soup is a parsing library and, as per the definition of a library, is less complex than Scrapy.

Beautiful Soup

For somebody just testing the web scraping waters, Beautiful Soup is the way to go. The library is ideal for the occasional scrape here and there and is so much easier and quicker to set up than Scrapy. Beautiful soup also relies on the programmer importing another package called requests to connect to the website, unlike Scrapy which has this functionality built in. It's a simple and quick way to scrape some information, but if you are looking to scrape multiple pages with multiple rows of records from a website, this is definitely not the best option. Beautiful soup is more of a parser, which essentially means it is a program used to break data down into more manageable chunks, but it can be built to become what's known as a crawler (i.e. a web scraper that scrapes multiple pages) similar to Scrapy. Doing so would be a waste of time and effort though, since there is already a program designed for the specific purpose - Scrapy.

Scrapy

Scrapy is an open sourced framework designed by a company called Zyte, formerly known as Scraping Hub. After playing around with Beautiful Soup for a little while, I tried out Scrapy. It was a bit more complicated to set up in the first place, but the documentation was very useful and provided clear step-by-step instructions on how to do it. Scrapy is actually specifically designed with web scraping as its purpose and is not just a parser like Beautiful Soup, but Beautiful Soup can be used in combination. It can also scrape more than one page at a time from the get go, with just a small amount of coding compared to Beautiful Soup.
Scrapy is designed to be asynchronous because it is built alongsude Twisted[7] which is used in building network applications[8]. This means that it can process requests more efficiently and is therefore faster than its counter parts that are not asynchronous. In addition to this, the company actually has a Scrapy Cloud, which is a cloud space where your Scrapy spider can be stored and run periodically if necessary. Due to the advantages of the ecosystem created by Zyte and the optimisation when it comes to scraping multiple pages in one go, I decided Scrapy would be the best option for my project.

3.2 Cloud hosting

Having decided to use Scrapy with Python to create my web scraper, I started to build it. At first I followed the examples, I tried to scrape a simple text website and it was pretty cool when I got the results. To begin with I stored the results in a text file as I was just learning how to use it and didn't want to over complicate things from the start.
However after playing around with it for a while I realised two things. Firstly, keeping the results in a text file wasn't a feasible option, they had to be stored somewhere that could fit a lot of information and that allowed easy retrieval. Secondly, although Scrapy gets the information asynchronously and it is fast, it's not immediate and asking the user to wait five minutes while the website gets scraped when they want to use the app is just not good enough, nobody would wait. The clear solution to the first problem was to use a database, it just makes sense - a database is a place where you can store and update your information for easy retrieval.

3.2.1 Databases

A database can be defined as a structured and organised collection of information. Usually a database is controlled by a DBMS (Database Management System) which is a program[9] such as Oracle, or PostgreSQL, that can catalog and organise data as well as allowing users to manipulate it. For a user to be able to use or "query" the information stored, they must use an SQL (structured query language).
There are many different types of databases, and they all have different purposes. For each category of database, there are also many options to choose from, and it can be quite overwhelming when trying to figure out which one to select for your own purposes. The two types of Database Management Systems I actually had some experience with were MySQL and MongoDB, and I figured one of these would definitely work for me and my needs so it narrowed the decision a little for me.

MongoDB

MongoDB is classified as a NoSQL database, which means that it does not store information in relational tables and due to this reason it can be more flexible. MongoDB being non-relational also means that the schema of the database doesn't have to be pre-defined, basically you can make it up as you go. A relational database simply means data is stored in a structured format - the relational table cannot have most of its rows have five columns, and a

few rows with six columns as that wouldn't be relational.
There are also different kinds of NoSQL databases - object oriented, document, key-value for example - and once again, they all serve different purposes. MongoDB uses a JSON (JavaScript object notation) document storage model, which makes it a general purpose database that can be used for most types of data, as well as easy to use alongside other programming languages. A NoSQL database can also sometimes be faster at retrieving information - 'joins' in relational databases can often take more time. In relational databases data is separated into different tables and if you are looking for information in a table that is related to a certain row in another table, you would find yourself using a 'join' that allows you to do so. Therefore instead of querying just one table, you would often be querying two, sometimes three tables when using a relational database instead of just one when using a NoSQL database.
MongoDB is also easily scalable, again due to the fact it is not relational, it doesn't have as many constraints and ties as an SQL Database Management System does. It is horizontally scalable, which means that by simply adding more servers the database will become larger making MongoDB a better option constantly growing and changing data. MongoDB is also vertically scalable by adding CPU or RAM[10] to the server which speeds up processing.
There are actually very few drawbacks of using MongoDB as the database, except minor things such as not supporting global transactions, i.e. only being able to access one resource at a time, which wouldn't really be an issue for my project.

MySQL

MySQL is another Database Management System, and is almost the polar opposite of MongoDB. It is a relational database - it uses tables and rigid schemas, the data is separated out into said tables instead of all being put in one place, and there are rules for the relationships between the tables. Rules for the relationships define how the data is organised, and the relationships can be one to one, or one to many, for example, which when enforced prevents inconsistent or duplicate entries. MySQL is also both vertically and horizontally scalable, however, when scaled horizontally by partitioning the data into smaller parts and spreading it across multiple servers for example, it can make querying slower but is quite inexpensive.
The fact that MySQL uses relational tables can be a positive or a negative thing depending on what your needs are. If you need to retrieve the values from the same columns every time, and you know what kind of data you're

going to be inserting from the beginning, then there's nothing wrong with using a relational database. However, if your data is going to be more dynamic or unpredictable, then the better option would definitely be a non relational database, such as MongoDB. Sometimes the uncertainty of what is going to be in the document when using MongoDB can even feel a little more messy, but sometimes it is quite a necessary chaos.
As I knew what kind of data my web scraper would be returning every time, it was predictable, and I decided that a relational database like MySQL would work perfectly for my use case. I liked the certainty of knowing exactly what was going to be in each table, what type of variables and knowing the structure of my database as well as how each table related to the others.

3.2.2 Cloud Platform

Now that I had decided on the Database Management System I needed to figure out where to keep my database servers. There are a few reasons why it would be better to store my servers on a cloud platform of some sort, instead of on my local machine.

- It is much easier to adjust the amount of storage available to you on a cloud platform than it is on your local machine.
- The database can be accessed from any machine, as long as it can connect to the internet, and in this day and age most devices can do that.
- When database servers are hosted on the cloud they are safer from accidents. You can't spill water on your cloud server and wipe the hard drive, but you can with your own device.
- A lot of the time it is actually cheaper in the long run to store your database on the cloud, especially for companies. There is no longer a need to buy all the expensive equipment, less space needed to store it and less effort and costs in maintaining the server when you can just do all of that on the cloud from one computer, and let the cloud host maintain its machines.

Due to all of the above reasons I decided it was vital my database be stored on a cloud somewhere and not on my own machine. The next step was choosing which cloud platform I would use for my database. Again, I wasn't stuck for options. Immediately there were three main contenders - Microsoft's Azure cloud, Amazon's AWS and Google cloud. All three cloud platforms offered options for MySQL databases.

Azure Cloud

Firstly, taking a look at Azure's website and trying to figure out which database would best suit my needs was not quite so simple. All I knew was that I needed somewhere to store my MySQL database. It seemed simple to me, but looking on Azure's website there were quite a few options to choose from and a table comparing features, such as HTAP and OSS based services, that had no relevancy nor meaning to me. It was quite overwhelming, all I wanted was a simple server, so I thought the Azure Database for MySQL option would work. It was a relational database and used MySQL, exactly what I needed. Azure had quite a lot of documentation available which is something that is greatly appreciated by students and experienced programmers alike. My mind wasn't made up yet though, I wanted to check out the other options.

Amazon Web Services

On AWS, I went through a similar process of not really knowing what option was best, just choosing something that seemed to be simple enough that fit my very basic criteria. The option that I thought would work best from AWS was the Amazon Aurora, and Amazon actually boasts that the Aurora is up to five times faster than MySQL. Again, this seemed like a viable option, but I still wanted to investigate Google's option, so far I didn't see much of an advantage or drawback for either option.

Google Cloud

Looking through my options on Google, it was by far the simplest website to navigate. Google uses a very clean layout and everything was explained clearly for each product. Knowing I wanted a MySQL server on the cloud, it was easy for me to understand Cloud SQL was the database service I was looking for, no ambiguity or confusion, even though there were plenty of other products. Google has so much documentation made readily available, which is so useful for someone like me who had never really used a cloud service like this before, although so did the other options. Another reason Cloud SQL appealed to me was the website was very clear about exactly which features would and would not work, the reasoning and how to get around the issues, which I really appreciated, even if none of them actually applied to my project. Neither Azure, nor AWS, mentioned any drawbacks to using any of their products, which to me felt like a sales pitch.
All three options had quite a lot in common and made good options, the only difference that I truly cared about between them was the pricing. Trying to

figure out the pricing was no easy feat, but I realised that all of them charge an hourly rate and a per Gigabyte rate, and it also varies depending on what location you select. So regardless of which cloud platform I chose it would all even out to about the same cost, therefore it didn't affect my decision too much. I actually also had a voucher for money off for Google's cloud platform, and that was really the deciding factor, considering there wasn't much of a difference between them otherwise.

3.2.3 Scrapy Cloud on Zyte

Now that I had created my web scraper using Python and Scrapy, and I was storing my scraped data in my Cloud SQL database, I was reminded of the second issue I had - Scrapy is fast, but it's not immediate. This, along with the fact I would need to scrape multiple times a day, everyday, to keep my information up to date meant I wanted my web scraper to run automatically somewhere in the background. At first I figured it would be easiest to just have the web scraper running on Google cloud as my database was on there as well, but soon I realised there was a better solution.
Zyte is the company that created Scrapy, the web scraping framework I am using to get my information from the website. Zyte also has their own website where they have something called a Scrapy Cloud. The Scrapy Cloud is also a cloud platform, but it is designed with one specific purpose in mind which is running scrapers. Using Scrapy Cloud was probably the easiest part of the entire project, I simply had to install a Python plugin and I could deploy my Scrapy spider within half an hour. Deploying and running the spider on the Scrapy Cloud was free, however I needed to set up automation.
To do this I had to set up a periodic job (a program that runs this scraper every fifteen minutes without me having to tell it to do it every time) which was not free, but is a set monthly charge regardless of how often you make your job run. Another advantage of using Scrapy Cloud is you can actually connect it up to GitHub, so instead of pushing changes to GitHub and then re-deploying to Scrapy Cloud, you can just push to GitHub and Scrapy Cloud will adopt the changes too.
Something I actually found very useful in Scrapy Cloud was how easy it was to use their logging features. When I first tried to use a function on Google Cloud to do the same thing, it wouldn't work and when I looked at the logs all I could see was that it crashed, but I wasn't able to see the error. I tried for days to figure out the logging to no avail, eventually giving up and trying to figure out what was happening a different way. With the function on Google Cloud I had tried putting in print statements, and importing logging packages, and more. With Scrapy Cloud it couldn't have been any simpler

- all my print statements from my Python script were being output to the logs. This made troubleshooting when something didn't work ten times more efficient, and my life a little easier.

3.3 The App

Having gathered all this information on the back end of my project, I wanted to allow the user to access it easily. In order to do this I had make some kind of user interface, like a program or an application, that somebody who has never seen a snippet of code before would be able to use. I needed to create the front end.
I very quickly landed on the choice to develop a smartphone app instead of a website or a computer program. There are actually a few reasons I chose to do this.

- Although most households in Ireland have access to a computer[11], a lot of people rely on their smartphones for most operations and would not reach for their computer if they could do the same thing just as easily on their phone.
- This is an app focused towards travellers, people on the go, that may not be bringing their computers with them on the journey, even if they do own one, due to extra weight and unnecessary risks.
- Even though it has been shown in recent surveys[12] that more people are using their smartphones to access websites than computers, websites are still not entirely geared towards viewing on a smartphone. I, for one, would rather not open websites on my phone as usually the formatting is off, words can overlap, or I might need to scroll through the entire website to find the button I'm looking for when it would be much more obvious on a computer. This is another reason why an app would be better for a smartphone - it is built specifically for that device.
- I really wanted to make an app that I would be able to have on my own phone, and even use for my own needs.

3.3.1 Android

Due to the fact that I, myself, have an Android phone and personally prefer the Android operating system, I decided to create an Android app. Android, initially released in 2008, is currently the world's most popular operating

system[13] as of 2021, and is actually open sourced. The Android Open Source Project is in fact led and sponsored by Google, and Google has become ingrained in the system with default apps, such as Google's Play Store and Gmail, coming pre-installed with each Android device. With Android and Apple's iOS being released within a year of each other, they became fast competitors and the rest of the operating systems became irrelevant - Windows phones for example - released in 2010 and even Bill Gates admitted he used an Android himself in 2017[14].
Android's Play Store offers millions of apps, I even put an app on there myself a couple of years ago, which means there are so many options to choose from, and if one app doesn't do what you would like it to you can find a dozen others that do. Another great thing about the apps on the play store is they are very often free apps even when their Apple equivalents on the App store are not. Although the Android operating systems on each phone will be the same, possibly different versions, each brand of Android smartphones have their own launchers - One UI launcher is used in Samsung devices, MIUI is used in Xiaomi devices, etc. One of the best parts about Android devices and the fact Android is open sourced, is that they are fully customisable, and if you don't like something you can usually change it, for example with the launcher, if you don't like the inbuilt one you can change it to a different one. Often Android devices can be more cost efficient, which again, can really be put down to the fact that the operating system is open sourced, so Android device manufacturers can spend less money on the development of the operating system than Apple would.
Something that people either really like or really dislike about Android devices is the amount of choice out there. There are now so many companies producing Android phones, like Samsung, Xiaomi, Lenovo, OnePlus, etc., and every company has lots of different models to choose from as well. The average person who doesn't know too much about technology could have a difficult time choosing an Android device for themselves when faced with all the options.
There are, however, some drawbacks to Android devices as well. For one, the apps on the Play store are not as thoroughly screened as the ones on the Apple App store, so they can potentially contain malware. Some Android phones do have inbuilt screening for such a thing when you download an app though. Every time I do, I get a screen for a few seconds to tell me my phone is checking if there's anything harmful in the app, and then assures me it is safe. This is also done when downloading from somewhere other than the Play store. I put my own app's APK (Android Package - an executable file for Androids) on Google Drive and tried to open it on my smartphone - I got the same pop up.

In older versions of Android, apps would be constantly running in the background and consuming a lot of battery in doing so. This has actually been resolved in recent years, there are now settings in the smartphone to prevent apps from doing this. Having such setting enabled, and being an avid user of my own smartphone, I can definitely say this works as I could easily go two days without charging my phone while using it as normal. Android is also known to be more difficult to develop for due to a number of reasons. Firstly, because there are so many different companies, creating all kinds of different Android smartphones and tablets, in all different sizes, it can be difficult to design a ”one size fits all” app. Testing an Android app can also be quite a nightmare. Different devices and Android versions will likely handle the app differently, so testing on only one device is not really an option - you wouldn't be getting the full picture of the app's behaviour.
Traditionally apps for Android devices were developed using Java and XML. Multiple XML files were needed for a single page of the app making it more complicated to develop the app. However this seems to be no longer the case, using dart I only needed one file per page, so although I don't have experience of developing an Android app using Java and XML, being able to use Flutter simplifies this.
Although my mind was already made up on which operating system I would be building my app for, as I only have an Android to test with, it was still interesting to find out why development could get difficult for these devices.

3.3.2 App development

Android Studio

When developing code it is important to choose the right environment. Android Studio is the official Android app development IDE (integrated development environment - the program in which code is written, in layman's terms), although there are other options available such as IntelliJ and Visual Studio. For the development of my Python script I actually used the Visual Studio IDE as it is easy to use and simple, no bells and whistles, perfect for coding in python when no extra features are really necessary. Android Studio, on the other hand, does offer some extra features that are geared towards developing apps.
Firstly, there is an inbuilt Android emulator in the IDE, which makes testing your app that much more convenient. An Android emulator is software that allows you to see and use a device that you do not physically have. As mentioned above, one of the difficulties of developing apps for Android is being able to ensure that the application will work on all Android devices

and not just the specific model you own. Using an Android emulator means you can test on all kinds of different virtual devices and within the Android studio there are over twenty models of smartphone and around five models of tablets to make use of, all with different specifications.
Another useful feature of the Android Studio is the "hot reload" option. When developing code, I found it useful to plug in my phone to test it as I went along and see if things were working how I expected. The hot reload feature offered by Android Studio meant that every time I made a change I didn't have to wait for the application to be re-downloaded onto my phone. Downloading the application for the first time, when I plugged it into the computer every day, would take up to two minutes. Not having to wait that long for every little change was a huge time saver.
Android Studio, like all of the best IDEs, offers suggestions and predictions in a drop down menu as you type your code. Something extra that Android Studio does is actually automatically comment your code for you, so in a tangled mess of bracket, braces and parentheses the IDE helps you keep track of them all. Of course, if an IDE doesn't do this it isn't usually a deal breaker, but it's just a nice add-on and I appreciated it.
Due to the Android Studio IDE being made specifically for Android development, it is incredibly easy to build your Android app executable file (the APK). This feature is not as readily available in other IDEs that aren't geared towards specifically Android app development, and can often be difficult to find. The Android Studio simplifies Android app development as much as possible, and makes something that's deemed as rather difficult a lot easier. Even though I do appreciate Visual Studio's simplicity, sometimes an IDE that's as feature-rich as Android Studio can be much more beneficial, depending on the project of course.

Dart and Flutter

All Flutter apps are written in the Dart programming language. Flutter is an open source SDK (software development kit) used to develop apps for Android and, of course, created by Google. Although the Dart programming language has been around for almost a full decade, Flutter is still quite new, as the first version was released in 2017. Since Flutter's release, it has actually proven to be exceedingly popular based on reviews from Gartner[15] and uses of Flutter on GitHub[16] exceeding those of React Native (which was previously the most commonly used in the genre). There are a number of reasons why it proved so popular, including the use of Dart and the advantages that come with it.

- Flutter uses Dart, which is an object oriented language that is optimised for building the user interface, i.e. it is client-optimised. This, in itself, is already a good reason for choosing Dart when developing an application, especially if most of the work is being done elsewhere, like in a Python script for example.

- Dart code compiles into native code for the smartphone which means that, unlike React Native, which is arguably one of Flutter's biggest competitors, it does not need that additional layer for communication with the platform[17]. This makes starting up the application that much quicker and supports hot reloading, as explained above, saving time while improving performance.
- Although not necessary in my case, Flutter does allow multiple platform development. This includes Android apps as well as iOS apps, and since the release of Flutter 2 it also includes Linux, MacOS, Windows and multiple Web platforms. This means that if you learn to develop apps using Flutter and Dart, you can really develop applications for most smartphones, tablets, computers, smart devices and even web apps. Since the release of Flutter 2 and its support for all of these platforms, it has become an even fiercer competitor for other major SDKs, such as React Native.
- Widgets are another major aspect of Flutter, and they act as blocks of the user interface. Every part of a flutter app is called a widget; every button, every piece of text or drop-down menu, even the margins and padding. There are different categories of widgets as well as different types. The options are vast and this is what makes the widgets so useful, you can create almost anything you can imagine by using them.
- Using Flutter for Android app development means you no longer need XML files to create the view as before. Previously, almost all Android apps would be developed using Java and XML files. The XML files would act as the views, or in other words the display, and they would have to be referenced in the Java code. Dart actually gets rid of that concept and allows it all to be in one place. It actually took me some time to understand this fully when I started developing the app, but once I realised I could include some logic in the view, in a way that was familiar to me, it really sped things up.

These are only some of the advantages of using Flutter, there are many more, but these were the ones that really sold me on using it for my project. Although there are alternative options with many similar benefits available, I was really interested in trying out this new, up-and-coming technology that fit my needs perfectly, as well as providing a challenge for me, which is the primary reason for choosing Flutter with Dart.

Chapter 4

System Design

The system for the app can be described as having three parts. Firstly, there is the back-end which includes the Python script that gets information from the website I used using the Scrapy framework. Then, there are the two clouds, the Google Cloud where the database is, and the Scrapy Cloud where the Python script runs on. Finally, there is the User interface in the form of an Android app, which was developed using Flutter. The app requires an internet connection and uses it to connect to the database on the cloud. The architecture of the system can be seen in figure 4.1.

4.1 The Back End

The back-end has multiple components that work together in order to update the database with useful information. The Python script uses the Scrapy API in order to retrieve information from the website.

4.1.1 The website

The website I am gathering information from is called flight-status.info and it contains thousands of records for different flights of over five hundred airlines. Each flight's status is updated in live time, so if you're catching a flight you can go to this website to check on the status of it. Flight numbers are used to denote the route and can be reused multiple times a week. Some flights are consistently late and that can be a factor when planning a trip for many people. Flight-status.info does not actually show trending information however, it only shows the status at that time. So if a flight is currently on time, the website would update the status to say it is on time, even if said flight tends to be consistently delayed. But if a person looks at that

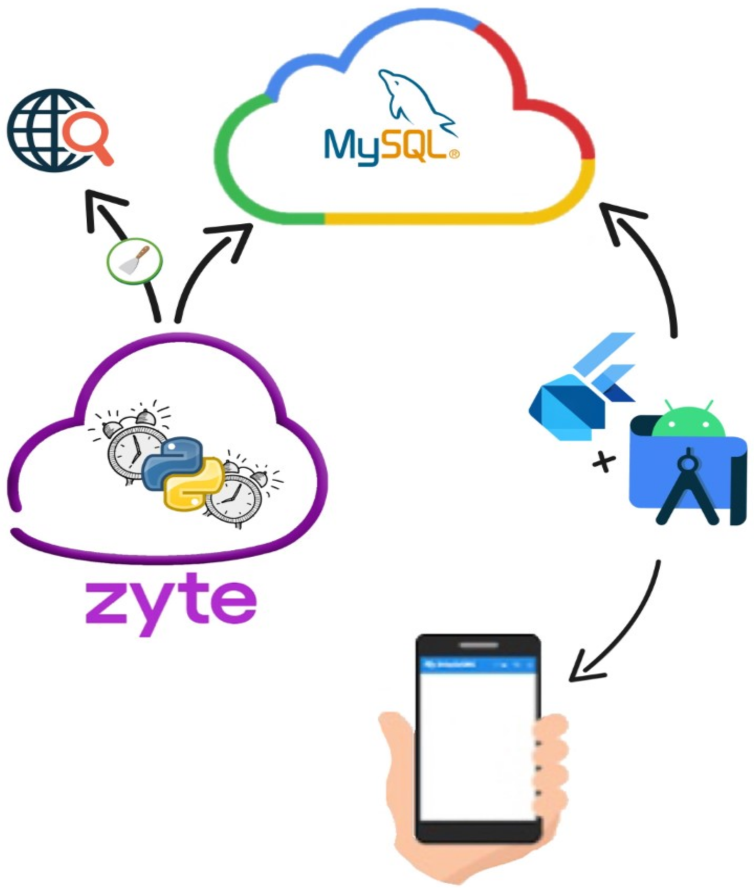


Figure 4.1: System Architecture

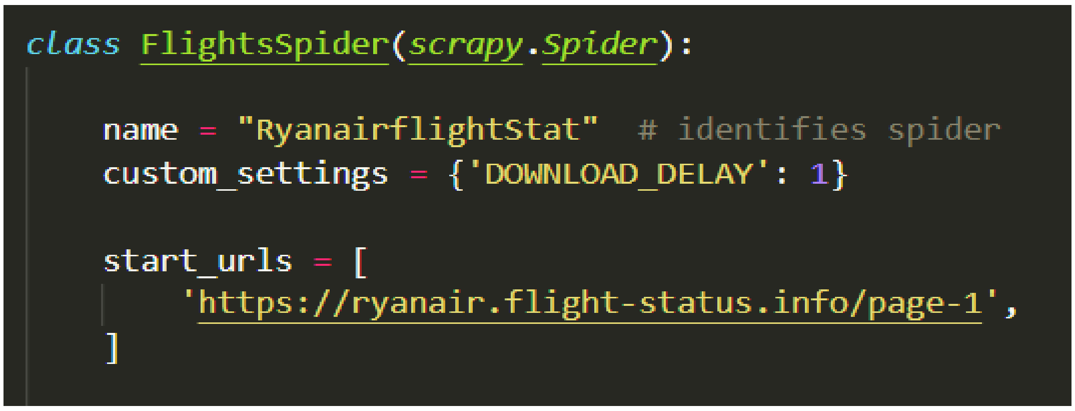
information to try and predict if that same flight they have booked for next week will be on time, and they see that it is currently ”on time”, then this information could be misleading or at the very least it is not showing the entire picture. My point is that this website has useful information, but it could be made more useful, which is one of the reasons it was a good option for me.
I wanted to take the information that website gathered and use it to create an app that would give the user more insight than just the current status of the flight. I wanted to allow the user to choose their flight by seeing the options available to them, and being able to compare them not just based on the current status and the scheduled duration, but by information that is updated over time to show a clearer image.

4.1.2 The Python script

The Python script has all of the logic for the app.

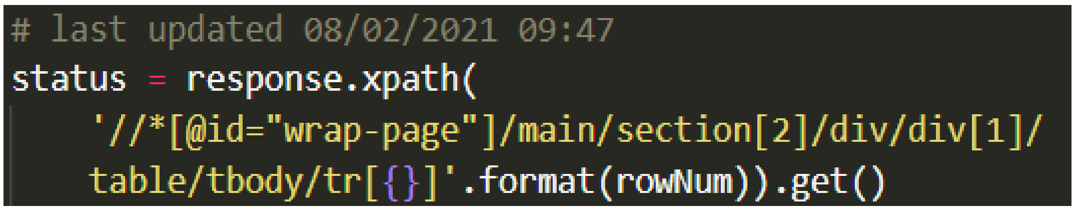
Scrapy

I used the Scrapy framework to gather the information from the flight-status.info website. To do this I first had to download Scrapy, then I created a project. I followed a tutorial from the Scrapy documentation to do this, and it was explained quite thoroughly, step-by-step. Scrapy uses GET requests to the given website to retrieve the information. When creating the scraper, a few things need to be defined, such as the name of the spider, the starting URL, and the parse method as can be seen in figure 4.2. Something else that can be seen in the figure, is that I defined a download delay of one second. This is done when downloading consecutive pages from a website in order to not overwhelm the servers of the website[18]. Due to the web scraping happening in the background on the cloud, this doesn’t actually affect the end user whatsoever.
In the parse method is where most of the work is done. The website is consistent, in that every page is named the same, but with an incrementing number, and every table on every page has the same amount of rows. The pages are ordered by time of update, so the most relevant information comes up first. I have a while loop in my parse method to go through and GET every row of the table as shown in figure 4.3. What I get back is the HTML (figure 4.4) for the row of the table, which I then pick apart to get the exact values I need and put into a dictionary. When I have processed the information I need, I move on to the next page by using a yield statement with my next request as seen in figure 4.5, and call the parse method on it again. In



```
class FlightsSpider(scrapy.Spider):

    name = "RyanairflightStat"  # identifies spider
    custom_settings = {'DOWNLOAD_DELAY': 1}

    start_urls = [
        'https://ryanair.flight-status.info/page-1',
    ]
```

Figure 4.2: Defining the Scrapy Spider



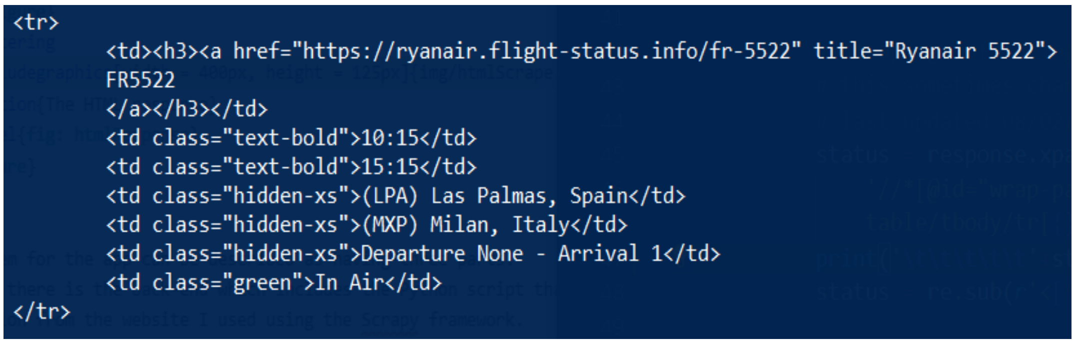
```
# last updated 08/02/2021 09:47
status = response.xpath(
    '//*[@id="wrap-page"]/main/section[2]/div/div[1]/
    table/tbody/tr[{}]'.format(rowNum)).get()
```

Figure 4.3: The GET Request

the figure you can see I have an if statement making sure I'm not at the final page that I want to scrape, and if I am the parsing simply finishes.

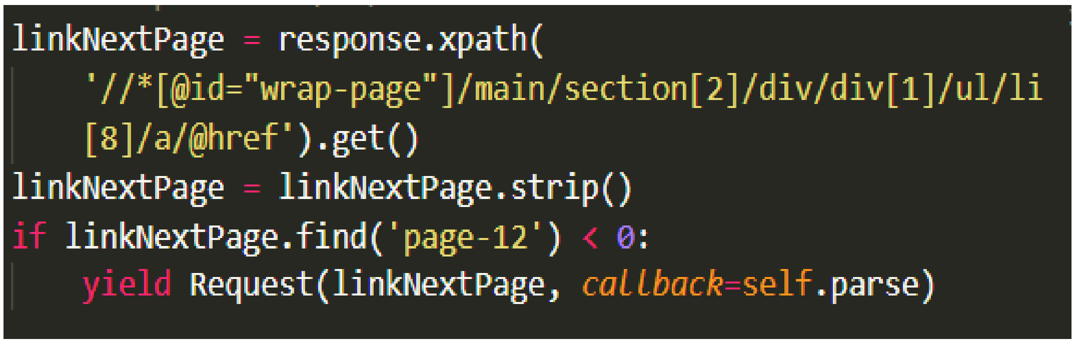
Connecting to the Database

My Python script is also from where the database gets populated. The database instance is on Google cloud and has its own public IP address, which is what the Python script uses to connect to it (as seen in figure 4.6) and update it (as seen in figure 4.7). The example is only one of the MySQL queries used but there are several different ones. The parse method aims to update the Ryanair Flights table (see figure 4.9), and if it isn't successful it will attempt to update the Airports table (see figure 4.10). This is because the reason I get an error when trying to update the Flights table is usually the fact that one of the airports I am using does not exist in the Airports table, and there is a relationship between the two tables, created through a foreign key. Once the script adds the new airport into the Airports table, it attempts to add the flight into the Ryanair Flights table again which is then possible.
After doing all of this, I run two more methods to update the Statistics table (figure 4.11) and the Fly Days table (figure 4.12). The Statistics table is



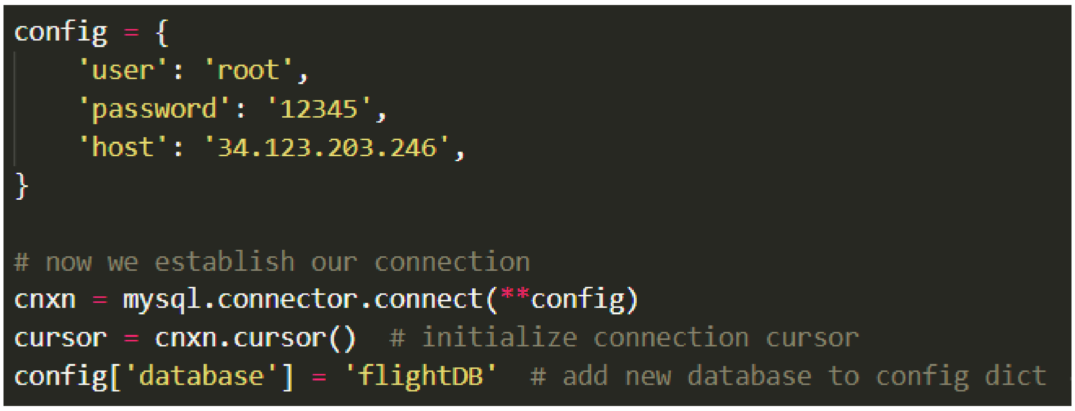
```
<tr>
        <td><h3><a href="https://ryanair.flight-status.info/fr-5522" title="Ryanair 5522">
        FR5522
        </a></h3></td>
        <td class="text-bold">10:15</td>
        <td class="text-bold">15:15</td>
        <td class="hidden-xs">(LPA) Las Palmas, Spain</td>
        <td class="hidden-xs">(MXP) Milan, Italy</td>
        <td class="hidden-xs">Departure None - Arrival 1</td>
        <td class="green">In Air</td>
</tr>
```

Figure 4.4: The HTML response



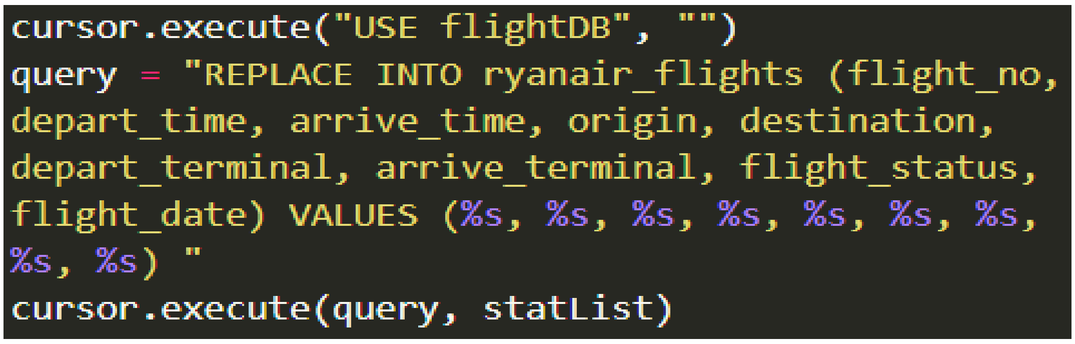
```
linkNextPage = response.xpath(
    '//*[@id="wrap-page"]/main/section[2]/div/div[1]/ul/li
    [8]/a/@href').get()
linkNextPage = linkNextPage.strip()
if linkNextPage.find('page-12') < 0:
    yield Request(linkNextPage, callback=self.parse)
```

Figure 4.5: Specifying The Next Page for Scraping



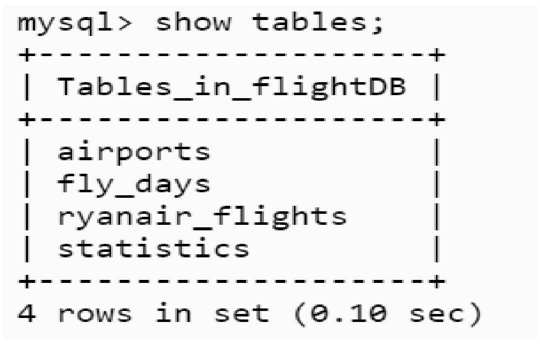
```
config = {
    'user': 'root',
    'password': '12345',
    'host': '34.123.203.246',
}

# now we establish our connection
cnxn = mysql.connector.connect(**config)
cursor = cnxn.cursor()  # initialize connection cursor
config['database'] = 'flightDB'  # add new database to config dict
```

Figure 4.6: Connecting to the Database



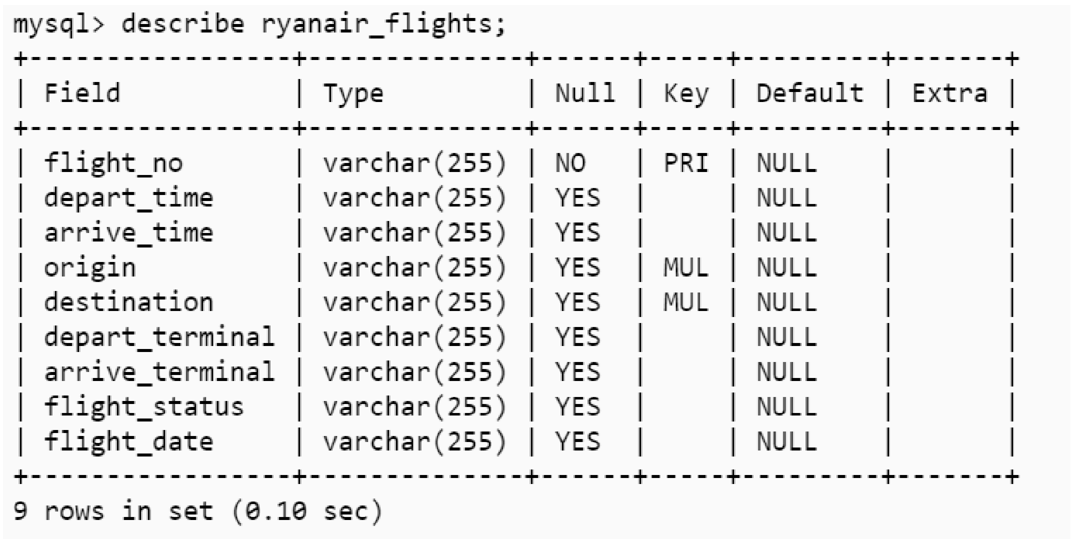
```
cursor.execute("USE flightDB", "")
query = "REPLACE INTO ryanair_flights (flight_no,
depart_time, arrive_time, origin, destination,
depart_terminal, arrive_terminal, flight_status,
flight_date) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, %s) "
cursor.execute(query, statList)
```

Figure 4.7: Executing the Query



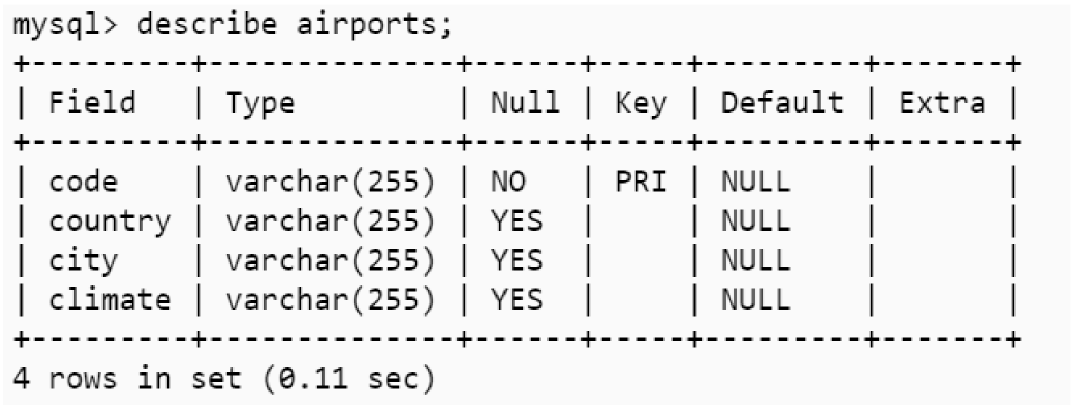
```
mysql> show tables;
+---------------------+
| Tables_in_flightDB |
+---------------------+
| airports            |
| fly_days            |
| ryanair_flights     |
| statistics          |
+---------------------+
4 rows in set (0.10 sec)
```

Figure 4.8: Show Tables Results for Database



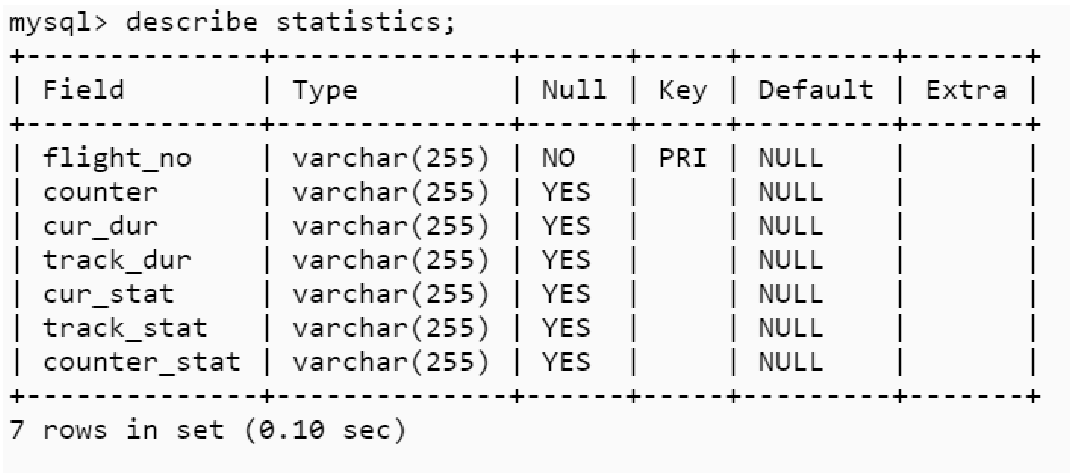
```
mysql> describe ryanair_flights;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| flight_no       | varchar(255) | NO   | PRI | NULL    |       |
| depart_time     | varchar(255) | YES  |     | NULL    |       |
| arrive_time     | varchar(255) | YES  |     | NULL    |       |
| origin          | varchar(255) | YES  | MUL | NULL    |       |
| destination     | varchar(255) | YES  | MUL | NULL    |       |
| depart_terminal | varchar(255) | YES  |     | NULL    |       |
| arrive_terminal | varchar(255) | YES  |     | NULL    |       |
| flight_status   | varchar(255) | YES  |     | NULL    |       |
| flight_date     | varchar(255) | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
9 rows in set (0.10 sec)
```

Figure 4.9: Ryanair Flights Table Schema



```
mysql> describe airports;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| code    | varchar(255) | NO   | PRI | NULL    |       |
| country | varchar(255) | YES  |     | NULL    |       |
| city    | varchar(255) | YES  |     | NULL    |       |
| climate | varchar(255) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
4 rows in set (0.11 sec)
```

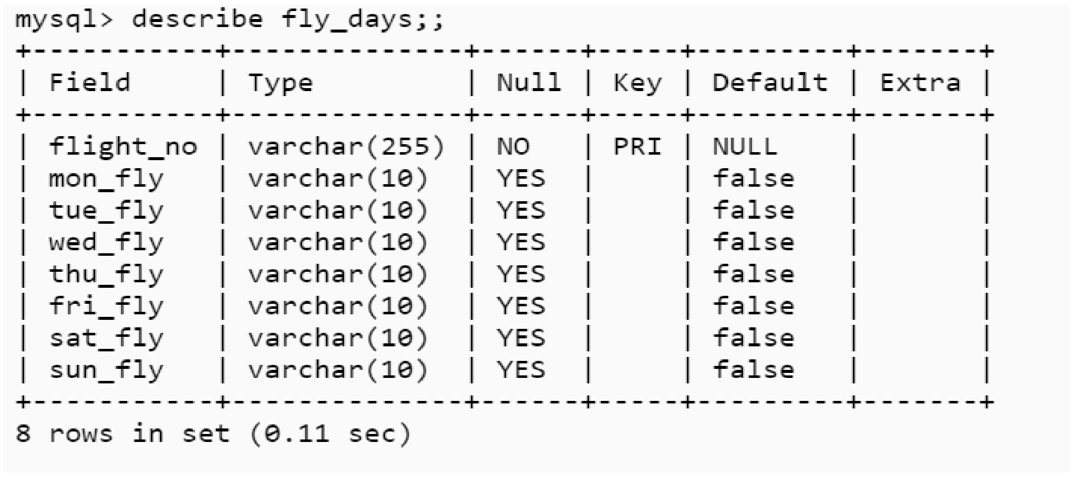
Figure 4.10: Airports Table Schema



```
mysql> describe statistics;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| flight_no    | varchar(255) | NO   | PRI | NULL    |       |
| counter      | varchar(255) | YES  |     | NULL    |       |
| cur_dur      | varchar(255) | YES  |     | NULL    |       |
| track_dur    | varchar(255) | YES  |     | NULL    |       |
| cur_stat     | varchar(255) | YES  |     | NULL    |       |
| track_stat   | varchar(255) | YES  |     | NULL    |       |
| counter_stat | varchar(255) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
7 rows in set (0.10 sec)
```

Figure 4.11: Statistics Table Schema

very important for my app as it contains the tracked on time percentages and tracked durations of the flights. In the statistics method, I connect to the database to check if there is any information for the flight already, and if there is I use it to update it with modified values. Otherwise I update the database with the first values.
The flight days method updates the fly days table to make sure every day the flight happens is set to true. So when someone using the app selects a date they want to fly, and it lands on a Sunday for example, they will only see the flights that historically have happened on a Sunday and will not see flights that don't happen on a Sunday.



```
mysql> describe fly_days;;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| flight_no | varchar(255) | NO   | PRI | NULL    |       |
| mon_fly   | varchar(10)  | YES  |     | false   |       |
| tue_fly   | varchar(10)  | YES  |     | false   |       |
| wed_fly   | varchar(10)  | YES  |     | false   |       |
| thu_fly   | varchar(10)  | YES  |     | false   |       |
| fri_fly   | varchar(10)  | YES  |     | false   |       |
| sat_fly   | varchar(10)  | YES  |     | false   |       |
| sun_fly   | varchar(10)  | YES  |     | false   |       |
+-----------+--------------+------+-----+---------+-------+
8 rows in set (0.11 sec)
```

Figure 4.12: Fly Day Table Schema

4.2 The Cloud

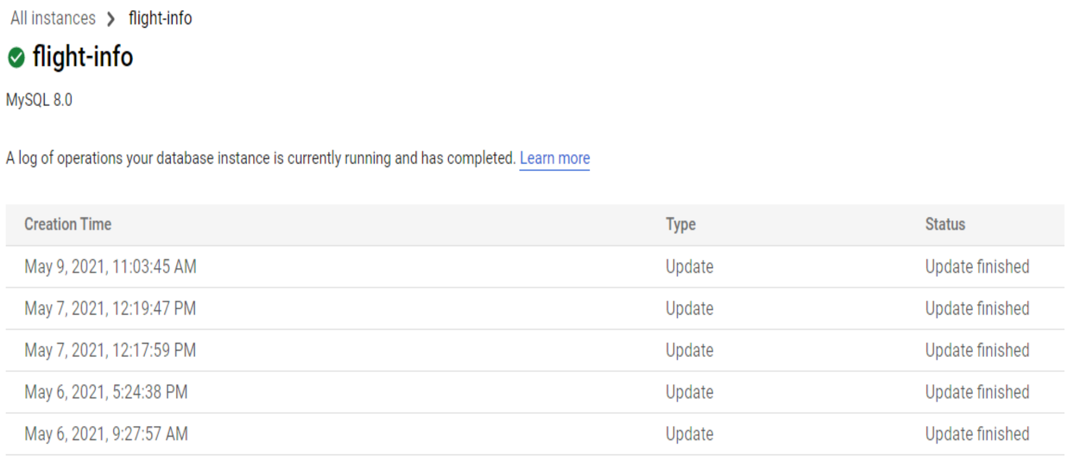
I used two different cloud platforms for my project, Google Cloud and Scrapy Cloud. I used separate cloud platforms because each one had a different purpose.

4.2.1 Google Cloud - CloudSQL

Google Cloud is where my database is stored, in a CloudSQL instance. My CloudSQL instance has a public IP address which allows any app to easily connect to it. Using the Google Cloud allows regular maintenance on my database, controlled by Google Cloud, which means I don't have to remember to do it myself and it doesn't inconvenience me as I can select times when the maintenance shouldn't occur. Google Cloud also performs daily backups of my database, so if anything were to go wrong and I needed restore a previous version of my database, I have that option. Using Google Cloud also allows me to see a log of what has been happening to my database since the last time I checked it, as shown in figure 4.13.

4.2.2 Scrapy Cloud

In Scrapy cloud is where my web scraping Python script actually runs. Scrapy Cloud is designed specifically for web scraping spiders to be hosted on, and hence it makes things easier than if I were to go about it in a roundabout way on a different cloud platform.


All instances › flight-info

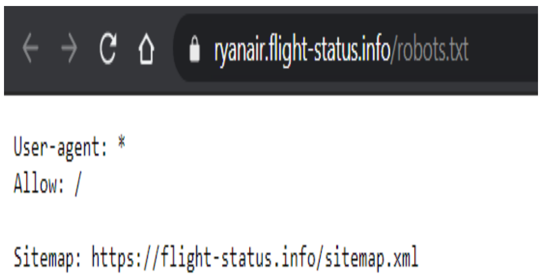
flight-info

MySQL 8.0

A log of operations your database instance is currently running and has completed. Learn more

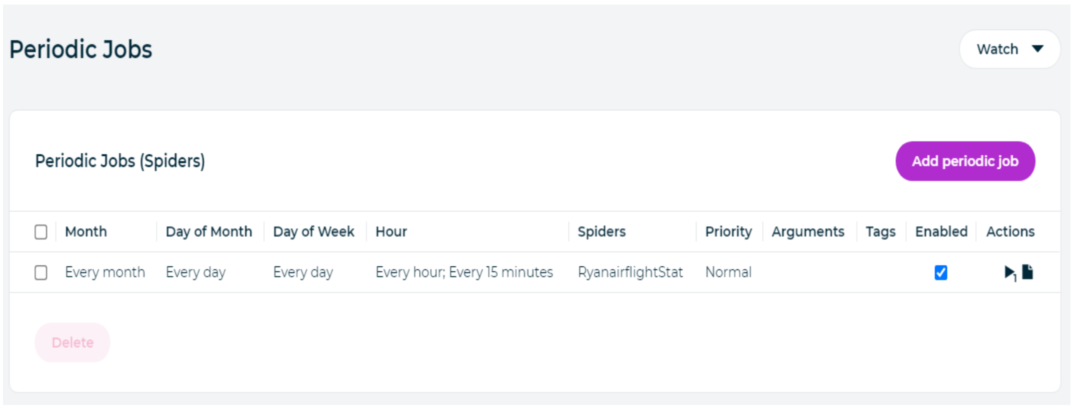
Creation Time	Type	Status
May 9, 2021, 11:03:45 AM	Update	Update finished
May 7, 2021, 12:19:47 PM	Update	Update finished
May 7, 2021, 12:17:59 PM	Update	Update finished
May 6, 2021, 5:24:38 PM	Update	Update finished
May 6, 2021, 9:27:57 AM	Update	Update finished

Figure 4.13: Google CloudSQL Instance Updates


ryanair.flight-status.info/robots.txt

```
User-agent: *
Allow: /

Sitemap: https://flight-status.info/sitemap.xml
```

Figure 4.14: Robots.txt for flight-status.info

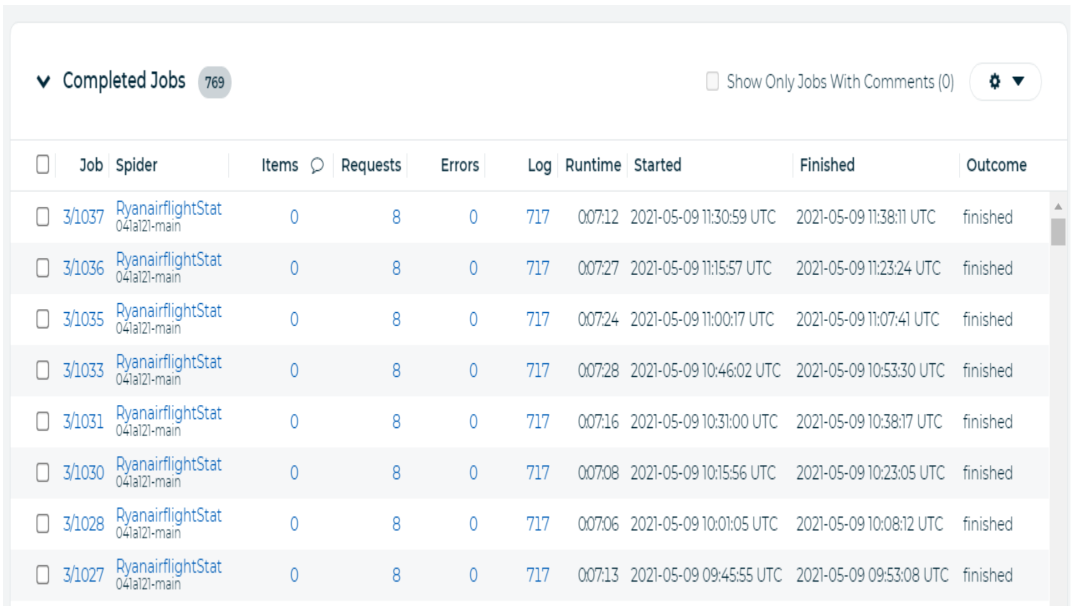


Periodic Jobs (Spiders)

	Month	Day of Month	Day of Week	Hour	Spiders	Priority	Arguments	Tags	Enabled	Actions
☐	Every month	Every day	Every day	Every hour; Every 15 minutes	RyanairflightStat	Normal			✓	

Figure 4.15: Scrapy Cloud Job

To use Scrapy Cloud, all that I needed to do was download their extension to my machine, and when I was ready to deploy my web scraper I just had to execute the ”shub deploy” command along with the project number on Scrapy Cloud. On Scrapy cloud I set up a job, as seen in figure 4.15, which specifies how often I want the scraper to run. For my project I set it to every fifteen minutes, but it could be more or less often, you can even set it to once a year if needed.
Every time my job executes and runs my scraper, the completed jobs list expands and gives me lots of information, as seen in figure 4.16. I can press on the numbers under 'Logs', 'Requests' and 'Errors' to view more information. In the logs section I would be able to see all of my print statements from the script, which is helpful in making sure everything looks as it should. The errors section usually stays empty, there's only ever anything in there if something isn't working correctly. If you go to the requests section, as seen in figure 4.17, there is lots of information about every request Scrapy makes, including the duration, the size of the response, the status and the URL for the website it is scraping.
In figure 4.17, Request 0 is actually starting the scrape on the robots.txt file for the website. As explained before, this file lists what is and isn't allowed to be scraped, and the one for this website can be seen in figure 4.14. There is a way to turn off compliance with the robots.txt file in Scrapy by setting ROBOTSTXT_OBEY in the settings file of the scraper to false, however I didn't want to go against any website's rules. Scrapy Cloud even allows changing your settings after the project is deployed on there, which I did with the download delay - I set it to 0.25 seconds instead of the one second specified in my script.

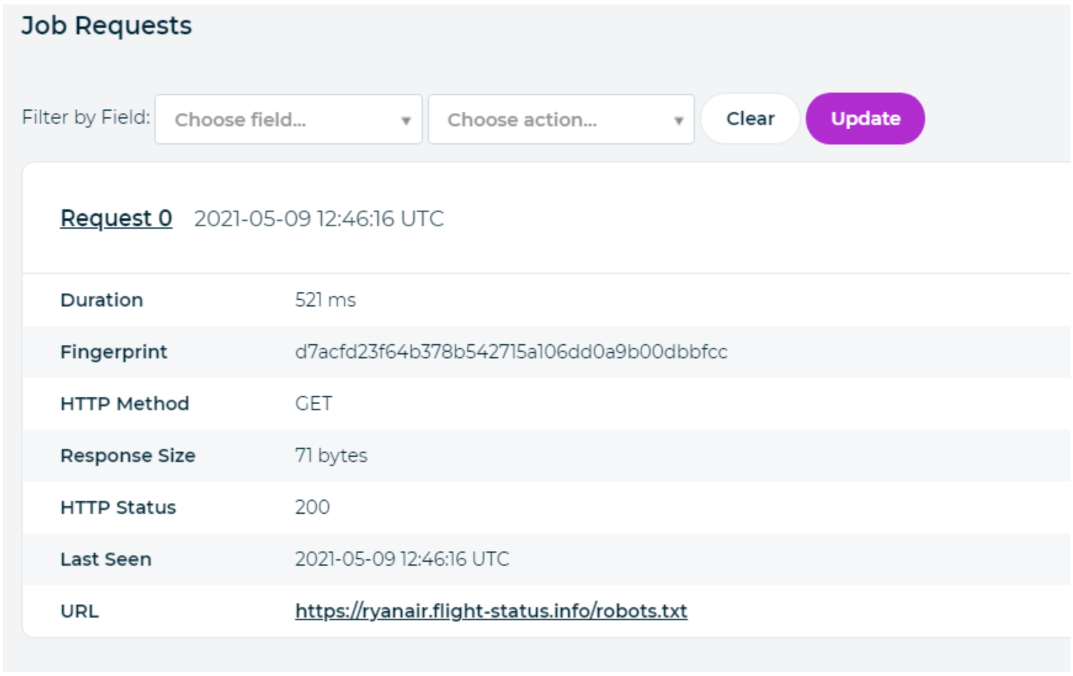


Completed Jobs 769

Show Only Jobs With Comments (0)

Job	Spider	Items	Requests	Errors	Log	Runtime	Started	Finished	Outcome
3/1037	RyanairflightStat 041a121-main	0	8	0	717	0:07:12	2021-05-09 11:30:59 UTC	2021-05-09 11:38:11 UTC	finished
3/1036	RyanairflightStat 041a121-main	0	8	0	717	0:07:27	2021-05-09 11:15:57 UTC	2021-05-09 11:23:24 UTC	finished
3/1035	RyanairflightStat 041a121-main	0	8	0	717	0:07:24	2021-05-09 11:00:17 UTC	2021-05-09 11:07:41 UTC	finished
3/1033	RyanairflightStat 041a121-main	0	8	0	717	0:07:28	2021-05-09 10:46:02 UTC	2021-05-09 10:53:30 UTC	finished
3/1031	RyanairflightStat 041a121-main	0	8	0	717	0:07:16	2021-05-09 10:31:00 UTC	2021-05-09 10:38:17 UTC	finished
3/1030	RyanairflightStat 041a121-main	0	8	0	717	0:07:08	2021-05-09 10:15:56 UTC	2021-05-09 10:23:05 UTC	finished
3/1028	RyanairflightStat 041a121-main	0	8	0	717	0:07:06	2021-05-09 10:01:05 UTC	2021-05-09 10:08:12 UTC	finished
3/1027	RyanairflightStat 041a121-main	0	8	0	717	0:07:13	2021-05-09 09:45:55 UTC	2021-05-09 09:53:08 UTC	finished

Figure 4.16: Completed Scrapy Cloud Jobs



Job Requests

Request 0 2021-05-09 12:46:16 UTC

Duration	521 ms
Fingerprint	d7acfd23f64b378b542715a106dd0a9b00dbbfcc
HTTP Method	GET
Response Size	71 bytes
HTTP Status	200
Last Seen	2021-05-09 12:46:16 UTC
URL	https://ryanair.flight-status.info/robots.txt

Figure 4.17: Scrapy Cloud Request

4.3 The App

The app itself, or the user interface, was made using Flutter along with multiple Dart files.

4.3.1 Database connection using the Future Class

For this app I imported a package called 'mysql1' to be able to connect to the database. In order to display results from my database, I had to use something called a Future class which allows delayed computation, as retrieving information from the cloud is not immediate. I also had to use this Future class for the date selection. The Future class allows the app to reload when something it was waiting for finishes buffering. For information from the database I have a buffering circle being displayed to let the user know something is loading, and then when the information is ready the user can see it.
The way I did this for every time I needed to query the database was in three steps.

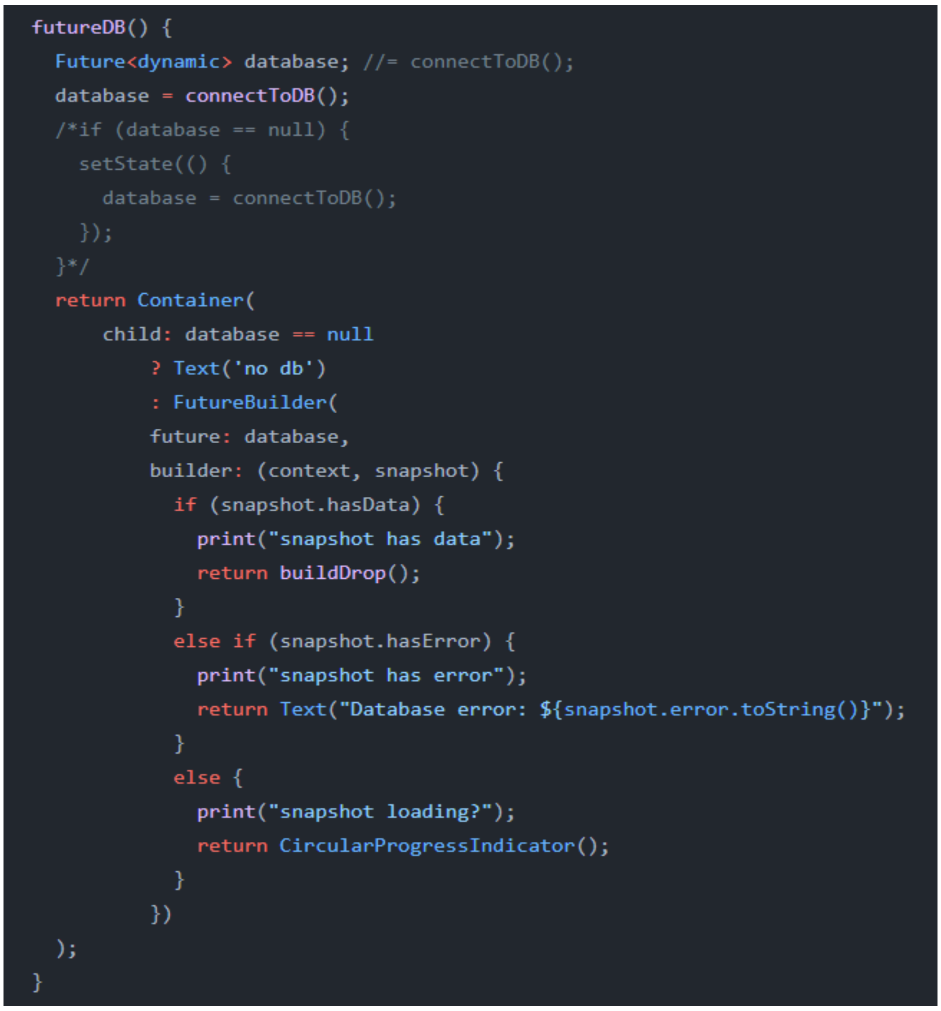
1. I call the future method in my widget.
2. The future method calls the method that connects to the database as shown in figure 4.18 and until this method returns, it shows a buffering icon, as shown in figure 4.21.
3. The connect to database method starts up as shown in 4.19, it connects to the database, retrieves the necessary information and does whatever processing it needs to, and then, when it is finished, it returns a Future object, 4.20, which tells the app the information has loaded and it refreshes.

4.3.2 Home Page

This is the page that the app opens up to at the beginning and is incredibly simple. There is only some text saying "Get Started" and a button in the shape of an airplane, this is shown in figure 4.22. When the user presses on the airplane it brings them to the next page.

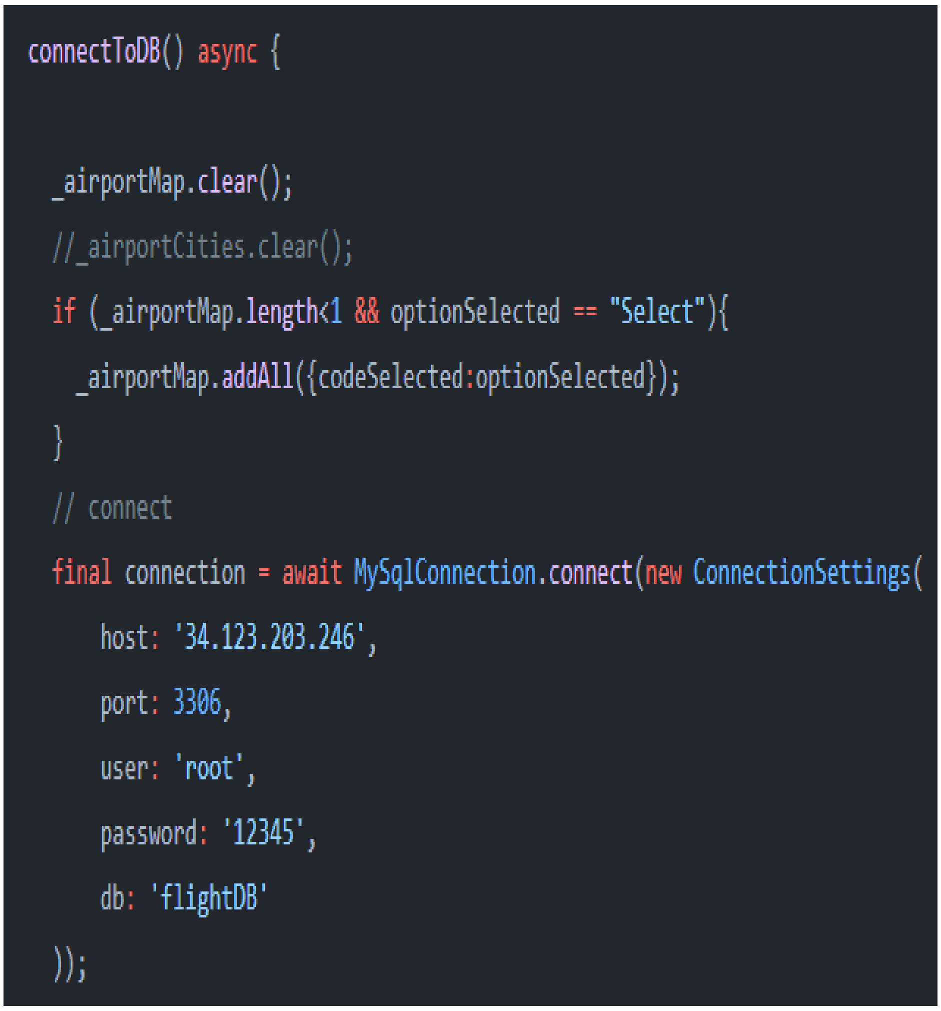
4.3.3 Origin and Date Selection Page

This page asks the user to make three choices - where they are starting the journey, whether they would like to choose the destination by climate or by



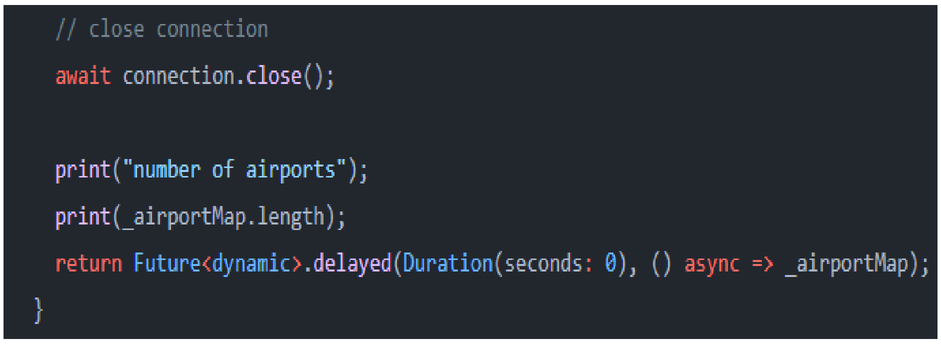
```
futureDB() {
  Future<dynamic> database; //= connectToDB();
  database = connectToDB();
  /*if (database == null) {
    setState(() {
      database = connectToDB();
    });
  }*/
  return Container(
      child: database == null
          ? Text('no db')
          : FutureBuilder(
          future: database,
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              print("snapshot has data");
              return buildDrop();
            }
            else if (snapshot.hasError) {
              print("snapshot has error");
              return Text("Database error: ${snapshot.error.toString()}");
            }
            else {
              print("snapshot loading?");
              return CircularProgressIndicator();
            }
          })
  );
}
```

Figure 4.18: Future Method



```
connectToDB() async {


  _airportMap.clear();
  //_airportCities.clear();
  if (_airportMap.length<1 && optionSelected == "Select"){
    _airportMap.addAll({codeSelected:optionSelected});
  }
  // connect
  final connection = await MySqlConnection.connect(new ConnectionSettings(
      host: '34.123.203.246',
      port: 3306,
      user: 'root',
      password: '12345',
      db: 'flightDB'
  ));
```

Figure 4.19: Connect to Database Method



```
  // close connection
  await connection.close();

  print("number of airports");
  print(_airportMap.length);
  return Future<dynamic>.delayed(Duration(seconds: 0), () async => _airportMap);
}
```

Figure 4.20: Returning Future from Connect to Database Method

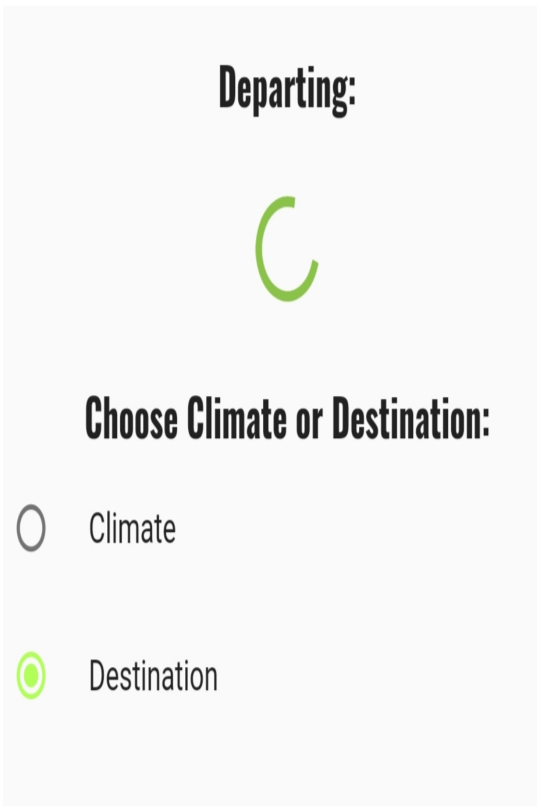


Figure 4.21: Buffering symbol

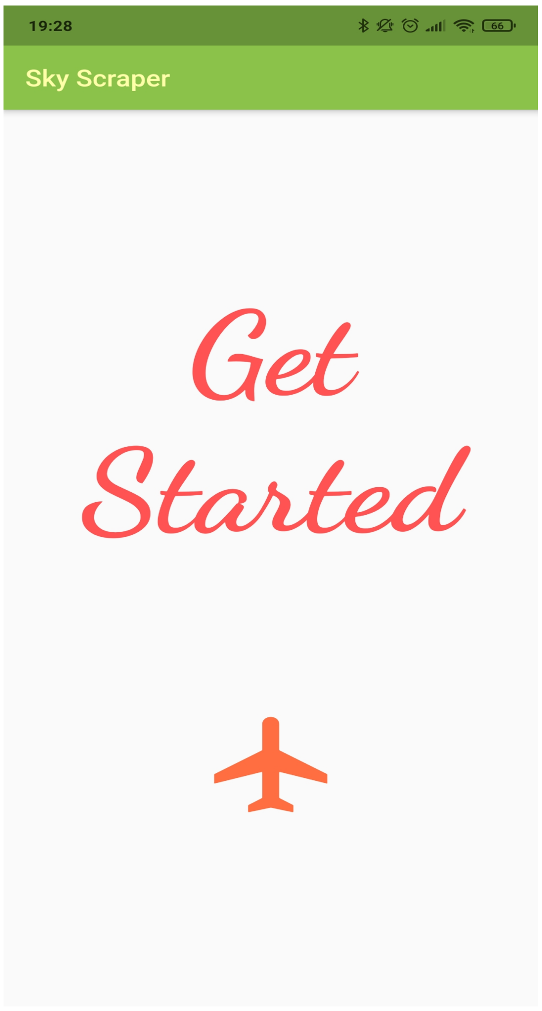


Figure 4.22: Home page

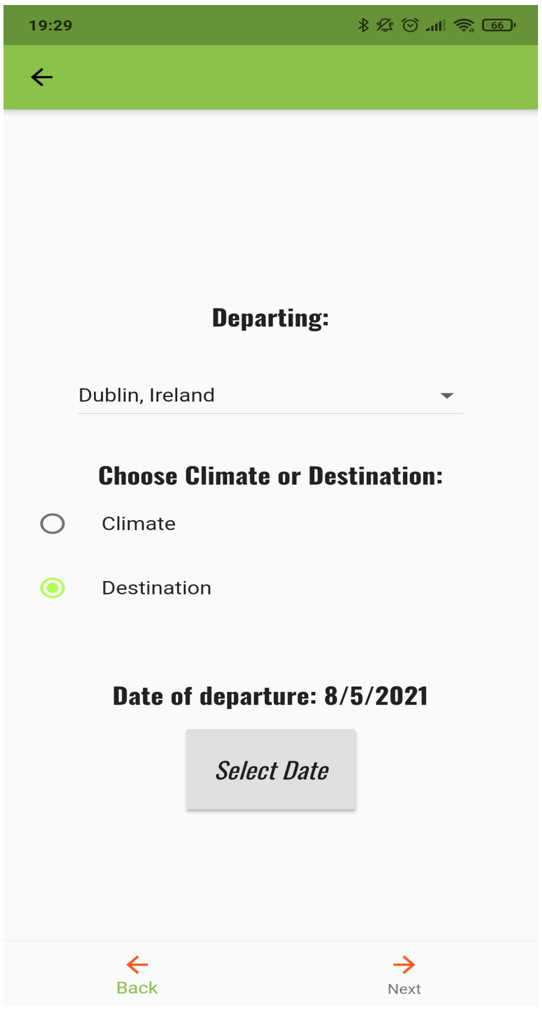


Figure 4.23: Origin selection page

destination, and the date of the departure. All of this can be seen in figure 4.23 The list of airports available to depart from is generated by selecting all of the airports from the airports table in the cloud database. The selection between climate and destination affects the next page where there is a list of destinations to pick from. The user needs to select what date they would like to view flights for (figure 4.24), as different flights fly on different days, although some fly everyday. When the user is ready, they press the arrow pointing right to move onto the next page.

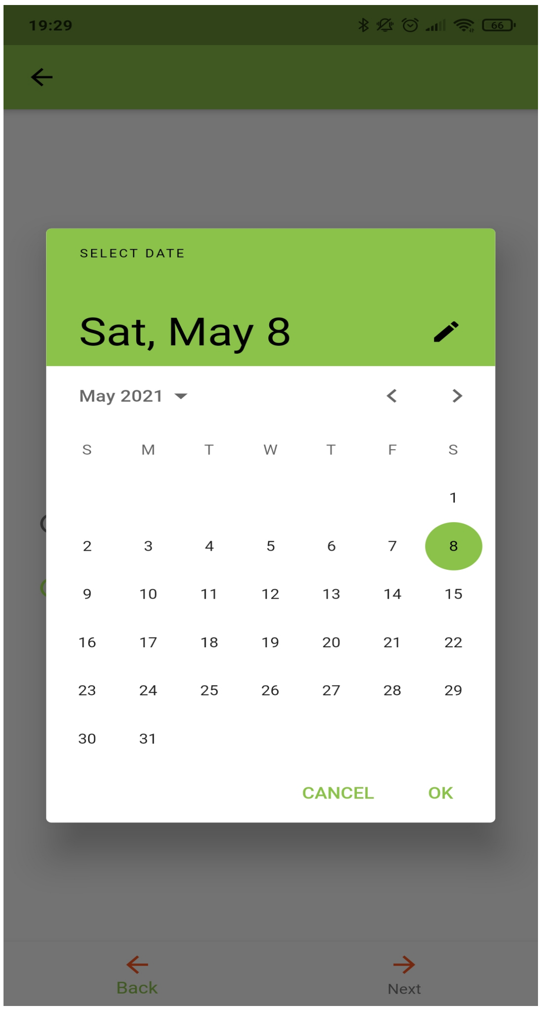


Figure 4.24: Date Selection

4.3.4 Destination Selection Page

Here the user gets to select multiple potential destinations they are interested in, as seen in figure 4.25. In this section, depending on whether the user selected the Climate or the Destination radio button, they will be able to choose a climate, like shown in figure 4.26. The climate options are generated by selecting each unique climate in the Ryanair Flights Table. Once a climate is selected, or if the Destinations button was chosen in the first place, the user is able to see a list of destinations available that fit the criteria, as seen in figure 4.27. The destinations are generated by selecting all the flights from the Ryanair table that have the correct origin and fly on the selected date (by joining this table with the fly day table), as well as having the correct climate if specified. When the user taps on the destinations they are interested in, they turn green and the checkbox on the row gets a checkmark, as per figure 4.28. At the top, the user is reminded of the origin, and at the bottom they are reminded of the date. Once the user is happy with their selections, they press the right arrow again and the flight selection page loads up.

4.3.5 Flight Selection page

On this page, the user is reminded of which destinations they have chosen, and they are asked to select up to two flights to compare, like in figure 4.29. The flights are generated by selecting them from the Ryanair flights table where the origin and destination match the specified choices, joined with the fly day table to make sure only the valid options are displayed. Once the user taps on a flight, the row goes green and the checkbox has a green checkmark. The text below also gets updated with the selected flights, so there is no ambiguity, shown in figure 4.30. If the user taps on a third flight, without deselecting one of the already chosen flights, nothing happens and the third flight is not added, as the user is being asked to limit their options to two. When the user is happy with the choices, again, they press the right arrow and they are redirected to the results page.

4.3.6 Results page

Here, the two chosen flights are compared side by side, as shown in figure 4.31. The information displayed is retrieved from the Ryanair flights table as well as the statistics table. This is really the end point of the app but the user is welcome to start again by pressing the home button and being redirected to the home page, or if they would like to compare different flights based on the same criteria they can press the left arrow to go back.

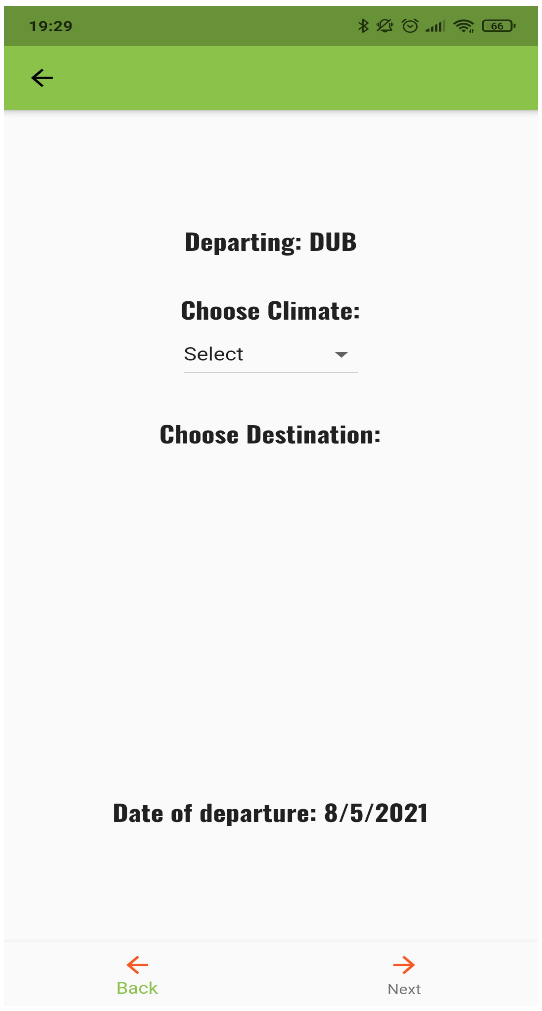


Figure 4.25: Destination Selection page

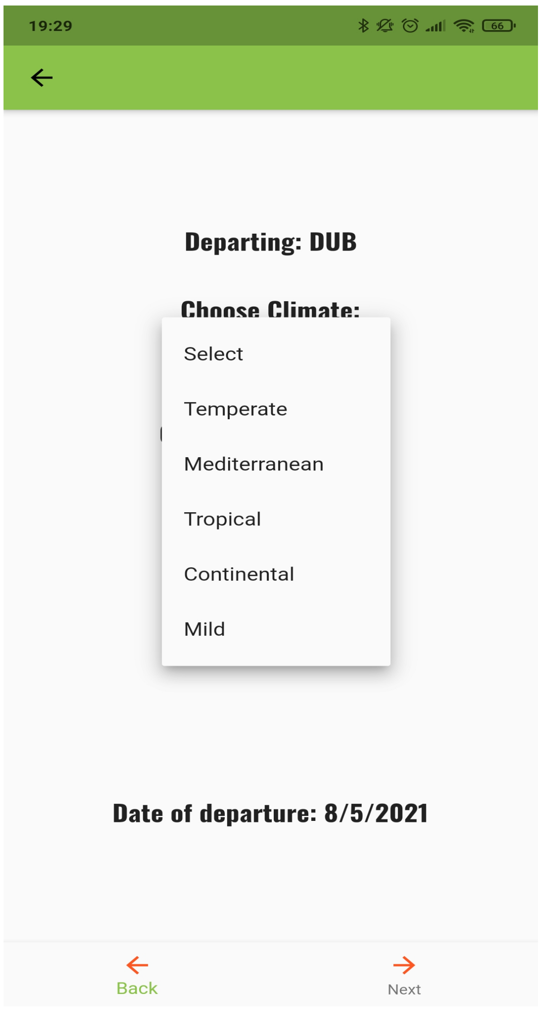


Figure 4.26: Climate options

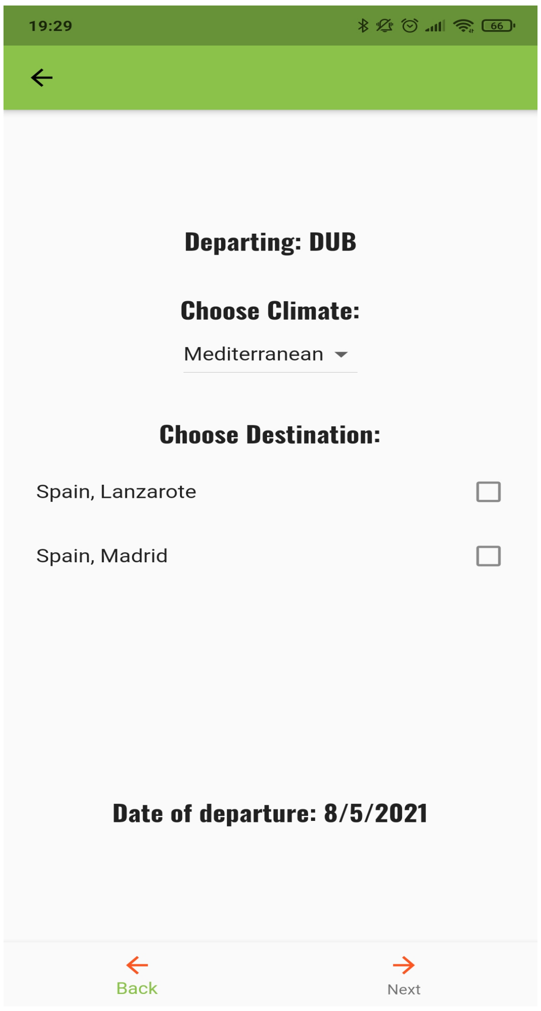
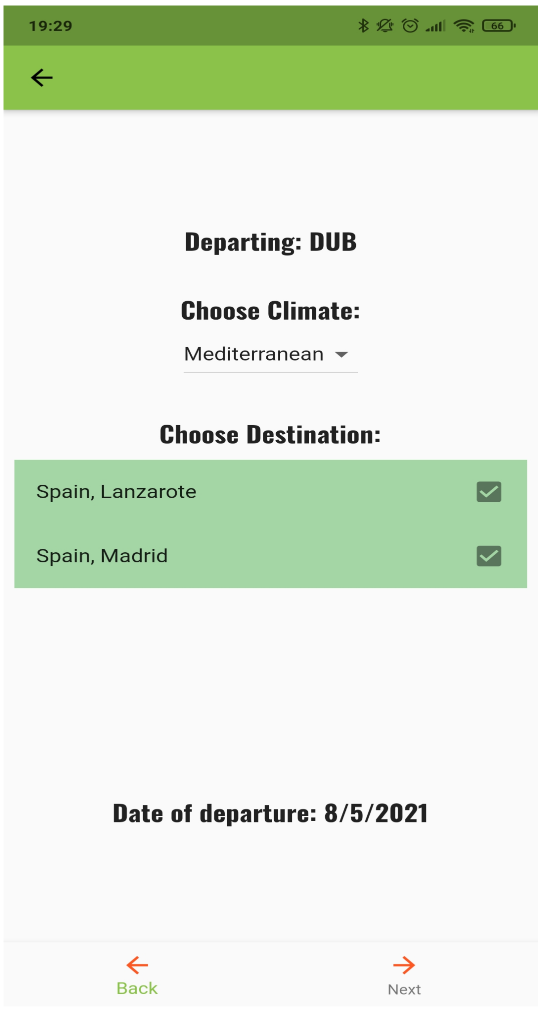


Figure 4.27: Destinations Available by Climate



Departing: DUB

Choose Climate:

Mediterranean

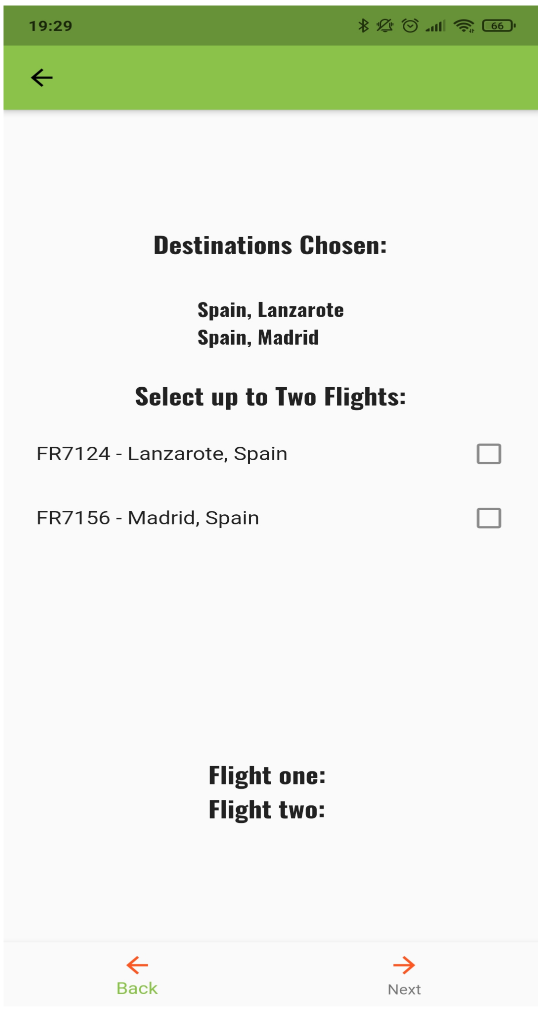
Choose Destination:

Spain, Lanzarote ✓

Spain, Madrid ✓

Date of departure: 8/5/2021

Figure 4.28: Destinations Chosen



Destinations Chosen:

Spain, Lanzarote
Spain, Madrid

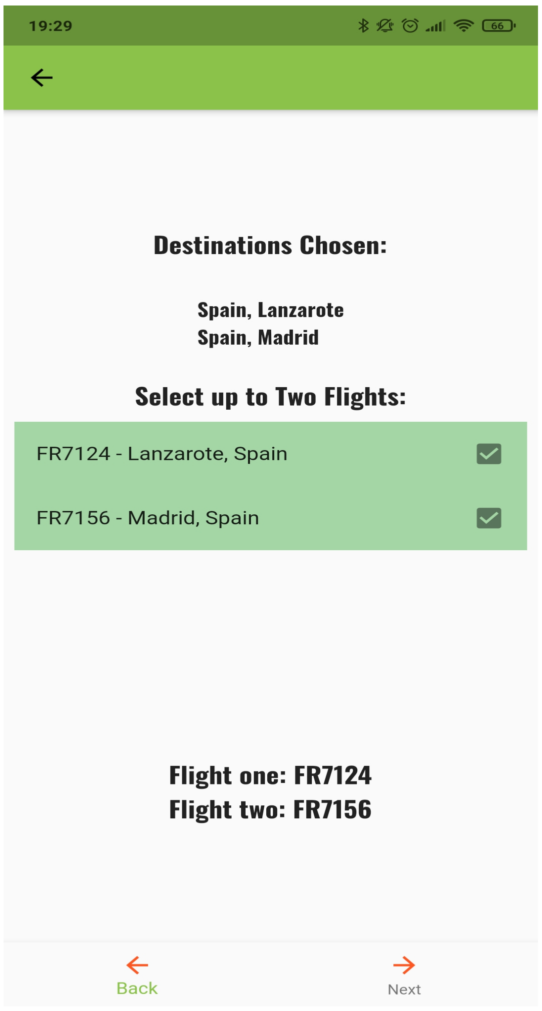
Select up to Two Flights:

FR7124 - Lanzarote, Spain ☐

FR7156 - Madrid, Spain ☐

Flight one:
Flight two:

Figure 4.29: Flight Selection page



Destinations Chosen:

Spain, Lanzarote
Spain, Madrid

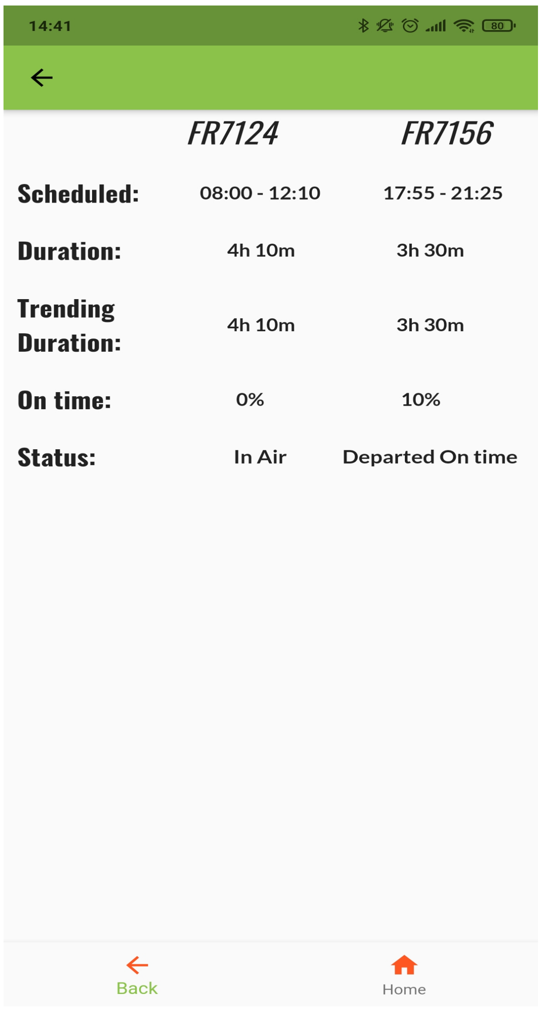
Select up to Two Flights:

FR7124 - Lanzarote, Spain ✓

FR7156 - Madrid, Spain ✓

Flight one: FR7124
Flight two: FR7156

Figure 4.30: Flights chosen on Flight Selection page



	FR7124	FR7156
Scheduled:	08:00 - 12:10	17:55 - 21:25
Duration:	4h 10m	3h 30m
Trending Duration:	4h 10m	3h 30m
On time:	0%	10%
Status:	In Air	Departed On time

Figure 4.31: Results page

Chapter 5

System Evaluation

In this chapter I will be evaluating the system I have designed based on a number of factors, including:

- Meeting the Objectives
- Scalability
- Performance against Testing
- Robustness of the Design
- Limitations

5.1 Did the results meet the objectives from the outset?

I set out to accomplish three things before starting the project. The first was to provide someone planning a journey with a convenient and easy way to compare flights so they could make an informed decision. I have achieved this goal, although in a small scale, as currently it is only showing Ryanair flights. I have created an easy to navigate app, that does allow the user to select their criteria and find flights based on the criteria, and allows them to compare flights too.
The second goal I wanted to achieve was to create a fully functioning system, from the back-end to the user interface, using multiple technologies and languages along the way. I have accomplished this, I have managed to connect my Python script to my database, and my Flutter code to the database, and in doing this I have created a fully functioning and automated system.

The final objective I set out for myself was to challenge myself and do something bigger than I had ever tried before, while learning new languages and technologies. This goal was also achieved, this is definitely the biggest and most challenging project I have done to date, and I have learned a tremendous amount from doing it.
I accomplished all three of the goals that I created for myself in doing this project.

5.2 Is this a scalable project?

Sky Scraper is an app designed to scrape a website with over five hundred airlines worth of information, and currently it is only scraping for one airline - Ryanair. It is definitely possible, and indeed quite easy to allow more airlines to be scraped. To do this I would create another Scrapy spider for each airline I wanted to include and deploy them all onto Scrapy cloud. I would also need more tables for the new airlines in my database, but if I were to do this I would possibly explore another kind of database, perhaps MongoDB.
I could also scale this app to be available to more than just Android devices, I would create an iOS equivalent for this app. I could release the app onto the Google Play store as well as Apple's App store to reach more users as well.
The project could be scaled to a point where it would no longer be just a flight comparison app, it could allow flight purchasing directly from the app if I were to make my scraper ignore the robots.txt files from airlines. To take it even further, it could be scaled to the point of allowing the user to compare different accommodations from different websites. This could be done by taking reviews and turning them into a points system, similarly to how I turned verbal statuses from the flight-status.info website into percentages for my app.
There are many things in this app that could be scaled and I ensured this while designing my app through naming conventions and how I designed my database tables.

5.3 How did Sky Scraper perform when tested?

I executed three types of testing, further outlined in my Methodologies chapter. When it came to the unit testing, each feature was tested and built upon until it passed the test. In the end all features of the project, front and back

end, were passing unit tests.
Regression testing was used once a new feature was passing its unit tests to make sure everything worked together. For the Python script, all of the regression tests were passing, however with Flutter this was not completely the case. During regression testing with Flutter I actually found that, while all features were working, some unexpected things happened as well. For example, when going through the different selections in the app, if I chose certain destinations, then chose two flights on the next page, and then went back to change my destination options, the two flights that were selected previously stayed selected and wouldn't allow the user to select more or remove those flights. This is definitely not ideal and needs to be resolved, however I didn't have enough time to do this.
The last type of testing I did was the acceptance testing. For the Python script, the acceptance tests included testing if I was updating my database with valid information from the website I was scraping, testing that the values in the database made sense, and testing to make sure the tables in the database were correctly related and made sense. After verifying that all of the above was true and editting my script when it wasn't until it passed, I moved on to acceptance testing the Flutter aspects.
The Flutter acceptance testing was more like the acceptance testing for the whole project, as this is where I used my project proposal to create my tests. I verified that the app behaved as it should when it was used as designed for the acceptance testing. This included making sure that when I selected the origin, date and destination I was getting correct flight information (which was verified using the actual website I was scraping, my own database and Ryanair's flight information). This also entailed checking that when two flights were selected, and I went on the results page, the statistics in my results page made sense.
This final part actually didn't completely pass the acceptance testing. At first glance it isn't noticeable, however the durations aren't correct due to different countries having different timezones, and the associated difficulties in implementing that. If I had some more time, I could have actually fixed that, but there isn't a readily available package to import that works exactly as I need it to, so it would have taken more time than I had left.
The other issue on the results page is the percentage of the flights being on time. Again, on first glance it could be correct, but in my opinion it seems quite low. However, I am not a mathematician so this was really my downfall in creating the statistics. Maybe with some more time I could have made it more accurate, but I just didn't have that extra time.

5.4 Is Sky Scraper a robust system?

Sky Scraper is robust in some ways, but it isn't in others. In my Python scripts I have multiple exception handlers that work to make sure that all errors I tested for would be handled, including certain airports not existing in the database and if request responses had extra symbols I wasn't expecting to get back. With such measures in place, the Python script can be said to be quite robust.
However the Flutter part is not as robust as I would like it to be. I have noticed certain issues throughout testing my application that aren't being handled in a satisfactory way, such as if there is no internet connection or if the user continues on to the next page without selecting anything. In these cases I get ugly errors, which I am aware of due to my extensive testing, however because of time constraints I actually didn't have leftover time to resolve such issues and make the user interface of my project more robust.

5.5 What limitations were encountered throughout development?

The biggest limitation throughout the project was really the time that was available. If I had more time I would have been able to ensure the user interface was more robust, and I could have handled the errors in a better way. With more time, I would have been able to use timezones when calculating the durations of flights which would have improved the accuracy of the results. More time would have also allowed me to make the calculation for the on-time percentage more accurate, as right now the on-time percentage gets to zero unreasonably quickly and it doesn't climb back up again.
The technology I was using really wasn't limiting at all for what I set out to design. However, if I now decided that I wanted to make this app for Apple devices as well it would be quite difficult for me to test due to lack of my own Apple devices as well as Apple emulators on the Android Studio. Of course, it's completely reasonable for the Android Studio to only have Android emulators. Since Flutter does allow cross platform development, and Android Studio and IntelliJ are the only options for development using Flutter, you might expect one of them to have an Apple emulator, but neither do. If I were to now try and make this app compatible with iOS I would need to purchase an iPhone or a Mac for testing purposes, and those costs are a deterrent for me.

Chapter 6

Conclusion

Creating Sky Scraper has been an interesting task, with ups and downs along the way. When I first started the project I worried I wouldn't be able to complete it in the time I had, or that I wasn't a good enough programmer to wrap my head around certain concepts. Being able to check all the features off of my to-do list has made me very proud of all I have accomplished through developing Sky Scraper. I can honestly say creating this entire system, that works so beautifully together, has been the biggest accomplishment of my time in GMIT.
I managed to accomplish all of the objectives I set out for myself at the beginning, I found out that I really can learn new languages and technologies by myself, and I created a great app that truly allows the user to compare flights. I believe this app could become the next big travel app with some further adjustments and additions.
I learned so much more than I set out to when I first chose this project. I learned how to work on my own initiative and manage my time better. My organisational and planning skills were improved massively, as well as my problem solving skills. My confidence as a developer has grown as well; it's easy to lose confidence when you get stuck on a problem as a programmer, but through perseverance I managed to get past all of the hurdles in my way.

6.1 Future Improvements and Developments

Although I am proud of my application so far, I'll be the first to admit it isn't perfect and there is definitely room for improvement.

- The first thing I would like to improve upon is the error handling in the user interface of the app. As it stands, it doesn't handle errors whatsoever and that needs to be resolved.

- The next thing I would do is transfer over my database to MongoDB, or look into some other kind of noSQL database, so I could add more airlines and make my app faster.
- Once I had moved over to a non-relational database, I would create more scrapers for different airlines that could be included in the comparison. The app isn't very useful when all of the flights being compared are for a single airline and a there are only a small number of them.
- When the Android app is perfected, I would like to get it onto the Play Store to actually allow people to find it and download it. What is the point of creating an app if no one gets to use it?
- Next, I would hopefully get my hands on an iOS device, so I could develop and test the Apple equivalent, and deploy that on the App Store as well.
- I would like to allow users to plan their entire journeys through my app, which would mean allowing accommodation comparison, bus and train journey comparison, booking reservations through the app, and possibly more.

In summary, I have achieved the goals I set out to achieve with this project, but that doesn't mean it is perfect, or that it's even finished. There are so many opportunities for growth with this app, the sky really is the limit.

Chapter 7

Appendix

7.1 GitHub

GitHub link: github.com/KlaudijaMedeksaite/FYP_SkyScraper

There are three main folders at my GitHub repository:

- Python Scripts
- Sky Scraper
- APK

There is also a requirements file, a read me file, a screen-cast, as well as this PDF.
Inside the Python scripts folder is the scraper that is hosted on Zyte's Scrapy Cloud. This can be found in 'Python Scripts/ Python Scripts/ tutorial/ ZYTE/ tutorial/ spiders/' and is called ryanair_flights.py. This is the most important file in the Python Scripts folder, but the folder also contains some Scrapy and Zyte generated files, as well as other Python scripts I used to create and configure the database on Google Cloud.
In the sky_scraper folder are all the files built by the Android studio. The dart source files containing the code are stored in the lib folder of sky_scraper.
The APK folder only has one file, the release APK file. If this is opened on an Android smartphone the user will download the app onto their device and be able to use it as long as there is an internet connection.
The requirements file specifies the necessary installs for the python scripts.
The Read Me file contains an overview of what this application does and how to run it if someone was to download it.

The screen cast shows an example of the app being used to give a better idea of what it does.

Bibliography

[1] Wikipedia, “Web scraping.”

[2] S. O’Dea, “Smartphone users worldwide 2016-2023,” 2021.

[3] R. Mitchell, *Web scraping with Python: Collecting more data from the modern web.* ” O’Reilly Media, Inc.”, 2018.

[4] G. Documentations, “Introduction to robots.txt.”

[5] P. Institute, “About python.”

[6] S. Cass, “The top programming languages: Our latest rankings put python on top-again-[careers],” *IEEE Spectrum*, vol. 57, no. 8, pp. 22–22, 2020.

[7] J. Wang and Y. Guo, “Scrapy-based crawling and user-behavior characteristics analysis on taobao,” in *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 44–52, IEEE, 2012.

[8] A. Fettig and G. Lefkowitz, *Twisted network programming essentials.* ” O’Reilly Media, Inc.”, 2005.

[9] P. Christensson, “Dbms definition,” 2006.

[10] MongoDB, “How to scale mongodb.”

[11] T. Alsop, “Share of households with home computers connected to the internet in ireland from 2011 to 2019, by household composition,” 2020.

[12] G. Stats, “Mobile and tablet internet usage exceeds desktop for first time worldwide,” 2016.

[13] S. O’Dea, “Share of households with home computers connected to the internet in ireland from 2011 to 2019, by household composition,” 2021.

[14] T. Warren, “Bill gates now uses an android phone,” 2017.

[15] Gartner, “Flutter reviews.”

[16] A. Begehr, “Top three reasons for choosing flutter over react native in 2020,” 2020.

[17] Altexsoft, “The good and the bad of flutter app development,” 2021.

[18] Scrapy, “Scrapy settings.”