City, University of London

BSc Computer Science

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

No proprietary arrangements, interests, outside help or other promises.

Consultant: [REDACTED]

Word count: 12,447

TABLE OF CONTENTS

Table of Contents **2**
Abstract **6**
1 Introduction **8**
1.1 Context 8
1.2 Problem to be solved 8
1.3 Project objectives 9
1.3.1 Main objective 9
1.3.2 Sub objectives 9
1.3.3 Project beneficiaries 10
1.4 Work performed 10
1.5 Scope limitations 11
2 Output summary **12**
3 Literature review **14**
3.1 Introduction 14
3.2 Similar applications and tools 14
3.2.1 Clear Sky Chart 14
3.2.2 AstroPlanner 14
3.3 Data sources 15
3.3.1 VizieR 15
3.3.2 Light pollution data 16
3.3.3 MapBox API 16
3.3.4 OpenWeatherMap API 16
3.4 Development 17
3.4.1 IDE 17
3.4.2 Deployment 17
3.4.3 Database 17
3.5 Conclusion 17
4 Methods **18**
4.1 Analysis 18
4.1.1 Planning 18
4.1.2 Requirements gathering 18
4.2 Design 20
4.3 Implementation 22
4.3.1 Architecture and deployment 22
4.3.2 Development environment 22
4.3.3 Authentication 23
4.3.4 APIs 23
4.3.5 Email template reuse 23
4.3.6 Object-relational mapper (O/RM) 24
4.3.7 Design patterns 24
4.3.8 Version control 26
4.3.9 Astronomical catalogues 26
4.3.10 Computational astronomy 26
4.4 Evaluation 27
4.4.1 Functional testing 27
4.4.2 Non-functional testing 27
5 Results **28**
5.1 Chapter introduction 28
5.2 Analysis 28

5.2.1 Planning ... 28
5.2.2 Requirements gathering ... 29
5.3 Design ... 34
5.3.1 Class Diagram ... 34
5.3.2 Entity relationship diagram ... 38
5.3.3 Deployment diagram ... 39
5.3.4 Logo ... 42
5.4 Implementation ... 43
5.4.1 Architecture and deployment ... 43
5.4.2 Development environment ... 51
5.4.3 APIs ... 53
5.4.4 Email template reuse ... 58
5.4.5 O/RM ... 61
5.4.6 Design patterns ... 64
5.4.7 Light pollution data ... 66
5.4.8 Rating weather conditions ... 70
5.4.9 Version control ... 71
5.4.10 Astronomical catalogues ... 73
5.4.11 Computational astronomy ... 78
5.5 Evaluation ... 83
5.5.1 Functional testing ... 83
5.5.2 Non-functional testing ... 84
6 Conclusions and discussion ... 85
6.1 Primary objective ... 85
6.1.1 Primary outcome ... 85
6.2 Sub objectives ... 86
6.2.1 Sub objective #1 ... 86
6.2.2 Sub objective #2 ... 86
6.2.3 Sub objective #3 ... 86
6.2.4 Sub objective #4 ... 86
6.2.5 Sub objective #5 ... 86
6.2.6 Sub objective #6 ... 87
6.2.7 Sub objective #7 ... 87
6.3 Discussion ... 88
6.4 Personal conclusions ... 89
6.4.1 Potential changes ... 89
7 Glossary of terms ... 90
8 Reference list ... 91
9 Appendices ... 96
9.1 Appendix A: Project Definition Document (PDD) ... 97
9.1.1 Proposal ... 98
9.1.2 Project objectives ... 99
9.1.3 Project beneficiaries ... 100
9.1.4 Work plan ... 100
9.1.5 Project risks ... 102
9.1.6 Ethics checklist ... 103
9.1.7 Glossary of terms ... 112
9.2 Appendix B: Reuse Summary ... 113
9.2.1 Libraries ... 113
9.2.2 Source listing ... 115
9.2.3 Other tools and reuse ... 118

9.3 Appendix C: AstroCue Participant Information Sheet 119
9.4 Appendix D: Requirements....120
9.4.1 Questionnaire 121
9.4.2 Requirements prioritisation....128
9.4.3 Volere requirements specifications....129
9.4.4 Use case diagram....136
9.4.5 Use case specifications....137
9.5 Appendix E: Design 147
9.5.1 Important preamble 148
9.5.2 Class diagram 149
9.5.3 Entity relationship diagram 150
9.5.4 Deployment diagram 151
9.5.5 AstroCue logo and font....152
9.6 Appendix F: Deployment and domain/DNS....154
9.6.1 Introduction....155
9.6.2 Deploying the backend to Azure 156
9.6.3 Frontend deployment to Vercel....174
9.6.4 Domain and DNS 176
9.7 Appendix G: EF Core database migration script....182
9.8 Appendix H: Introduction to celestial coordinate systems 183
9.8.1 Celestial coordinate systems 184
9.8.2 Equatorial coordinate system....184
9.8.3 Horizontal coordinate system....187
9.9 Appendix I: Testing 188
9.9.1 Introduction....189
9.9.2 Testing results 190
9.9.3 Use case testing....191
9.9.4 Unit testing....198
9.9.5 Maintainability testing....202
9.10 Appendix J: Software build guide 203
9.10.1 Introduction....204
9.10.2 Important note....204
9.10.3 Prerequisites 204
9.10.4 Steps....205
9.11 Appendix K: Code snippets 219
9.11.1 Introduction....220
9.11.2 Full OpenWeatherMap request with JSON parsing....221
9.11.3 Full MailGun API request using templates 222
9.11.4 LightPollutionService constructor 223
9.11.5 LightPollutionService retrieving light pollution levels for coordinates....224
9.11.6 Bortle scale methods 225
9.11.7 Computational astronomy calculations 226
9.11.8 Weather index sorting 229
9.12 Appendix L: Full codebase 230
9.13 Appendix M: User interface / all web pages....231
9.13.1 Introduction....232
9.13.2 Authentication 233
9.13.3 Observation locations 234
9.13.4 Observations....236
9.13.5 Reports....237
9.13.6 Observation logs....238

ABSTRACT

For millennia, humanity and its ancestors have looked to the sky in wonder. *What is out there? What is our purpose? Are the stars there for us, or are they just there?*

With modern computing, there are unprecedented opportunities to bring these wonders of the universe, and the questions that arise from them, into the hands of anyone.

AstroCue had the goal to help inspire more people about the universe and the incredible story it can tell by making it easier to plan, execute, and log the results of astronomical observations. It aimed to use localised weather and light pollution data as well as computational astronomy to provide predicted optimal viewing times for chosen stars, galaxies, and other objects.

The following document covers the research, planning, implementation, and results of this endeavour.

***Important note**: all references to section numbers in this PDF are clickable for easy navigation. The document has also been exported with navigation data, so you can view a navigable copy of the contents table in most PDF viewer software or web browsers to aid in quickly jumping between various sections.*

I would like to thank my consultant, [REDACTED], for his advice and help throughout this project.

I would also like to thank the many space agencies, researchers, authors, and engineers whose work made this possible.

1 INTRODUCTION

1.1 CONTEXT

There are many pieces of software that allow users to locate astronomical objects in the night sky. Many of these are in the form of augmented reality (AR) mobile apps – users point their phones at the sky and the internal compass and accelerometer are used to show a projected image of the constellations, galaxies, and other such astronomical objects. These are great for on-the-spot observations, or for identifying astronomical objects that have already been seen. An example of one such app, Stellarium, is shown in both mobile and desktop screenshots below:

Figure 2: Orion constellation in Stellarium v. 1.8.3 running on Android 12 (Zotti, et al., 2020)

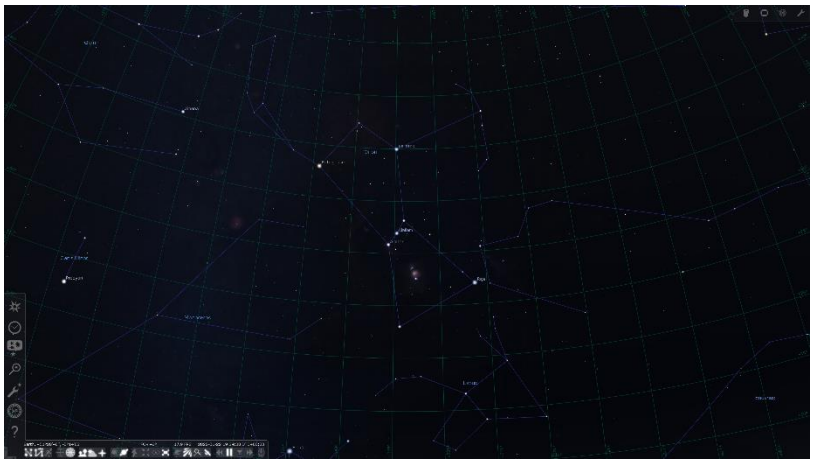


Figure 1: Orion constellation in Stellarium v. 0.21.3 running on Windows 11 (Zotti, et al., 2020)

1.2 PROBLEM TO BE SOLVED

A shortcoming of many of these types of software (not just the mobile apps) is that they do not provide automated facilities for planning observations in advance; fewer still consider the user's location beyond its geographic coordinates and local time. For example, it is of little use to tell a user that a star they want to observe is visible at celestial coordinates (x, y) if the skies are overcast and they are in the middle of a city with extreme light pollution – this is exacerbated when the user may want to carry out their observations at a future time, or at a different location to their home address and cannot simply go outside to check the conditions for themselves. Additionally, on top of their lack of automated planning facilities, many provide no way to take an astronomical observation log – a key part of any astronomy session.

1.3 PROJECT OBJECTIVES

1.3.1 MAIN OBJECTIVE

The main objective of AstroCue was to provide a set of integrated and unified web facilities for the automated planning and subsequent user logging of astronomical observations. It would consider geographic and celestial coordinates, local time, weather, and light pollution data to provide predicted best viewing times over a short upcoming period, and then allow users to log any observations that were made.

1.3.2 SUB OBJECTIVES

	Objective	Test
1	AstroCue shall produce reports on optimal viewing times for an astronomical object	Combinations of observation locations and astronomical objects should produce periodic reports on viewing times.
2	AstroCue shall export these reports to an external medium for future reference.	After a report has been generated, an email should be sent from the server to the user's email address via the MailGun API.
3	AstroCue shall be able to calculate positions of astronomical objects in the horizontal coordinate system with an accuracy of ± 10°.	Compare outputs to known correct test cases via unit testing.
4	AstroCue shall allow users to register observation locations with any valid latitude and longitude.	Any observation location with a latitude value in the range -90° to +90° combined with a longitude value in the range -180° to +180° should be accepted.
5	AstroCue shall be able to retrieve weather forecasts for any valid observation location from the OpenWeatherMap API.	Any observation location (with a valid longitude and latitude) can have a weather forecast retrieved for it.
6	AstroCue shall allow users to create astronomical observation logs linked to the reports it generates (see sub objective 1).	Observation logs can be made against a generated report.
7	AstroCue shall be able to retrieve light pollution levels for arbitrary observation locations.	Compare outputs to known correct test cases via unit testing.

1.3.3 PROJECT BENEFICIARIES

The main beneficiaries of AstroCue were first and foremost be the author, followed by other amateur and hobby astronomers.

1.3.3.1 BENEFITS

The benefit of AstroCue was its automated calculation of optimal viewing times. Users did not need to manually check if a star is visible where they are planning to observe, what the weather was like over the next few days there, or if this location had too much light pollution to easily observe their chosen object – AstroCue took all this information into account to produce succinct, easy to understand reports.

1.4 WORK PERFORMED

To meet the objectives, the author followed a software development lifecycle based on Agile. As this was a software development project, the delivery of working software was prioritised.

The author began with a literature review (section 3) which involved background research in the area. Several similar pieces of software were found, and their advantages/disadvantages weighed up against the goals of AstroCue. This also led to the discovery of several existing data sets that would go on to be used in development.

Next, methods were defined (section 4). These included both design approaches, implementation details such as design patterns, and the approach taken towards testing.

The main section of this report is section 5 (results). This summarises all the main work that went into the development of AstroCue. There you can find summaries of the design work, lots of implementation records with code snippets, details about the computational astronomy calculations used, as well as testing results.

The overall outcomes can be found in the conclusion (section 6).

1.5 SCOPE LIMITATIONS

There were several actions taken to limit the scope of AstroCue:

- All timestamps were outputted in UTC. The author did not attempt to internationalize these.
- Celestial coordinates were not adjusted for Earth's precession or nutation (see 9.8).
- Report generation times were based on UTC. Again, this was not internationalized.
- Planets were not included, only stars and deep sky objects.
- The Moon's impact on light pollution was not calculated.
- Earth's atmospheric refraction was not considered when calculating positions in the horizontal coordinate system.

2 OUTPUT SUMMARY

1	**Description:** AstroCue Server.	**Beneficiaries:** End users (inc. author), markers.
Type of output: An ASP.NET Core web API written in C# comprised of 10,782[1] lines of source code split across 86 classes (18 of which were automatically generated). Hosted live on the web using Microsoft Azure. Functionality can be accessed via the frontend at astrocue.co.uk.		
Results chapter(s) link: 5.4.1.1 – server overview in results.		
Appendices link: 9.6, 9.12 – deployment to the cloud, and raw code respectively.		

2	**Description:** Server unit tests.	**Beneficiaries:** Author, future developers, markers.
Type of output: A Visual Studio unit test project written in C# comprised of 2,983[1] lines of source code split across 20 classes.		
Results chapter(s) link: 5.5 – unit test results.		
Appendices link: 9.9.4, 9.12 – unit test complete outputs, and raw code respectively.		

3	**Description:** AstroCue Client.	**Beneficiaries:** End users (inc. author), markers.
Type of output: A React web app written in TypeScript. Comprised of 4,300[2] lines of source code, split across 73[2] files. It can be accessed live on the web to test at astrocue.co.uk.		
Results chapter(s) link: 5.4.1.2 – client overview in results.		
Appendices link: 9.6.3, 9.12, 9.13 – deployment, raw code, and all web pages respectively.		

4	**Description:** Requirements document.	**Beneficiaries:** Author, markers.
Type of output: Word document or exported PDF.		
Results chapter(s) link: 5.2.2 – requirements results.		
Appendices link: 9.4 – requirements details in full.		

5	**Description:** Design document.	**Beneficiaries:** Author, future developers, markers.
Type of output: Word document or exported PDF.		
Results chapter(s) link: 5.3 – design results.		
Appendices link: 9.5 – design details in full.		

[1] According to Visual Studio code metrics
[2] According to VS Code Counter

6	**Description:** Deployment document.	**Beneficiaries:** Future developers, markers.
Type of output: Word document or exported PDF.		
Results chapter(s) link: 5.4.1.1.1, 5.4.1.2.1 – server and client deployment results.		
Appendices link: 9.6 – full document.		

7	**Description:** Introduction to celestial coordinate systems.	**Beneficiaries:** Markers.
Type of output: Word document or exported PDF.		
Results chapter(s) link: 5.4.11 – computational astronomy results.		
Appendices link: 9.8 – full document.		

8	**Description:** Testing document.	**Beneficiaries:** Markers.
Type of output: Word document or exported PDF.		
Results chapter(s) link: 5.5 – testing results.		
Appendices link: 9.9 – full document.		

9	**Description:** Software build guide.	**Beneficiaries:** Markers.
Type of output: Word document or exported PDF.		
Results chapter(s) link: N/A.		
Appendices link: 9.10 – full document.		

3 LITERATURE REVIEW

3.1 INTRODUCTION

When creating software, it is important to consider existing work in the field. This process will help to provide a solid foundation to start the required work and give a background as to what has already been built and can be utilised/built upon, easing the development process.

3.2 SIMILAR APPLICATIONS AND TOOLS

This section covers existing software and available tools that fall into the same domain as AstroCue.

3.2.1 CLEAR SKY CHART

Clear Sky Chart is a weather forecast service targeted at astronomers. The generated forecasts are in the form of tabular images that provide a clear view on several upcoming days and their optimal times for observations (Danko, 2022). While these forecasts provide a visual way to discern clear nights, a large limitation is the fact that they are limited to a select number of locations in the US, Canada, and Mexico. However, the general concept of a sliding window approach will prove useful as a source of inspiration for a method of weather analysis when AstroCue approaches the same problem as part of its development.

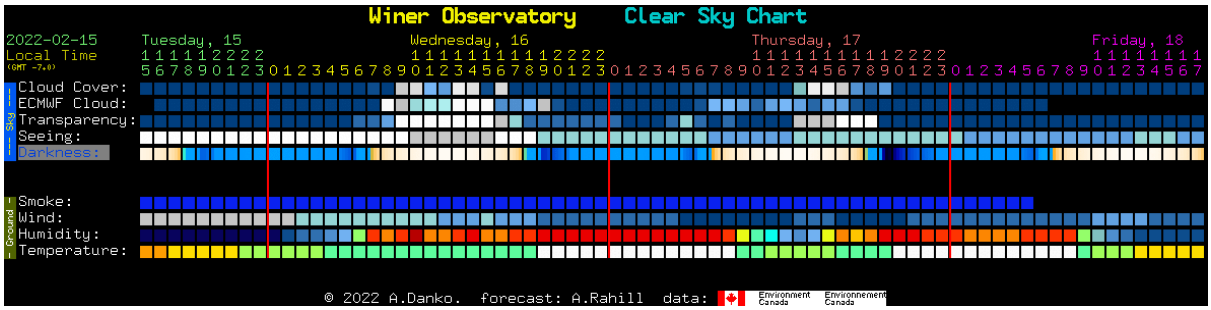


Figure 3: An example of a Clear Sky Chart for Winer Observatory, Arizona (Danko, 2022)

3.2.2 ASTROPLANNER

AstroPlanner is a desktop application for MacOS and Windows that allows users to create observation plans that are then visualised and can have logs taken against them (iLanga Inc, 2020). It has a large suite of tools including lunar cycles, support for vast collections of astronomical catalogues (totalling hundreds of millions of objects), can control telescopes via motorised 'GoTo' mounts, and can download images of observation targets from the Sloan Digital Sky Survey (SDSS).

A downside, however, is that it only tracks a limited amount of information about the observer's location – artificial light pollution is not considered, and there is also no use of local weather

forecasts to advise any observation plans. The plans themselves are also entirely manual. Users need to enter dates and times and AstroPlanner does not attempt to provide any guidance on them.

On the other hand, AstroPlanner offers lots of logging options. Users can record seeing levels (astronomical level of visibility) in many different scales and can attach images to their logs. These features will help to guide the development of similar functionalities in AstroCue.

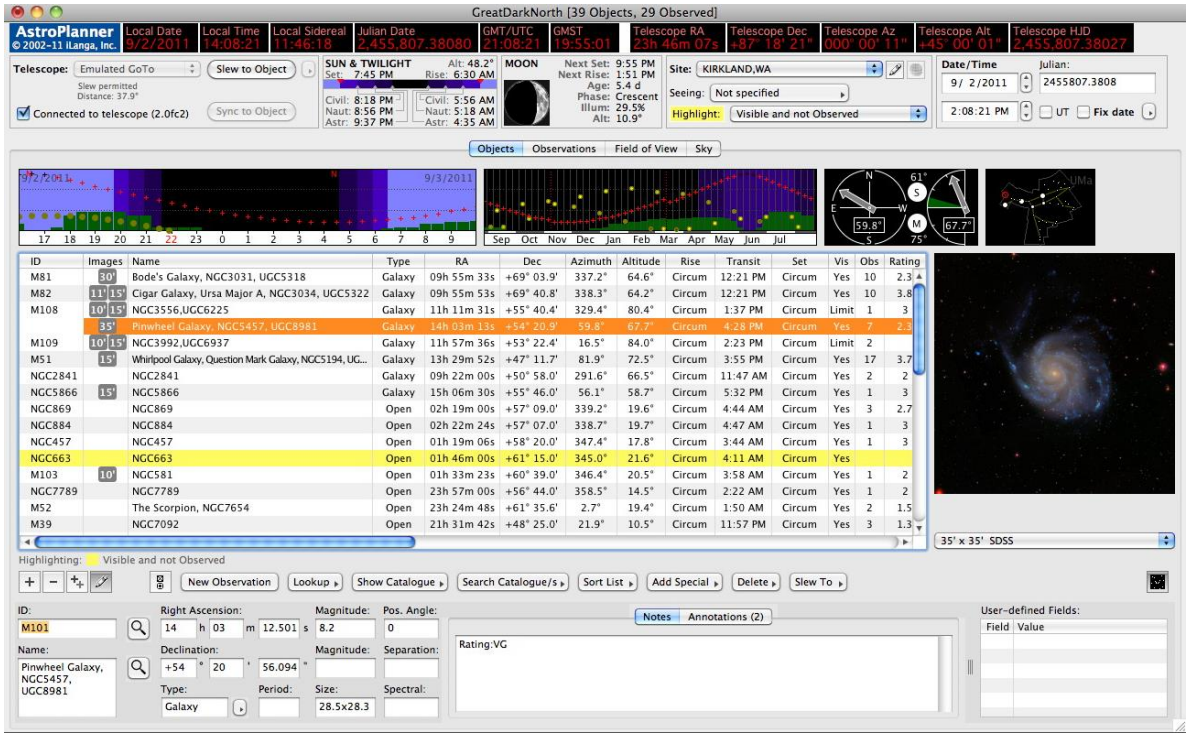


Figure 4: AstroPlanner running on MacOS (iLanga Inc, 2020)

3.3 DATA SOURCES

This section covers the Application Programming Interfaces (APIs) for weather and mapping data, light pollution datasets, as well as astronomical catalogues for stellar and deep sky object data.

3.3.1 VIZIER

To reason about astronomical objects, a list of their details is required. These details come in the form of astronomical catalogues. Over the years, there have been many space missions whose goals have been simply to observe and document astronomical objects in said catalogues. The Centre de Données astronomiques de Strasbourg (CDS) is a data centre whose goal it is to gather and consolidate many of these catalogues under a single queryable database service called VizieR (Ochsenbein, et al., 2000). Once a catalogue is selected, users can pick and choose exactly what details they wish to export, and in what format.

For AstroCue, the core pieces of information required for export will be celestial coordinates (explained in detail in a later section) and visible apparent magnitudes (how bright an astronomical object is when viewed from Earth). These will be exported in machine-readable formats.

3.3.2 LIGHT POLLUTION DATA

The new world atlas of artificial night sky brightness is a worldwide set of measurements of light pollution levels. It was created using collated measurements from a mixture of Sky Quality Meter (SQM) and charged-couple device (CCD) devices, as well as data taken from NASA's Visible Infrared Imaging Radiometer Suite (VIIRS) instruments aboard the NOAA-20 satellite (Falchi, et al., 2016a, p. 1) (NASA Earthdata, 2022). An advantage of using this data is that the brightness map was released as a supplement in the form of a machine readable 2.9GB GeoTIFF file (Falchi, et al., 2016b).

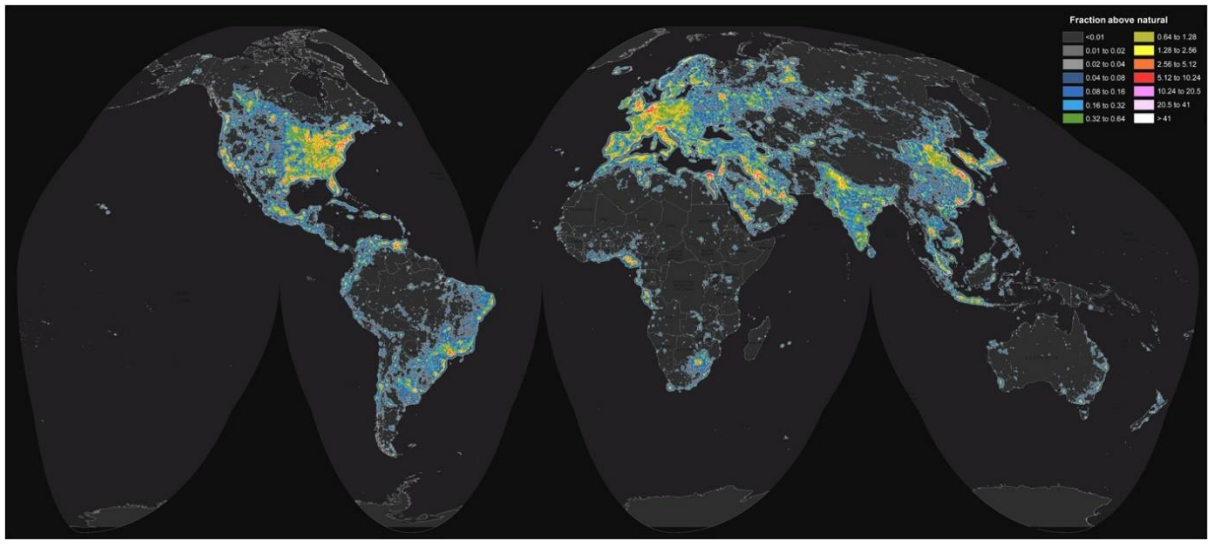


Figure 5: "World map of artificial sky brightness" (Falchi, et al., 2016a, p. 2)

3.3.3 MAPBOX API

AstroCue requires mapping data both for observation locations and carrying out forward geocoding (converting textual addresses to coordinates, such as 'Northampton Square, London' → 51.5274° N, 0.1022° W). The MapBox API offers both features and has the added benefit of a substantial free tier of 50,000 static map requests and 100,000 geocoding requests per month (MapBox, n.d.).

3.3.4 OPENWEATHERMAP API

To take weather conditions into account, a forecast service was needed that could be queried using any coordinates. That use case is covered by the OpenWeatherMap API (OpenWeather, n.d.). As with MapBox, there is a free tier, but this can be further increased under the student initiative (OpenWeather, n.d.).

3.4 DEVELOPMENT

This section covers software and related tools to be used in development.

3.4.1 IDE

For development, the author chose Visual Studio. There are 2 main reasons for this: Visual Studio has native integration with Microsoft Azure, the targeted deployment platform, and it has support for JetBrains' ReSharper extension (available via a student license), providing useful tools such as unit test coverage and advanced refactoring options (JetBrains, n.d.).

3.4.2 DEPLOYMENT

For deployment, the choice was mostly guided by the desire to use C# for the server development due to familiarity. Since C# is developed by Microsoft, deploying to the Azure App Service offers the combined benefit of having first party support for both the language and cloud operator (Microsoft, 2022) – meaning large amounts of documentation and online support are available.

3.4.3 DATABASE

A quick analysis of the data that will need to be stored leads to the fact that it is highly relational (users owning locations which are linked to observations which are linked to astronomical objects etc…). This suits a relational database well. The author decided to use Microsoft SQL Server due to it having both local (development) and Azure (deployment) versions – sharing data access code across both scenarios (Microsoft, 2022).

3.5 CONCLUSION

After having conducted this review on existing software, data sources and development tools, AstroCue has a solid set of building blocks that can be used to kickstart development. After catalogues have been exported from VizieR, and all the required APIs have been integrated, the only major step left will be to combine them all together to produce the desired functionality. As a result, less time will be spent researching during development, hopefully leading to as few delays as possible.

4 METHODS

The following section encompasses the work that was carried out to ensure AstroCue's success. It covers the major stages of the project, beginning at analysis and requirements gathering and working through implementation all the way to the final evaluation.

4.1 ANALYSIS

This section covers the planning and requirements gathering that took place prior to development.

4.1.1 PLANNING

4.1.1.1 GANTT CHART

To guide the general progress needed to complete AstroCue, a Gantt chart was created. The original version of this can be seen in Appendix A: Project Definition Document (PDD), however there was an unforeseen set of delays due to university strikes. It took longer than expected to gain ethics approval, and to reflect this an updated chart was created which can be seen in section 5.

4.1.1.2 METHODOLOGY

In terms of software development methodologies, a variant of agile was adopted. First and foremost, AstroCue was an *application development project*; the degree to which any requirements may or may not have been satisfied would have been determined by whether a functional piece of software was delivered in a timely manner. Since one of the founding principles of agile is that working software should be valued above all and delivered frequently, the author planned for 4 sprints throughout which there was consistent unit testing of key business logic to ensure that functionality was correct based on requirements, resilient, and delivered on time.

However, some parts of agile were not implemented. These included cross functional teams and group sprint retrospectives which did not apply as there was only a single developer.

4.1.2 REQUIREMENTS GATHERING

4.1.2.1 QUESTIONNAIRE

To gather an initial set of requirements, a study needed to be carried out on what features members of the target audience would find the most useful in a platform like AstroCue. This was facilitated via an anonymous online questionnaire. The first page of this questionnaire displayed the participant information sheet (PIS) and was there to gather consent from participants before any questions were asked. This PIS used heavily borrowed from City, University of London's PIS template (City, University of London, n.d.).

4.1.2.2 Sampling method

An opportunity sample was used to gather participants. This is a type of convenience sampling that involves directly asking members of the target population if they would agree to take part. While it has the downside of often adding a bias to the data that is received, it has the huge upside of speeding up the sampling process (Macleod, 2019). All participants can be gathered in a single day if required. With only limited time to complete AstroCue, a sizeable percentage of which was targeted at the implementation, it was of paramount importance to ensure that no unnecessary delays were introduced. To this end, opportunity sampling was deemed an appropriate choice.

4.1.2.3 Requirements prioritisation

Once a set of potential requirements had been gathered by the questionnaire, the MoSCoW (must have, should have, could have, will not have this time) method of prioritisation was used to ratify the set of requirements that would be taken forward. Once these had been decided, Volere snow cards were created to provide some further context and rationale to each requirement to aid development and testing.

4.1.2.4 Use cases

To build upon the use cases that arose from the requirements, a use case diagram was created showing the actors, use cases, and their relationships and dependencies. This would provide the groundwork to build a set of use case specifications from which tests could be carried out.

As AstroCue had many secondary actors (all the third-party APIs and time) as opposed to just a single primary actor (the user), the use case diagram was useful as it allowed the visualisation of the degree to which different sections of functionality relied on secondary actors to complete their goals.

Finally, use case specifications were created. These specifications showed the order of execution of the operations that made up various use cases, and where in their flow the extension and inclusion relationships to other use cases were invoked.

4.2 DESIGN

This section covers how the design of AstroCue was carried out.

4.2.1.1 CLASS DIAGRAM

As mentioned in section 3.4, the author opted for C# for server development. This was driven primarily by experience; a year at university was spent learning Java (syntactically similar), and the author's industrial placement allowed them to gain experience developing industry standard software in C#. C# itself is an object-oriented language, meaning the paradigm of programming focuses on objects (data) and their methods (operations on data). A class diagram was created to provide a visual representation of these different classes, along with their relationships to one another. It also showed inheritance, demonstrating exactly what shared behaviour was extracted to common, reusable parent classes.

This acted as a springboard; defining packages and hierarchies before any time had been committed to development.

4.2.1.2 ENTITY RELATIONSHIP DIAGRAM

Section 3.4 also touched on the decision to use a relational database for the persistence of data. To begin, there needs to be some ideating of the different entities that form a part of the database (there is not a direct 1:1 link between the class diagram's entities and the entities that will be stored in the database). The ERD entities typically represent real world things or concepts, in the case of AstroCue, this ends up covering things such as users and astronomical objects.

ERDs can also denote relationships between different sets of entities, and in some cases, how these relationships can go on to form relationship entities of their own (Chen, 1976, p. 19). This concept applies directly to AstroCue, such as users having zero or more observation locations, and these observation locations being tied to zero or more observations of astronomical objects.

4.2.1.3 DEPLOYMENT DIAGRAM

AstroCue was a distributed piece of software deployed to the cloud. A deployment diagram was created to show the different elements of this deployment, and how and where communication between various nodes and their software took place.

This was created because cloud deployments quickly gain complexity and having a diagram showing both where each node of the system should be located conceptually, and how it is allowed to communicate with other nodes and their software, would help when creating the real deployment.

Additionally, AstroCue integrated with external APIs. Showing where these were accessed from within the deployment further solidified these interactions and which elements of the deployment facilitated them.

4.2.1.4 Logo

AstroCue needed a logo. One was created with a free planet icon and 'sci-fi' genre font to give a bespoke space themed look.

4.3 IMPLEMENTATION

This section covers the implementation details of AstroCue.

4.3.1 ARCHITECTURE AND DEPLOYMENT

Broadly, AstroCue followed a distributed client-server architecture.

4.3.1.1 SERVER

The server was an ASP .NET Core web API written in C# and deployed to Azure. It exposed HTTP endpoints that could be contacted via the standard HTTP verbs (GET, SET etc…) to carry out operations. The deployment involved an App Service, a database, and the Docker image itself which was deployed to the Docker Hub. Also deployed to Azure, but not related to the server, was a Blob Storage instance for static email assets.

4.3.1.2 CLIENT

The client was a React and Next.js application written in TypeScript deployed to Vercel. It defined AstroCue's user interface. Based on the user's actions, it displayed their data, communicated with the server, and generally allowed them to directly realise the use cases. The deployment only involved a single element, the client application itself.

4.3.2 DEVELOPMENT ENVIRONMENT

4.3.2.1 IDES / EDITORS

As mentioned in section 3.4.1, the author decided to use Visual Studio (VS) for the server development. Again, partly due to familiarity but also because of native integrations with Azure. The author made use of VS's ASP .NET Core web API template to speed up initial development.

For the frontend, VS Code was used. It's important to make a distinction here that VS ≠ VS Code; the former is an IDE, the latter a text editor. VS Code provides useful type annotations for TypeScript code and has a plethora of extensions that can be used to aid development of web applications. The author also made use of a starter template project here to get a barebones configuration up and running faster.

4.3.2.2 DOCKER

.NET Core was chosen over .NET Framework due to its cross-platform abilities. It can be hosted on Linux systems in the cloud, saving in hosting costs due to no operating system licensing being imposed. At the time of writing, a B1 tier Azure App Service Windows instance cost just under 6x more than its Linux counterpart (£9.898/month vs £51.686/month) (Microsoft, 2022). As the author

was using the $100 student credit offer, hosting as cheaply as possible was of paramount importance.

To make sure the code behaved the same in development as it would in the cloud, the server was containerised using Docker. Docker packages the code, OS libraries and all required dependencies into a single image that will run consistently across many environments (IBM, 2021). Microsoft's ASP .NET Core template offers Docker support out of the box, and the author reused the resulting Dockerfile to avoid needing to write one from scratch.

4.3.2.3 Database

Testing development code against a production database incurs unneeded costs and has larger overheads. Due to this, the author used Microsoft SQL Server's developer edition, which offers a development database that can be ran locally alongside the code (Microsoft, 2022).

4.3.2.4 API testing with Postman

The web API would expose different endpoint that could be contacted for data. To test these during development, Postman was used.

4.3.3 Authentication

For authentication, JSON Web Tokens (JWTs) were used. These are small, cryptographically signed pieces of data that the server can use to identify clients. Once signed in, a client would receive a JWT that they send with each subsequent request for authentication.

The author considered cookies, but due to the distributed nature of the client and server, they were on separate domains. Cookies are only sent by browsers to the same domain they were issued for, so this was considered unfeasible, and any domain or software workarounds too time intensive.

To implement this, the author leant on code that he had already written for the IN3046: Cloud Computing module (see Appendix B: Reuse Summary). This was done to save development time so time could spent focusing more on AstroCue's core features.

4.3.4 APIs

AstroCue interfaced with the MailGun, OpenWeatherMap and MapBox APIs. These were chosen above other similar services due to their generous free tiers, as mentioned in section 3.3.

4.3.5 Email template reuse

AstroCue's emails were based on free templates from Unlayer (https://unlayer.com/). This saved time during development; the author could focus more on the content inside the emails rather than

their appearance. To make these templates dynamic, Handlebars templating was added (https://handlebarsjs.com/).

4.3.6 Object-relational mapper (O/RM)

The author used Microsoft's Entity Framework Core. It allows developers to work with a database directly via entity classes and a context object. EF Core creates migrations that bring the database's tables and relationships in line with the state of the entity classes. (Microsoft, 2021). This saves time when writing data access code – an important factor under the time-limited circumstances.

4.3.7 Design patterns

4.3.7.1 MVC and deviations

AstroCue followed the Model-View-Controller (MVC) pattern. Except rather than the view layer being integrated as part of the same codebase as the models and controllers, it was a separate application hosted on a different domain.

However, a deviation AstroCue took from pure MVC was the introduction of a service layer to handle the database access and large percentages of the business logic. This then conceptually becomes MVC(S), with the services adding an extra 'layer' to the pattern. The service classes had their specific operations represented by interfaces to provide abstractions.

This service layer pattern is useful in situations where there are multiple clients to the same pieces of logic (i.e., controller X and service Y both requiring access to logic in service Z). Extracting reusable and related operations into reusable services reduces duplication (Fowler, 2012, p. 133).

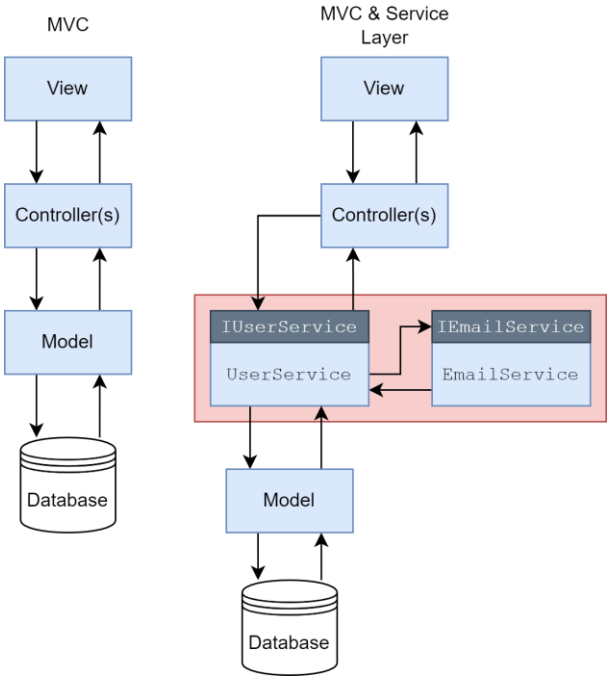


Figure 6: MVC vs. MVC(S); services highlighted red. Services may or may not be directly invoked by controllers and can have interdependencies amongst themselves. In this case, both UserService and a controller both have a dependency on EmailService.

These abstractions constitute the 'I' from the 'SOLID' principles of Object-Oriented Design: "Make fine grained interfaces that are client specific" (Martin, 2005). The benefit from this was that controllers stayed thin, and the service layer classes became more reusable and maintainable.

4.3.7.2 Dependency injection (DI)

Building on the idea of a service layer, DI was used to inject concrete classes into other classes that depended on their abstractions. This is the 'D' from 'SOLID': "Depend on abstractions, not on concretions" (Martin, 2005). Again, increasing maintainability and reusability.

Figure 7: 'SOLID' Principles of OOD (InterVenture, 2015). I and D being the last 2 in the list.

4.3.8 VERSION CONTROL

4.3.8.1 GIT AND GITHUB

The author used Git for version control, and GitHub for repository hosting. Separate repositories were created for the client and the server.

To track issues and tasks being carried out on the codebases, a GitHub Projects Kanban board was created. This sorts tasks into categories of 'to do', 'in progress' and 'done' (GitHub, n.d.). The two repositories were added to this project so issues could be tracked against their various branches and pull requests.

4.3.9 ASTRONOMICAL CATALOGUES

As mentioned in section 3.3.1, VizieR was used to source astronomical catalogues. The author exported 3 catalogues directly, but a 4th catalogue was also sourced from the International Astronomical Union (IAU) for official star names. Custom parsers were created to read these catalogues and insert them into the database.

4.3.10 COMPUTATIONAL ASTRONOMY

To aid in the computational astronomy (talked about more in the results chapter), various equations from the widely acclaimed 1998 book 'Astronomical Algorithms' by Jean Meeus were implemented (Meeus, 1998).

4.4 EVALUATION

4.4.1 FUNCTIONAL TESTING

The author created use case tests based on the use case specifications and carried out unit testing of as much core logic as was feasible in the available time.

4.4.2 NON-FUNCTIONAL TESTING

For non-functional testing, the author measured software maintainability. As AstroCue contained lots of different classes and layers, ensuring that it was easy to integrate new features to was of high importance. Within Visual Studio, there is an option to calculate code metrics. One of these metrics includes a 'Maintainability Index'. This option "calculates an index value between 0 and 100 that represents the relative ease of maintaining the code" (Microsoft, 2021). The author made frequent use of this objective measurement during development to ensure consistent high levels of maintainability.

5 RESULTS

5.1 CHAPTER INTRODUCTION

This chapter aims to cover the outputs from the planning, design, implementation, and testing of AstroCue. It begins with the initial planning and requirements gathering stage, followed by the design and accompanying diagrams. It goes into depth on the implementation details mentioned in the methods chapter, including design patterns, tools, and computational astronomy algorithms, and ends with the evaluation results.

5.2 ANALYSIS

5.2.1 PLANNING

5.2.1.1 GANTT CHART

During the project definition stage, a Gantt chart was created covering the stages of development as they were initially envisioned. However, as mentioned in section 4.1.1.1, there were some unforeseen delays that led to a newer updated version of this chart being created.

It is not largely different from the PDD's version, except for the extension of the requirements stage and few minor additions to the sprints (on top of them being pushed back).

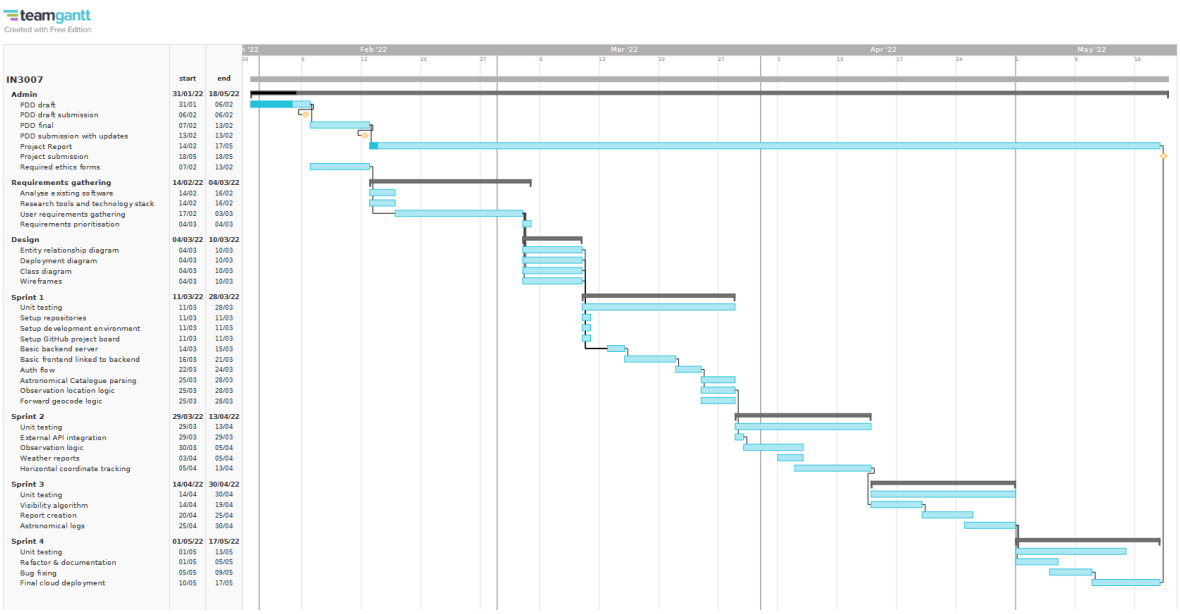


Figure 8: Gantt chart created with TeamGantt (TeamGantt, 2022).

5.2.2 REQUIREMENTS GATHERING

The questionnaire used to gather the set of requirements that would be used for AstroCue's development was created with Qualtrics. The questions ranged from what pieces of information users would find most useful in the reports, to how often they thought the reports themselves should be generated.

This questionnaire made use of City, University of London's participant information sheet as a template (City, University of London, n.d.). The author changed some of its content where required and set it as the front page of the questionnaire. This can be seen partly below.

CITY
UNIVERSITY OF LONDON
EST 1894

AstroCue Requirements Gathering Questionnaire

Invitation

We would like to invite you to take part in a research study. Before you decide whether you would like to take part it is important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information. You will be given a copy of this information sheet to keep.

What is the purpose of the study?

The aim of this study is to gather an initial set of requirements for 'AstroCue' – a platform for planning and logging astronomical observations.

Why have I been invited to take part?

You have been invited to take part in this research study due to your interest in astronomy and/or space.

Do I have to take part?

Participation in this survey is voluntary, and you can choose not to participate in part or all of the survey. You can withdraw at any stage of the project without being penalised or disadvantaged in any way. It is up to you to decide whether or not to take part. If you decide to take part you are still free to withdraw at any time and without giving a reason.

The point at which you can no longer withdraw your data is the final submission. Withdrawal at any point prior to this will result in any data collected thus far not being retained. Withdrawal occurs 1 hour after you have last edited a response and have still not reached the final submission.

Figure 9: AstroCue participant information sheet via Qualtrics

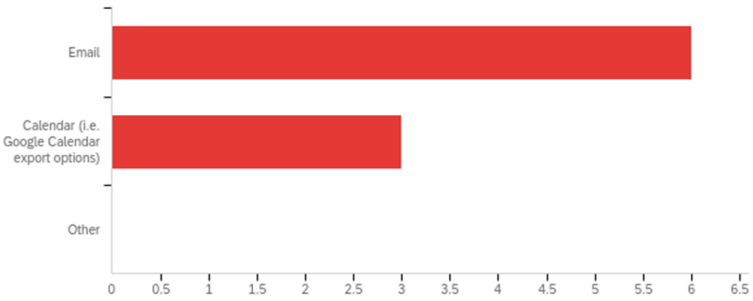
This also acted to gather consent. There was a yes/no consent question at the bottom to which only 'yes' answers would be able to continue to the questions. To see the full content of the participant information sheet, see Appendix C: AstroCue Participant Information Sheet.

The questions themselves covered various topics, an example of which is given below. In total, 6 responses were gathered. The responses were then exported from Qualtrics, including results tables and diagrams:

AstroCue aims to export the reports it generates. To what medium would these exports prove to be the most useful?

Tick all that apply.

- [ ] Email
- [ ] Calendar (i.e. Google Calendar export options)
- [ ] Other



For a full list of the questions and results from the survey, see section 9.4.1.

5.2.2.1 REQUIREMENTS PRIORITISATION

Once all the results from the survey had been collated, they were used to inform the requirements of AstroCue, which were ordered by level of importance using the MoSCoW method - **M**ust have, **S**hould have, **C**ould have, and **W**ill not have (this time). This allowed the author to assess not only what features would be the most in demand, but also their relative order of importance.

The top 2 sections of the prioritisation were as follows (the bottom 2 can be seen in section 9.4.2):

Must have:

[1] Observation reports must contain the forecasted weather.

[2] Observation reports must contain the IAU's formalized name if available.

[3] Observation logs must be able to contain a textual description of what was observed.

[4] Observation logs must be able to contain the weather at the time of the observation.

[5] Observation logs must be able to contain who carried out the observation.

[6] Observation logs must be able to contain where the observation took place.

[7] Observation reports must be generated every ~3 days.

[8] Observation reports must be exported via email.

Should have:

[1] Observation reports should contain the object's rise and set times.

[2] Observation reports should contain a link to discover more information about the object.

[3] Observation logs should be able to contain the type of observation made.

[4] Observation reports should contain calendar export options.

[5] A filter for popular/bright stars should be available when setting up observations.

These 2 sections informed the key requirements for AstroCue.

5.2.2.2 VOLERE SNOW CARDS

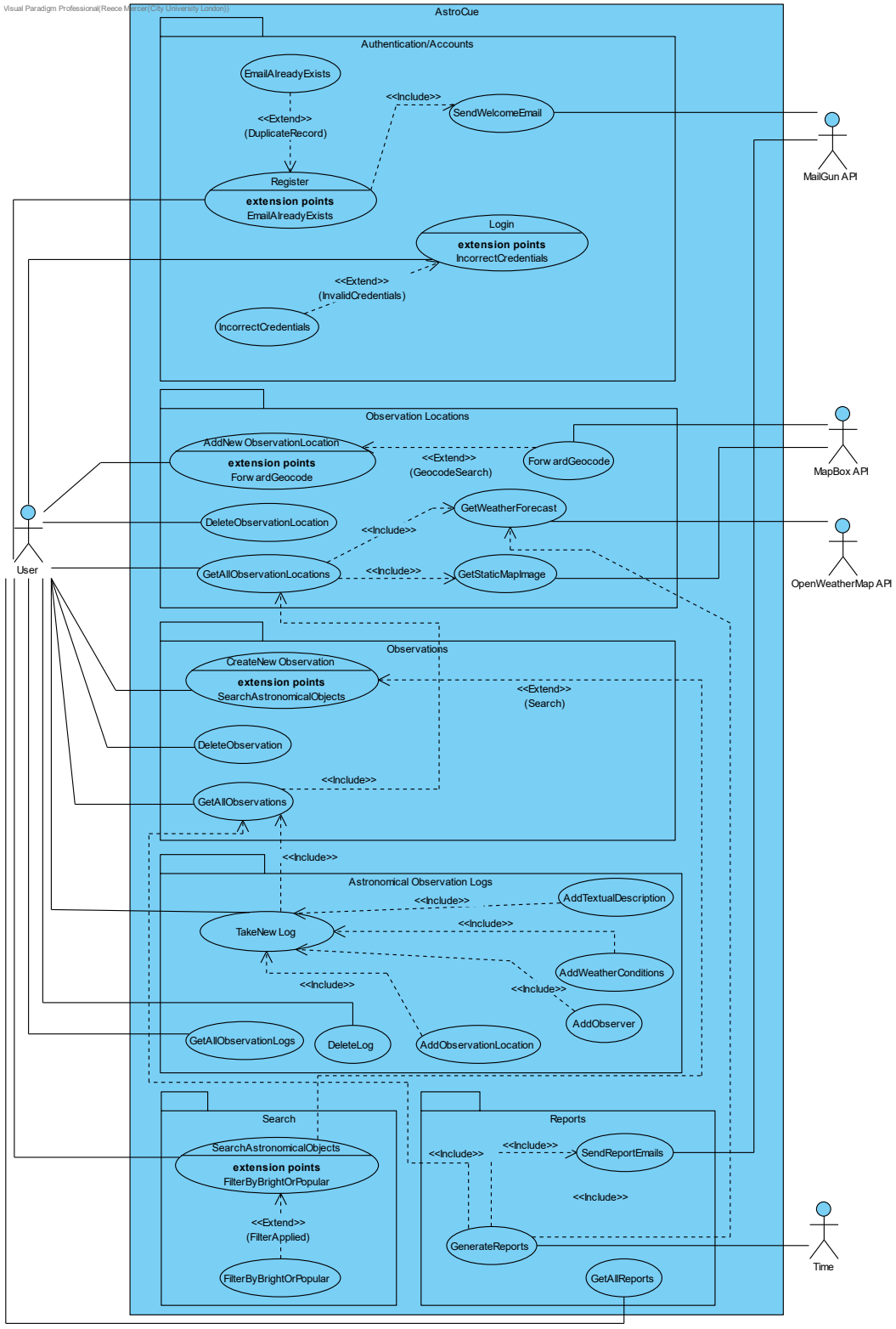
Once the requirements were prioritised, the author created Volere snow cards to consolidate the reasoning behind their existence. In total, 13 cards were created. An example is given below, with the full set in appendix 9.4.3.

#5	**Type:** Functional requirement
Description: Observation logs must be able to contain who carried out the observation	
Rationale: When carrying out an observation, the user may not be alone. When taking a log, they should be able to record who carried out the observation – this could be as simple as a list of names.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When taking an observation log, an entry area for the name(s) of the observer's must be present.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

Figure 10: Volere snow card for an observation log requirement

5.2.2.3 USE CASES AND DIAGRAM

Once a set of uses cases had been drawn up from the requirements and their rationales, a use case diagram was created. As mentioned in section 4.1.2.4, a key point here is the primary to secondary actor ratio:



For each of the use cases in this diagram, a use case specification was created. These created the basis for use case tests, covered later. Where present, alternative flows were given their own specifications.

Use case: AddNewObservationLocation
ID: 4
Brief description: Allows a user to add a new observation location to their account
Primary actors: User
Secondary actors: None
Preconditions: 1. The user is logged in
Main flow: 1. User enters the observation location details 2. Extension point: GeocodeSearch 3. System saves new location to database
Postconditions: 1. The user's new observation location is added to their account
Alternative flows: None

5.3 Design

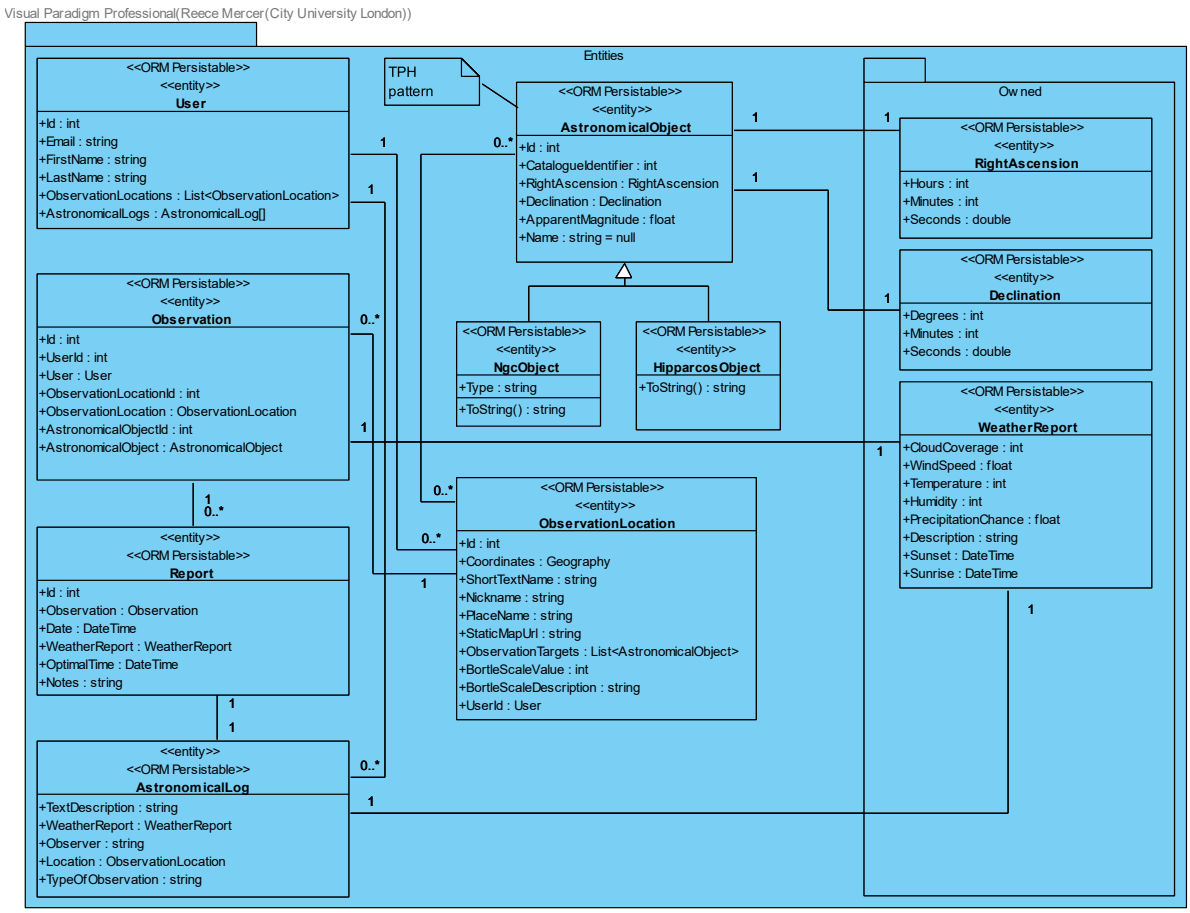
5.3.1 Class Diagram

Most of the logic needed to make AstroCue function would be (and was) housed in the server. This fact was known before development began, and to ensure that the design and layout of all the classes had a starting point rather than being made up as the author went along, a class diagram was created to show a large-scale overview of many of the envisioned classes and packages. Overall, there were 7 different packages in the full diagram, each encompassing a different element of the server. Only key packages are shown here. To view the diagram in its entirety, see Appendix E: Design (9.5.2).

5.3.1.1 Entities package

This package contains the entities ('models' from MVC) – the classes that mapped the database tables to C# objects. These were used by Entity Framework Core to create migrations; as the author added and updated these classes, EF Core was used to mirror them to the database.

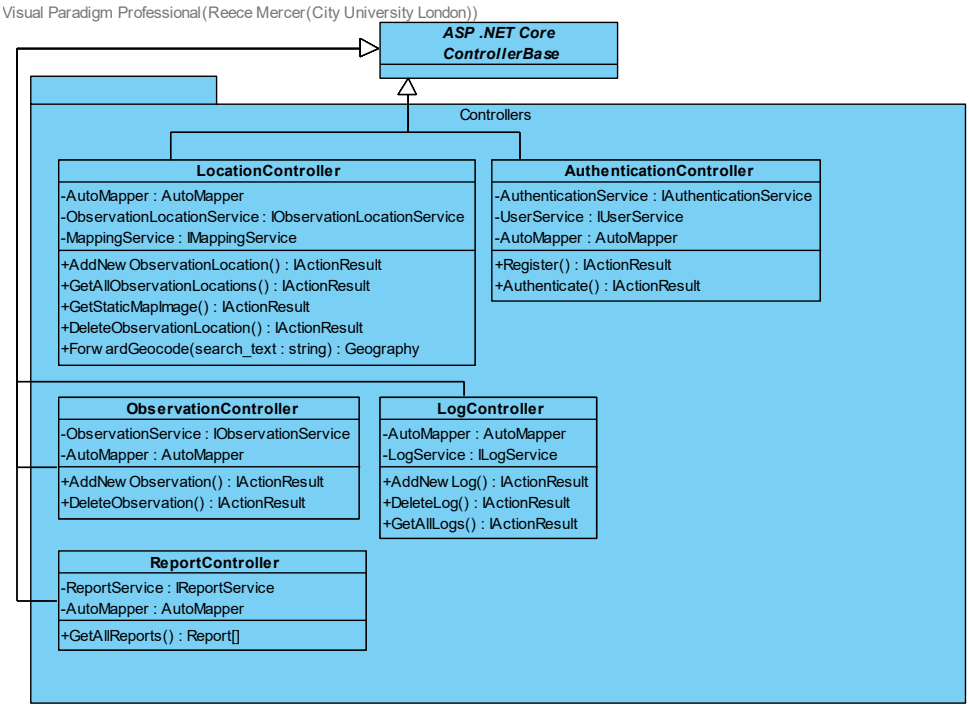
The entities in the 'Owned' sub-package did not form database tables directly, but rather became extra embedded fields in the tables of the entities that had them as child properties.



5.3.1.2 CONTROLLERS PACKAGE

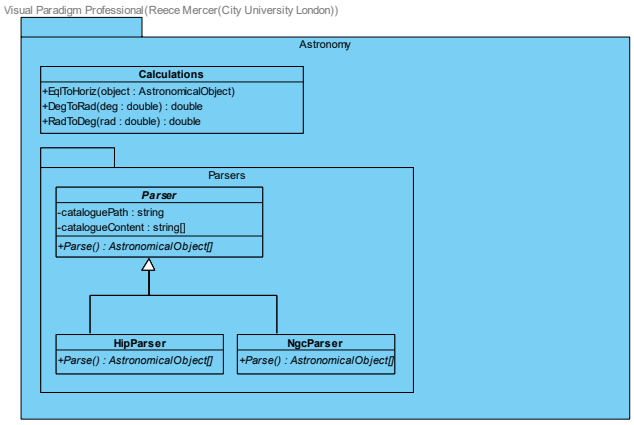
The controllers defined the public API to the web. They included all the possible endpoints that clients could contact for various resources and operations, as well as the required HTTP verbs (GET, POST, DELETE etc…), and parameters that these endpoints were expecting.

Note the inheritance from the abstract base class *ControllerBase* – this class is part of ASP. NET Core itself and inheriting from it provided lots of useful features such as responding with HTTP codes and accessing the HTTP context for each request that arrives.



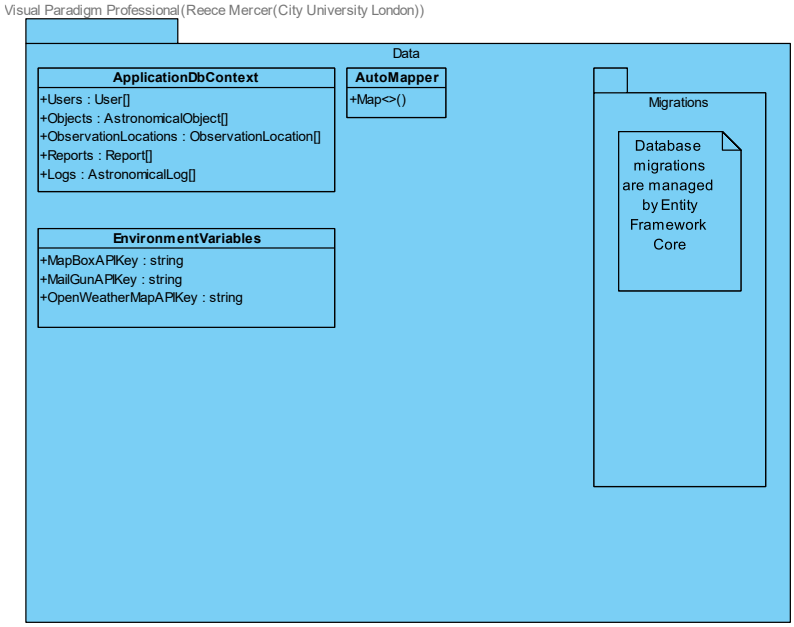
5.3.1.3 ASTRONOMY PACKAGE

This package contained all the computational astronomy logic (talked about later in section 5.4.11).



5.3.1.4 DATA PACKAGE

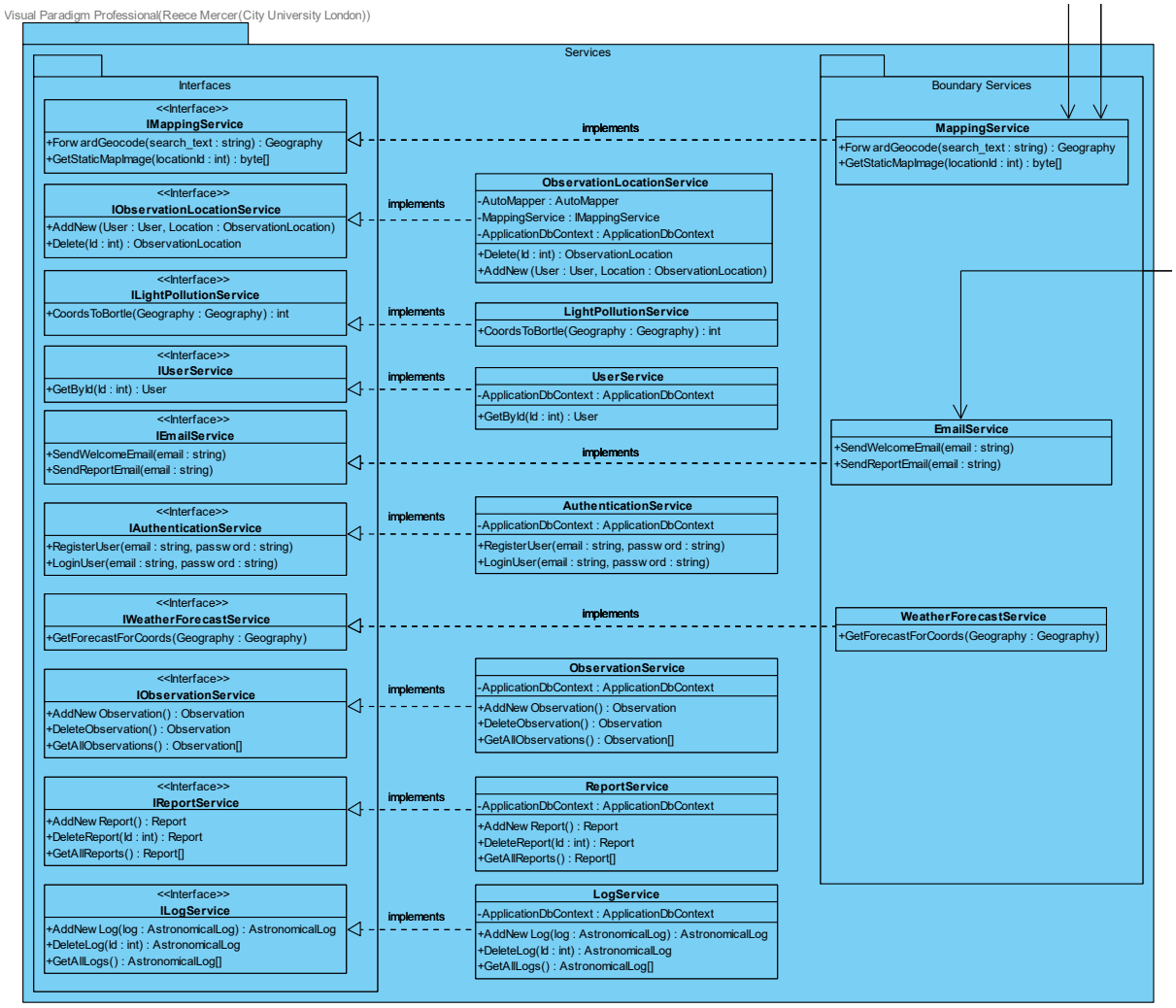
This package included any classes involved in interaction with external data (e.g., reading from astronomical catalogues, loading environment variables etc…). Note that in the final implementation, the parsers included in the previous package (section 5.3.1.3) were moved into this package. Also, the migrations ended up elsewhere.



5.3.1.5 SERVICES PACKAGE

Most of the data access logic and general domain specific logic was housed in this package. Each class here had its operations represented via a specific interface (see section 4.3.7.1 on MVC deviations). The interfaces provided both an abstraction for each class relying on a particular service, as well as the ability to create mocks during unit testing (section 5.5.1.2). On top of this, it provided the ability to change the concrete implementation of a given service under specific conditions (i.e., in development, the author often instantiated a special development version of the email service to prevent unnecessarily spamming himself during testing).

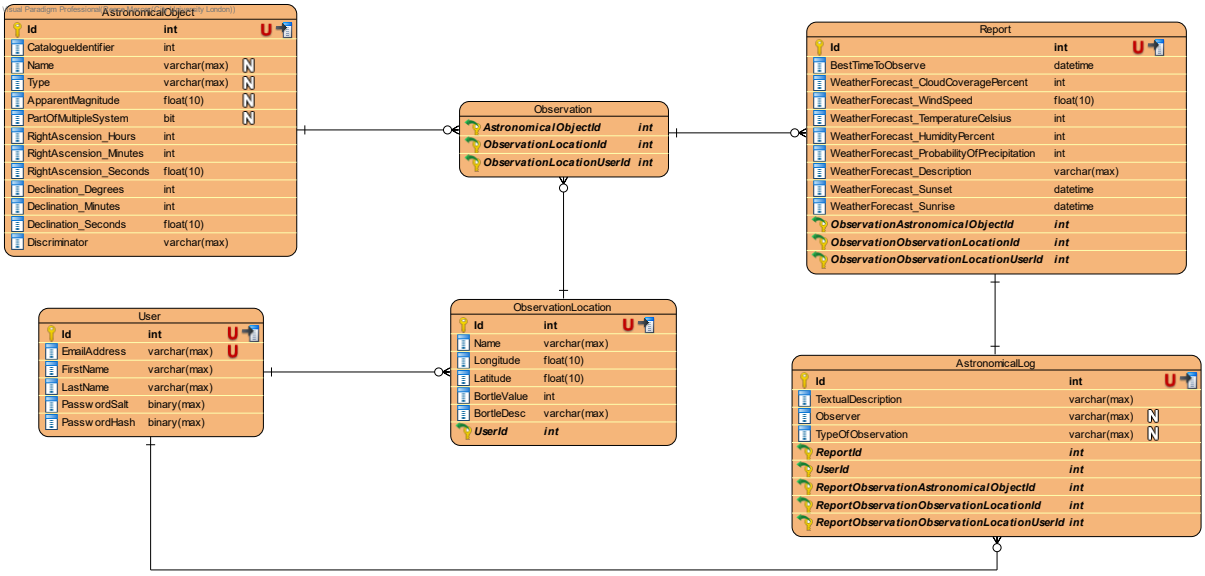
The sub-package 'Boundary Services' contained the services that communicated with any external service (i.e., `MappingService` made requests to the MapBox API).



5.3.2 ENTITY RELATIONSHIP DIAGRAM

To visualise the different entities within AstroCue and how their database tables would be structured, the author created an entity relationship diagram. Notice a few key points:

1. Despite two astronomical object entities existing in the class diagram (Entities package), there's only one `AstronomicalObject` table. In that table, however, there is a `Discriminator` field. This field is set to one of two possibilities to denote which type of `AstronomicalObject` a particular field represents. This is called the Table-per-hierarchy (TPH) pattern – the use of a single table to store multiple types from a single hierarchy (Microsoft, 2021).
2. The many-to-many relationship between `ObservationLocation` and `AstronomicalObject` forming the `Observation` joining table.



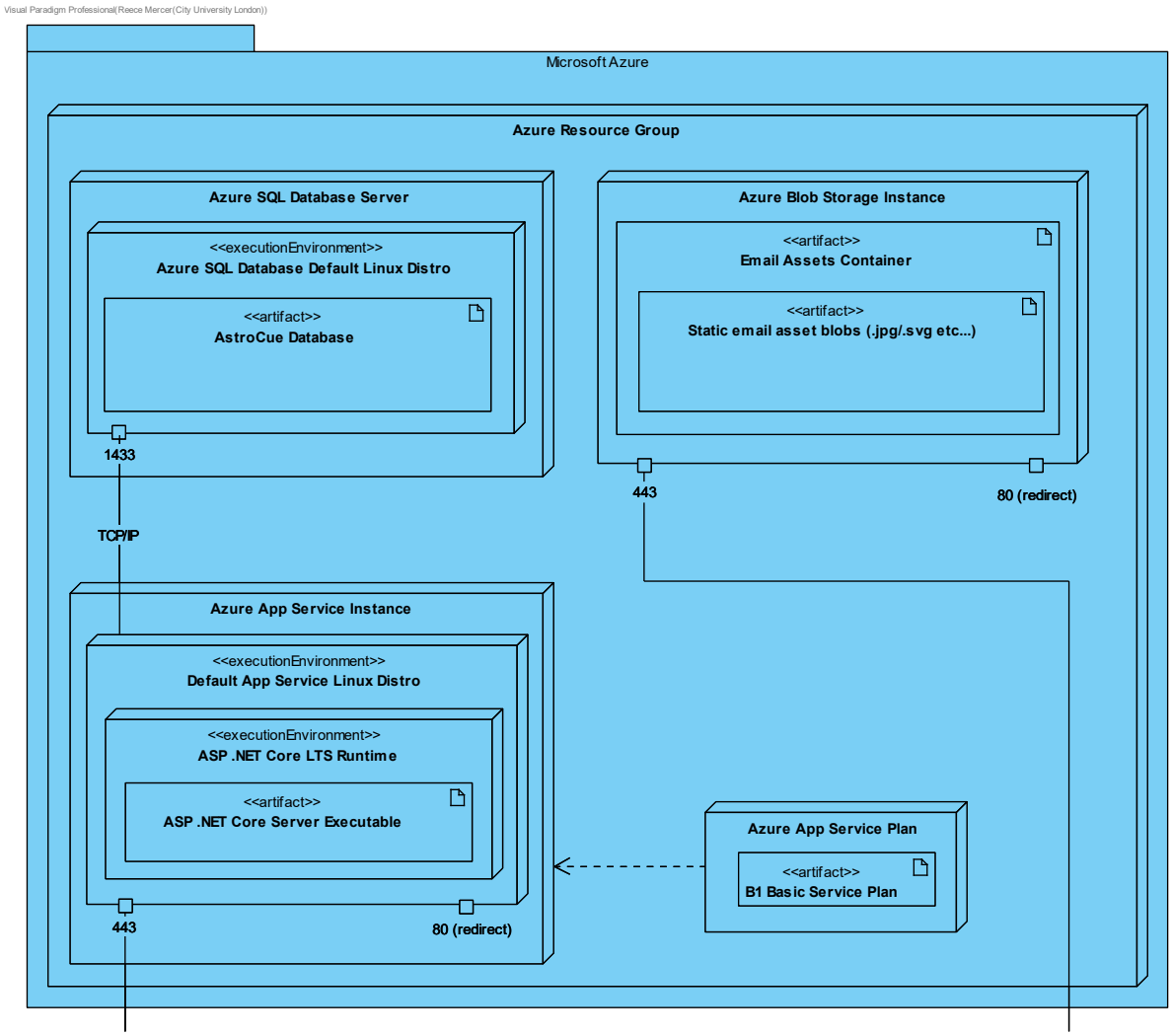
5.3.3 DEPLOYMENT DIAGRAM

A deployment diagram was created to document the different distributed sections of AstroCue, and how they were conceptually deployed and linked to one another. The full combined diagram can be seen in section 9.5.4.

5.3.3.1 AZURE

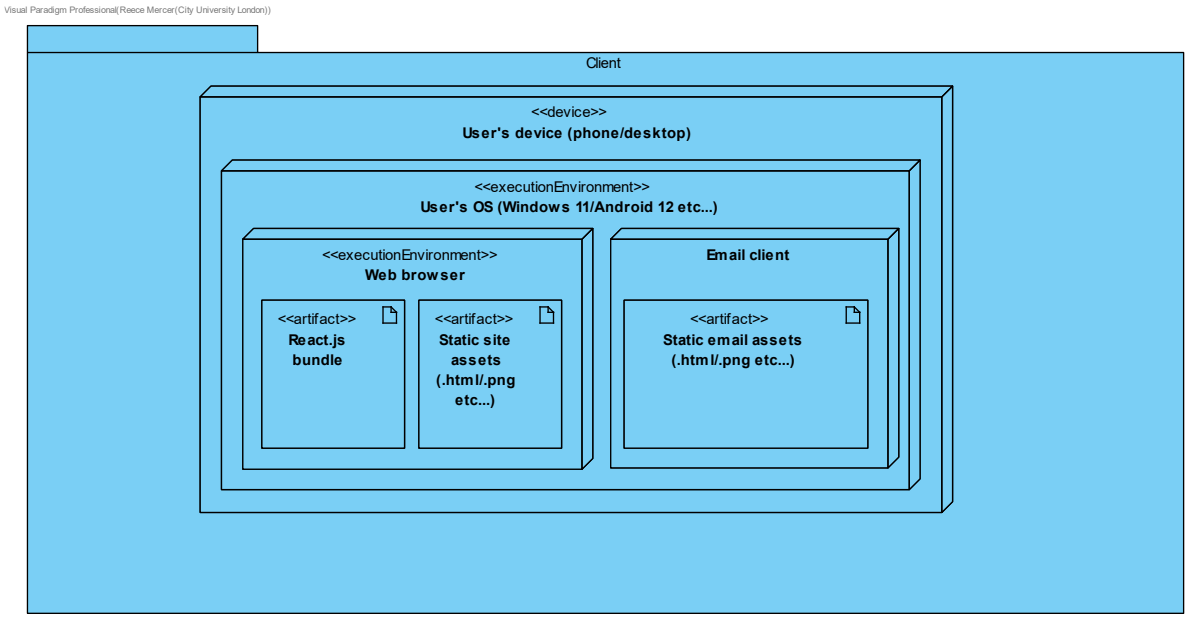
This package of the deployment diagram displays:

- The database server (top left)
- Blob Storage for email assets (top right)
- The App Service and its Service Plan (bottom two)



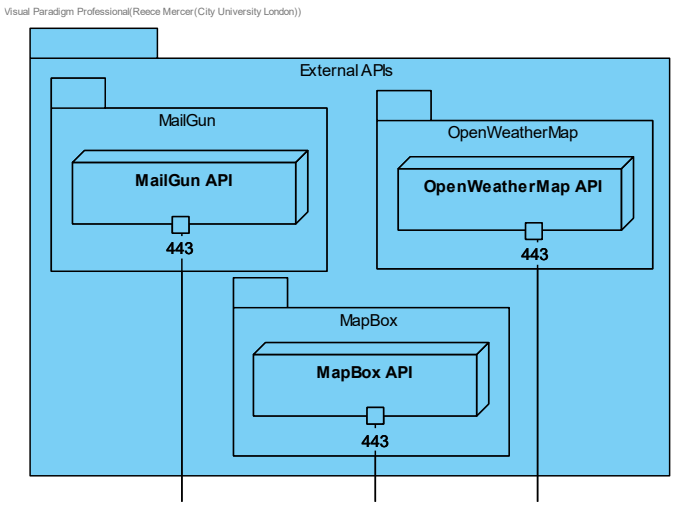
5.3.3.2 CLIENT

This package shows a typical user setup. The core parts of software required on their systems are a web browser and an email client. The software functions the same on phones as well as desktops/laptops, so the diagram makes this distinction clear.



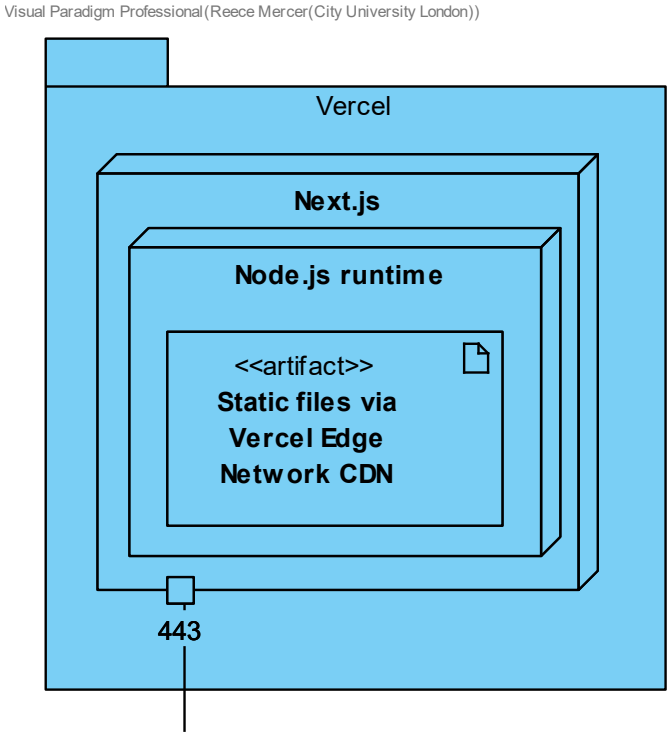
5.3.3.3 EXTERNAL APIS

This package simply adds the 3 external APIs for completeness. As their internal deployment is not public, they were just added as nodes with the HTTPS port open.



5.3.3.4 VERCEL

This package shows the frontend deployment. Vercel is much simpler due to its mostly static file nature. The core element for AstroCue is its static web assets being served by the Vercel Edge Network – a CDN (Content Delivery Network) with support for serverless functions if required (though not used in this case) (Vercel, 2022).



5.3.4 LOGO

The AstroCue logo (seen on the front page of this document and on each section's title), was made by the author using a combination of a free to use font and icon. Both light and dark versions were created:

The logo was used in this document as well as on the navigation bar on the frontend:

The icon and font were sourced from flaticon.com and 1001freefonts.com respectively (Flaticon, n.d.) (1001freefonts, n.d.).

For a full look into the design, see Appendix E: Design (9.5.5).

5.4 IMPLEMENTATION

5.4.1 ARCHITECTURE AND DEPLOYMENT

AstroCue was a distributed client/server application – and those two different aspects were implemented in separate projects with different languages.

5.4.1.1 SERVER OVERVIEW

The server structure largely matches the class diagram (section 5.3.1). The author created a folder per package:

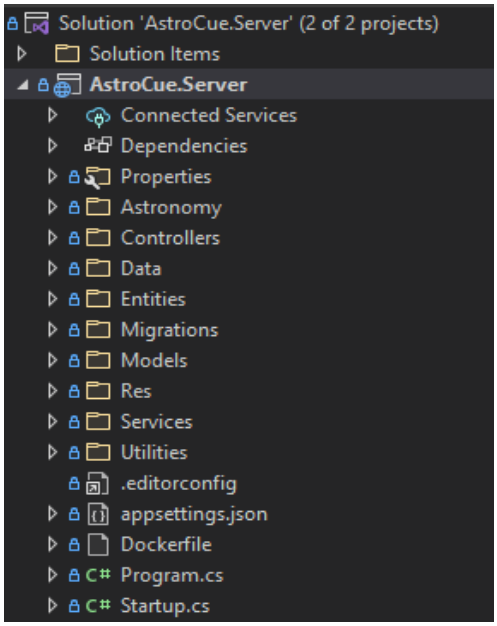


Figure 11: Server in VS

At the core of the API were the controllers. The following snippet shows an endpoint from `ObservationLocationController`. This is a HTTP GET endpoint, and its body shows it leaning largely on a service-layer method `GetAllAsync`. This endpoint returns all observation locations belonging to a user:

```
public async Task<ActionResult<List<OutboundObsLocationModel>>>
GetAllObservationLocations()
{
    int reqUserId = (int)this.HttpContext.Items[Constants.HttpContextReqUserId]!;
    return this.Ok(
        await  this._observationLocationService.GetAllAsync(reqUserId)
    );
}
```

The invoked service method is in `ObservationLocationService`. This service itself also relies on other services to complete its tasks. For example, when retrieving all locations, the `GetAllAsync` method invokes a method in `WeatherForecastService`:

```
IEnumerable<Task> forecastTasks = outboundObservationLocations.Select(
    async location =>
{
    location.SingleForecast =
        await this._weatherForecastService.GetCurrentWeatherAsync(
            location.Longitude,
            location.Latitude);
});
```

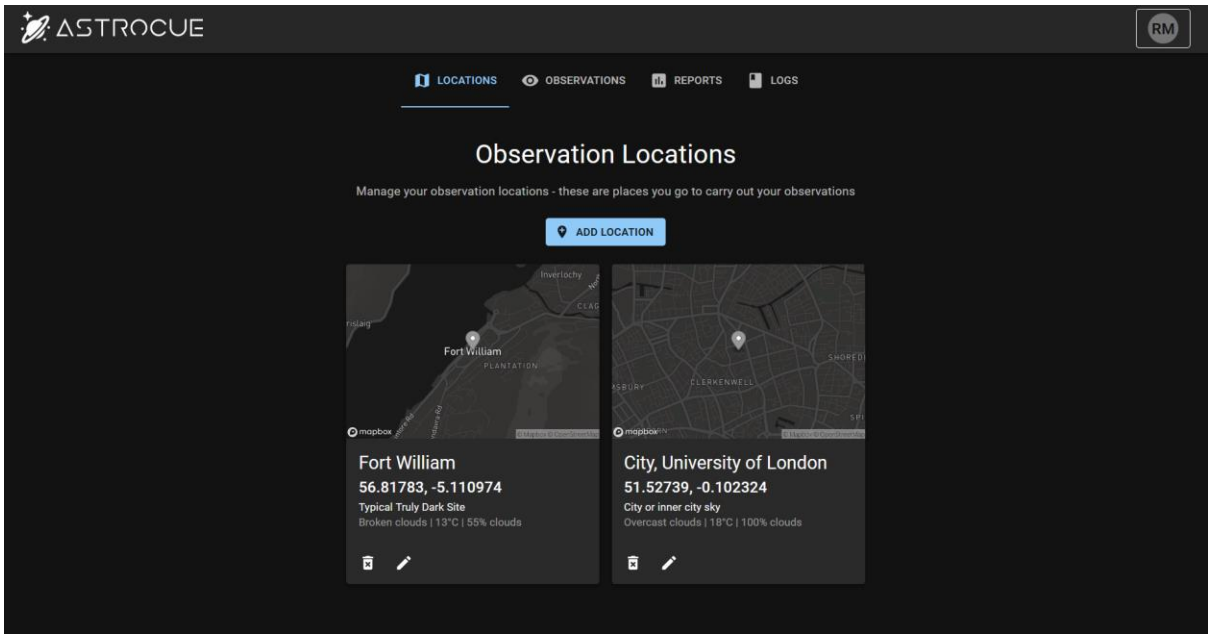
This pattern of controllers orchestrating what service layer to contact for data, and those services themselves often relying on other services, is a good example of MVC(S) from section 4.3.7.1.

The controller then serialises this data before passing it back to the client:


```
Headers  Cookies  Request  Response  Timings  Stack Trace  Security
Filter properties
JSON
▼ 0: Object { id: 1084, name: "Fort William", longitude: -5.110974, ... }
    id: 1084
    name: "Fort William"
    longitude: -5.110974
    latitude: 56.81783
    bortleScaleValue: 2
    bortleDesc: "Typical Truly Dark Site"
  ▼ singleForecast: Object { cloudCoveragePercent: 55, temperatureCelcius: 13, description: "Broken clouds", ... }
      cloudCoveragePercent: 55
      temperatureCelcius: 13
      description: "Broken clouds"
      retrievedAt: "2022-05-06T17:50:37"
▼ 1: Object { id: 1085, name: "City, University of London", longitude: -0.102324, ... }
    id: 1085
    name: "City, University of London"
    longitude: -0.102324
    latitude: 51.52739
    bortleScaleValue: 8
    bortleDesc: "City or inner city sky"
  ▼ singleForecast: Object { cloudCoveragePercent: 100, temperatureCelcius: 18, description: "Overcast clouds", ... }
      cloudCoveragePercent: 100
      temperatureCelcius: 18
      description: "Overcast clouds"
      retrievedAt: "2022-05-06T17:50:37"
```


Figure 12: JSON received by client (Firefox).

Once the client receives this data, it can display it:



5.4.1.1.1 DEPLOYMENT

Full deployment in Appendix F: Deployment and domain/DNS.

As the server was containerised, the author first deployed it to Docker Hub, directly from VS:

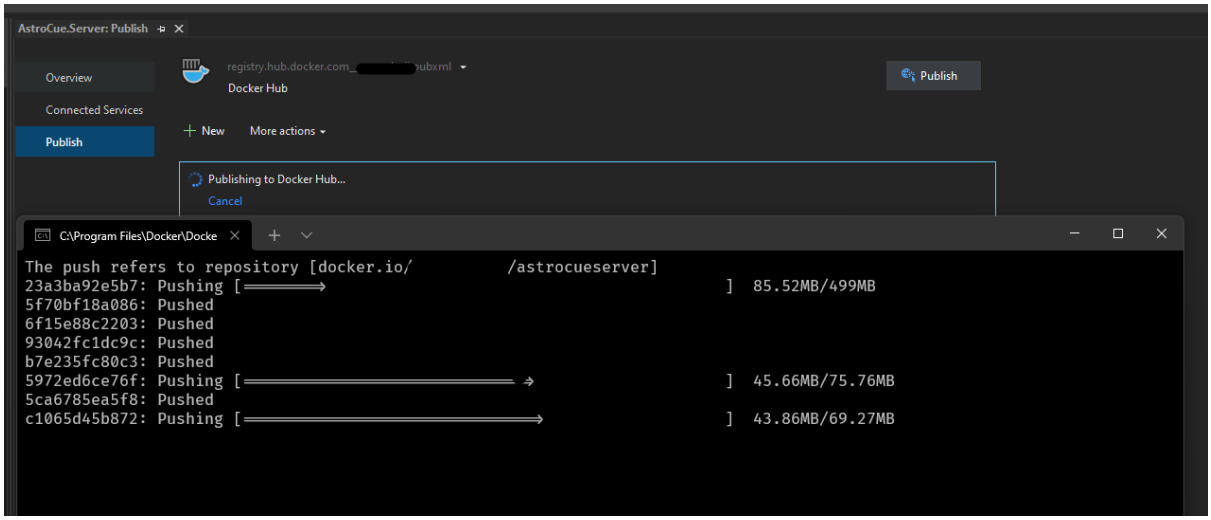


Figure 13: VS publishing to Docker Hub

In Azure, the author then created a new 'Web App for Containers' resource and directed it to pull the image from Docker Hub. A database was also required. Since the author had chosen MS SQL Server, an Azure SQL Database was created.

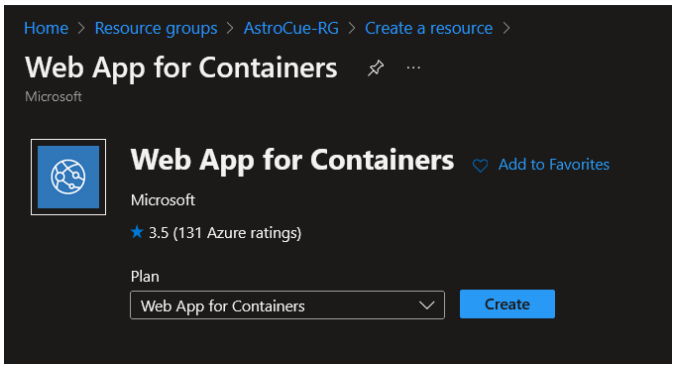


Figure 14: Web app

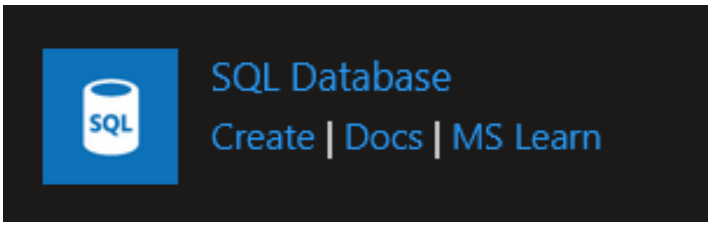
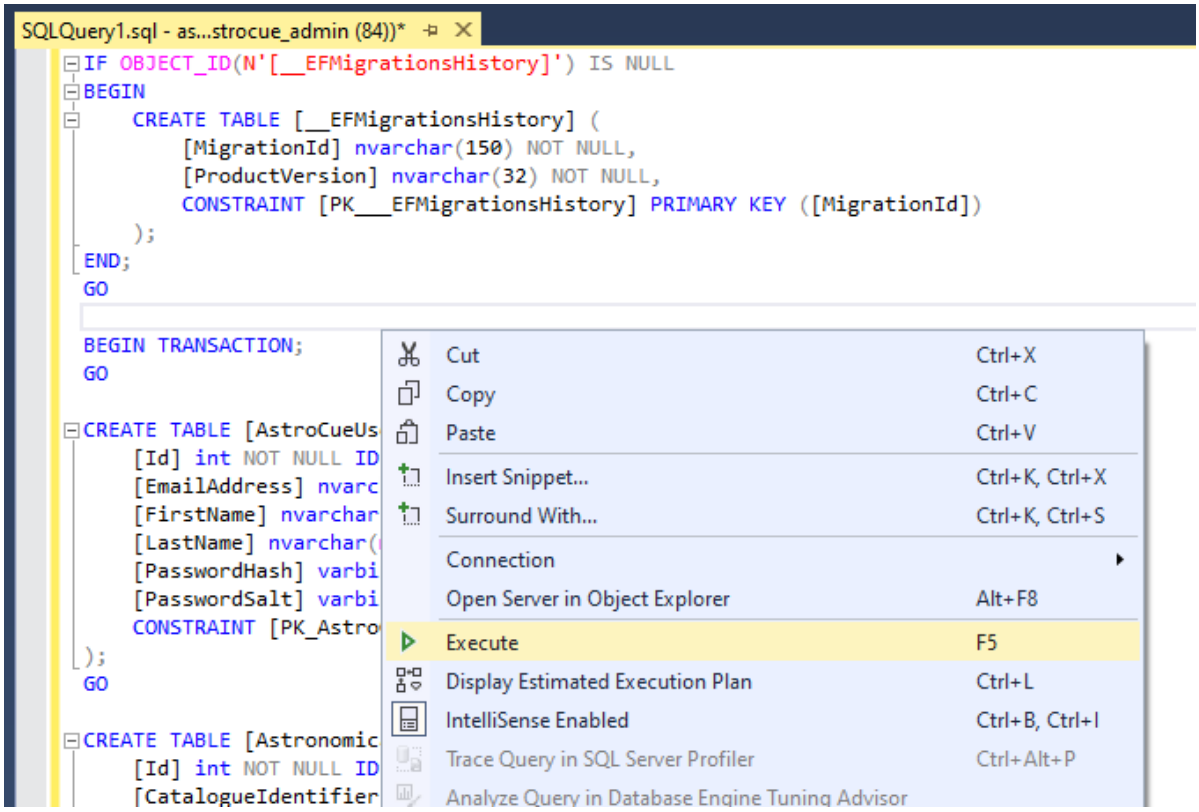


Figure 15: DB to be linked to web app

The database migrations then needed to be ran against this new database so that it had the correct tables and overall structure.

To do this, the author logged in to it remotely, and executed the script migration, which is an SQL version of the migrations created with EF Core. For the full script, see Appendix G: EF Core database migration script.



After all these steps had been completed, the server was rebooted. This caused it to carry out its database seeding and populate the astronomical object tables directly from the astronomical catalogues. It was then ready for traffic.

5.4.1.2 CLIENT OVERVIEW

The client's structure revolves around the top-level pages in the `pages` directory and the rest of the UI components and logic in the `src` directory:

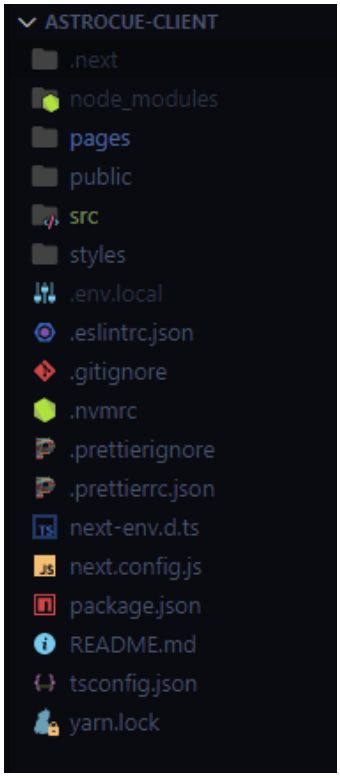


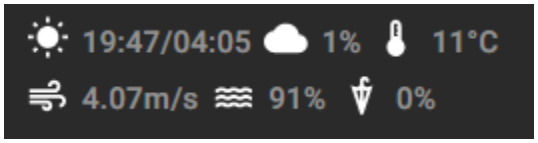
Figure 16: Client in VS Code

The client web application, written in React, comprised all the UI shown to users. As with most React applications, the UI is largely component based. This means that the UI is broken into individual reusable functional components that change what they display based on their provided data.

For example, the weather forecast component (shortened):

```
const Forecast = ({ ...props }: IForecastProps) => {
  return (
    <Stack direction='column' spacing={0.5}>
      <Stack direction='row' spacing={0.75}>
        <WbSunnyIcon sx={{ fontSize: iconSize }} />
        <Typography display='inline' variant='subtitle2' color='text.disabled'>
          {`${sunset}/${sunrise}`}
        </Typography>
        <CloudIcon sx={{ fontSize: iconSize }} />
        <Typography display='inline' variant='subtitle2' color='text.disabled'>
          {props.forecast?.cloudCoveragePercent}%
        </Typography>
        <ThermostatIcon sx={{ fontSize: iconSize }} />
        <Typography display='inline' variant='subtitle2' color='text.disabled'>
          {props.forecast?.temperatureCelcius}°C
        </Typography>
      </Stack>
    </Stack>
  );
};
```

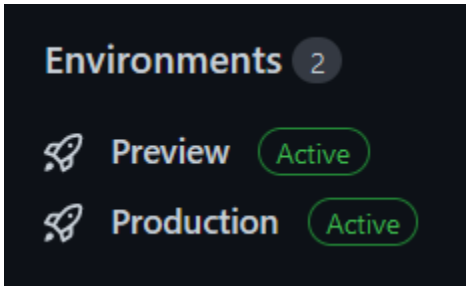
An interface defining a set of values is passed into the function. The UI tags inside that component can then access the values and display them:



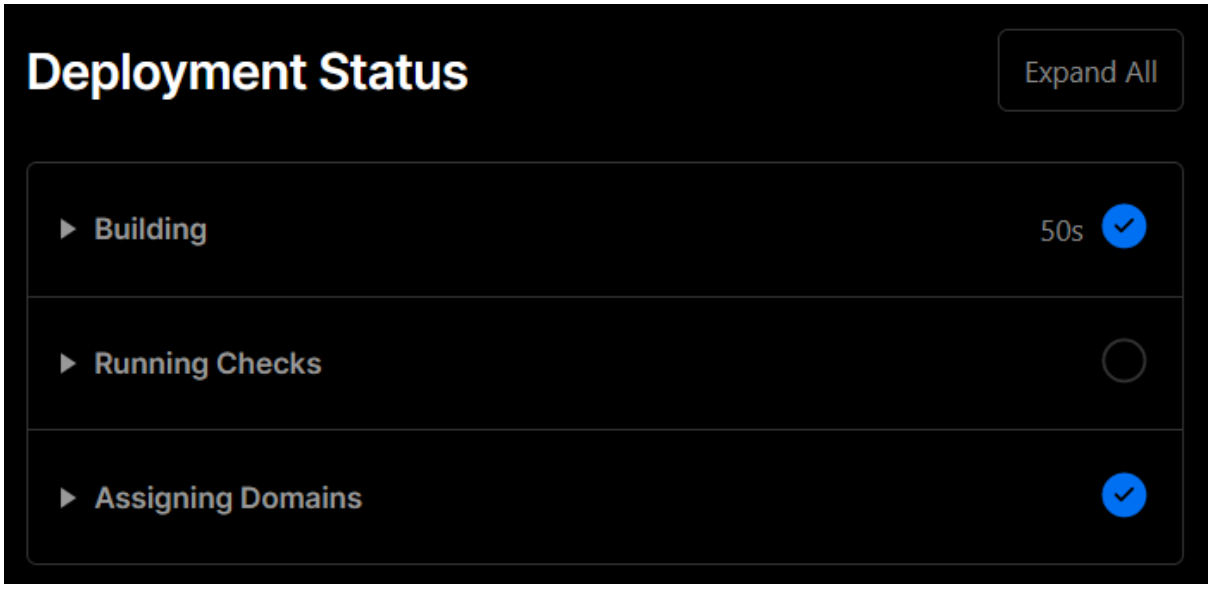
5.4.1.2.1 Deployment

Full deployment in Appendix F: Deployment and domain/DNS.

The client was deployed to Vercel. This occurred directly from GitHub. Inside Vercel, the author selected the client repository. Vercel then setup 2 deployment environments:



Because of these environments, the deployment is automated. Every time a commit is merged into the master branch, a new deployment is issued:



5.4.2 DEVELOPMENT ENVIRONMENT

5.4.2.1 NOTE ON TEMPLATE PROJECTS

Any template files that went on to be used will be documented in Appendix B: Reuse Summary.

5.4.2.2 VISUAL STUDIO

VS was used for the server development. As mentioned in section 4.3.2.1, the author started development from VS via a template project. This automatically generates boilerplate code, including project files and, importantly, a `Dockerfile`; a set of instructions to Docker that describe how to build the project code into an image that can be ran by the Docker Engine.

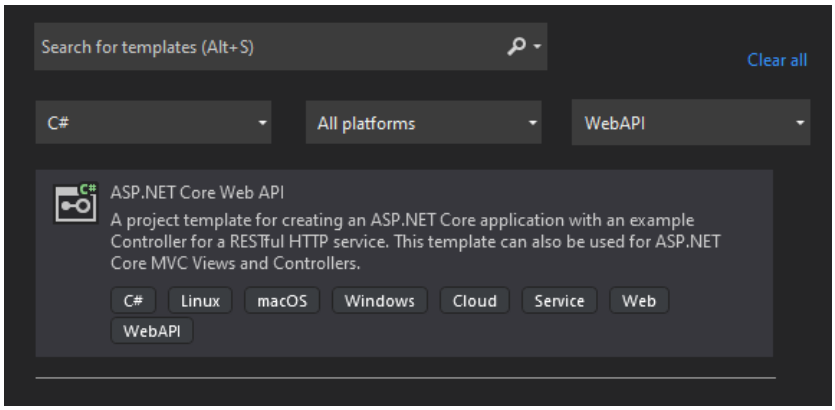


Figure 17: Web API template

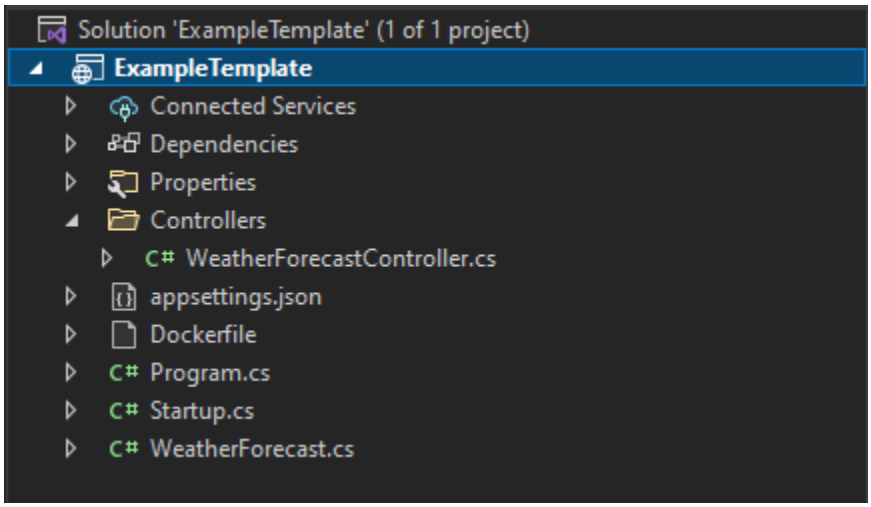


Figure 18: The template's files

5.4.2.3 VISUAL STUDIO CODE

For frontend development, VS Code was used.

The author made use of a Next.js template project. It was created via NPX, a command line tool used to execute NPM packages:

```
~$ npx create-next-app@latest --ts
```

Notice the `--ts` switch on the command used to enable TypeScript support. This command uses the create-next-app library.

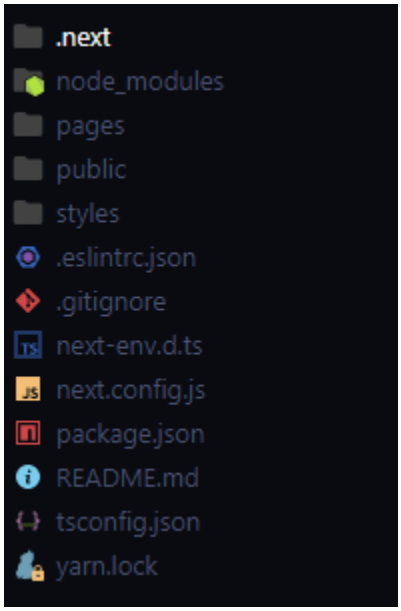


Figure 19: The template's files

5.4.3 APIs

5.4.3.1 Library

Communication with external APIs was implemented with a C# library called RestSharp. RestSharp provides an extremely succinct syntax for HTTP requests. It lets you create request objects, add parameters/files to them and then asynchronously execute and wait for the result.

5.4.3.2 OpenWeatherMap

OWM was used to gather all weather data used by AstroCue. Below, the code that retrieves a single current forecast can be seen:

```
RestRequest req = new(this._environmentManager.BaseOwmCurrentWeatherUrl);
req.AddParameter("lat", latitude);
req.AddParameter("lon", longitude);
req.AddParameter("units", "metric");
req.AddParameter("appid", this._environmentManager.OpenWeatherMapApiKey);
RestResponse res = await this._client.GetAsync(req);
```

This queries the OWM current weather endpoint using a pair of coordinates. You can see the units being specified, and the API key being inserted as a parameter so the request can be authenticated.

A full example of a OWM query can be seen in appendix 9.11.2.

5.4.3.3 MapBox

2 endpoints were used from MapBox: forward geocoding and static map images. Forward geocoding was used to allow users to search for locations when creating new observation locations, and the static map endpoint was used to retrieve map images that are displayed on the website and in the report emails:

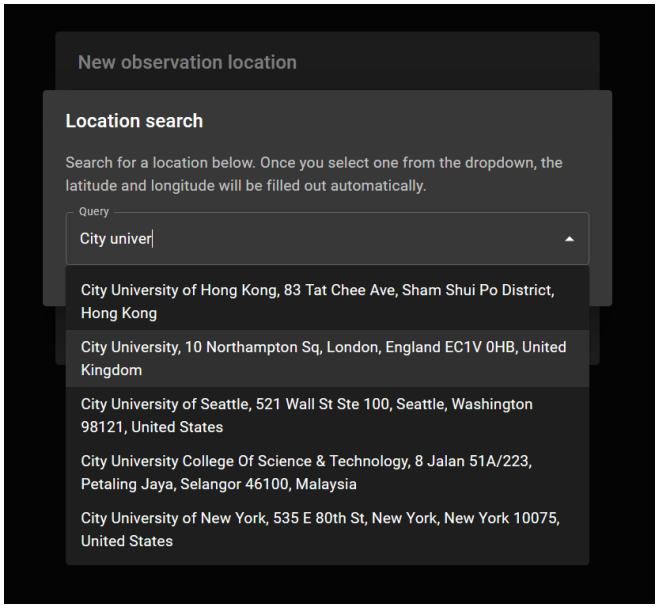


Figure 20: Forward geocoding

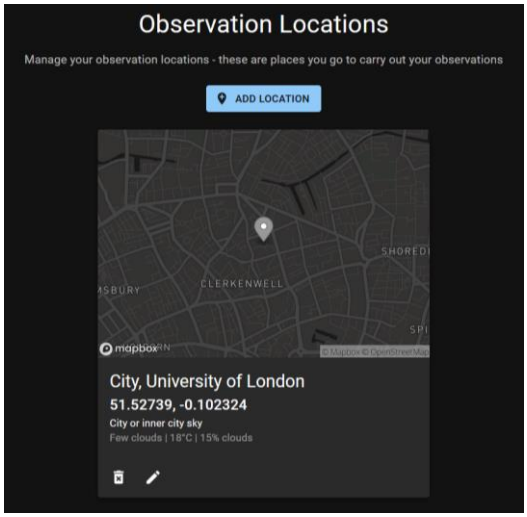


Figure 21: Static map on web

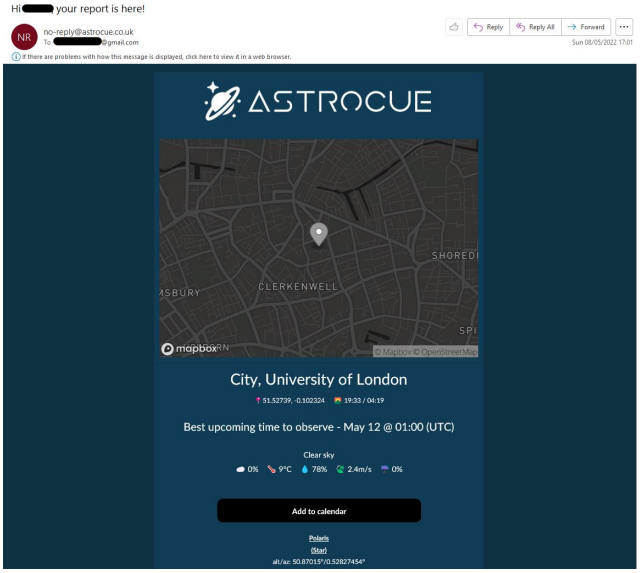


Figure 22: Static map in email

Again, RestSharp was used. Below is the method used to retrieve a static map. As this MapBox endpoint returns binary data rather than JSON, no parsing was needed. It was simply received as a byte array:

```
RestRequest req = new(this.GenStaticMapImageUrl(longitude, latitude));
req.AddParameter("access_token", this._environmentManager.MapBoxApiKey);
return await this._client.DownloadDataAsync(req);
```

As MapBox maps can be customised, a separate method was implemented that generated the URL based on the author's desired map appearance, which included a pin overlay and zoom level.

The integration of forward geocoding required query validation, as MapBox imposes some limits on query lengths and characters. The author checked against both, returning errors to the user if their query was invalid. This ensured that all queries were valid before they were even executed:

```
if (query.Split(' ').Length > 20)
{
    throw new ArgumentOutOfRangeException(
        nameof(query),
        "Query needs to be no more than 20 words");
}

if (query.Contains(';'))
{
    throw new ArgumentException("Query contains illegal character ';'",
nameof(query));
}
```

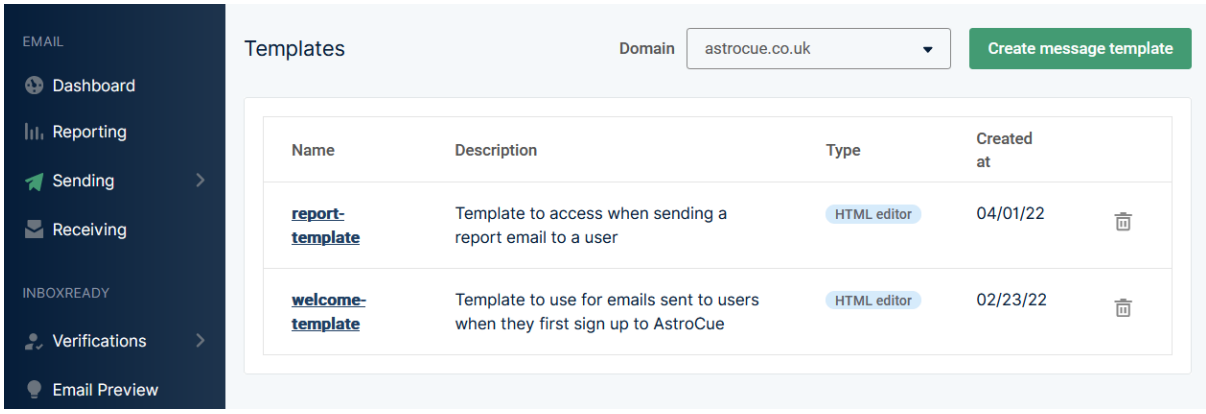
5.4.3.4 MAILGUN

Emails were sent to users using the MailGun API. Rather than forcing the use of SMTP (or other traditional email protocols), MailGun allows communication via HTTP. There were two kinds of emails sent by AstroCue: welcome emails when a user first signed up, and report emails every ~3 days, or on demand.

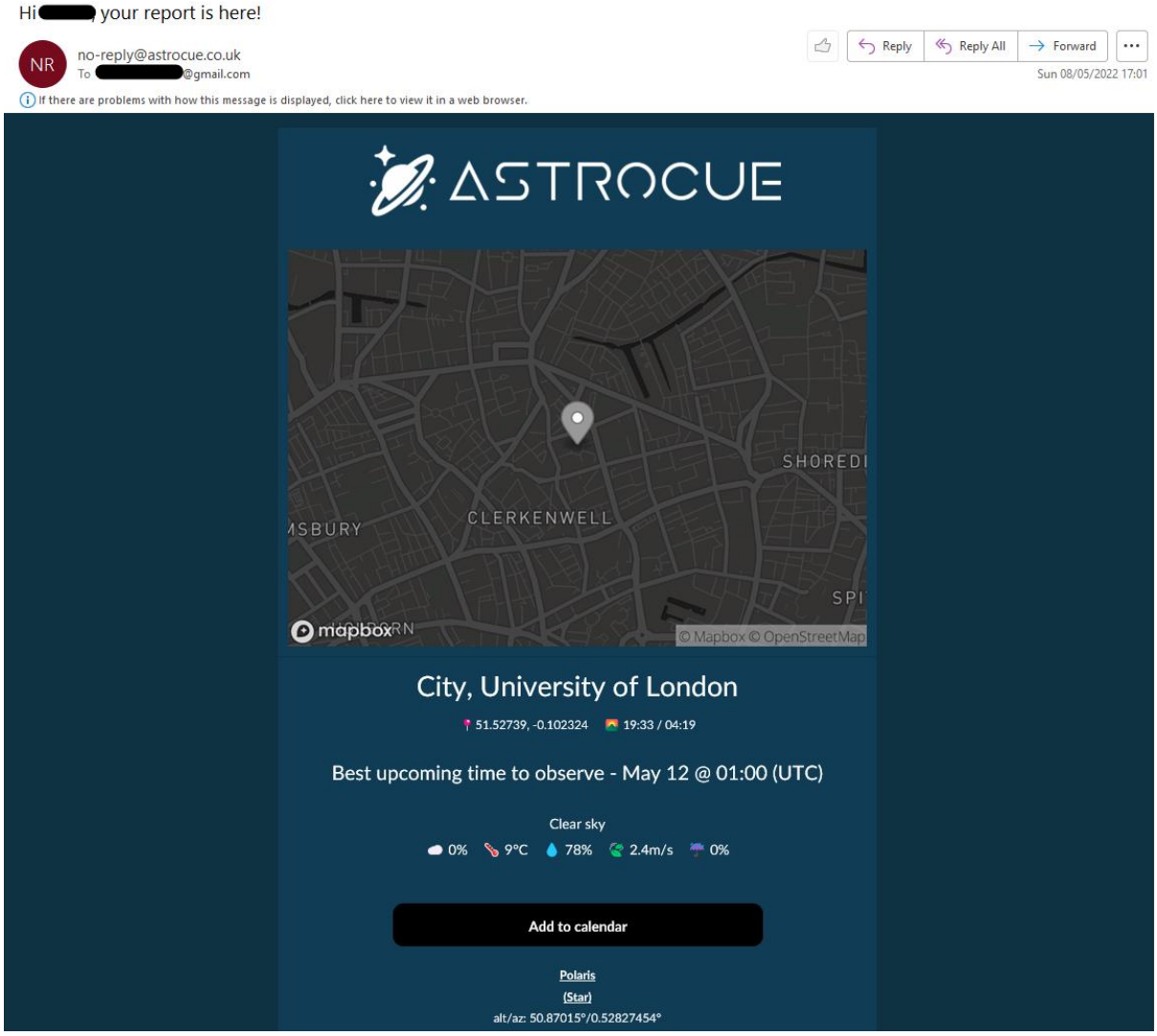
When sending a request to MailGun, the `template` parameter points to an email template present on the MailGun account the API key belongs to. For example, here, the author is using the welcome email template (covered later in section 5.4.4). You can also see the email target address and subject being set as regular HTTP parameters:

```
RestRequest req = new();
req.AddParameter("domain", "astrocue.co.uk", ParameterType.UrlSegment);
req.Resource = "{domain}/messages";
req.AddParameter("from", "AstroCue <no-reply@astrocue.co.uk>");
req.AddParameter("to", user.EmailAddress);
req.AddParameter("subject", $"Welcome, {user.FirstName}!");
req.AddParameter("template", WelcomeEmailTemplate);
req.AddParameter("h:X-Mailgun-Variables", templateData);
```

On the MailGun account, templates were uploaded and listed under 'Sending':



Some emails contained inline attachments, such as static map images used in report emails:



To add these, the author appended their binary data to the RestSharp request as an inline file, along with a key denoting a file name:

```
req.AddFile("inline", value, key);
```

Adding the file inline allowed access to it from within the contents of the email itself. If it was a regular attachment, it would be added as a separate downloadable file.

5.4.4 EMAIL TEMPLATE REUSE

The two emails sent by AstroCue (welcome and report emails) are both based on a free template called "Game Activation" (Unlayer, 2022).

Before editing, the template looked as follows:

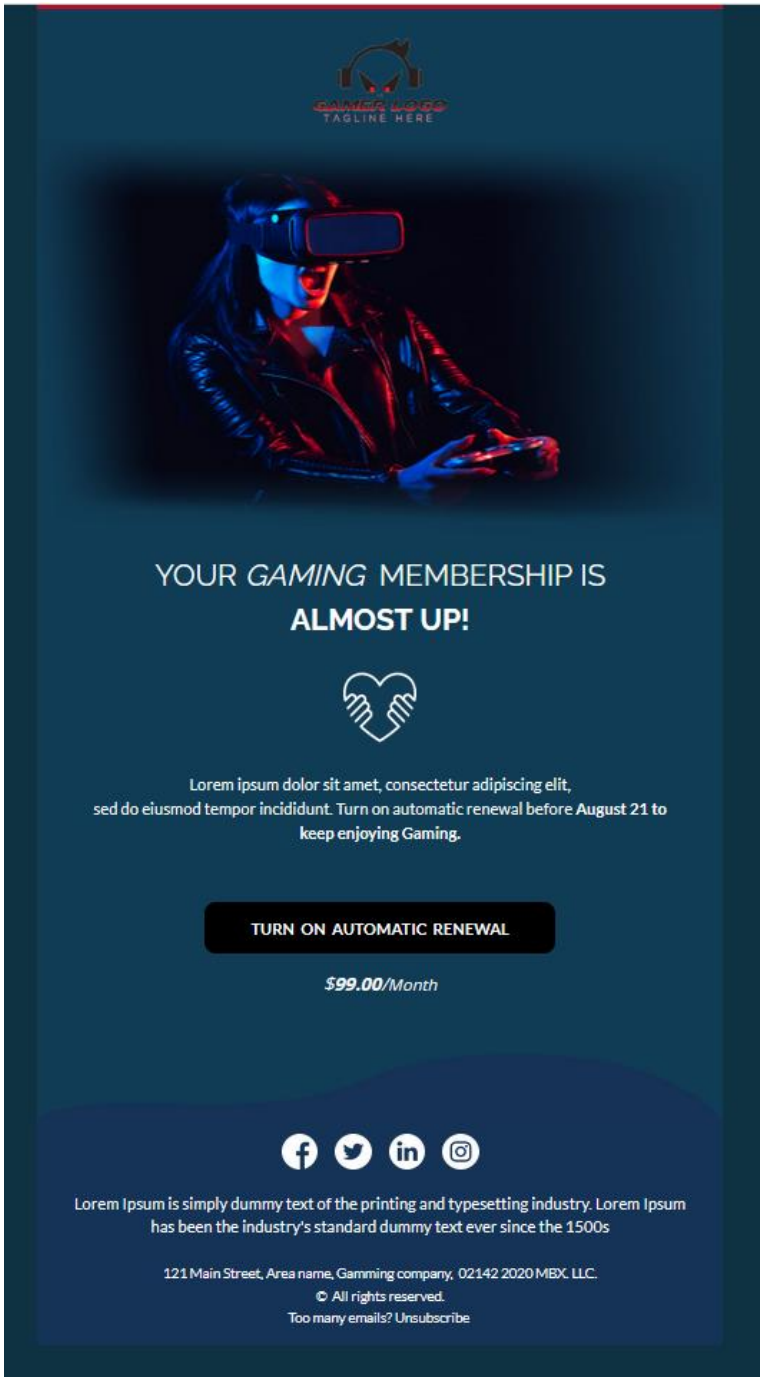
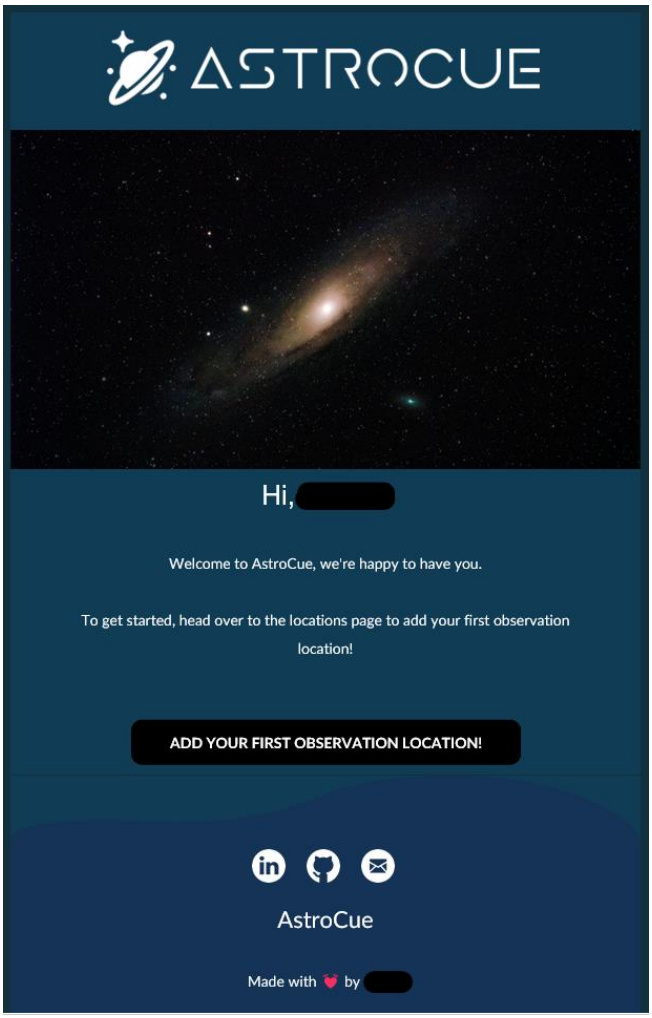
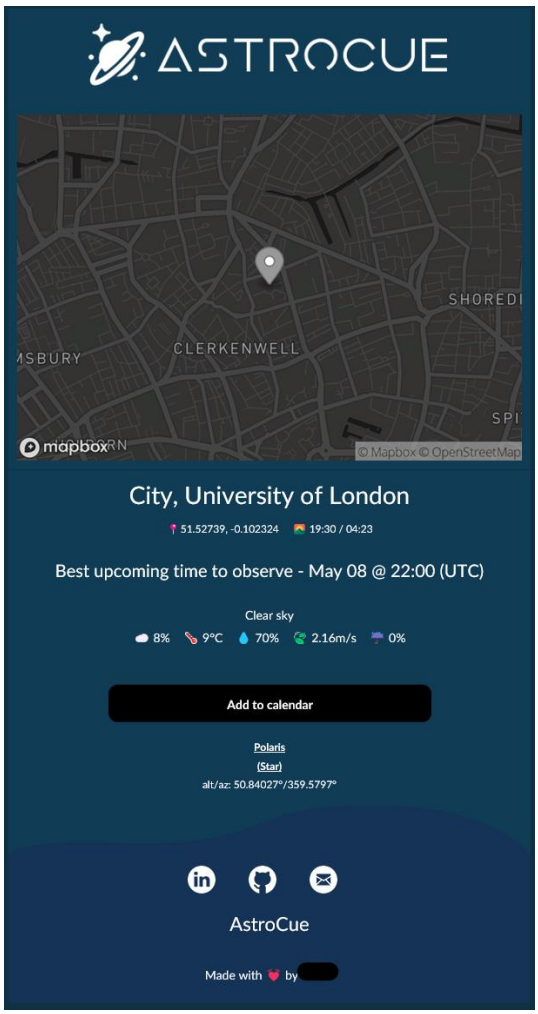


Figure 23: Template (Unlayer, 2022).

AstroCue's emails were then built on this template:





The left email was sent to newly registered users, and the right email was the report email – sent to users to inform them of the best observation opportunities.

One thing that stands out about these emails is that they are dynamic; the content differs depending on who they are being sent to. This was achieved with Handlebars.

5.4.4.1 HANDLEBARS

Handlebars is a templating language that allows you to embed dynamic values into HTML. For example, take the Handlebars template:

```
<p>{{value1}} {{value2}}</p>
```

And the template values:

```
{
    value1: "something",
    value2: "interesting"
}
```

The resulting HTML, once executed with Handlebars, will be:

```
<p>something interesting</p>
```

When specifying a template for MailGun (the `template` parameter), you can also pass a set of template values to populate it with. This is how AstroCue generated its dynamic emails - pointing MailGun at one of its templates and providing templating data. This can be seen if we take another look at the welcome email method. You can see the template data being generated by serialising an object for a particular user, and then it being attached to the request beneath where the template was specified.

```
string templateData =
JsonConvert.SerializeObject(this._mapper.Map<WelcomeEmailModel>(user));
req.AddParameter("template", WelcomeEmailTemplate);
req.AddParameter("h:X-Mailgun-Variables", templateData);
```

The author's modifications to the original template involved adding templating code where necessary. These changes can be seen in the reuse summary.

A full MailGun request can be seen in appendix 9.11.3.

5.4.5 O/RM

Entity Framework Core was used to interface with the database. There were 3 main steps to working with EF Core:

1. Create entities (the models from MVC)
2. Create a database context class that holds collections of entities
3. Create and execute database migrations

At the beginning of development, the author started by creating the required entities to represent AstroCue's data. As an example, below is the entity representing a user's observation:

```
public class Observation
{
    public int Id { get; set; }

    public int ObservationLocationId { get; set; }

    public ObservationLocation ObservationLocation { get; set; }

    public int AstronomicalObjectId { get; set; }

    public AstronomicalObject AstronomicalObject { get; set; }
}
```

Once models were created, sets of them were added as properties on the DB context class. For example:

```
public DbSet<Observation> Observations { get; set; }
```

Any many-to-many relationships had to be configured manually in the DB context class. The author needed to do this for the relationship between observation locations and astronomical objects to form the observations table as referenced by the entity above. This is done by overriding the `OnModelCreating` method:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Observation>()
        .HasOne(l => l.ObservationLocation)
        .WithMany(a => a.Observations)
        .HasForeignKey(l => l.ObservationLocationId);

    modelBuilder.Entity<Observation>()
        .HasOne(l => l.AstronomicalObject)
        .WithMany(a => a.Observations)
        .HasForeignKey(l => l.AstronomicalObjectId);
}
```

To update the local database to match the models and context, two commands are ran from the VS Package Manager Console.

Firstly, we create a new migration with a name:

```
Package Manager Console
Package source: All
PM> Add-Migration AddedNewModels
```

And then we apply any non-applied migrations to the database:

```
Package Manager Console
Package source: All
PM> Update-Database
```

This creates new tables based on added entities, or updates existing tables based on changes to entities that were already migrated.

5.4.5.1 PRODUCTION ERD

Below is an ERD of the tables in the production database that were created directly by EF Core migrations:

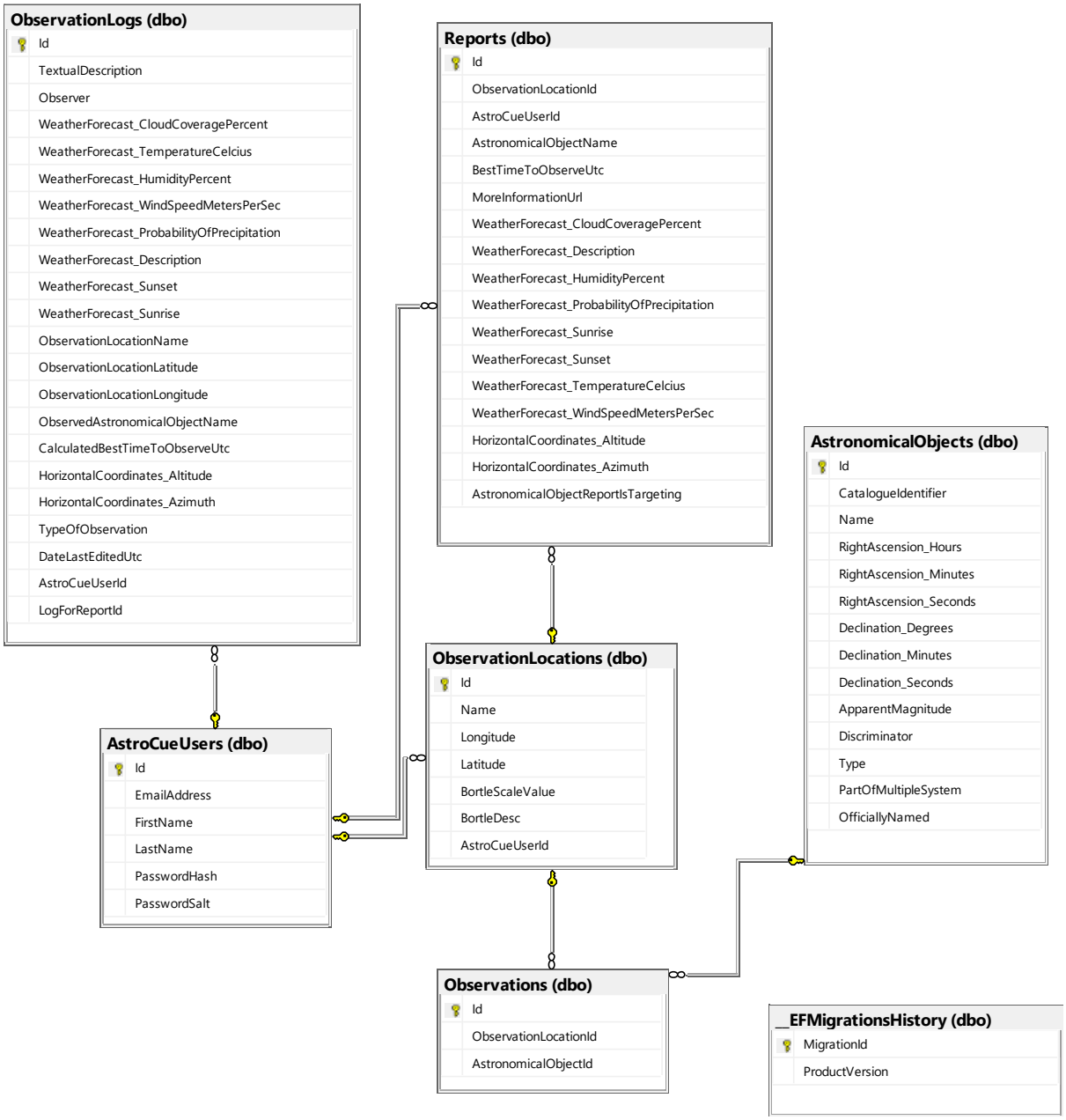


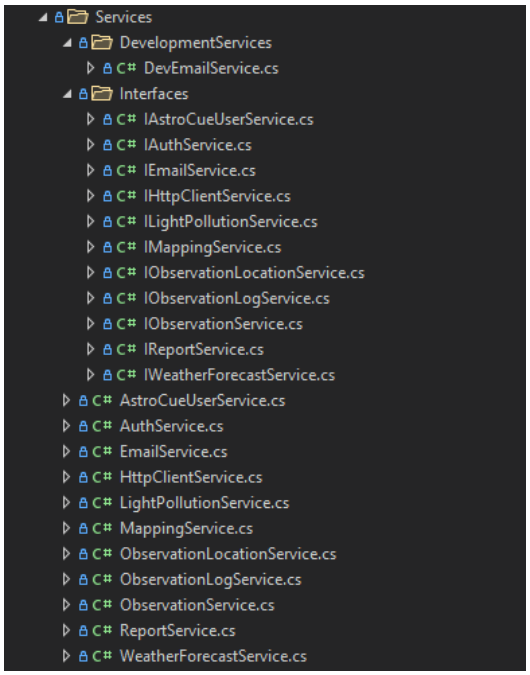
Figure 24: Production ERD. Created with MS SQL Server Management Studio

5.4.6 DESIGN PATTERNS

5.4.6.1 MVC(S)

As mentioned in section 4.3.7.1, the author deviated from MVC by adding a service layer – becoming MVC(S). As models (entities) and controllers have already been covered, we can explore services.

In VS, the `Services` folder can be seen to contain all the service classes:



Each service class has its operations represented via interfaces. For example, inside the `IWeatherForecastService` interface, there are 2 method signatures:

```
Task<SingleForecast> GetCurrentWeatherAsync(float longitude, float latitude);

Task<FourDayForecastReport> GetForecastNextFourDaysAsync(float longitude, float
latitude);
```

Implementations for these two methods can be found in the corresponding `WeatherForecastService` class, which itself implements the interface:

```
public class WeatherForecastService : IWeatherForecastService
```

5.4.6.2 DEPENDENCY INJECTION

Once a service has been created, it needs to be 'registered' so that other classes are able to use it. Service registration is done in `Startup.cs`:

```
services.AddScoped<IWeatherForecastService, WeatherForecastService>();
```

This registers the concrete weather service against its interface. The 'injection' can now occur. Any class that depends on this service by putting the interface in its constructor's parameters can be said to be depending on it. It is then injected where required.

For example, here is another service with `IWeatherForecastService` in its constructor:

```
public ObservationLocationService(
    ApplicationDbContext context,
    IMapper mapper,
    IWeatherForecastService weatherForecastService,
    IMappingService mappingService,
    ILightPollutionService lightPollutionService)
{
```

When this class is instantiated, the DI container will first look at all the interface parameters, then resolve their concrete implementations via the service registrations such that they can be 'injected' into the class for it to use.

Interfaces can also be implemented by different concrete classes. This was used by the author to register a development (dev) version of the email service. Rather than the implementation of this service contacting the MailGun API, it simply printed the action instead. This prevented the author from spamming his own inbox during testing.

The dev service registration was handled by an environment variable that was either true or false, and could be changed at will via VS:

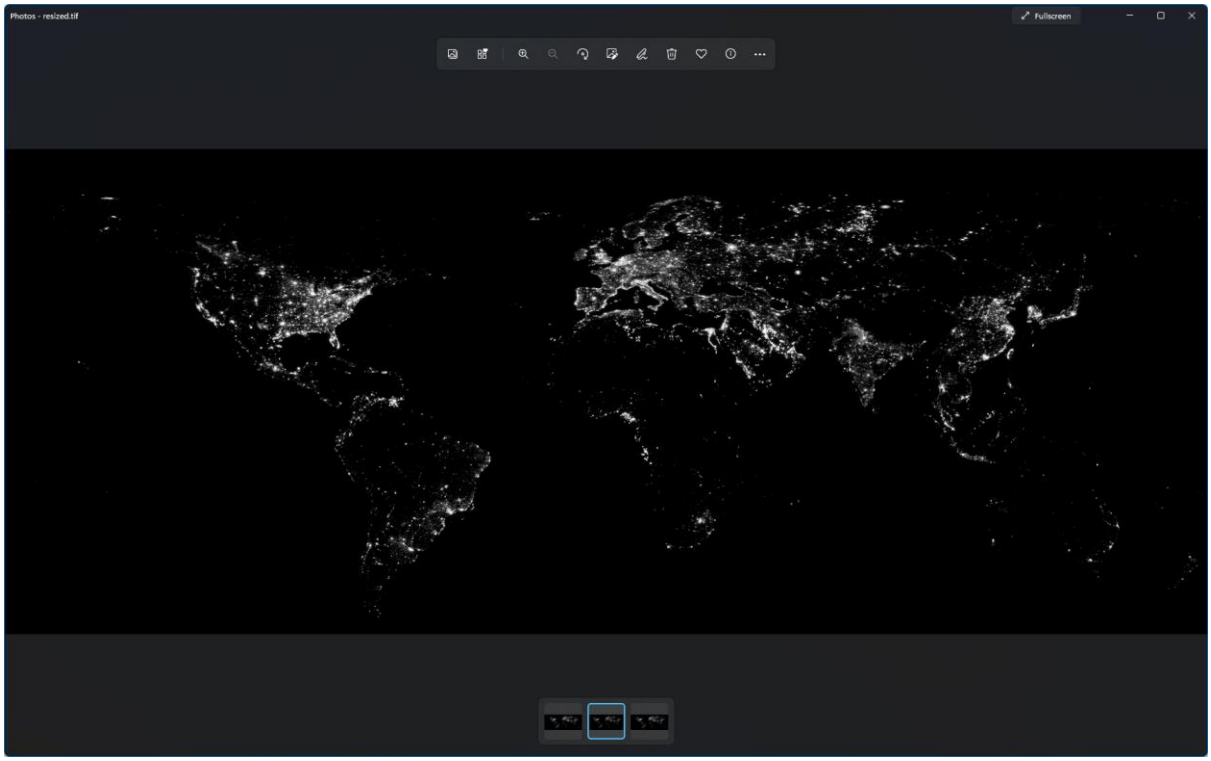
```
if (this._environmentManager.RegisterDevServices)
{
    services.AddScoped<IEmailService, DevEmailService>();
}
else
{
    services.AddScoped<IEmailService, EmailService>();
}
```

5.4.7 LIGHT POLLUTION DATA

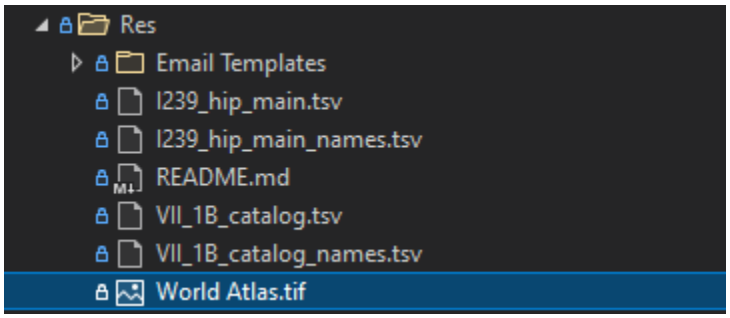
The light pollution data used by AstroCue came from the GeoTIFF file that resulted from the large-scale study mentioned in section 3.3.2 (Falchi, et al., 2016b). In its raw form, this file is 2.8GB (43200 x 17406 pixels) – far too unwieldy to work with.

To begin, the author used GDAL (Geospatial Data Abstraction Library); software specifically designed to work with geospatial data formats (GDAL/OGR contributors, 2022a). One of the GDAL tools, `gdalwarp`, was used to resample the GeoTIFF down to a sixth of its original size (7200 x 2901):

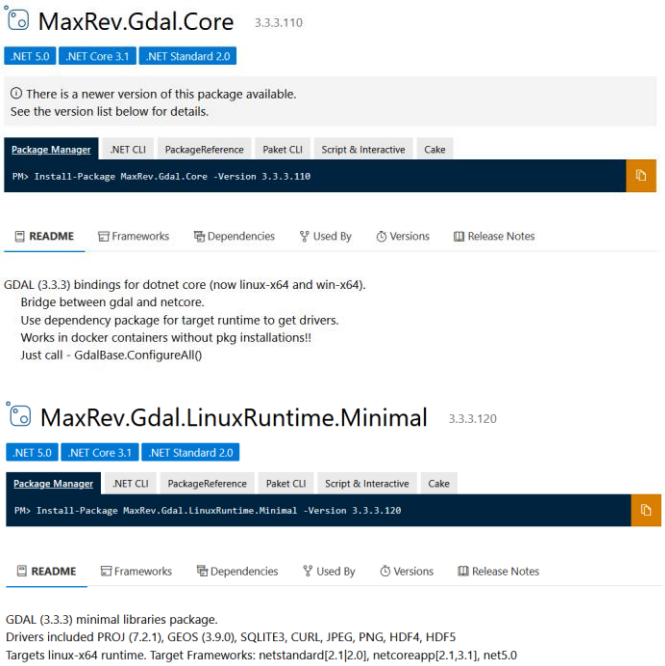
```
~$ gdalwarp -ts 7200 2901 -r near -of GTiff ./World_Atlas_2015.tif ./resized.tif
```



The file, now ~80mb, was placed in the `Res` folder in the server:



To access the light pollution data from code, GDAL bindings were needed such that the author could call the native GDAL binaries from code. To achieve this, the following libraries were used:



These provide both the GDAL binaries, as well as the .NET bindings to access them from C#. To begin, the author created the `LightPollutionService` class. The constructor of this class was used to initialise GDAL and load the GeoTIFF. As part of the constructor, GDAL was used to extract a geo transform matrix from the dataset:

```
this._geoTransformMatrix = new double[6];
this._worldAtlasDataset.GetGeoTransform(this._geoTransformMatrix);
```

The full constructor with error checking can be seen in appendix 9.11.4.

The geo transform matrix is an array of coefficients that allows mapping between georeferenced space and pixels (GDAL/OGR contributors, 2022b). For example, if you have coordinates (51.5953, 0.0653) and want to read the corresponding pixel value from the GeoTIFF, they need to be mapped first.

This mapping was implemented as follows:

```
int xOffset = (int)((longitude - this._geoTransformMatrix[0]) /
this._geoTransformMatrix[1]);
int yOffset = (int)((latitude - this._geoTransformMatrix[3]) /
this._geoTransformMatrix[5]);
```

This was combined with a bounds check to ensure that the result was inside the dataset. Full methods can be seen in appendix 9.11.5.

To work out which coefficients needed to be used, the author consulted the *Geographic Information Systems* Stack Exchange, and found a post attempting a similar operation which was then adapted for the needs of AstroCue (Georg, 2017).

The GeoTIFF's unit of brightness was millicandela/m^2 (mcd/m^2). The author needed a way to relate this unit to the dimming effect on objects in the sky. In other words: *at a given mcd/m^2 level, what is the dimmest object visible to an observer*? This was needed to provide users feedback on observations they were creating (i.e., warning users if a given object is likely to be too dim given their local light pollution levels).

5.4.7.1 THE BORTLE SCALE

In 2006, astronomer John Bortle created a scale based on almost 50 years of his own personal observation experience. Its purpose was to provide a method of relating light pollution levels to the effect on astronomy (Bortle, 2006). The scale features 9 levels, ranging from a class 1 (excellent dark sky site) to a class 9 (inner city sky).

Each level has a different naked-eye limiting magnitude (NELM): the dimmest a visible object could be to the naked-eye (Bortle, 2006). Because celestial apparent magnitudes work on a reverse logarithmic scale (i.e., a magnitude 1 object is brighter than a magnitude 10 object), the brighter the sky, the lower the NELM.

However, the Bortle scale provides no mcd/m^2 → Bortle conversion. For this, the author used a nomogram created by the Dark Skies Awareness project for an approximate conversion:

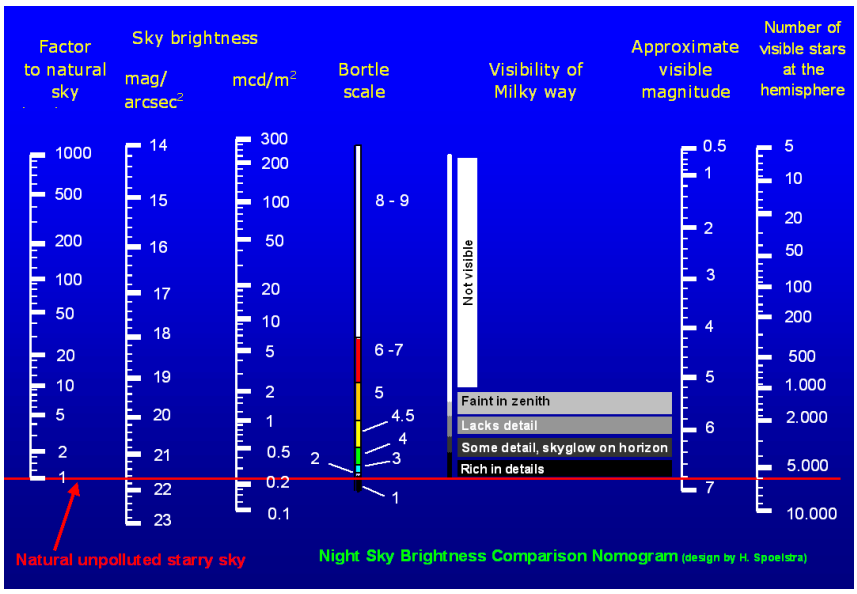


Figure 25: Sky Brightness Nomogram (Spoelstra, 2009).

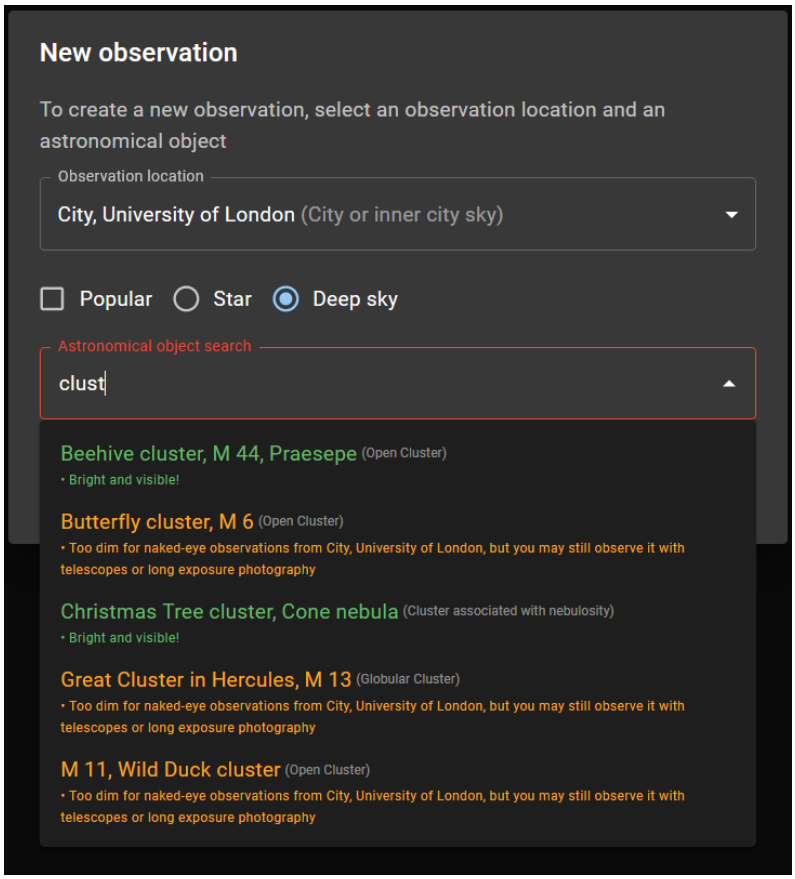
This was then converted to code. Following the nomogram, class 8 and 9 were combined:

```
public static int McdM2ToBortle(float mcdM2)
{
    return mcdM2 switch
    {
        <= 0.25f => 1,
        > 0.25f and <= 0.275f => 2,
        > 0.275f and <= 0.325f => 3,
        > 0.325f and <= 0.5f => 4,
        > 0.5f and <= 2.5f => 5,
        > 2.5f and <= 4f => 6,
        > 4f and <= 6.85f => 7,
        _ => 8
    };
}
```

Another method for converting the scale values to their descriptions was created (Bortle, 2006). This can be seen in appendix 9.11.6.

A good example of the Bortle scale's use in AstroCue was the new observation dialog. When a user was searching for an object, they would be warned if the object was likely too dim. This was done using a direct comparison of their location's NELM and the target's apparent magnitude:

This warning was presented in the UI as follows:



5.4.8 RATING WEATHER CONDITIONS

As mentioned in the literature review section 3.2.1, inspiration could be drawn from Clear Sky Chart regarding rating weather conditions for astronomical observations.

All the weather elements considered (inc. cloud coverage, wind speed, probability of precipitation etc…) are all better for observing when they are lower. This makes ratings easier than it initially seems. For each forecast being compared, simply sum these values to create an index, and then take the lowest of these.

During testing the author realised that cloud coverage was often higher than anticipated in calculations, so to combat this, its weighting was doubled. As this is the only metric that can **completely** obscure the sky (the rest, at worst, only hinder visibility), this led to much more acceptable ratings.

The index was calculated as follows:

```
float index = 0f;

index += hourlyForecast.CloudCoveragePercent * 2.0f;
index += hourlyForecast.WindSpeedMetersPerSec;
index += hourlyForecast.ProbabilityOfPrecipitation;
index += hourlyForecast.HumidityPercent;

return index;
```

The code calculating the lowest from a set of indexes can be seen in appendix 9.11.8.

5.4.9 VERSION CONTROL

The author made use of two separate GitHub repositories; one for the frontend and one for the backend. This allowed the code itself, along with the tracking of pull requests and issues to remain separate.

5.4.9.1 REPOSITORY IMAGES

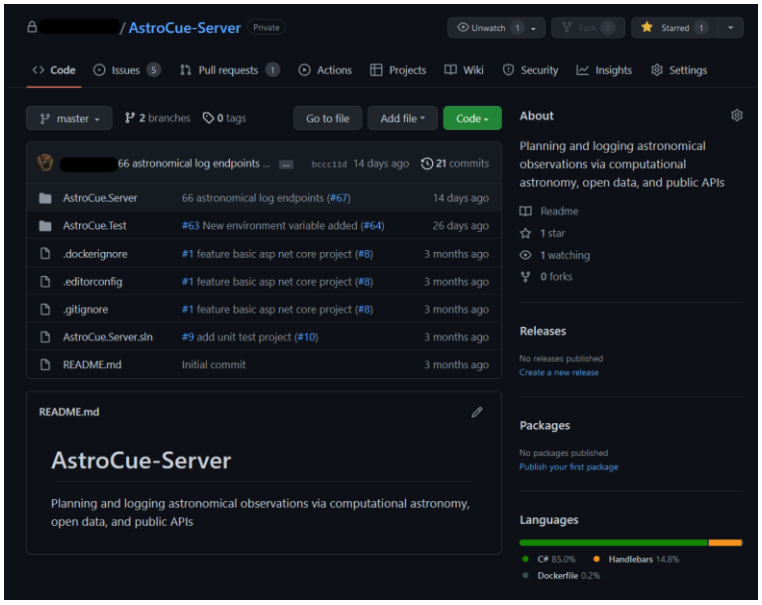


Figure 26: Server repo

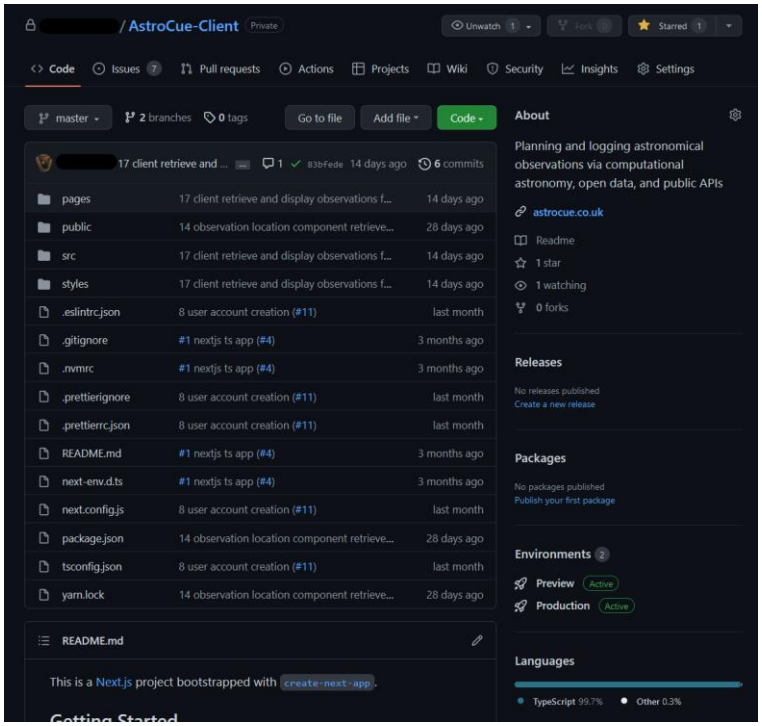
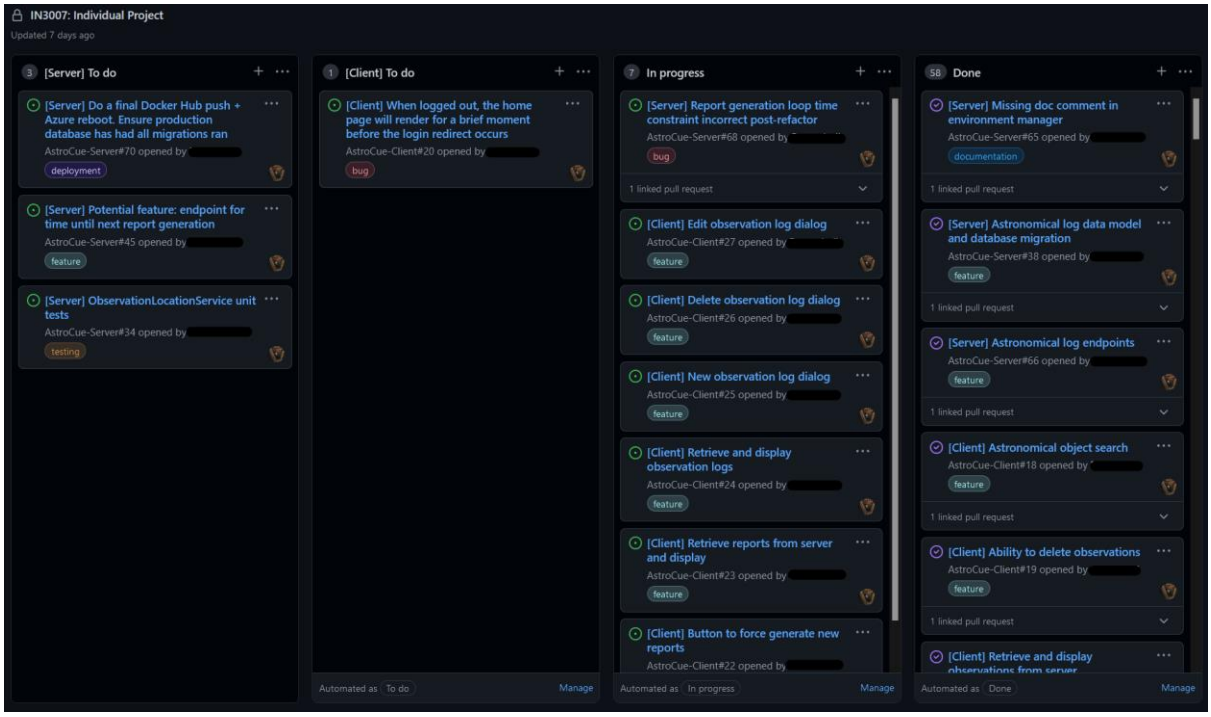


Figure 27: Client repo

5.4.9.2 GitHub Projects

As mentioned in section 4.3.8, the author used GitHub Projects to track tasks across both repositories. Once an issue had been created, it could be assigned directly to either repository. This board was automated; when pull requests were merged, the affected tasks would automatically be moved to the 'Done' section.



5.4.10 ASTRONOMICAL CATALOGUES

Sections 3.3.1 and 4.3.9 covered the use of VizieR as a source of astronomical catalogues. When it came to exporting these catalogues, the author used a certain set of export options to reduce the file size and to ensure the files were machine readable.

To find catalogues, the author started at the VizieR home page, and searched their names:

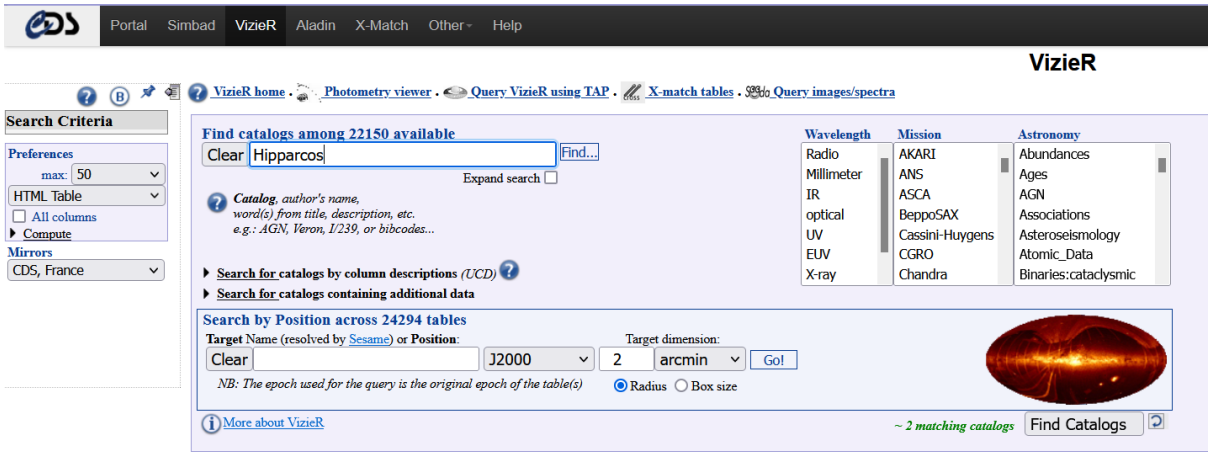


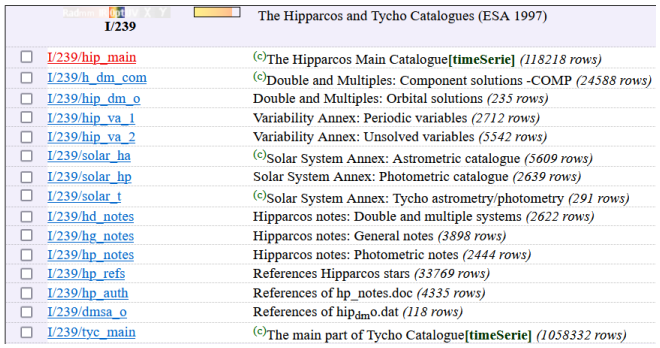
Figure 28: VizieR search

In each catalogue, the author selected a set of preferences that controlled what data was exported and in what format.

*A **magnitude** column, where present, refers to apparent magnitude – a value that denotes how visibly bright an object appears when viewed from the surface of Earth.*

5.4.10.1 HIPPARCOS CATALOGUE (EUROPEAN SPACE AGENCY, 1997)

118,218 with VizieR name `I/239/hip_main`:



I/239	The Hipparcos and Tycho Catalogues (ESA 1997)
I/239/hip_main	(c)The Hipparcos Main Catalogue[timeSerie] *(118218 rows)*
I/239/h_dm_com	(c)Double and Multiples: Component solutions -COMP *(24588 rows)*
I/239/hip_dm_o	Double and Multiples: Orbital solutions *(235 rows)*
I/239/hip_va_1	Variability Annex: Periodic variables *(2712 rows)*
I/239/hip_va_2	Variability Annex: Unsolved variables *(5542 rows)*
I/239/solar_ha	(c)Solar System Annex: Astrometric catalogue *(5609 rows)*
I/239/solar_hp	Solar System Annex: Photometric catalogue *(2639 rows)*
I/239/solar_t	(c)Solar System Annex: Tycho astrometry/photometry *(291 rows)*
I/239/hd_notes	Hipparcos notes: Double and multiple systems *(2622 rows)*
I/239/hg_notes	Hipparcos notes: General notes *(3898 rows)*
I/239/hp_notes	Hipparcos notes: Photometric notes *(2444 rows)*
I/239/hp_refs	References Hipparcos stars *(33769 rows)*
I/239/hp_auth	References of hp_notes.doc *(4335 rows)*
I/239/dmsa_o	References of hip_{dm}o.dat *(118 rows)*
I/239/tyc_main	(c)The main part of Tycho Catalogue[timeSerie] *(1058332 rows)*

5.4.10.1.1 EXPORT

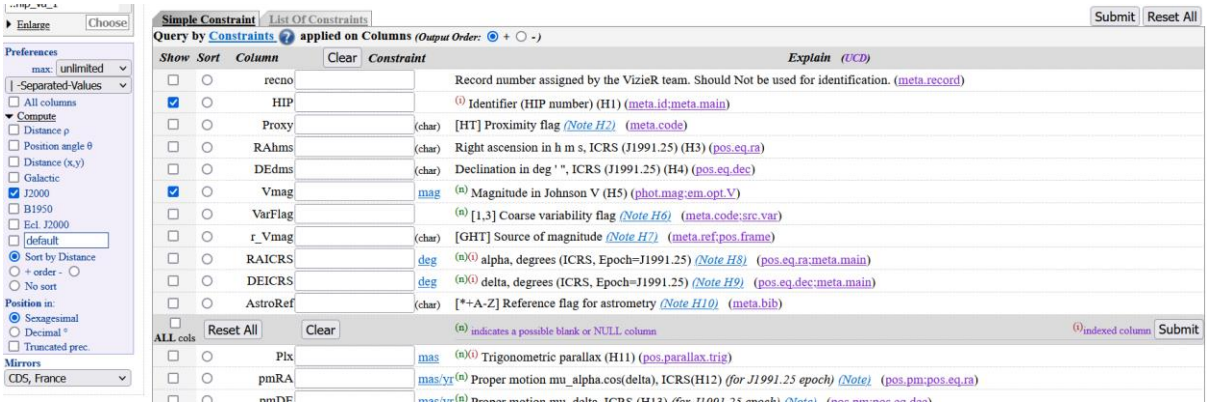
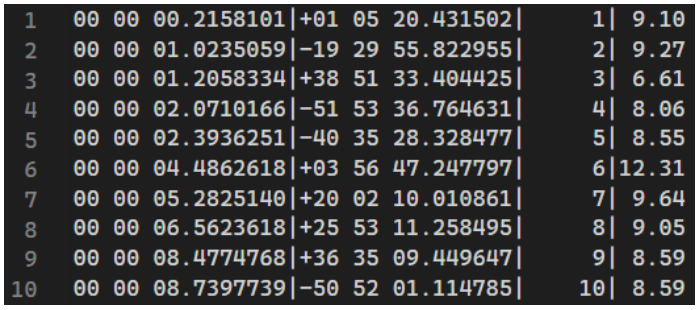


Figure 29: Export settings

5.4.10.1.2 FILE



1	00 00 00.2158101	+01 05 20.431502	1	9.10
2	00 00 01.0235059	-19 29 55.822955	2	9.27
3	00 00 01.2058334	+38 51 33.404425	3	6.61
4	00 00 02.0710166	-51 53 36.764631	4	8.06
5	00 00 02.3936251	-40 35 28.328477	5	8.55
6	00 00 04.4862618	+03 56 47.247797	6	12.31
7	00 00 05.2825140	+20 02 10.010861	7	9.64
8	00 00 06.5623618	+25 53 11.258495	8	9.05
9	00 00 08.4774768	+36 35 09.449647	9	8.59
10	00 00 08.7397739	-50 52 01.114785	10	8.59

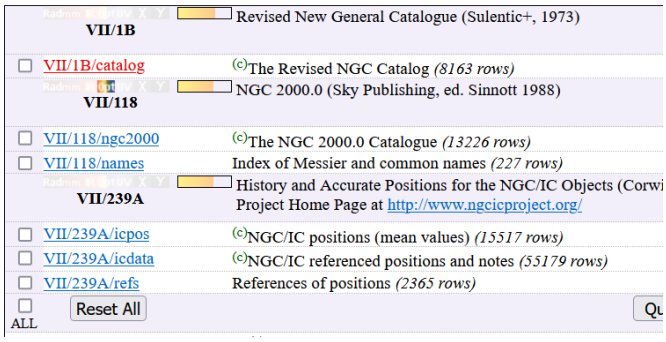
Figure 30: First 10 rows

```
#Column _RAJ2000        (A16)   Right ascension (FK5, Equinox=J2000.0) at Epoch=J2000, proper motions taken into account        [ucd=pos.eq.ra]
#Column _DEJ2000        (A16)   Declination (FK5, Equinox=J2000.0) at Epoch=J2000, proper motions taken into account    [ucd=pos.eq.dec]
#Column HIP     (I6)    Identifier (HIP number) (H1)    [ucd=meta.id;meta.main]
#Column Vmag    (F5.2)  ? Magnitude in Johnson V (H5)   [ucd=phot.mag;em.opt.V]
_RAJ2000|_DEJ2000|HIP|Vmag
"h:m:s"|"d:m:s"| |mag
```

Figure 31: Export format

5.4.10.2 Revised New General Catalogue (Sulentic, et al., 1973)

8,163 deep sky objects with VizieR name VII/1B/catalog:



5.4.10.2.1 Export

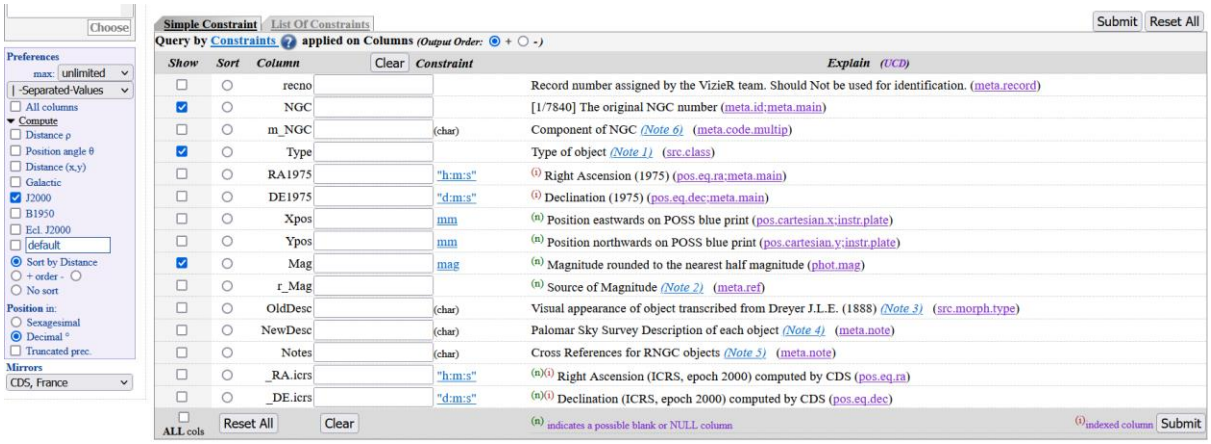
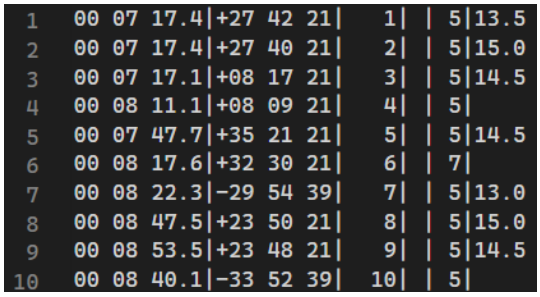


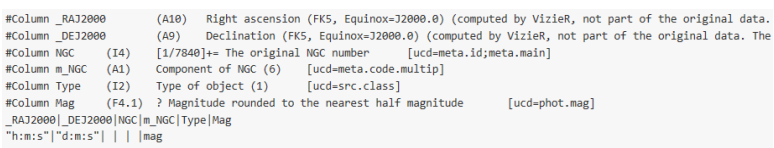
Figure 32: Export settings

5.4.10.2.2 File



```
1   00 07 17.4|+27 42 21|   1| | 5|13.5
2   00 07 17.4|+27 40 21|   2| | 5|15.0
3   00 07 17.1|+08 17 21|   3| | 5|14.5
4   00 08 11.1|+08 09 21|   4| | 5|
5   00 07 47.7|+35 21 21|   5| | 5|14.5
6   00 08 17.6|+32 30 21|   6| | 7|
7   00 08 22.3|-29 54 39|   7| | 5|13.0
8   00 08 47.5|+23 50 21|   8| | 5|15.0
9   00 08 53.5|+23 48 21|   9| | 5|14.5
10  00 08 40.1|-33 52 39|  10| | 5|
```

Figure 33: First 10 rows

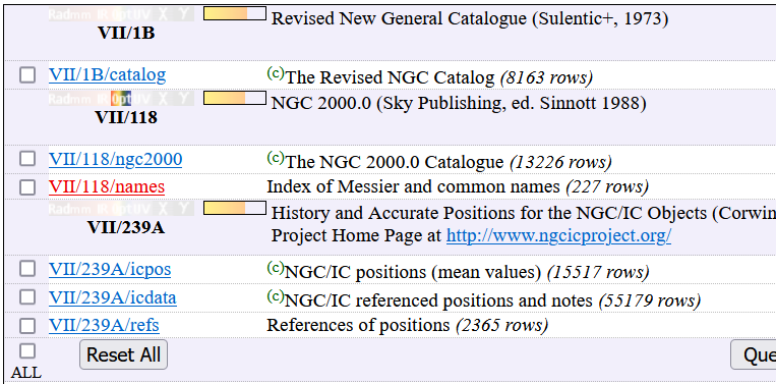


```
#Column _RAJ2000        (A10)   Right ascension (FK5, Equinox=J2000.0) (computed by VizieR, not part of the original data.
#Column _DEJ2000        (A9)    Declination (FK5, Equinox=J2000.0) (computed by VizieR, not part of the original data. The
#Column NGC     (I4)    [1/7840]+= The original NGC number      [ucd=meta.id;meta.main]
#Column m_NGC   (A1)    Component of NGC (6)    [ucd=meta.code.multip]
#Column Type    (I2)    Type of object (1)      [ucd=src.class]
#Column Mag     (F4.1)  ? Magnitude rounded to the nearest half magnitude       [ucd=phot.mag]
_RAJ2000|_DEJ2000|NGC|m_NGC|Type|Mag
"h:m:s"|"d:m:s"| | | |mag
```

Figure 34: Export format

5.4.10.3 NGC NAMES (SINNOTT, 1997)

227 NGC object names with VizieR name VII/118/names:



5.4.10.3.1 EXPORT

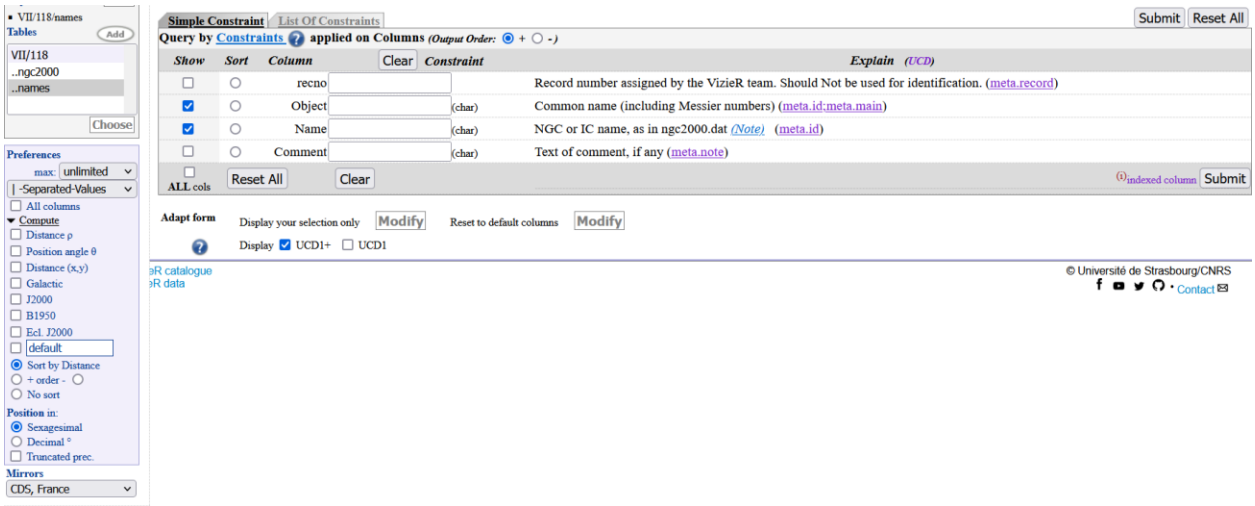


Figure 35: Export settings

5.4.10.3.2 FILE

```
47 Tuc                                |  104
Antennae                              | 4038
Antennae                              | 4039
Barnard's galaxy                      | 6822
Baxendell's unphotographable nebula| 7088
Bear Paw galaxy                       | 2537
Beehive cluster                       | 2632
Black-eye galaxy                      | 4826
Blinking planetary                    | 6826
Blue Snowball                         | 7662
```

Figure 36: First 10 rows

```
#Column Object  (a35)   Common name (including Messier numbers) [ucd=meta.id;meta.main]
#Column Name    (A5)    NGC or IC name, as in ngc2000.dat       [ucd=meta.id]
Object|Name
```

Figure 37: Export format

5.4.10.4 IAU NAMES (INTERNATIONAL ASTRONOMICAL UNION, 2022)

The IAU are the organisation behind star names. When talking about 'Polaris' or 'Sirius', for example, while they didn't come up with them all, the IAU is the arbiter of these names and their broad recognition.

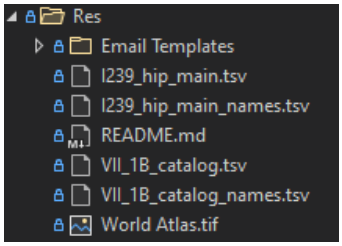
They offer the names as a file which very usefully contains columns mapping the names themselves to their corresponding IDs across a variety of catalogues (including the Hipparcos catalogue).

1	Absolutno	Absolutno	XO-5	_	_	Lyn	_	_	12.13	_	_	116.716506	39.094572	2019-12-17
2	Acamar	Acamar	HR 897	tet01	θ1	Eri	A	02583-4018	2.88	13847	18622	44.565311	-40.304672	2016-07-20 *
3	Achernar	Achernar	HR 472	alf	α	Eri	A	_	0.45	7588	10144	24.428523	-57.236753	2016-06-30
4	Achird	Achird	HR 219	eta	η	Cas	A	00491+5749	3.46	3821	4614	12.276213	57.815187	2017-09-05
5	Acrab	Acrab	HR 5984	bet	β	Sco	Aa	16054-1948	2.56	78820	144217	241.359300	-19.805453	2016-08-21
6	Acrux	Acrux	HR 4730	alf	α	Cru	Aa	12266-6306	1.33	60718	108248	186.649563	-63.099093	2016-07-20
7	Acubens	Acubens	HR 3572	alf	α	Cnc	Aa	08585+1151	4.26	44066	76756	134.621761	11.857700	2016-07-20
8	Adhafera	Adhafera	HR 4031	zet	ζ	Leo	Aa	10167+2325	3.43	50335	89025	154.172567	23.417312	2016-07-20
9	Adhara	Adhara	HR 2618	eps	ε	CMa	A	06586-2858	1.50	33579	52089	104.656453	-28.972086	2016-08-21
10	Adhil	Adhil	HR 390	ksi	ξ	And	_	_	4.87	6411	8207	20.585080	45.528778	2016-08-21

Figure 38: First 10 rows

The columns in question are the 1st and 10th. These are the star's name and HIP ID respectively. As an example, the star 'Acrab' (line 5) is the 78820th Hipparcos entry.

All files were placed in the server's Res folder:



5.4.10.5 CATALOGUE PARSERS

2 custom parsers were implemented to read the astronomical catalogues and insert their values into the database. They both inherited from an abstract *Parser* class. This abstract class handled the shared requirements of checking file existence and reading into memory.

One parser targeted the HIP catalogue, and another the NGC. Each of these concrete parsers implemented specific behaviour based on their target. This included, for example, reading the object's names from the name catalogues and reading the catalogue's rows themselves. When being instantiated, these concrete parsers passed their catalogue's names to *Parser*:

```
public HipParser(IFileSystem fs)
    : base(fs, "I239_hip_main.tsv", "I239_hip_main_names.tsv") { }
```

5.4.11 COMPUTATIONAL ASTRONOMY

To calculate the positions of astronomical objects in the sky from different positions on Earth at different times, AstroCue needed to implement various computational astronomy algorithms and work in multiple coordinate systems.

The two different celestial coordinate systems supported by AstroCue were:

1. Equatorial
2. Horizontal

The equatorial system is used as a fixed coordinate system to locate objects in the sky, and the horizontal coordinate system is used by observers on Earth to locate said objects from *specific locations at specific times*.

5.4.11.1 EQUATORIAL SYSTEM

The core of the equatorial system is the concept of the 'celestial sphere'. The idea is that distant stars can be thought of as being projected onto the inside of a large sphere surrounding the earth. Of course, this isn't the case, but it allows us to project conceptual lines of celestial longitude and latitude, respectively called Right Ascension (α/RA) and Declination (δ/Dec), onto its interior. These are angular units, and they allow the assignment of fixed coordinates to distant objects.

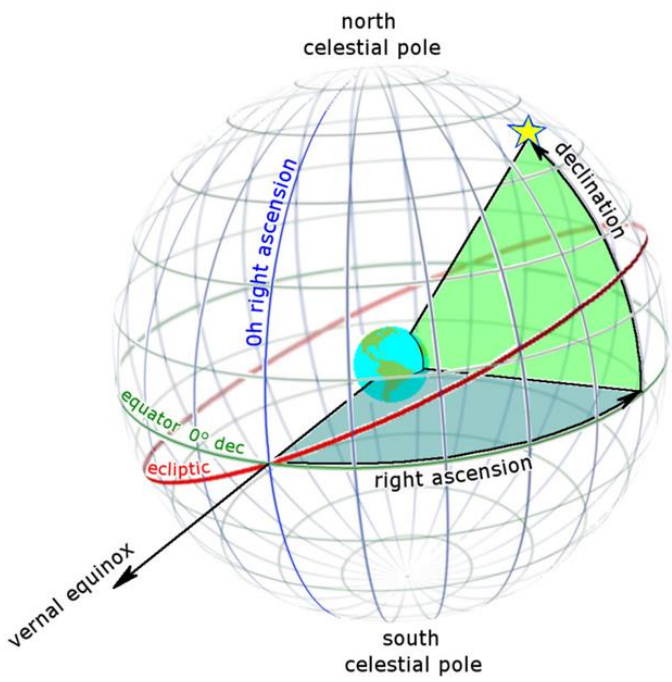


Figure 39: Equatorial system and the celestial sphere (Sky & Telescope, 2019).

5.4.11.2 HORIZONTAL SYSTEM

Equatorial coordinates are fixed. This means that they aren't entirely useful for an observer who wants to see an object from a specific location on the surface of the Earth at a specific time. For this case, the horizontal system exists.

It uses the observer's local horizon as the key plane and is a combination of two units called altitude and azimuth. The altitude represents how far up they need to look (from $0°$ at the horizon), and the azimuth is how far around a full spin they need to look (from $0°$ South).

The core algorithm used by AstroCue was a coordinate transform from the equatorial space into the horizontal space. All the astronomical catalogue exports from VizieR had equatorial coordinates. To tell an observer at location (x, y) where their object would be at time z, this transform was required.

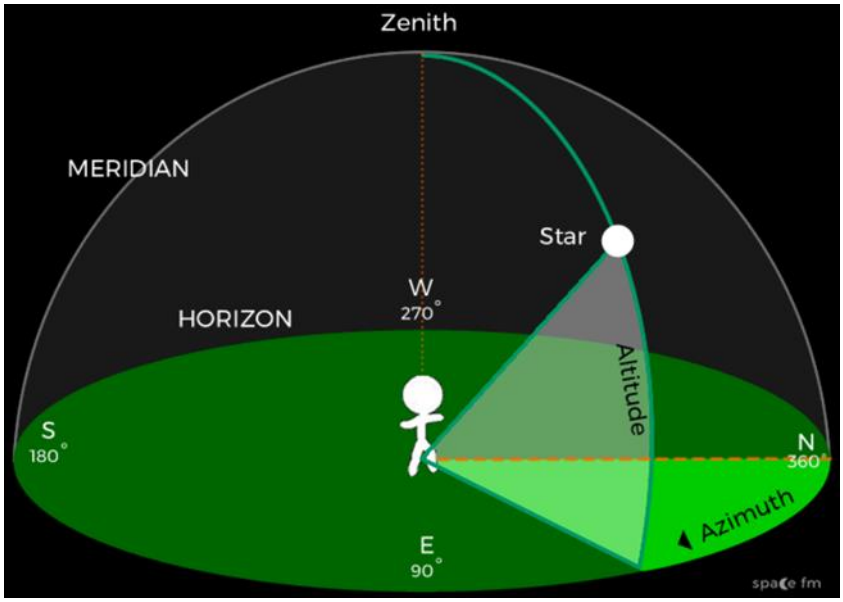


Figure 40: Horizontal system (space.fm, n.d.).

There are several intermediate steps before this can be calculated, and they are documented beginning overleaf.

More details in appendix 9.8.

5.4.11.3 JULIAN DAY

Calendars have been a part of civilisation since antiquity. Over the course of millennia, there have been various calendars that have seen widespread use, including the Julian Calendar popularised by Julius Caesar, and the Gregorian calendar that most people use today. There have, of course, been many others.

The attempted reconciliation of the movement of Earth (and consequently what humans observe to be days, months, and years) with rigid calendars leads to inaccuracies, which must be accounted for via periodic adjustments (leap years being a well-known example).

When attempting to calculate the positions of objects, early astronomers came up with the concept known as the Julian Day (JD) to get around these complications. The JD is simply the exact number of fractional days that have elapsed since the year -4712 (Meeus, 1998, p. 59).

The calculation of a JD value for a given instant UTC in the Gregorian calendar is given as:

$$JD = \lfloor 365.25(Y + 4716) \rfloor + \lfloor 30.6001(M + 1) \rfloor + D + B - 1524.5$$

$$where$$

$$Y = year\ (instant) \quad M = month\ (instant) \quad D = fractional\ day\ (instant)$$

$$and$$

$$A = \left\lfloor \frac{Y}{100} \right\rfloor \qquad B = 2 - A + \left\lfloor \frac{A}{4} \right\rfloor$$

$$and$$

$$M \leq 2 \implies Y -= 1 \ and \ M += 12$$

(Meeus, 1998, pp. 61, eq. 7.1)

Author's implementation:

```
int a = (int)Math.Floor(year / 100.0);
int b = 2 - a + (int)Math.Floor(a / 4.0);

return Math.Floor(365.25 * (year + 4716)) + Math.Floor(30.6001 * (month + 1)) +
dayFrac + b - 1524.5;
```

(Full code in appendix 9.11.7.1).

5.4.11.4 SIDEREAL TIME

Sidereal time is a method of timekeeping based on the movement of Earth against distant stars rather than its orbit around the Sun. It builds upon the JD and is used so that, when trying to locate objects, astronomers can ignore the quirks of the Earth's orbit (the root cause of the calendrical inaccuracies).

The Sidereal time, θ_0, for a given instant in the Gregorian calendar is:

$$\theta_0 = 280.46061837 + 360.98564736629\,(JD - 2451545) + 0.000387933\,T^2 - \frac{T^3}{38710000}$$

$$where$$

$$T = \frac{JD - 2451545}{36525}$$

$$T\ is\ then\ brought\ in\ range\ 0° - 360°\ by\ adding\ or\ subtracting\ multiples\ of\ 360°$$

(Meeus, 1998, pp. 87-88, eq. 12.1 & 12.4)

Author's implementation:

```
double jd = DateToJulianDay(instant);
double t = (jd - 2451545.0) / 36525;

double mst = 280.46061837 + 360.98564736629 * (jd - 2451545.0) + 0.000387933 * (t *
t) - t * t * t / 38710000;
```

(Full code in appendix 9.11.7.2)

5.4.11.5 EQUATORIAL TO HORIZONTAL TRANSFORM

Once all the previous calculations had been implemented, the author was able to implement the equatorial → horizontal coordinate transformation, defined as:

$$h = altitude, A = azimuth\ (horizontal)$$
$$\alpha = right\ ascension, \delta = declination\ (equatorial)$$
$$\varphi = observer's\ latitude, H = local\ hour\ angle$$
$$L = observer's\ longitude$$
$$\ldots$$
$$A = arctan2\left(\frac{\sin H}{\cos H\ \sin\varphi - \tan\delta\ \cos\varphi}\right)$$
$$\sin h = \sin\varphi\ \sin\delta + \cos\varphi\ \cos\delta\ \cos H$$
$$where$$
$$H = \theta_0 - L - \alpha$$

(Meeus, 1998, pp. 91-94, eq. 13.5 & 13.6)

First, α needs to be converted into degrees. Take, as an example, $9^h 14^m 55^s.8$:

$$\alpha = 15\left(9 + \frac{14}{60} + \frac{55.8}{3600}\right)$$

(Meeus, 1998, p. 8)

These are spherical trigonometry equations that calculate angular distances on the celestial sphere mentioned in section 5.4.11.1.

Author's implementation:

```
double azimuthRad = Math.Atan2(Math.Sin(hourAngleRad),
    Math.Cos(hourAngleRad) * Math.Sin(latitudeRad) - Math.Tan(decRad) *
Math.Cos(latitudeRad));

double altitudeRad = Math.Sin(latitudeRad) * Math.Sin(decRad) +
                     Math.Cos(latitudeRad) * Math.Cos(decRad) *
Math.Cos(hourAngleRad);

float az = (float)RadToDeg(azimuthRad);
float alt = (float)RadToDeg(Math.Asin(altitudeRad));

AltAz coordinates = new()
{
    Azimuth = az + 180,
    Altitude = alt
};
```

(Full code in appendix 9.11.7.3).

5.5 EVALUATION

The author carried out various evaluation activities to ensure that the output of the development process met the objectives and fulfilled all originally envisioned use cases.

5.5.1 FUNCTIONAL TESTING

5.5.1.1 USE CASE TESTING

13 use case tests were created. The reason for this number being less than the number of uses cases (9.4.5), is that many of the tests combined multiple use cases together. Below is an example of one of the use case tests created. These cases were built directly from the use cases they cover. This test can be seen to cover use cases 1 and 3.

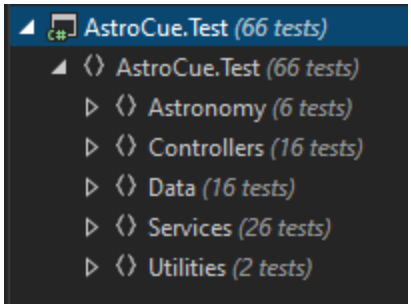
Test no.: 1	**Covers use case(s) ID:** 1 (Register), 3 (SendWelcomeEmail)		
Preconditions	User has internet access, and an email account with access to the inbox		
Steps	1. Open a web browser and navigate to astrocue.co.uk 2. You will be on the login screen, click the 'Register' link at the bottom 3. Enter a valid email address whose inbox you have access to 4. Enter your first name and last name in the corresponding boxes 5. Enter a password (note this down somewhere for later tests) 6. Click the 'Register' button		
Expected result	• You are redirected to the login screen, and a notification appears thanking you for registering • You have received a welcome email to the email address you used to register ○ If this is not present, check your junk folder		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

The full set of use cases tests, all of which passed, can be seen in appendix 9.9.

5.5.1.2 UNIT TESTING

Overall, the author wrote 66 total unit tests for AstroCue. While not entirely comprehensive in either a coverage or use case basis, the unit tests covered several key controllers, many service methods, all the computational astronomy calculations, the catalogue parsers, and some utility classes. All of the unit tests passed.

Below is the test breakdown per package, shown by Visual Studio:



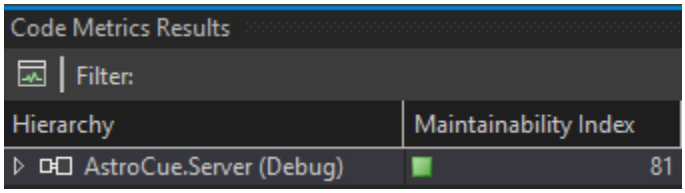
The full set of unit tests, including results, can be seen in appendix 9.9.4. Some of these unit tests were created to specifically prove that certain AstroCue objectives were met.

Many unit tests made use of mocking – the process of using interfaces to override the implementation of a dependency for a class under testing. For example, when testing `MappingService`, the author didn't want to have to include a real set of environment variables as unit tests should be isolated and atomic. To get around this, the `EnvironmentManager`, along with the other dependencies, was mocked, and the author could override behaviours where required:

```
private const string FakeMapBoxApiKey = "B76C88D4-AEC5-46B2-9FBA-CFEB505173B0";

private readonly Mock<IEnvironmentManager> _mockEnvironmentManager;

this._mockEnvironmentManager
    .Setup(i => i.MapBoxApiKey)
    .Returns(FakeMapBoxApiKey);

new MappingService(this._mockEnvironmentManager.Object, [other dependencies])
```

5.5.2 NON-FUNCTIONAL TESTING

VS's Maintainability Index was used to calculate code maintainability. A final score of 81/100 was recorded. This represented very maintainable code, and therefore a success. This score was largely driven by the author's reusable set of service classes (4.3.7.1). More details in appendix 9.9.5.



6 CONCLUSIONS AND DISCUSSION

To determine the success of AstroCue, it was useful to return to the original objectives laid out in section 1.

6.1 PRIMARY OBJECTIVE

"The main objective of AstroCue was to provide a set of integrated and unified web facilities for the automated planning and subsequent user logging of astronomical observations. It would consider geographic and celestial coordinates, local time, weather, and light pollution data to provide predicted best viewing times over a short upcoming period, and then allow users to log any observations that were made." – section 1.3.1, the same as the PDD (appendix 9.1).

6.1.1 PRIMARY OUTCOME

This main objective was met.

The outcome of the development process was a web service that automatically planned observations for users and allowed them to log the results. User's geographic coordinates were considered via their observation locations, and AstroCue's computational astronomy covered 2 celestial coordinate systems. Time, weather, and light pollution were used to generated reports and inform users of potential difficulties they may come across during their unique observation scenarios.

Once all of this had come together in the reports, users could easily log their results for future reference.

6.2 SUB OBJECTIVES

The sub objectives listed here are from section 1, which are also unchanged from their original definitions in the PDD in appendix 9.1.

6.2.1 SUB OBJECTIVE #1

"AstroCue shall produce reports on optimal viewing times for an astronomical object".

This objective was met - these reports can be seen when manually generated and appear every ~3 days. Both cases were confirmed via use case testing section 9.9.3.10.

6.2.2 SUB OBJECTIVE #2

"AstroCue shall export these reports to an external medium for future reference".

This objective was met – the external email exports of reports was also confirmed in use cast testing section 9.9.3.10.

6.2.3 SUB OBJECTIVE #3

"AstroCue shall be able to calculate positions of astronomical objects in the horizontal coordinate system with an accuracy of ± 10°".

This objective was met – the accuracy of the celestial coordinate transforms were confirmed via unit tests. The specific test that confirmed this objective is documented in unit testing section 9.9.4.2.1.

6.2.4 SUB OBJECTIVE #4

"AstroCue shall allow users to register observation locations with any valid latitude and longitude".

This objective was met[3] - the registration of observation locations was confirmed via use case testing section 9.9.3.5.

6.2.5 SUB OBJECTIVE #5

"AstroCue shall be able to retrieve weather forecasts for any valid observation location from the OpenWeatherMap API".

This objective was met – the forecasts being retrieved for an observation location was confirmed directly in use case test section 9.9.3.5.

[3] Within the confines of the dataset as documented in section 9.9.4.2.2. As this limitation is not due to a lack of effort from the author, and because with a more complete dataset all coordinates would have been supported, the author still regards this objective as having been met. Furthermore, even with this limitation, the number of potentially impacted users is extremely small.

6.2.6 SUB OBJECTIVE #6

"AstroCue shall allow users to create astronomical observation logs linked to the reports it generates (see sub objective 1)".

This objective was met – the creation of logs against reports can be seen in use cast testing section 9.9.3.11.

6.2.7 SUB OBJECTIVE #7

"AstroCue shall be able to retrieve light pollution levels for arbitrary observation locations".

This objective was met – this was confirmed by unit testing and can be seen in section 9.9.4.2.3.

6.3 DISCUSSION

AstroCue solved the original issue recognised with many pieces of software that allow users to find astronomical objects – the lack of automated observation planning and observation log facilities. It integrated these two features together, along with consideration for the user's local weather and light pollution levels, providing bespoke observation results.

Some of the implementation details for AstroCue were influenced by the literature review stage of this project. For example, the weather rating system (5.4.8) was inspired by the concept behind Clear Sky Chart (CSC), an astronomy weather forecast service discussed in section 3.2.1. However, AstroCue builds upon the great work of CSC by integrating this system as part of a larger set of unified tools available to users. They no longer need to plan observations and find optimal forecasts separately – they can be done under the same roof and are now automated.

These automations also improved upon other similar pieces of software such as AstroPlanner (section 3.2.2). While these other pieces of software may provide more in-depth information, they still lack such automated observation planning facilities.

AstroCue also successfully brought together lots of data from different sources in different formats and combined them to solve the same problem. VizieR's astronomical catalogues (3.3.1) were used to seed the database with a large collection of astronomical objects for users to access. The MapBox and OpenWeatherMap APIs (3.3.3 and 3.3.4) were also used to provide the bespoke location and weather-based data.

There are, however, some drawbacks to AstroCue. Firstly, a substantial contributor to light pollution, the Moon, was not considered. With more time, it would have been beneficial to integrate information about the current lunar cycle into the report generation process to provide a more accurate picture of light pollution levels.

Another downside is that times were not internationalised. Report generation times and best viewing times were all outputted in UTC. With some extra time and implementation, making the time zones specific to a user (or maybe their individual observation locations) would have added even more uniqueness to the already bespoke observation reports, and removed the need for users to convert these times to their local time zone manually.

6.4 PERSONAL CONCLUSIONS

The author was very happy with the outcome of AstroCue. Overall, it achieved all the high-level objectives set in the original PDD, with no changes needed.

The implementation went very smoothly, and the author managed to write large swaths of the code without any major issues or delays. The only part of the implementation that slowed down development considerably was the computational astronomy (5.4.11) – as this was a mostly new area to the author, especially the mathematical aspect. This was one key moment where standing on the shoulders of existing knowledge paid huge dividends – 'Astronomical Algorithms' (Meeus, 1998) was a priceless resource in this regard. Its importance to the outcome of this project cannot be understated.

6.4.1 POTENTIAL CHANGES

If the author could start again and try a different implementation, AstroCue would likely be implemented as a mobile app. Astronomers often carry their phones with them when they go observing, and in most cases, they still have a network signal – giving them constant access. Having access to push notifications would also make observation reminders easier and automated. This would mostly be a case of implementing the app as a separate standalone client, as the server aspect already exists and can be reused.

7 GLOSSARY OF TERMS

Term/abbreviation	Definition
API	Application Programming Interface
AWS	Amazon Web Services
Azure	Microsoft's cloud platform
Celestial	Positioning relating to the sky or space
DI	Dependency Injection
DNS	Domain Name System
DTU	Database Transaction Unit
Equatorial coordinate system	A type of celestial coordinate system used to locate astronomical coordinates in the night sky.
GeoTIFF	An extension of the .TIFF image format for geodetically correct raster georeferencing capabilities (Ritter & Ruth, 1997)
Hash	A mathematical function used to map data of an arbitrary size to a fixed size
Horizontal coordinate system	A type of celestial coordinate system centred on the observer, using their local horizon as a zero point for its coordinates.
IAU	International Astronomical Union
IDE	Integrated Development Environment
JD	Julian Day
JSON	JavaScript Object Notation
JWT	JSON Web Token
O/RM	Object-relational mapper
OS	Operating System
Rainbow table	Pre-computed table of hash outputs used in password cracking
Salt	Random data appended to the input parameter of a hash function to protect against rainbow tables
SQL	Structured Query Language
SQL injection	Using maliciously formed inputs to execute SQL and gain unauthorised access/control over a database
UML	Unified Modelling Language

8 REFERENCE LIST

1001freefonts, n.d. *Earth Orbiter Font.* [Online]
Available at: https://www.1001freefonts.com/earth-orbiter.font
[Accessed 24 January 2022].

Bortle, J. E., 2006. *Gauging Light Pollution: The Bortle Dark-Sky Scale.* [Online]
Available at: https://skyandtelescope.org/astronomy-resources/light-pollution-and-astronomy-the-bortle-dark-sky-scale/
[Accessed 9 May 2022].

Chen, P. P.-S., 1976. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems,* 1(1), pp. 9-36.

City, University of London, n.d. *Research participant information and consent.* [Online]
Available at: https://www.city.ac.uk/research/support/integrity-and-ethics/guidance-and-resources/participants
[Accessed 7 March 2022].

Danko, A., 2022. *Clear Sky Chart.* [Online]
Available at: http://www.cleardarksky.com/csk/index.html
[Accessed 15 February 2022].

European Space Agency, 1997. *The Hipparcos and Tycho Catalogues. Astrometric and Photometric Star Catalogues dervied from the ESA Hipparcos Space Astrometry Mission.* [Online]
Available at: https://www.cosmos.esa.int/documents/532822/552851/vol1_all.pdf
[Accessed 30 April 2022].

Falchi, F. et al., 2016a. The new world atlas of artificial night sky brightness. *Science Advances,* 2(6), pp. 1-25.

Falchi, F. et al., 2016b. *Supplement to: The New World Atlas of Artificial Night Sky Brightness.* [Online]
Available at: https://dataservices.gfz-potsdam.de/contact/showshort.php?id=escidoc:1541893&contactform
[Accessed 16 February 2022].

Flaticon, n.d. *Saturn free icon.* [Online]
Available at: https://www.flaticon.com/free-icon/saturn_2224678
[Accessed 24 January 2022].

Fowler, M., 2012. *Patterns of Enterprise Application Architecture.* Boston, MA: Pearson Education.

G. c., 2022a. *GDAL/OGR Geospatial Data Abstraction. Open Source Geospatial Foundation..* [Online]
Available at: https://gdal.org
[Accessed 8 May 2022].

G. c., 2022b. *Raster Data Model.* [Online]
Available at: https://gdal.org/user/raster_data_model.html
[Accessed 8 May 2022].

Georg, 2017. *Convert latitude longitude pair to pixels in geotiff.* [Online]
Available at: https://gis.stackexchange.com/questions/245472/convert-latitude-longitude-pair-to-pixels-in-geotiff
[Accessed 8 May 2022].

GitHub, n.d. *About project boards.* [Online]
Available at: https://docs.github.com/en/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards
[Accessed 14 March 2022].

IBM, 2021. *Docker.* [Online]
Available at: https://www.ibm.com/uk-en/cloud/learn/docker
[Accessed 14 March 2022].

iLanga Inc, 2020. *AstroPlanner.* [Online]
Available at: https://www.astroplanner.net/index.html
[Accessed 16 February 2022].

International Astronomical Union, 2022. *Division C WG Star Names.* [Online]
Available at: https://www.iau.org/science/scientific_bodies/working_groups/280/
[Accessed 1 May 2022].

InterVenture, 2015. *InterVenture Best Practices: Tomatoes Rot, and So Does the Code.* [Online]
Available at: https://www.interventure.info/blog/interventure-best-practices-tomatoes-rot-and-so-does-the-code/
[Accessed 12 April 2022].

JetBrains, n.d. *ReSharper - The Visual Studio Extension for .NET Developers.* [Online]
Available at: https://www.jetbrains.com/resharper/
[Accessed 16 February 2022].

Macleod, S. A., 2019. *Sampling methods. Simply Psychology.* [Online]
Available at: www.simplypsychology.org/sampling.html
[Accessed 7 March 2022].

MapBox, n.d. *Mapbox pricing.* [Online]
Available at: https://www.mapbox.com/pricing
[Accessed 16 February 2022].

Martin, R. C., 2005. *The Principles of OOD.* [Online]
Available at: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
[Accessed 14 March 2022].

Meeus, J., 1998. *Astronomical Algorithms.* 2nd ed. Richmond, Virginia: Willmann-Bell Inc.

Microsoft, 2021. *Code metrics values.* [Online]
Available at: https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022
[Accessed 24 April 2022].

Microsoft, 2021. *Entity Framework Core.* [Online]
Available at: https://docs.microsoft.com/en-us/ef/core/
[Accessed 14 March 2022].

Microsoft, 2021. *Inheritance.* [Online]
Available at: https://docs.microsoft.com/en-us/ef/core/modeling/inheritance
[Accessed 24 April 2022].

Microsoft, 2022. *App Service overview.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/app-service/overview
[Accessed 16 February 2022].

Microsoft, 2022. *App Service Pricing.* [Online]
Available at: https://azure.microsoft.com/en-us/pricing/details/app-service/linux/
[Accessed 14 March 2022].

Microsoft, 2022. *Try SQL Server on-premises or in the cloud.* [Online]
Available at: https://www.microsoft.com/en-gb/sql-server/sql-server-downloads
[Accessed 6 March 2022].

NASA Earthdata, 2022. *Visible Infrared Imaging Radiometer Suite (VIIRS).* [Online]
Available at: https://earthdata.nasa.gov/earth-observation-data/near-real-time/download-nrt-data/viirs-nrt
[Accessed 16 February 2022].

Ochsenbein, F., Bauer, P. & Marcout, J., 2000. The VizieR database of astronomical catalogues. *Astronomy and Astrophysics Supplement,* Volume 143, pp. 23-32.

OpenWeather, n.d. *OpenWeatherMap Weather API.* [Online]
Available at: https://openweathermap.org/api
[Accessed 16 February 2022].

OpenWeather, n.d. *Student initiative.* [Online]
Available at: https://openweathermap.org/our-initiatives/student-initiative
[Accessed 16 February 2022].

Pavlov, D., 2014. *Split a List into smaller lists of N size.* [Online]
Available at: https://stackoverflow.com/questions/11463734/split-a-list-into-smaller-lists-of-n-size
[Accessed 15 May 2022].

Perkic, A., n.d. *Understanding and Using the Celestial Coordinates.* [Online]
Available at: https://orionbearastronomy.com/2019/02/05/using-the-celestial-coordinates/
[Accessed 27 April 2022].

Ritter, N. & Ruth, M., 1997. The GeoTiff data interchange standard for raster geographic images. *International Journal of Remote Sensing,* 18(7), pp. 1637-1647.

Sinnott, W. R., 1997. *VizieR Online Data Catalog: NGC 2000.0,* s.l.: Sky Publishing Corporation.

Sky & Telescope, 2019. *Right Ascension & Declination: Celestial Coordinates for Beginners.* [Online]
Available at: https://skyandtelescope.org/astronomy-resources/right-ascension-declination-celestial-coordinates/
[Accessed 27 April 2022].

space.fm, n.d. *Horizontal Coordinate System - Altitude & Azimuth.* [Online]
Available at: https://www.space.fm/astronomy/starsgalaxies/horizoncoordinates.html
[Accessed 27 April 2022].

space.fm, n.d. *Precession.* [Online]
Available at: https://www.space.fm/astronomy/planetarysystems/precession.html
[Accessed 28 April 2022].

Spoelstra, H., 2009. *Sky Brightness Nomogram.* [Online]
Available at: http://www.darkskiesawareness.org/nomogram.php
[Accessed 9 May 2022].

Sulentic, J. W., Tifft, W. G. & Dreyer, J. L. E., 1973. *The revised new catalogue of nonstellar astronomical objects,* Tucson: University of Arizona Press.

TeamGantt, 2022. *TeamGantt.* [Online]
Available at: https://teamgantt.com
[Accessed 6 February 2022].

Unlayer, 2022. *Game Activation.* [Online]
Available at: https://unlayer.com/templates/game-activation
[Accessed 9 May 2022].

Vercel, 2022. *Edge Network.* [Online]
Available at: https://vercel.com/docs/concepts/edge-network/overview
[Accessed 28 April 2022].

Zotti, G. et al., 2020. The Simulated Sky: Stellarium for Cultural Astronomy Research. *Journal of Skyscape Archaeology,* 6(2), pp. 221-258.

9 APPENDICES

	Appendix	Submission method
A	**[Required]** Project Definition Document (PDD)	This document (section 9.1)
B	**[Required]** Reuse Summary	This document (section 9.2)
C	AstroCue Participant Information Sheet	This document (section 9.3)
D	Requirements	This document (section 9.4)
E	Design	This document (section 9.5)
F	Deployment and domain/DNS	This document (section 9.6)
G	EF Core database migration script	Moodle submission.
H	Introduction to celestial coordinate systems	This document (section 9.8)
I	Testing	This document (section 9.9)
J	Software build guide	This document (section 9.10) and additional files via Moodle.
K	Code snippets	This document (section 9.11)
L	Full codebase	Moodle submission.
M	User interface / all web pages	This document (section 9.13)

9.1 APPENDIX A: PROJECT DEFINITION DOCUMENT (PDD)

Project Definition Document

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

No proprietary arrangements, interests, outside help or other promises.

Consultant: [REDACTED]

Word count: 1166

9.1.1 PROPOSAL

9.1.1.1 CONTEXT

There are many pieces of software that allow users to locate astronomical objects in the night sky. Many of these are in the form of augmented reality (AR) mobile apps – users point their phones at the sky and the internal compass and accelerometer are used to show a projected image of the constellations, galaxies, and other such astronomical objects. These are great for on-the-spot observations, or for identifying astronomical objects that have already been seen. An example of one such app, Stellarium, is shown in both mobile and desktop screenshots below:

Figure 42: Orion constellation in Stellarium v. 1.8.3 running on Android 12 (Zotti, et al., 2020)

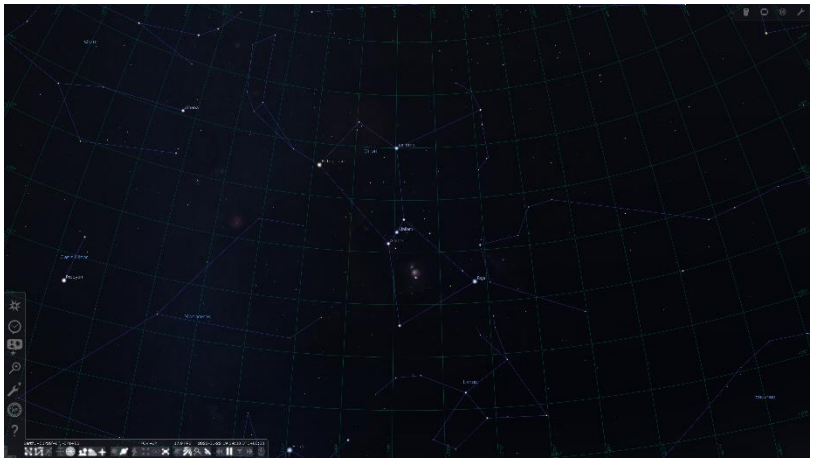


Figure 41: Orion constellation in Stellarium v. 0.21.3 running on Windows 11 (Zotti, et al., 2020)

9.1.1.2 PROBLEM TO BE SOLVED

A shortcoming of many of these types of software (not just the mobile apps) is that they do not provide automated facilities for planning observations in advance; fewer still consider the user's location beyond its geographic coordinates and local time. For example, it is of little use to tell a user that a star they want to observe is visible at celestial coordinates (x, y) if the skies are overcast and they are in the middle of a city with extreme light pollution – this is exacerbated when the user may want to carry out their observations at a future time, or at a different location to their home address and cannot simply go outside to check the conditions for themselves. Additionally, on top of their lack of automated planning facilities, many provide no way to take an astronomical observation log – a key part of any astronomy session.

When considering the following objectives, note that AstroCue does not aim to replace the existing pieces of planetarium/astronomy software, but rather be used in tandem as another tool available to astronomers during and after their observations.

9.1.2 PROJECT OBJECTIVES

9.1.2.1 MAIN OBJECTIVE

The main objective of AstroCue is to provide a set of integrated and unified web facilities for the automated planning and subsequent user logging of astronomical observations. It will consider geographic and celestial coordinates, local time, weather, and light pollution data to provide predicted best viewing times over a short upcoming period, and then allow users to log any observations that were made.

9.1.2.2 SUB OBJECTIVES

	Objective	Test
1	AstroCue shall produce reports on optimal viewing times for an astronomical object	Combinations of observation locations and astronomical objects should produce periodic reports on viewing times.
2	AstroCue shall export these reports to an external medium for future reference.	After a report has been generated, an email should be sent from the server to the user's email address via the MailGun API.
3	AstroCue shall be able to calculate positions of astronomical objects in the horizontal coordinate system with an accuracy of ± 10°.	Compare outputs to known correct test cases via unit testing.
4	AstroCue shall allow users to register observation locations with any valid latitude and longitude.	Any observation location with a latitude value in the range -90° to +90° combined with a longitude value in the range -180° to +180° should be accepted.
5	AstroCue shall be able to retrieve weather forecasts for any valid observation location from the OpenWeatherMap API.	Any observation location (with a valid longitude and latitude) can have a weather forecast retrieved for it.
6	AstroCue shall allow users to create astronomical observation logs linked to the reports it generates (see sub objective 1).	Observation logs can be made against a generated report.
7	AstroCue shall be able to retrieve light pollution levels for arbitrary observation locations.	Compare outputs to known correct test cases via unit testing.

9.1.3 PROJECT BENEFICIARIES

The main beneficiaries of AstroCue will first and foremost be the author, followed by other amateur and hobby astronomers.

9.1.3.1 BENEFITS

The benefit of AstroCue will be its automated calculation of optimal viewing times. Users will not need to manually check if a star is visible where they are planning to observe, what the weather is like over the next few days there, or if this location has too much light pollution to easily observe their chosen object – AstroCue will take all this information into account and produce succinct, easy to understand reports.

9.1.4 WORK PLAN

The work plan for this project will involve various stages. Before any implementation can start, there are several things that need to be completed, including this document (PDD), requirements gathering and designs (including UML diagrams and wireframes for the user interface).

Once these planning, design and pre-implementation stages are done, they will be followed by a series of agile sprints focusing on the technical implementation itself along with its outputs.

This plan can be seen overleaf in the form of a Gantt chart, broken down into tasks, deadlines, and outputs.

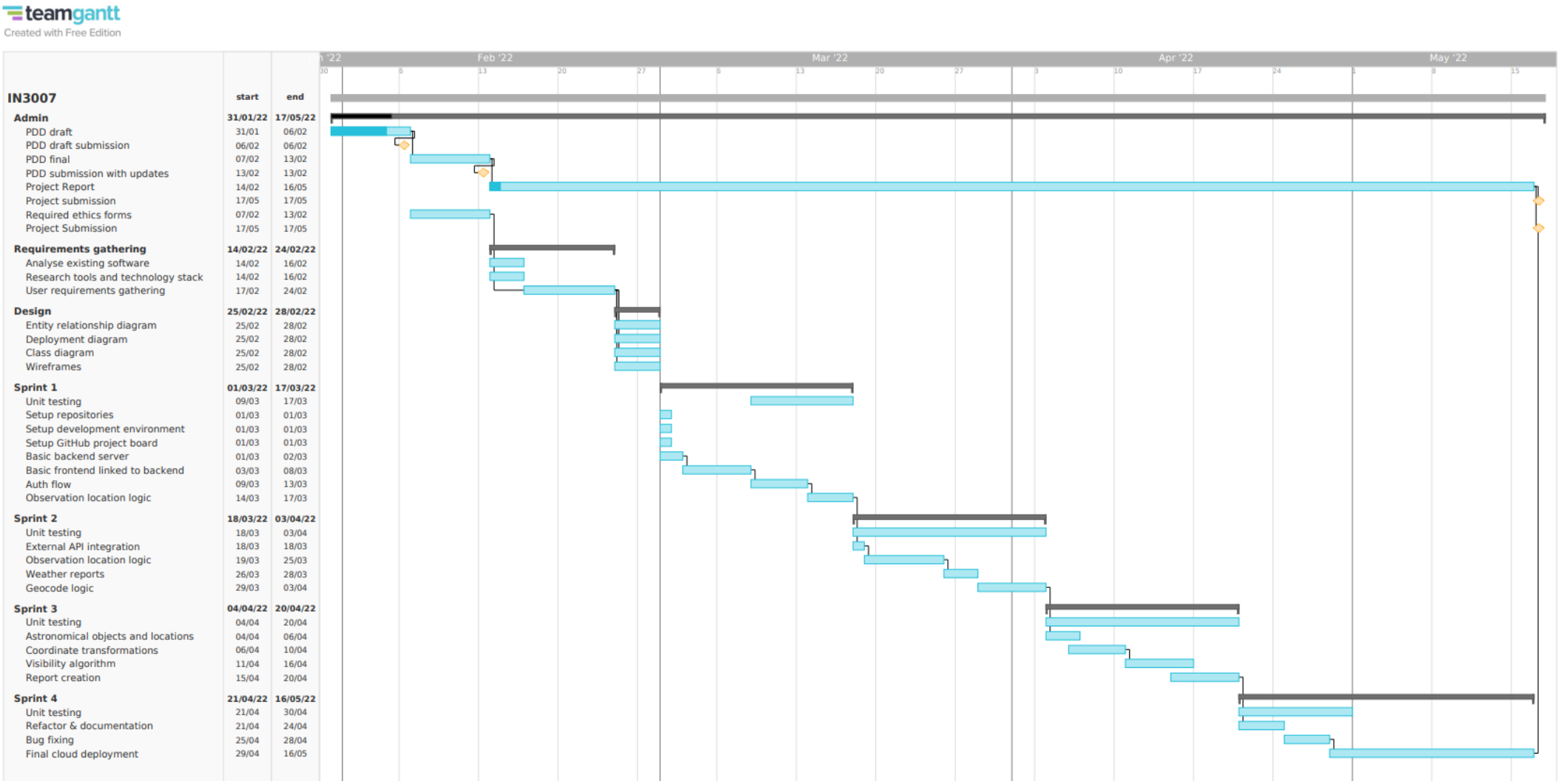


Figure 43: Gantt chart produced with TeamGantt Free Edition (TeamGantt, 2022)

9.1.5 PROJECT RISKS

	Risk	Probability	Impact	Mitigation(s)
1	Extended outage of production cloud provider (i.e., Azure).	Very low	Moderate	Another cloud service can be used as a temporary replacement (i.e., AWS).
2	Extended outage of external API. (i.e., the OpenWeatherMap API goes down).	Very low	Moderate	All interactions with external services should be loosely coupled to the rest of the system such that providers can be switched if the need arises.
3	Hardware failure.	Low	Low	All source code will be checked into version control, and documentation will be backed up.
4	Difficulty in parsing GeoTIFF files for light pollution data.	Low	High	Research suitable tutorials and take into consideration open-source code that achieves the same functionality.
5	Difficulty integrating functionality reliant on third-party APIs.	Low	High	Refer closely to official documentation (especially where examples are given) and assign more time to the API integration tasks to accommodate any delays.
6	Difficulty in understanding and implementing required computational astronomy algorithms.	Moderate	High	Use relevant literature as a guide – 'Astronomical Algorithms' being a good example (Meeus, 1998). Use stringent unit testing with known correct test cases.
7	Unauthorised access to passwords.	Low	High	Passwords will be hashed (and salted to defend against rainbow tables). Inputs will be sanitised to protect against SQL injection.
8	Teams/Zoom outage before consultant meeting(s).	Low	Low	Communications can resume via email.

9.1.6 ETHICS CHECKLIST

Research Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

http://www.city.ac.uk/department-computer-science/research-ethics

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form. Students who have answered "no" to all questions in A1, A2 and A3 and "yes" to question 4 in A4 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk. The approval may be ***provisional*** – *identifying the planned research as* likely to involve MINIMAL RISK. In such cases you must additionally seek ***full approval*** from the supervisor as the project progresses and details are established. ***Full approval*** must be acquired in writing, before beginning the planned research.

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)?	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act?	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation?	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent?	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects?	**NO**

2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study?	**NO**
2.6	Does your research involve invasive or intrusive procedures?	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)?	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London?	**NO**
3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**

A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data?	**YES**

PART B: Ethics Proportionate Review Form

If you answered YES to question 4 and NO to all other questions in sections A1, A2 and A3 in PART A of this form, then you may use PART B of this form to submit an application for a proportionate ethics review of your project. Your project supervisor has delegated authority to review and approve this application under proportionate review. You must receive final approval from your supervisor in writing before beginning the planned research.

However, if you cannot provide all the required attachments (see B.3) with your project proposal (e.g. because you have not yet written the consent forms, interview schedules etc), the approval from your supervisor will be ***provisional***. You **must** submit the missing items to your supervisor for approval prior to commencing these parts of your project. Once again, you must receive written confirmation from your supervisor that any provisional approval has been superseded by with ***full approval*** of the planned activity as detailed in the full documents. **Failure to follow this procedure and demonstrate that final approval has been achieved may result in you failing the project module.**

Your supervisor may ask you to submit a full ethics application through Research Ethics Online, for instance if they are unable to approve your application, if the level of risks associated with your project change, or if you need an approval letter from the CSREC for an external organisation.

B.1 The following questions must be answered fully. **All grey instructions must be removed.**		*Delete as appropriate*
1.1.	Will you ensure that participants taking part in your project are fully informed about the purpose of the research?	**YES**
1.2	Will you ensure that participants taking part in your project are fully informed about the procedures affecting them or affecting any information collected about them, including information about how the data will be used, to whom it will be disclosed, and how long it will be kept?	**YES**
1.3	When people agree to participate in your project, will it be made clear to them that they may withdraw (i.e. not participate) at any time without any penalty?	**YES**

1.4	Will consent be obtained from the participants in your project? Consent from participants will be necessary if you plan to involve them in your project or if you plan to use identifiable personal data from existing records. "Identifiable personal data" means data relating to a living person who might be identifiable if the record includes their name, username, student id, DNA, fingerprint, address, etc.	**YES**
1.5	Have you made arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential?	**YES**

B.2 If the answer to the following question (B2) is YES, you must provide details		*Delete as appropriate*
2	Will the research be conducted in the participant's home or other non-University location?	**YES**

B.3 Attachments **ALL of the following documents MUST be provided to supervisors if applicable.** **All must be considered prior to final approval by supervisors.** **A written record of final approval must be provided and retained.**	***YES***	***NO***	***Not Applicable***
Details on how safety will be assured in any non-University location, including risk assessment if required (see B2) **Participants will be able to carry out all tasks remotely from safe and suitable locations of their choice, free from any danger.**	**X** (See left)		
Details of arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential (see B1.5) **No personal information will be collected from participants prior to, or during their participation. Questions that are not strictly necessary for AstroCue's requirements analysis will be avoided.** **The data collection will take place solely via City's Qualtrics survey software, the author will not use Google Forms or any other third-party survey provider.**	**X** (See left)		
Full protocol for any workshops or interviews**			**X**
Participant information sheet(s)**	**X** (Still in draft form at the time of submission)		

Consent form(s)**	**X** (Still in draft form at the time of submission)		
Questionnaire(s)**	**X** (Still in draft form at the time of submission)		
Topic guide(s) for interviews and focus groups**			**X**
Permission from external organisations or Head of Department**			**X**

***If these items are not available at the time of submitting your project proposal, then **provisional approval** can still be given, under the condition that you must submit the final versions of all items to your supervisor for approval at a later date. **All** such items **must** be seen and approved by your supervisor before the activity for which they are needed begins. Written evidence of **final approval** of your planned activity must be acquired from your supervisor before you commence.*

Changes

If your plans change and any aspects of your research that are documented in the approval process change as a consequence, then any approval acquired is invalid. If issues addressed in Part A (the checklist) are affected, then you must complete the approval process again and establish the kind of approval that is required. If issues addressed in Part B are affected, then you must forward updated documentation to your supervisor and have received written confirmation of approval of the revised activity before proceeding.

Templates for Consent and Information

You must use the templates provided by the University as the basis for your participant information sheets and consent forms. You **must** adapt them according to the needs of your project before you submit them for consideration.

Participant Information Sheets, Consent Forms and Protocols must be consistent. Please ensure that this is the case prior to seeking approval. Failure to do so will slow down the approval process.

We strongly recommend using Qualtrics to produce digital information sheets and consent forms.

Further Information

http://www.city.ac.uk/department-computer-science/research-ethics

https://www.city.ac.uk/research/ethics/how-to-apply/participant-recruitment

https://www.city.ac.uk/research/ethics

9.1.7 GLOSSARY OF TERMS

Term/abbreviation	Definition
API	Application Programming Interface
Apparent Magnitude	How bright an astronomical object appears from the surface of Earth.
AWS	Amazon Web Services (Amazon's cloud platform)
Azure	Microsoft's cloud platform
Celestial	Positioning relating to the sky or space
GeoTIFF	An extension of the .TIFF image format for geodetically correct raster georeferencing capabilities (Ritter & Ruth, 1997)
Hash	A mathematical function used to map data of an arbitrary size to a fixed size
Horizontal coordinate system	A type of celestial coordinate system centred on the observer
O/RM	Object-Relational Mapper
Rainbow table	Pre-computed table of hash outputs used in password cracking
Salt	Random data appended to the input parameter of a hash function to protect against rainbow tables
SQL	Structured Query Language
SQL injection	Using maliciously formed inputs to execute SQL and gain unauthorised access/control over a database
UML	Unified Modelling Language

(*The original bibliography of the PDD was merged into the main report's bibliography in section 8*).

9.2 APPENDIX B: REUSE SUMMARY

9.2.1 LIBRARIES

This section documents all the libraries used by AstroCue in both the server and client codebases.

9.2.1.1 SERVER / BACKEND

Library	Version	License	Link
AutoMapper	11.0.1	MIT	Nuget
AutoMapper.Extensions.Microsoft.DependencyInjection	11.0.0	MIT	Nuget
coverlet.collector	3.1.0	MIT	Nuget
FluentAssertions	6.5.0	Apache-2.0	Nuget
Hangfire.AspNetCore	1.7.28	GitHub Link	Nuget
HangFire.Core	1.7.28	GitHub Link	Nuget
HangFire.SqlServer	1.7.28	GitHub Link	Nuget
MaxRev.Gdal.Core	3.3.3.110	MIT	Nuget
MaxRev.Gdal.LinuxRuntime.Minimal	3.3.3.120	MIT	Nuget
MaxRev.Gdal.WindowsRuntime.Minimal	3.3.3.110	MIT	Nuget
Microsoft.AspNetCore.Authentication.JwtBearer	5.0.15	Apache-2.0	Nuget
Microsoft.EntityFrameworkCore	5.0.15	Apache-2.0	Nuget
Microsoft.EntityFrameworkCore.Design	5.0.15	Apache-2.0	Nuget
Microsoft.EntityFrameworkCore.InMemory	5.0.15	Apache-2.0	Nuget
Microsoft.EntityFrameworkCore.SqlServer	5.0.15	Apache-2.0	Nuget
Microsoft.EntityFrameworkCore.Tools	5.0.15	Apache-2.0	Nuget
Microsoft.NET.Test.Sdk	16.11.0	Nuget Link	Nuget
Microsoft.VisualStudio.Azure.Containers.Tools.Targets	1.14.0	Nuget Link	Nuget
Moq	4.16.1	GitHub Link	Nuget
MSTest.TestAdapter	2.2.7	MIT	Nuget
MSTest.TestFramework	2.2.7	MIT	Nuget
Newtonsoft.Json	13.0.1	MIT	Nuget
RestSharp	107.3.0	Apache-2.0	Nuget
RichardSzalay.MockHttp	6.0.0	MIT	Nuget
Swashbuckle.AspNetCore	5.6.3	MIT	Nuget
Swashbuckle.AspNetCore.Annotations	6.3.0	MIT	Nuget
System.IO.Abstractions	16.1.23	MIT	Nuget

9.2.1.2 CLIENT / FRONTEND

Library	Version	License	Link
@emotion/react	^11.7.1	MIT	NPM
@emotion/styled	^11.6.0	MIT	NPM
@mui/icons-material	^5.5.1	MIT	NPM
@mui/lab	^5.0.0-alpha.75	MIT	NPM
@mui/material	^5.4.1	MIT	NPM
@rehooks/local-storage	^2.4.4	MIT	NPM
axios	^0.26.1	MIT	NPM
axios-hooks	^3.0.1	MIT	NPM
formik	^2.2.9	Apache-2.0	NPM
next	12.0.10	MIT	NPM
react	17.0.2	MIT	NPM
react-dom	17.0.2	MIT	NPM
react-toastify	^8.2.0	MIT	NPM
@types/material-ui	^0.21.12	MIT	NPM
@types/node	17.0.17	MIT	NPM
@types/react	17.0.39	MIT	NPM
eslint	8.9.0	MIT	NPM
eslint-config-next	12.0.10	MIT	NPM
eslint-config-prettier	^8.5.0	MIT	NPM
prettier	^2.6.2	MIT	NPM
typescript	4.5.5	Apache-2.0	NPM

9.2.1.2.1 MISCELLANEOUS CLIENT / FRONTEND

Library	Version	License	Link
create-next-app *see section 5.4.2.3	12.1.6	MIT	NPM

9.2.2 SOURCE LISTING

The following tables show a full source code listing for the server and the client. For each item, the table provides a status colour code that denotes original files, reused files, reused with adaptations etc… For items marked as directories (beginning with '/'), any status applies recursively.

The author created `.diff` files, where required, to show larger reuse. These files have their names given here and were submitted separately via Moodle. To find the changes that the author made to a reused/templated file, reference the names in the table in 9.2.2.2 and 9.2.2.3 against files themselves that can be found in the `.zip` file provided via Moodle submission. These files can be rendered with colours via most modern editors such as VS Code. However, they are plain text so can be opened in essentially any available editor.

9.2.2.1 KEY

Type	Colour	Notes
Original work		N/A
Reused with changes		N/A
Auto generated		N/A
Visual Studio template		Files created via the Visual Studio template (see section 5.4.2.2).
`create-next-app` template		Files created via the `create-next-app` template library (see section 5.4.2.3)

9.2.2.2 SERVER / BACKEND

Item	Status	Original file	Diff file showing changes
`.dockerignore`		See section 5.4.2.2	Unchanged from template
`.editorconfig`		N/A	N/A
`.gitignore`		N/A	N/A
`/Astronomy`		N/A	N/A
`/Controllers/Parameters`		N/A	N/A
`/Data`		N/A	N/A
`/Entities`		N/A	N/A
`/Migrations`		N/A	N/A
`/Models`		N/A	N/A
`/Utilities`		N/A	N/A
`appsettings.Development.json`		See section 5.4.2.2	`appsettings.Development.json.diff`
`appsettings.json`		See section 5.4.2.2	`appsettings.json.diff`
`AstroCue.Server.csproj`		N/A	N/A
`AstroCue.Server.csproj.user`		N/A	N/A
`AstroCue.Server.sln`		N/A	N/A
`AstroCue.Server.sln.DotSettings.user`		N/A	N/A
`AstroCue.Test`		N/A	N/A
`AstroCueReportEmail.hbs`		Original	`AstroCueReportEmail.hbs.diff`[4]
`AstroCueUserService.cs`		N/A	N/A
`AstroCueWelcomeEmail.hbs`		Original	`AstroCueWelcomeEmail.hbs.diff`[4]

[4] HTML changes were made within Unlayer so do not constitute work by the author. Only *tag content* changes and Handlebars integrations were new work. However, the diffs were too complex to reference this, so it is noted here.

AuthController.cs		Author's submission for the IN3046 module	AuthController.cs.diff
AuthService.cs		Author's submission for the IN3046 module	AuthService.cs.diff
DevEmailService.cs		N/A	N/A
Dockerfile		See section 5.4.2.2	Unchanged from template
EmailService.cs		N/A	N/A
GeoController.cs		N/A	N/A
HttpClientService.cs		N/A	N/A
IAstroCueUserService.cs		N/A	N/A
IAuthService.cs		Author's submission for the IN3046 module	IAuthService.cs.diff
IEmailService.cs		N/A	N/A
IHttpClientService.cs		N/A	N/A
ILightPollutionService.cs		N/A	N/A
IMappingService.cs		N/A	N/A
IObservationLocationService.cs		N/A	N/A
IObservationLogService.cs		N/A	N/A
IObservationService.cs		N/A	N/A
IReportService.cs		N/A	N/A
IWeatherForecastService.cs		N/A	N/A
LightPollutionService.cs		N/A	N/A
MappingService.cs		N/A	N/A
ObservationController.cs		N/A	N/A
ObservationLocationController.cs		N/A	N/A
ObservationLocationService.cs		N/A	N/A
ObservationLogController.cs		N/A	N/A
ObservationLogService.cs		N/A	N/A
ObservationService.cs		N/A	N/A
Program.cs		See section 5.4.2.2	Program.cs.diff
ReportController.cs		N/A	N/A
ReportService.cs		N/A	N/A
Startup.cs		See section 5.4.2.2	Startup.cs.diff
WeatherForecastService.cs		N/A	N/A

9.2.2.3 CLIENT/ FRONTEND

Item	Status	Original file	Diff file showing changes
.eslintrc.json		See section 5.4.2.3	N/A
.gitignore		See section 5.4.2.3	N/A
.nvmrc		N/A	N/A
.prettierignore		N/A	N/A
.prettierrc.json		N/A	N/A
/src		N/A	N/A
_app.tsx		See section 5.4.2.3	_app.tsx.diff
_document.tsx		Source	_document.tsx.diff
config.ts		N/A	N/A
globals.css		See section 5.4.2.3	globals.css.diff
index.tsx		See section 5.4.2.3	index.tsx.diff
login.tsx		N/A	N/A
next.config.js		See section 5.4.2.3	N/A
next-env.d.ts		See section 5.4.2.3	N/A
package.json		N/A	N/A
package-lock.json		N/A	N/A
README.md		See section 5.4.2.3	Unchanged from template
register.tsx		N/A	N/A
tsconfig.json		N/A	N/A
yarn.lock		N/A	N/A

9.2.3 OTHER TOOLS AND REUSE

- The author made use of the GDAL (Geospatial Data Abstraction Library) command line tools for GeoTIFF raster resampling. These are available from https://gdal.org/index.html# and are released under an MIT style license.
- Geographic Information Systems: *Convert latitude longitude pair to pixels in geotiff* (Georg, 2017) (5.4.7).
- Stack Overflow: *Split a List into smaller lists of N size* (Pavlov, 2014).

9.3 APPENDIX C: ASTROCUE PARTICIPANT INFORMATION SHEET

AstroCue Requirements Gathering Questionnaire

Invitation

We would like to invite you to take part in a research study. Before you decide whether you would like to take part it is important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information. You will be given a copy of this information sheet to keep.

What is the purpose of the study?

The aim of this study is to gather an initial set of requirements for 'AstroCue' – a platform for planning and logging astronomical observations.

Why have I been invited to take part?

You have been invited to take part in this research study due to your interest in astronomy and/or space.

Do I have to take part?

Participation in this survey is voluntary, and you can choose not to participate in part or all of the survey. You can withdraw at any stage of the project without being penalised or disadvantaged in any way. It is up to you to decide whether or not to take part. If you decide to take part you are still free to withdraw at any time and without giving a reason.

The point at which you can no longer withdraw your data is the final submission. Withdrawal at any point prior to this will result in any data collected thus far not being retained. Withdrawal occurs 1 hour after you have last edited a response and have still not reached the final submission.

What will happen if I take part?

You will fill out an anonymous online survey that should take no longer than 10 minutes. No personal information will be collected. You will be asked to answer a series of questions about potential features and requirements for the AstroCue platform.

What are the possible disadvantages and risks of taking part?

None.

What will happen to the results?

The results of this survey will go into influencing the initial requirements and design documents that will inform the upcoming development of the AstroCue platform. The fully anonymous results of this survey will be included in the final report that will be produced alongside this development process.

Confidentiality & anonymity

This survey collects no personal data, asks no questions for which answers containing personal data are required, and does **not** log the IP addresses of participants.

Further information and contact details

For any inquiries about this research, please contact either:

9.4 APPENDIX D: REQUIREMENTS

Requirements

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

Consultant: [REDACTED]

9.4.1 QUESTIONNAIRE

As part of the requirements gathering stage, an anonymous questionnaire was used to ascertain which features members of the target audience would find the most useful. This document covers the questions, their answers, and the requirements that resulted.

9.4.1.1 QUESTIONS ASKED

The following are the questions that were included, taken directly from Qualtrics' preview.

9.4.1.1.1 Q1: PARTICIPANT INFORMATION AND CONSENT

This question followed the participant information that can be found elsewhere in the appendices.

I understand what taking part in this study will involve.
I agree to take part in this study.

- ○ Yes
- ○ No

9.4.1.1.2 Q2: AMATEUR AND HOBBYIST SPLIT

As AstroCue aimed to cater for 2 types of astronomers (amateurs and hobbyists) the first question allowed participants to select which one they felt they fell under.

Which of these best describes your current astronomy/stargazing ability?

- ○ Amateur
- ○ Hobbyist

9.4.1.1.3 Q3: INFORMATION INCLUDED IN OBSERVATION REPORTS

One of the main goals of AstroCue was to be able to generate reports on when the best time to go and view an astronomical object would be. To this end, the next question asked which information the participant would find the most useful in such a report. There was also an open-ended option for any other suggestions not mentioned in the question.

If you were looking at a report about the best time to view an astronomical object, what pieces of information would you find useful?

Tick all that apply.

- ☐ The forecasted weather at the point of the recommended observation
- ☐ The object's rise and set times
- ☐ The object's transit time
- ☐ Altitude/Azimuth values that can be used in tracking telescopes
- ☐ The apparent magnitude (visible brightness) of the object
- ☐ If available for a star, the IAU (International Astronomical Union) formalised name.(I.e. Sirius, Betelgeuse, Polaris etc...)
- ☐ A link to discover more information about the object
- ☐ Other

9.4.1.1.4 Q4: Information included in astronomical logs

In a similar fashion, the next question asked what pieces of information the participant would find most useful to be able to log against an observation. There was also an open-ended option for any other suggestions not mentioned in the question.

When taking an observation log, what pieces of information would you find it the most useful to be able to include?

Tick all that apply

- [ ] A textual description of what was observed
- [ ] Pictures (if any were taken)
- [ ] The weather conditions at the time
- [ ] The type of observation made (naked eye, telescope, long exposure photography etc...)
- [ ] Other

9.4.1.1.5 Q5: Frequency of generated reports

As a platform for generating reports, it was important to consider how often said reports are generated. The next question offered participants to select on a slider how often (per week) they think reports should be created.

How often do you think observation reports should be created?

With 1 being every day of the week, and 7 being once a week.

1 2 3 4 4 5 6 7

Days between each report

9.4.1.1.6 Q6: Report export medium

One of the sub-objectives of AstroCue defined in the PDD was the ability to export the reports it generates to an external medium. Question 6 was aimed at finding out which export type was the most popular. There was also an open-ended option for any other suggestions not mentioned in the question.

AstroCue aims to export the reports it generates. To what medium would these exports prove to be the most useful?

Tick all that apply.

- [ ] Email
- [ ] Calendar (i.e. Google Calendar export options)
- [ ] Other

9.4.1.1.7 Q7: Closing remarks

The last question was an optional text entry box for participants to enter any other thoughts or suggestions they may have had that the set of questions did not touch on. Unlike the others, this question did not require an answer and participants could skip if they wished.

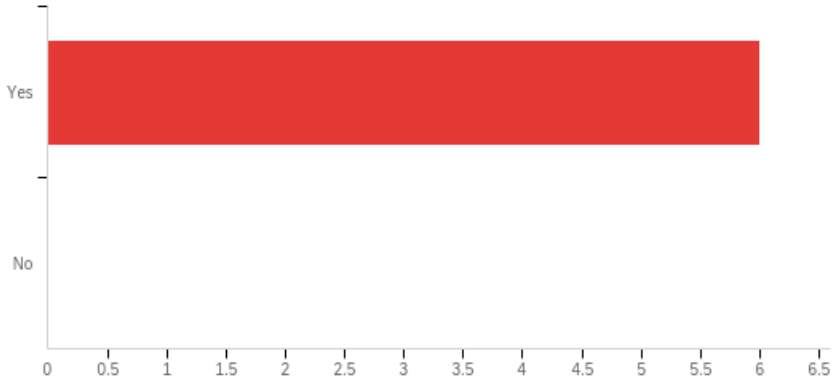
Is there anything else you would like to add, or any other features you would like to suggest that were not mentioned? If not, you can skip this question and submit your responses.

9.4.1.1.8 End of questionnaire

The final page once again gave contact details and allowed participants a chance to download a copy of the information sheet for themselves.

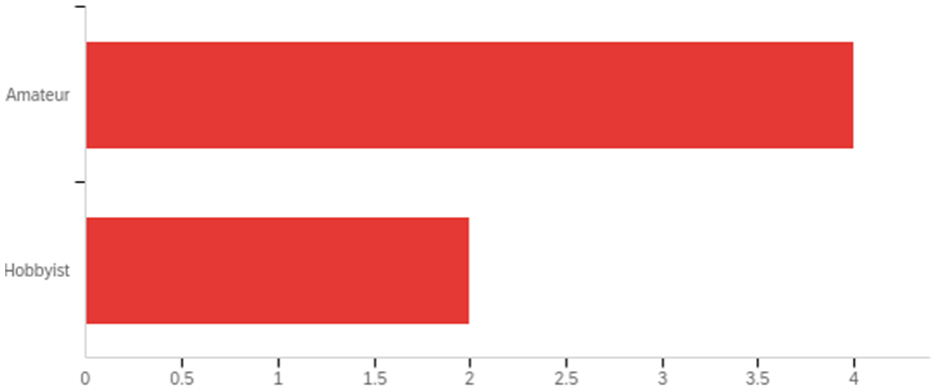
9.4.1.2 RESULTS

9.4.1.2.1 Q1



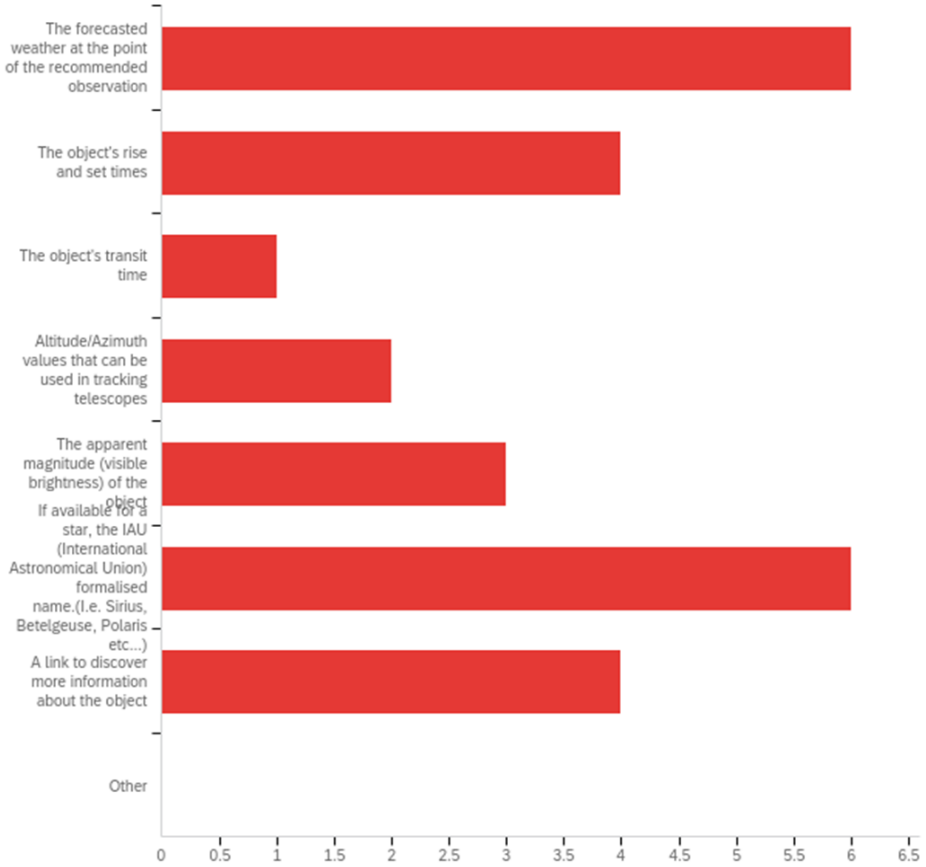
#	Answer	%	Count
1	Yes	100.00%	6
2	No	0.00%	0
	Total	100%	6

9.4.1.2.2 Q2



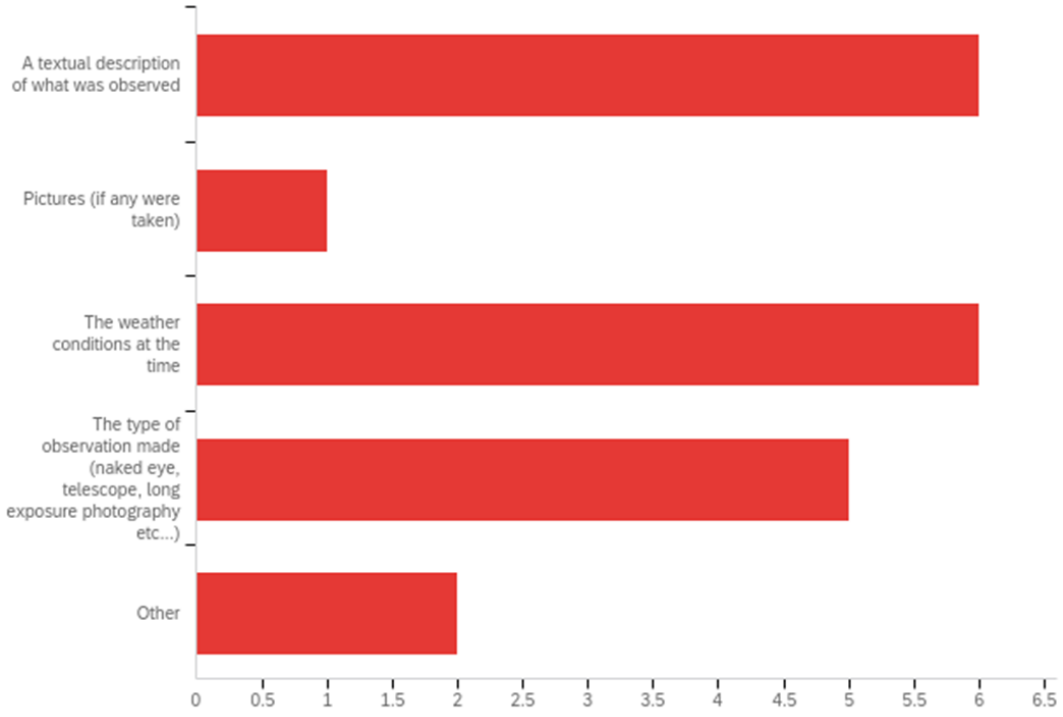
#	Answer	%	Count
1	Amateur	66.67%	4
2	Hobbyist	33.33%	2
	Total	100%	6

9.4.1.2.3 Q3



#	Answer	%	Count
1	The forecasted weather at the point of the recommended observation	23.08%	6
2	The object's rise and set times	15.38%	4
3	The object's transit time	3.85%	1
4	Altitude/Azimuth values that can be used in tracking telescopes	7.69%	2
5	The apparent magnitude (visible brightness) of the object	11.54%	3
6	If available for a star, the IAU (International Astronomical Union) formalised name. (I.e. Sirius, Betelgeuse, Polaris etc...)	23.08%	6
7	A link to discover more information about the object	15.38%	4
8	Other	0.00%	0
	Total	100%	26

9.4.1.2.4 Q4



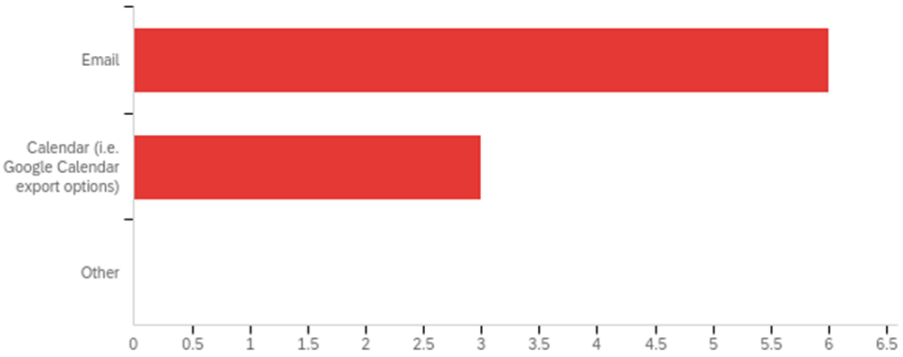
#	Answer	%	Count
1	A textual description of what was observed	30.00%	6
2	Pictures (if any were taken)	5.00%	1
3	The weather conditions at the time	30.00%	6
4	The type of observation made (naked eye, telescope, long exposure photography etc...)	25.00%	5
5	Other	10.00%	2
	Total	100%	20

Other – Text
Who made the observation
Where the observation took place

9.4.1.2.5 Q5

#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Days between each report	3.00	4.00	3.17	0.37	0.14	6

9.4.1.2.6 Q6



#	Answer	%	Count
1	Email	66.67%	6
2	Calendar (i.e. Google Calendar export options)	33.33%	3
3	Other	0.00%	0
	Total	100%	9

9.4.1.2.7 Q7

Is there anything else you would like to add, or any other features you would like to suggest that were not mentioned? If not, you can skip this question and submit your responses.
There are lots of stars, so maybe a filter for for popular or Very bright ones would help people
A feature that lets you 'like' stars so you can access them faster in the future
I think a facility for searching through available stars etc would be useful when setting up an observation

9.4.2 REQUIREMENTS PRIORITISATION

Requirements were prioritised using the MoSCoW method: **M**ust have, **S**hould have, **C**ould have, and **W**ill not have (this time).

9.4.2.1.1 MUST HAVE

[1] Observation reports must contain the forecasted weather.

[2] Observation reports must contain the IAU's formalized name if available.

[3] Observation logs must be able to contain a textual description of what was observed.

[4] Observation logs must be able to contain the weather at the time of the observation.

[5] Observation logs must be able to contain who carried out the observation.

[6] Observation logs must be able to contain where the observation took place.

[7] Observation reports must be generated every ~3 days.

[8] Observation reports must be exported via email.

9.4.2.1.2 SHOULD HAVE

[1] Observation reports should contain the object's rise and set times.

[2] Observation reports should contain a link to discover more information about the object.

[3] Observation logs should be able to contain the type of observation made.

[4] Observation reports should contain calendar export options.

[5] A filter for popular/bright stars should be available when setting up observations.

9.4.2.1.3 COULD HAVE

[1] Observation reports could contain altitude and azimuth values for tracking telescopes.

[2] Observation reports could contain the brightness of an astronomical object.

[3] A feature for 'liking' stars could be included.

9.4.2.1.4 WILL NOT HAVE (THIS TIME)

[1] Observation reports will not contain the object's transit time.

[2] Observation logs will not be able to contain pictures.

9.4.3 VOLERE REQUIREMENTS SPECIFICATIONS

The following Volere snow cards only cover the 'must have' and 'should have' MoSCoW categories.

#1	**Type:** Functional requirement
Description: Observation reports must contain the forecasted weather at the point of observation	
Rationale: Forecasts will add context to reports when users are looking at them	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When viewing an observation report, the weather forecast at the time the report was generated for should be present	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#2	**Type:** Functional requirement
Description: Observation reports must contain the IAU's formalised name if it is available	
Rationale: The International Astronomical Union's formalised names (i.e., Polaris, Sirius etc…) are often far more recognisable than the numeric designations otherwise used.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When viewing an observation report, if available, the IAU formalised star name must be present.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#3	**Type:** Functional requirement
Description: Observation logs must be able to contain a textual description of what was observed	
Rationale: This gives users a chance to write, in their own words, what they saw during their observation – allowing much deeper records to be taken.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When taking an observation log, users should be presented with a blank text entry area where they can enter whatever they wish about their observation.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#4	**Type:** Functional requirement
Description: Observation logs must be able to contain the weather at the time of the observation	
Rationale: When users are looking back at previous observation logs, a weather forecast will allow them to have more context about what conditions were like at the point in time that the logged observation was taking place.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When taking an observation log, the weather forecast from the observation must be able to be included.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#5	**Type:** Functional requirement
Description: Observation logs must be able to contain who carried out the observation	
Rationale: When carrying out an observation, the user may not be alone. When taking a log, they should be able to record who carried out the observation – this could be as simple as a list of names.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When taking an observation log, an entry area for the name(s) of the observer's must be present.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#6	**Type:** Functional requirement
Description: Observation logs must be able to contain where the observation took place	
Rationale: As with containing a weather forecast at the point of observation, adding the location will add much needed context when users are looking back through their logs.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When taking an observation log, the observation should be able to be included.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#7	**Type:** Functional requirement
Description: Observation reports must be generated every ~3 days	
Rationale: As Earth rotates about its axis and orbits the sun, and the weather forecasts change day to day, generated reports will only be so accurate ahead of time. Generating them roughly every 3 days will provide a good combination of advance planning and accuracy.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: Roughly every 3 days, a new set of observation reports will be generated for every user.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#8	**Type:** Functional requirement
Description: Observation reports must be exported via email	
Rationale: Exporting to an easy access external medium will allow users to access their reports quickly after they have been created.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: After creating a new observation, users should receive reports in email form every ~3 days.	
Customer satisfaction: 5	**Customer dissatisfaction:** 5
Priority: High	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#9	**Type:** Functional requirement
Description: Observation reports should contain the object's rise and set times	
Rationale: Even if users do not manage to observe the object at the recommended time, having a rise and set time will give them more ability to judge the length of time available to them either side of the recommended point that may also be used if need be.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: In observation reports, the rise and set time of each object should be calculated and included	
Customer satisfaction: 4	**Customer dissatisfaction:** 3
Priority: Medium	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

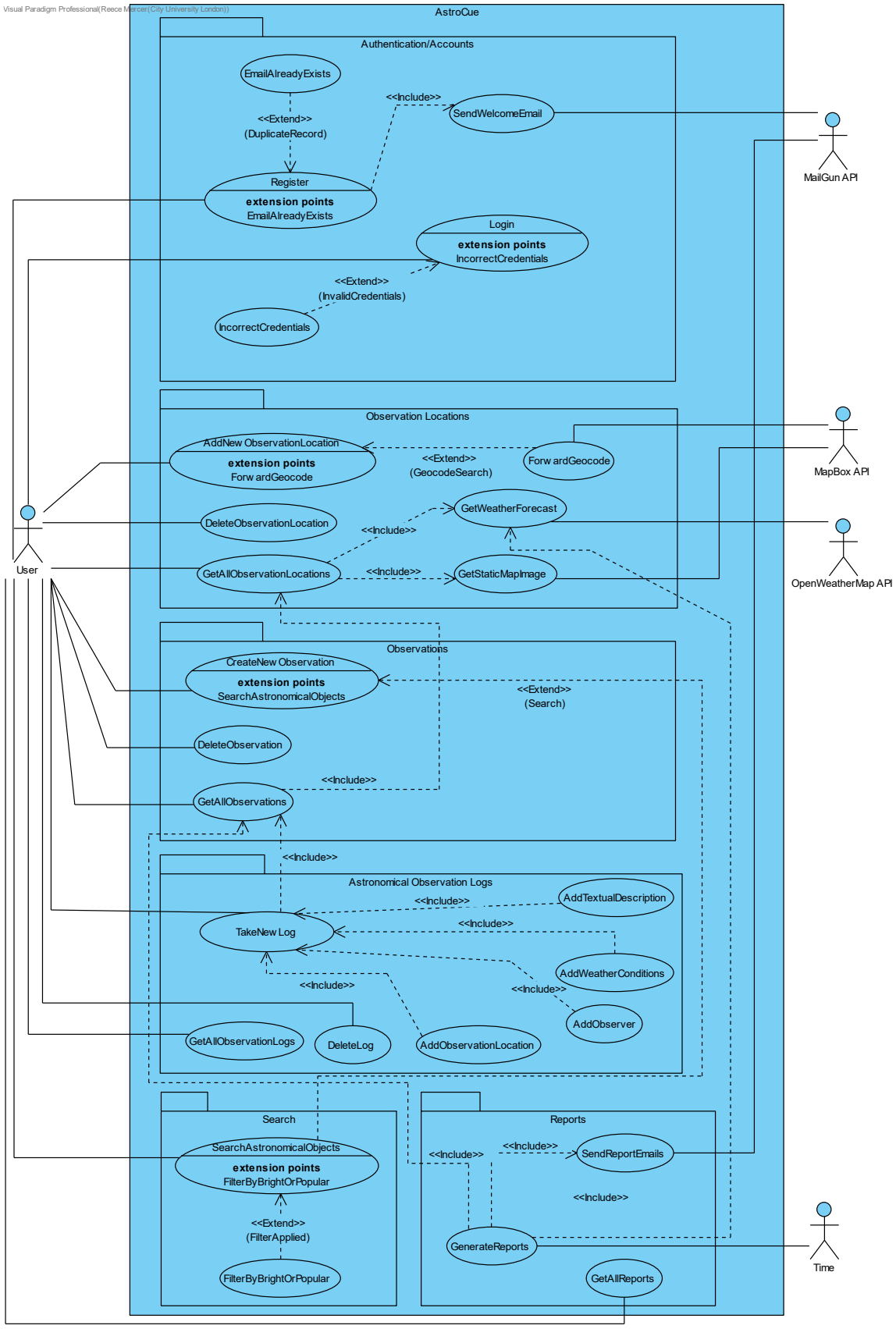
#10	**Type:** Functional requirement
Description: Observation reports should contain a link to discover more information about the object.	
Rationale: While some information can be provided in the reports, they are not a place to discover all the information about the object. A link to an external database (such as CDS/Simbad) should be included for further research to be carried out.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: In reports, an external link should be present that users can visit to discover more information about the object.	
Customer satisfaction: 4	**Customer dissatisfaction:** 3
Priority: Medium	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#11	**Type:** Functional requirement
Description: Observation logs should be able to contain the type of observation made	
Rationale: Not all observations are alike. Some are naked eye, some are through telescopes/binoculars, and others are via long exposure photography. The type of observation a user makes should be able to be recorded	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: In an observation report, users should be able to take note of the type of observation that was made.	
Customer satisfaction: 4	**Customer dissatisfaction:** 3
Priority: Medium	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#12	**Type:** Functional requirement
Description: Observation reports should contain calendar export options	
Rationale: On top of email exports, calendar options will also allow quick report access. Push notifications could also be configured if the user wishes.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: Each report should contain an option to export to a user's calendar (specifically Google Calendar, but others can be integrated without major difficulty).	
Customer satisfaction: 4	**Customer dissatisfaction:** 3
Priority: Medium	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

#13	**Type:** Functional requirement
Description: A filter for popular/bright stars should be available when setting up observations	
Rationale: When setting up an observation, there will be many stars/galaxies etc… to choose from. A filter for well-known/bright stars (and other objects) such as the north star (Polaris), brightest star (Sirius), popular galaxies (Andromeda) etc… will aid in making the choice easier.	
Source/originator: Anonymous questionnaire and subsequent requirements prioritisation	
Fit criterion: When picking an astronomical object for an observation, there should be an option to filter down to bright and well-known objects.	
Customer satisfaction: 5	**Customer dissatisfaction:** 3
Priority: Medium	**Conflicts:** None
Supporting Materials: None	Volere Copyright © Atlantic Systems Guild
History: None	

9.4.4 USE CASE DIAGRAM



9.4.5 USE CASE SPECIFICATIONS

Please note that the *test section* numbers are clickable in the PDF to quickly hop to the relevant use case tests.

Use case: Register	Test section 9.9.3.1
ID: 1	
Brief description: Allows the user to register a new account with AstroCue	
Primary actors: User	
Secondary actors: MailGun API	
Preconditions: None	
Main flow: 1. User enters an email and password 2. System checks that no account exists with the same email 3. Extension point: DuplicateRecord 4. Include (SendWelcomeEmail) 5. Save new account to database	
Postconditions: 1. The system has stored the user's new account in the database 2. A welcome email has been sent	
Alternative flows: None	

Extension use case: EmailAlreadyExists	Test section 9.9.3.2
ID: 1.1	
Brief description: Segment 1: User is told that an account with the provided email already exists	
Primary actors: User	
Secondary actors: None	
Segment 1 preconditions: 1. A user has entered an email that already belongs to a registered account	
Segment 1 flow: 1. The system notifies the user that the email is already in use	
Segment 1 postconditions: An error has been displayed	

Use case: Login	Test section 9.9.3.4
ID: 2	
Brief description: Allows the user to login to AstroCue	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user has an account registered	
Main flow: 1. User enters email and password 2. System validates user details against database 3. Extension point: InvalidCredentials	
Postconditions: 1. The user has been logged in	
Alternative flows: None	

Extension use case: IncorrectCredentials	Test section 9.9.3.3
ID: 2.1	
Brief description: Segment 1: Notify the user that either the email or password they entered is incorrect	
Primary actors: User	
Secondary actors: None	
Segment 1 preconditions: 1. The user has entered an email and password combination that does not correspond to an existing AstroCue account	
Segment 1 flow: 1. The system notifies the user that the no account matches the details they have provided	
Segment 1 postconditions: 1. An error has been displayed	

Use case: SendWelcomeEmail	Test section 9.9.3.1
ID: 3	
Brief description: Sends an email to a newly registered user welcoming them to AstroCue	
Primary actors: User	
Secondary actors: MailGun API	
Preconditions: 1. The user has successfully registered a new account with a valid email address 2. The user is logged in	
Main flow: 1. The system uses the MailGun API to send a welcome email to the address the user used to register	
Postconditions: 1. An email has been sent via the MailGun API	
Alternative flows: None	

Use case: AddNewObservationLocation	Test section 9.9.3.5
ID: 4	
Brief description: Allows a user to add a new observation location to their account	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in	
Main flow: 1. User enters the observation location details 2. Extension point: GeocodeSearch 3. System saves new location to database	
Postconditions: 1. The user's new observation location is added to their account	
Alternative flows: None	

Extension use case: ForwardGeocode	Test section 9.9.3.6
ID: 4.1	
Brief description: Segment 1: Carry out a forward geocode search using the MapBox API. For example, turning the query 'Northampton Square, London' into 51.5274° N, 0.1022° W.	
Primary actors: User	
Secondary actors: MapBox API	
Segment 1 preconditions: 1. The user is logged in	
Segment 1 flow: 1. The user enters a text query 2. The query is validated 3. The query is sent to the MapBox API 4. The result is returned	
Segment 1 postconditions: 1. A MapBox Forward Geocode result has been returned to the user	

Use case: DeleteObservationLocation	Test section 9.9.3.7
ID: 5	
Brief description: Allows a user to delete an observation location from their account	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in	
Main flow: 1. User selects the observation location they want to delete 2. The system remove the location from the user's account	
Postconditions: 1. The chosen observation is removed from the account	
Alternative flows: None	

Use case: GetAllObservationLocations	Test section 9.9.3.5
ID: 6	
Brief description: Allows a user to retrieve all their observation locations from their account	
Primary actors: User	
Secondary actors: OpenWeatherMap API, MapBox API	
Preconditions: 1. The user is logged in	
Main flow: 1. User navigates to the observation locations page 2. The system retrieves all their observation locations 3. For each observation location 3.1. Include (GetWeatherForecast) 3.2. Include (GetStaticMapImage) 4. The system displays the observation locations	
Postconditions: 1. The user's observation locations have been retrieved from the database	
Alternative flows: None	

Use case: GetWeatherForecast	Test section 9.9.3.5
ID: 7	
Brief description: Retrieves a weather forecast for a given location from the OpenWeatherMap API	
Primary actors: User	
Secondary actors: OpenWeatherMap API	
Preconditions: 1. The user is logged in 2. The user has at least one observation location saved to their account	
Main flow: 1. Fetch a weather forecast for a given latitude and longitude value belonging to an observation location owned by a user	
Postconditions: 1. The user's observation has a weather forecast fetched for it from the OpenWeatherMap API	
Alternative flows: None	

Use case: GetStaticMapImage	Test section 9.9.3.5
ID: 8	
Brief description: Retrieves a static map image for a given location from the MapBox API	
Primary actors: User	
Secondary actors: MapBox API	
Preconditions: 1. The user is logged in 2. The user has at least one observation location saved to their account	
Main flow: 1. Fetch a static map image for a given latitude and longitude value belonging to an observation location owned by a user	
Postconditions: 1. The user's observation has a static map image fetched for it from the MapBox API	
Alternative flows: None	

Use case: CreateNewObservation	Test section 9.9.3.8
ID: 9	
Brief description: Allows a user to set up a new astronomical observation	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation location saved to their account	
Main flow: 1. The user enters the details of their new astronomical observation (location, object etc…) 2. Extension point: Search 3. The system saves the new observation to the database	
Postconditions: 1. The new astronomical observation has been saved to the database	
Alternative flows: None	

Use case: DeleteObservation	Test section 9.9.3.13
ID: 10	
Brief description: Allows a user to delete an existing astronomical observation	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account	
Main flow: 1. The user chooses an observation to delete 2. The system deletes the observation from the user's account	
Postconditions: 1. The chosen observation has been removed from the user's account	
Alternative flows: None	

Use case: GetAllObservations	Test section 9.9.3.8
ID: 11	
Brief description: Allows a user to see all the astronomical observations they have set up	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account	
Main flow: 1. The user navigates to the observations page 2. The system retrieves all the observations that the user has set up 3. Include (GetAllObservationLocations) 4. The system displays the observations	
Postconditions: 1. All the user's observations have been retrieved from the database and displayed	
Alternative flows: None	

Use case: TakeNewLog	Test section 9.9.3.11
ID: 12	
Brief description: Allows a user to take a new astronomical observation log	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it	
Main flow: 1. The user navigates to the reports page 2. The user choses a report to take a log against 3. Include (AddTextualDescription, AddWeatherConditions, AddObserver, AddObservationLocation) 4. The system stores the log in the database	
Postconditions: 1. The user's log has been stored in the database	
Alternative flows: None	

Use case: AddTextualDescription	Test section 9.9.3.11
ID: 13	
Brief description: Allows a user to add a text description of what they observed to an astronomical log	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it 4. The user has started a new astronomical log	
Main flow: 1. The user types in plain English what they observed. 2. The system attaches the text description to the log	
Postconditions: 1. The user's log has a plain text description attached to it	
Alternative flows: None	

Use case: AddWeatherConditions	Test section 9.9.3.11
ID: 14	
Brief description: Allows weather conditions to be included inside an astronomical log	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it 4. The user has started a new astronomical log	
Main flow: 1. The user selects weather conditions to be included 2. The system attaches weather conditions from the report to the log	
Postconditions: 1. The user's log has the weather forecast attached to it	
Alternative flows: None	

Use case: AddObserver	Test section 9.9.3.11
ID: 15	
Brief description: Allows users to record who carried out an observation	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it 4. The user has started a new astronomical log	
Main flow: 1. The user enters names of the observers 2. The system adds the names onto the log	
Postconditions: 1. The user's log has observer's name(s) attached to it	
Alternative flows: None	

Use case: AddObservationLocation	Test section 9.9.3.11
ID: 16	
Brief description: Allows users to log where an observation was carried out	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it 4. The user has started a new astronomical log	
Main flow: 1. The user selects the observation location to be recorded 2. The system adds the observation location to the log	
Postconditions: 1. The user's log has the observation location attached to it	
Alternative flows: None	

Use case: DeleteLog	Test section 9.9.3.12
ID: 17	
Brief description: Allows users to delete an astronomical log from the system	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it 4. The user has recorded a log against the report	
Main flow: 1. The user selects the log for deletion 2. The system deletes the log from the database	
Postconditions: 1. The selected log has been removed from the database	
Alternative flows: None	

Use case: GetAllObservationLogs	Test section 9.9.3.11
ID: 18	
Brief description: Allows users to retrieve all the astronomical logs they have taken	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation saved to their account 3. The observation has had at least 1 report generated for it 4. The user has recorded a log against the report	
Main flow: 1. The user navigates to the astronomical log page 2. The system retrieves all the user's astronomical logs from the database	
Postconditions: 1. All the user's logs are displayed to them	
Alternative flows: None	

Use case: SearchAstronomicalObjects	Test section 9.9.3.8
ID: 19	
Brief description: Allows users to search the system for astronomical objects (stars, galaxies etc…)	
Primary actors: User	
Secondary actors: None	
Preconditions: 1. The user is logged in 2. The user has at least one observation location added to their account 3. The user has started to create a new observation	
Main flow: 1. The user enters a text query 2. Extension point: FilterApplied	
Postconditions: 1. The system returns any results matching the text query and the optional filter	
Alternative flows: None	

Extension use case: FilterByBrightOrPopular	Test section 9.9.3.9
ID: 19.1	
Brief description: Segment 1: Narrows down an astronomical object search result to just well-known and bright objects	
Primary actors: User	
Secondary actors: None	
Segment 1 preconditions: 1. The user is logged in 2. The user is searching through astronomical objects	
Segment 1 flow: 1. The user applies the bright and popular object filter 2. The system applies the filter on top of the user's original text query	
Segment 1 postconditions: 1. The filtered search results are returned to the user	

Use case: GenerateReports	Test section 9.9.3.10
ID: 20	
Brief description: Allows users to retrieve all the astronomical logs they have taken	
Primary actors: Time	
Secondary actors: MailGun API	
Preconditions: 1. There is at least one observation set up in the database	
Main flow: 1. This use case is triggered every ~3 days 2. Include (GetAllObservations) 3. For every observation 3.1. Generate reports 3.2. Include (SendReportEmails) 3.3. Store report in database	
Postconditions: 1. All the observations have had reports generated for them and the emails have been sent	
Alternative flows: None	

Use case: SendReportEmails	Test section 9.9.3.10
ID: 21	
Brief description: Sends a set of reports generated for a given user to their registered email address	
Primary actors: MailGun API	
Secondary actors: None	
Preconditions: 1. There is at least one observation set up in the database	
Main flow: 1. Send the generated report to the user via their email address	
Postconditions: 1. All reports generated in a given run have been sent via the MailGun API to the user	
Alternative flows: None	

9.5 APPENDIX E: DESIGN

Design

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

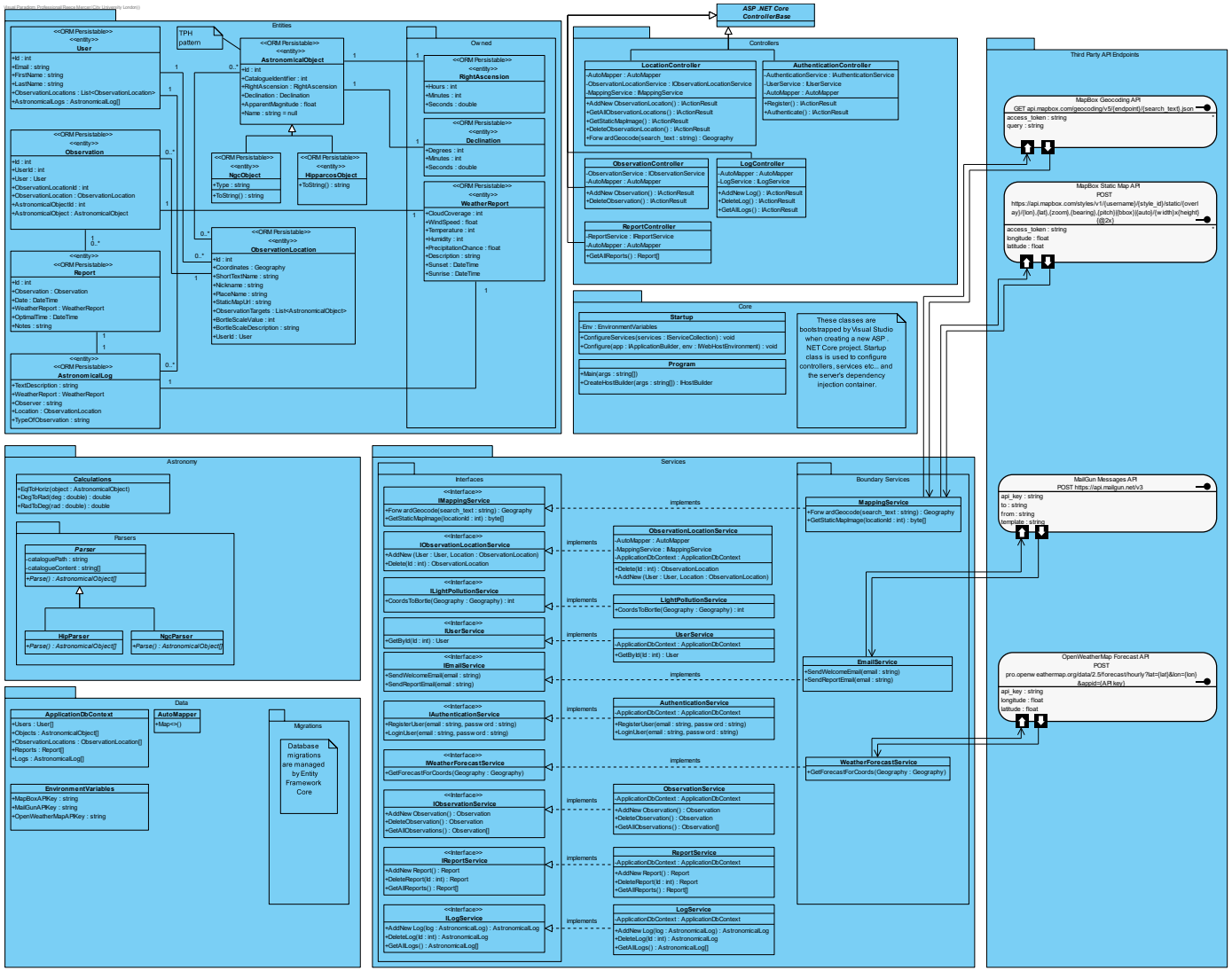
Consultant: [REDACTED]

9.5.1 IMPORTANT PREAMBLE

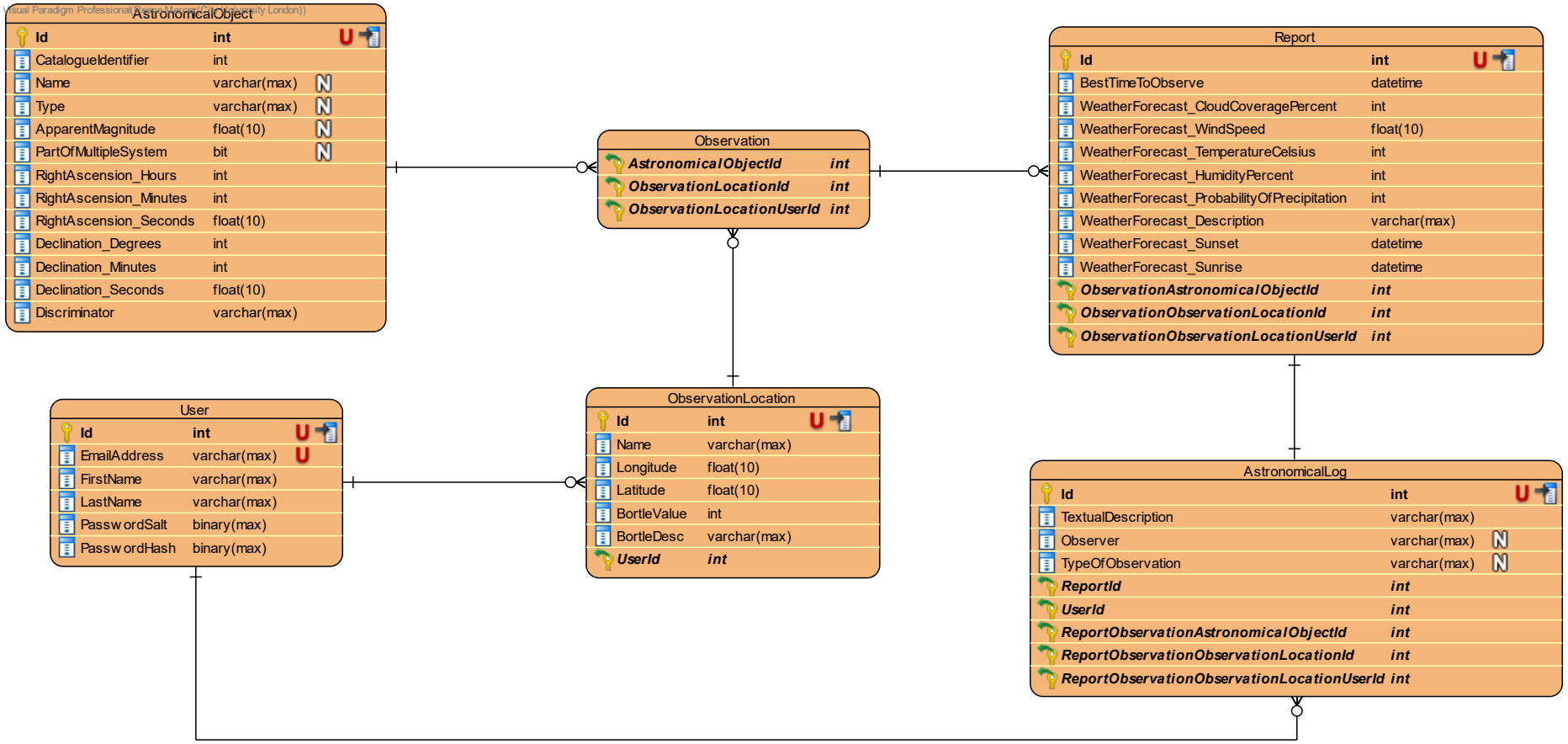
All large design diagrams in this document are in SVG format. Zoom in on the PDF a few hundred percent for more detail; quality will not be lost.

Diagrams begin overleaf.

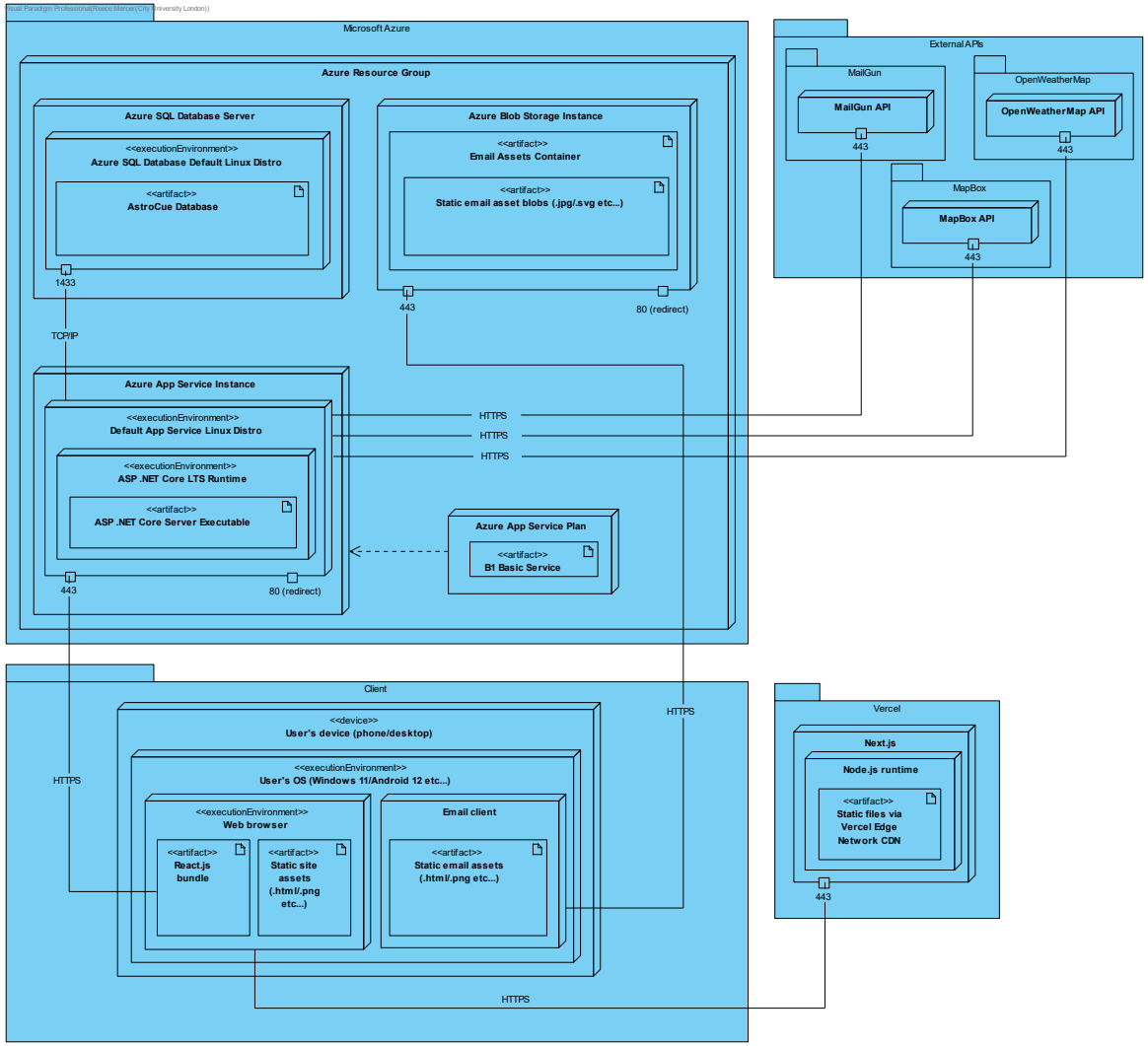
9.5.2 CLASS DIAGRAM



9.5.3 ENTITY RELATIONSHIP DIAGRAM



9.5.4 DEPLOYMENT DIAGRAM



9.5.5 AstroCue logo and font

The AstroCue logo (seen on the front page of this document and on each section's title), was made by the author using a combination of a free to use font and icon. Both light and dark versions were created:

9.5.5.1 Saturn free icon

The icon element of the AstroCue logo was downloaded from flaticon.com and has the "Free for personal and commercial purpose with attribution" license (Flaticon, n.d.):

Required attribution: "*This logo has been designed using resources from Flaticon.com*" (Flaticon, n.d.).

9.5.5.2 Earth Orbiter font

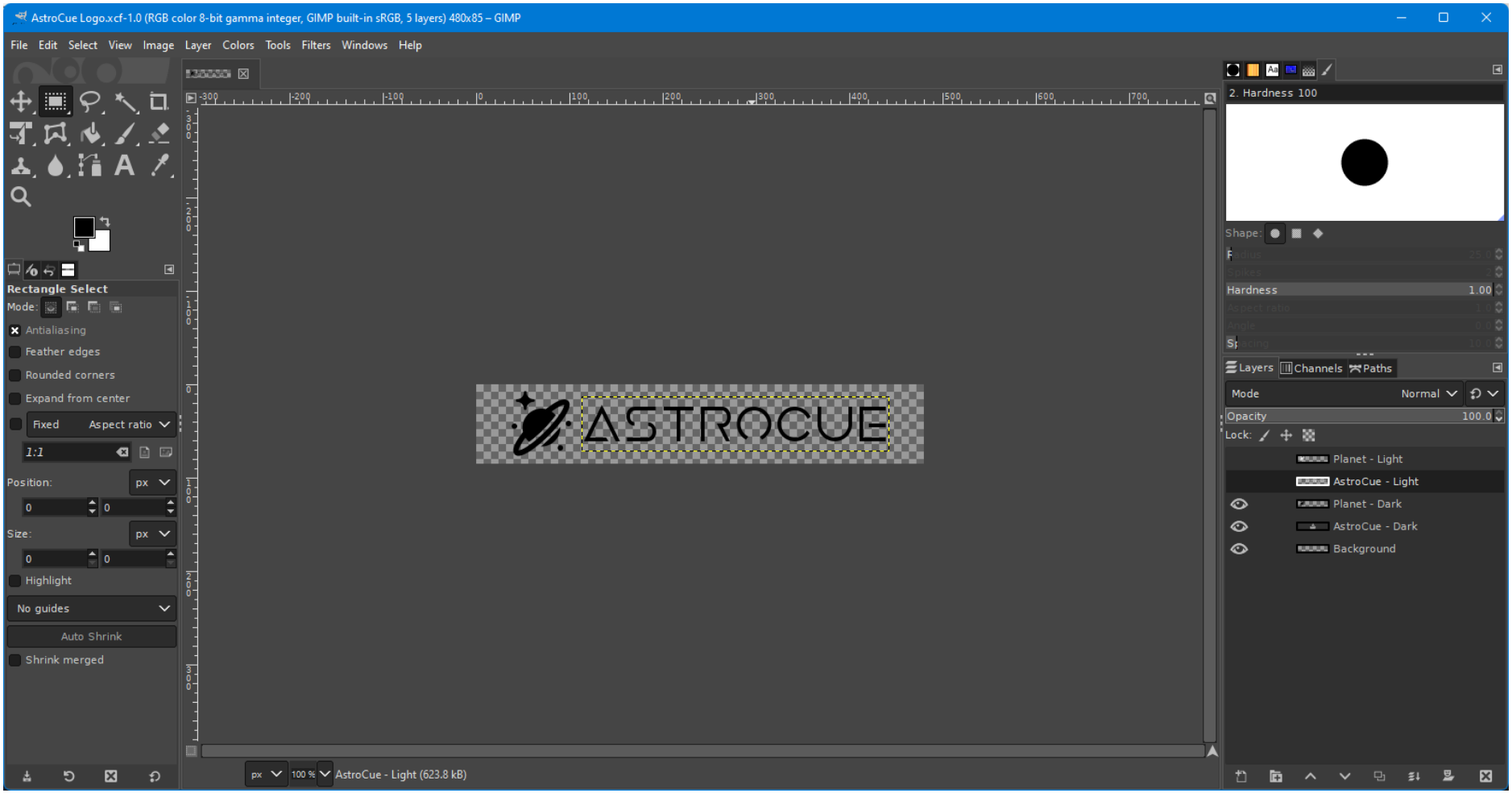
The font used in the logo is the 'Earth Orbiter Font' and was downloaded from 1001freefonts.com:

EARTH ORBITER | ASTROCUE

This font is free for non-commercial use (1001freefonts, n.d.).

9.5.5.3 Final logo creation

To create the final logos seen at the top of this page, the author combined the font and icon into a final single image using the GNU Image Manipulation Program (GIMP). This is shown on the next page.


AstroCue Logo.xcf-1.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 5 layers) 480x85 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Rectangle Select
Mode:
Antialiasing
Feather edges
Rounded corners
Expand from center
Fixed Aspect ratio
1:1
Position: px
Size: px
Highlight
No guides
Auto Shrink
Shrink merged
ASTROCUE
2. Hardness 100
Shape:
Hardness 1.00
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Planet - Light
AstroCue - Light
Planet - Dark
AstroCue - Dark
Background
px 100 % AstroCue - Light (623.8 kB)

9.6 APPENDIX F: DEPLOYMENT AND DOMAIN/DNS

Deployment and domain/DNS

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

Consultant: [REDACTED]

9.6.1 INTRODUCTION

This appendix covers how to deploy the backend to Azure, the frontend to Vercel, and how to set up the domain's DNS settings to work with the website client and MailGun's API.

Instructions begin overleaf.

9.6.2 DEPLOYING THE BACKEND TO AZURE

9.6.2.1 PREREQUISITES

- An Azure subscription with an available balance
- Docker Desktop and a Docker Hub account (https://www.docker.com/products/docker-desktop/)
- Visual Studio installed (preferably 2022 - https://visualstudio.microsoft.com/vs/)
- The server code, complete with Dockerfile, on your development system
- Azure Storage Explorer (https://azure.microsoft.com/en-gb/features/storage-explorer/)
- Microsoft SQL Server Management Studio (https://docs.microsoft.com/en-us/sql/ssms/)

9.6.2.2 INTRODUCTION

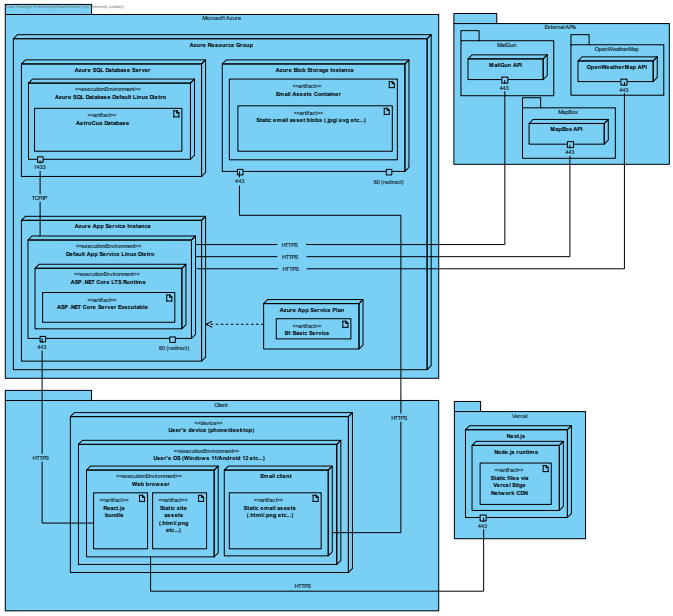
The backend, or 'server', component of AstroCue is deployed to Microsoft's cloud platform, Azure. It is comprised of 3 major elements:

1. Azure App Service instance running the server executable (via its Docker container)
2. Azure SQL instance to which the server can persist its data
3. Azure Blob storage instance where static email assets (such as images) are stored

Before any resources can be created in Azure, a **Resource Group** needs to be created. Think of this a logical container within which any cloud-based resources you spin up sit.

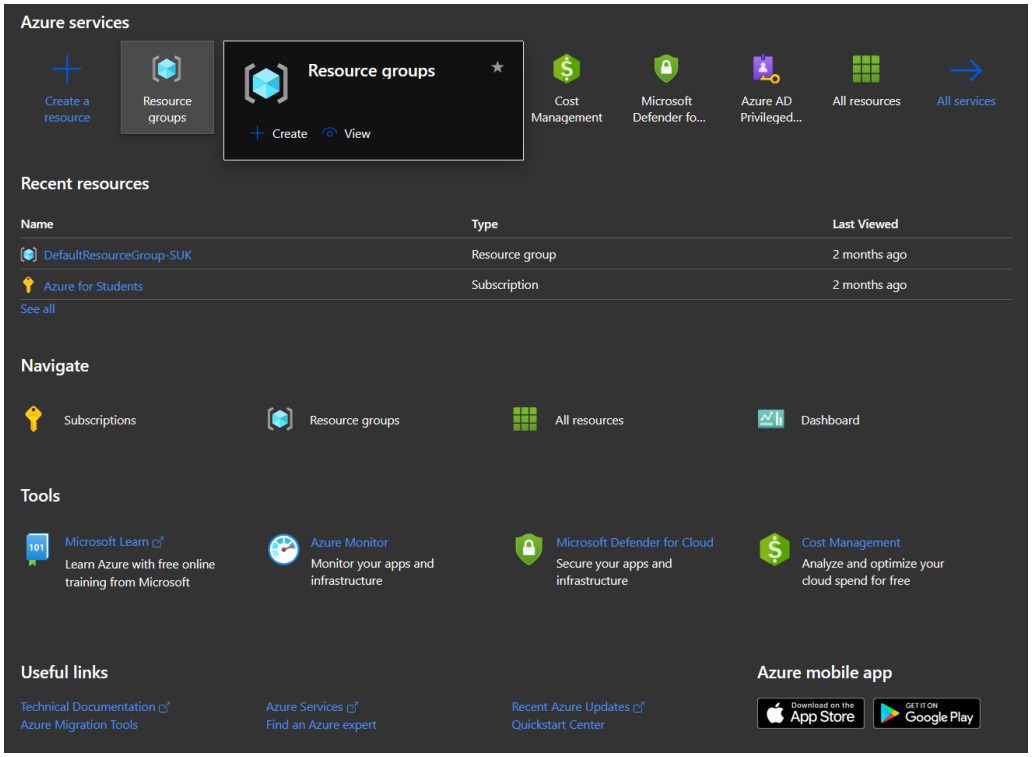
Before the deployment instructions begin overleaf, below is a copy of the deployment diagram from appendix item 9.5.4 for completeness.

Reminder that this is an SVG image, zoom in at least 300% to see details. Quality will not be lost.

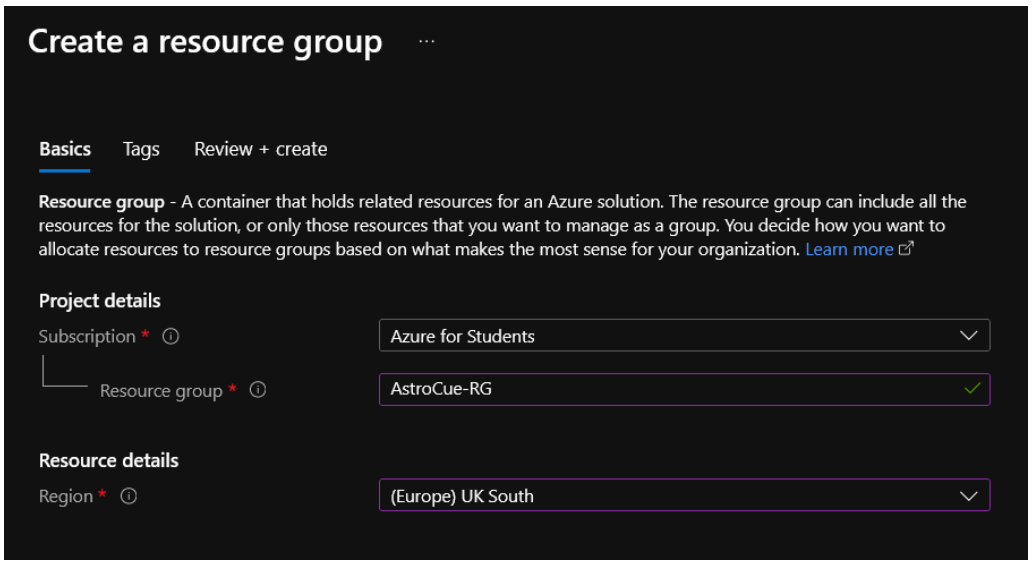


9.6.2.3 Creating a Resource Group

Head to the Azure Portal (https://portal.azure.com) and click the available prompt up the top to create a new Resource Group.

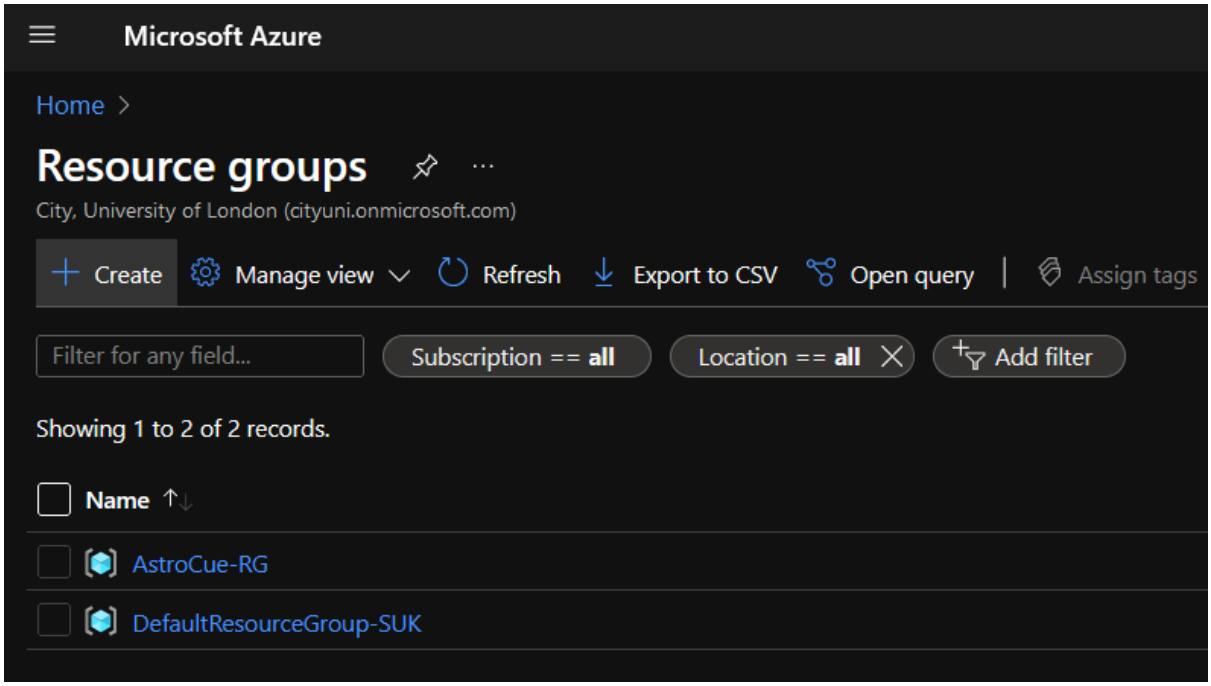


Enter a sensible name and choose a region (typically choose the country in which you are based and in which most of the users of the resources will be located).



Once you have created the Resource Group, it should be visible in the list of Resource Groups under your account. It is sometimes the case that you will have pre-existing Resource Groups. This is normal but creating a new one just for AstroCue itself is beneficial for the sake of keeping things neat.

Resource Groups themselves, if empty, incur no costs.

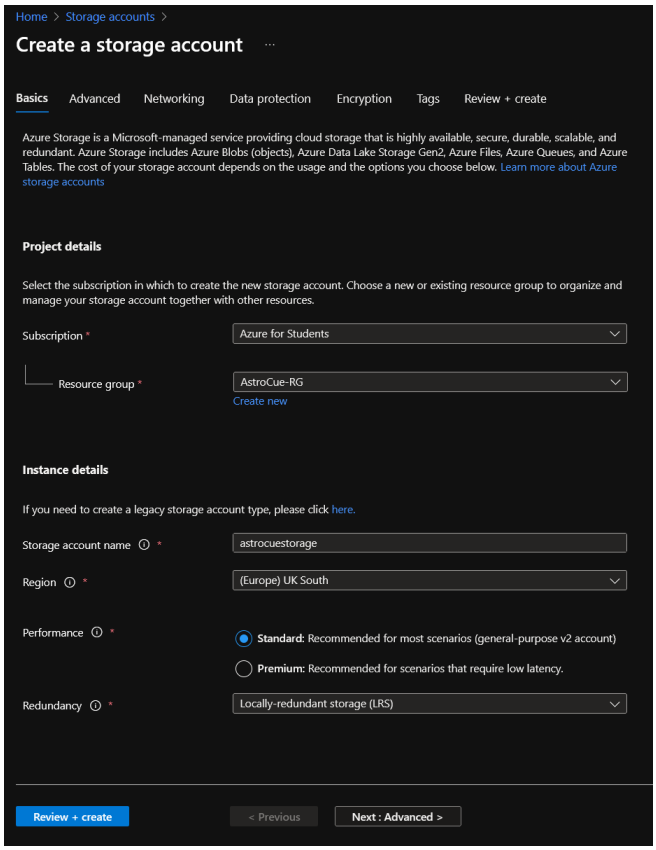


9.6.2.4 CREATING BLOB STORAGE FOR EMAIL ASSETS

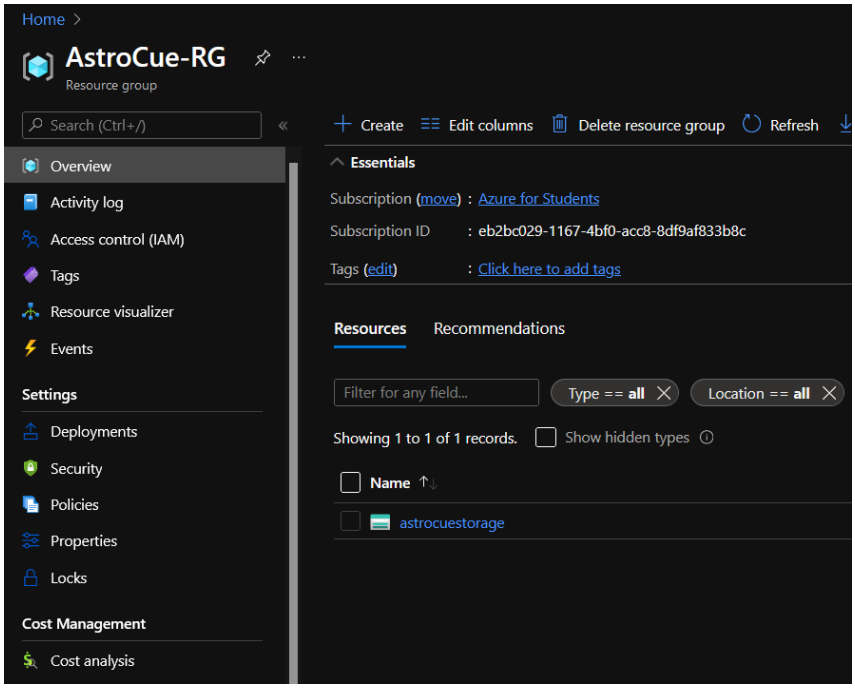
When you view emails in an email client, most of the time you are viewing HTML content. Media within these emails, (images etc…) are either inlined within the email itself or are retrieved from an external domain using regular HTML tags.

For AstroCue's emails, the templates were designed to retrieve their assets from an external domain. To this end, blob storage was used to provide quick public access to the images that were needed.

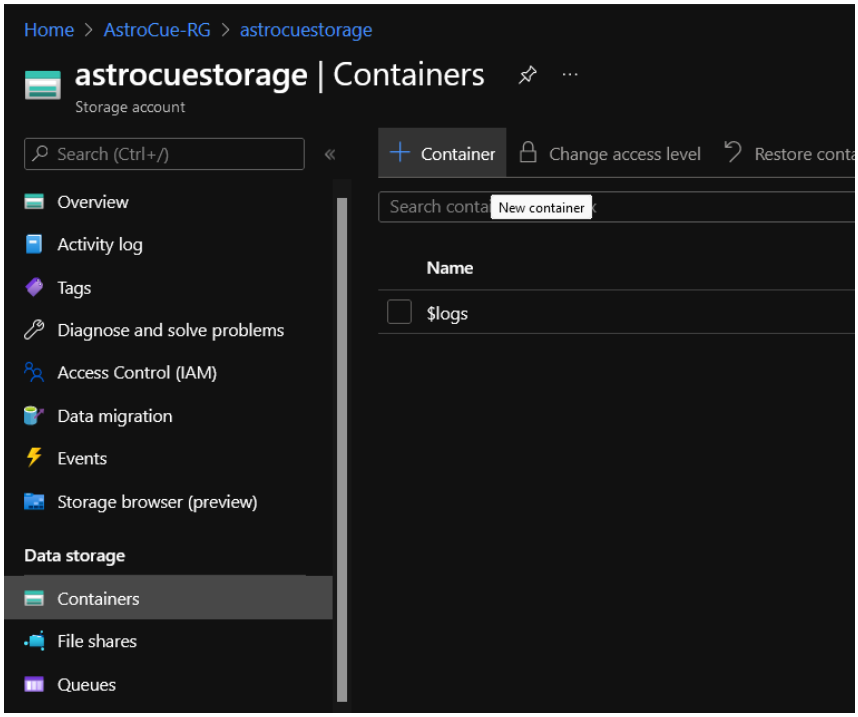
Begin creating a new storage account – this is the parent of any blob storage instances you will create. When creating this resource, ensure that you create it under the Resource Group you created in the previous step. Again, enter a sensible name and pick the region. The standard performance and LRS redundancy should be left as default.



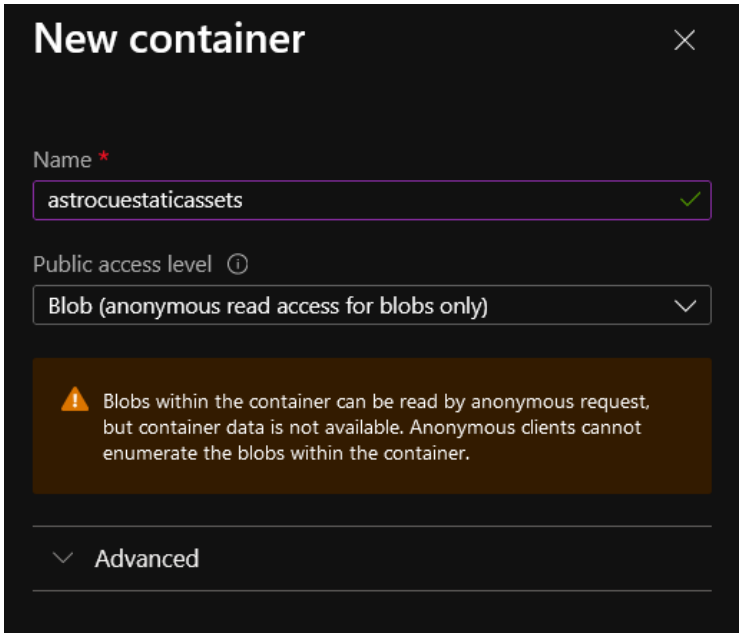
This new storage account should then appear as one of the resources within your Resource Group.



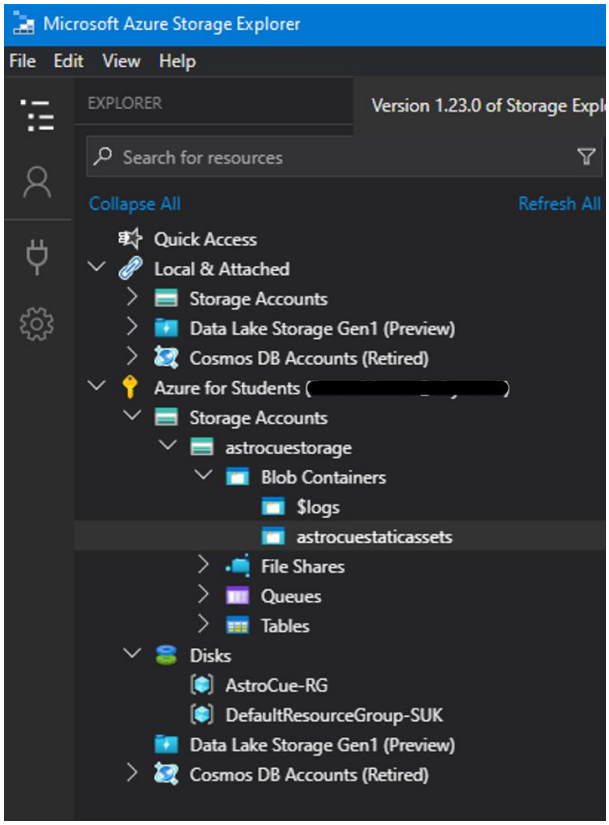
You can now create the blob storage instance itself. Click on the newly created storage account, and in the 'Containers' tab, click the new container button.



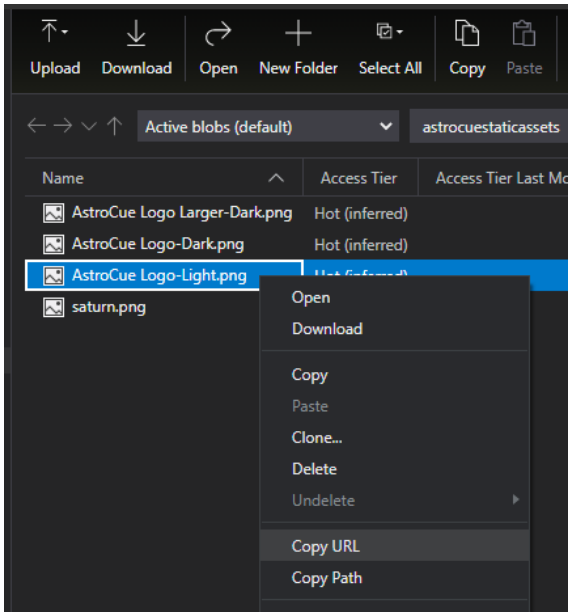
Give it a sensible name, and make sure that anonymous read access is enabled, else the public web will not be able to access the resources you add to it.



Open Azure Storage Explorer (linked under heading 9.6.2.1) and navigate to the blob containers under your storage account. The newly created container should be visible.



Using the storage explorer, upload any resources needed for the email templates (logo images, footer dividers etc…) into the container. You should be able to right click any of the uploaded resources and be able to copy their URL.

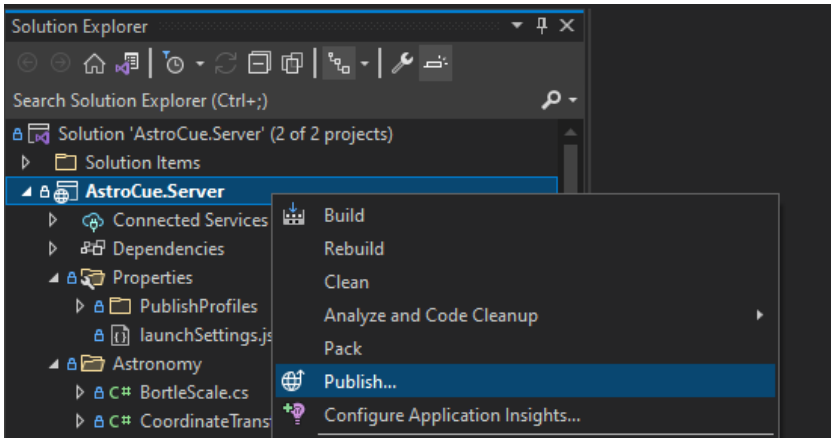


When you visit this URL in a browser, it should return the resource. As the blob container was set up with anonymous public access, this should work even in incognito tabs.

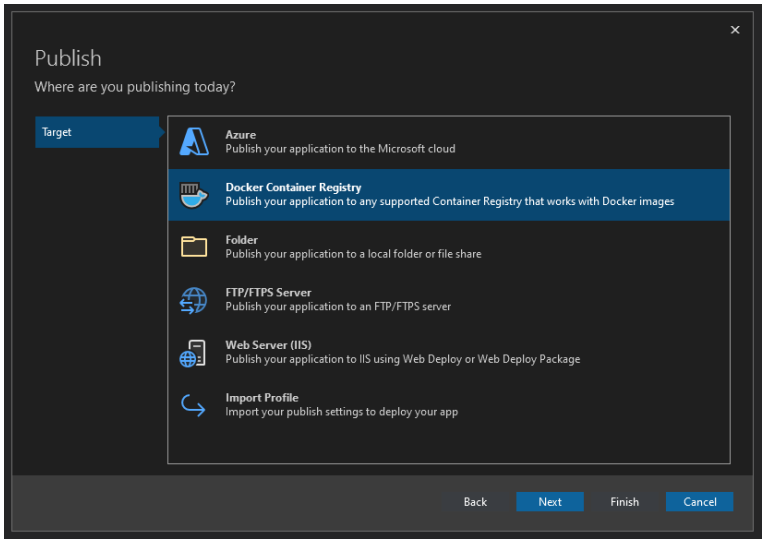
9.6.2.5 DEPLOYMENT OF THE SERVER AND ITS DATABASE

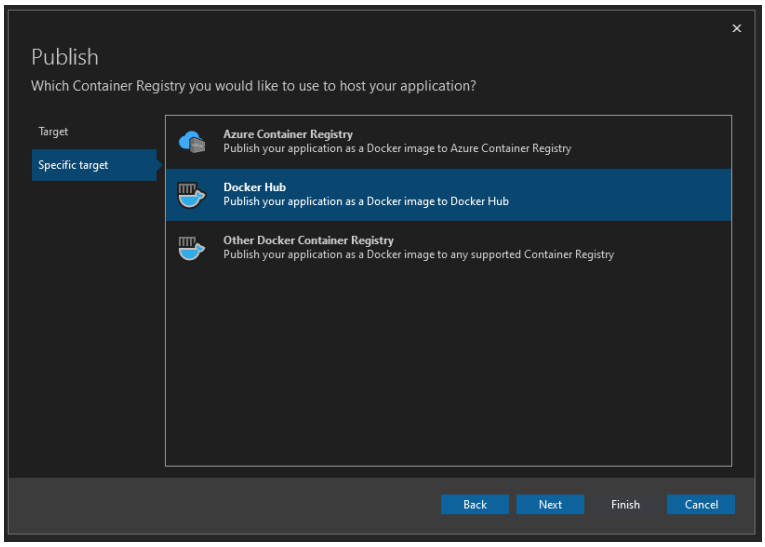
As mentioned, the server code is containerised using Docker. Because of this, instead of directly publishing to Azure via Visual Studio, the process is a little different.

Start by right clicking the server project inside Visual Studio and entering the 'Publish' menu.

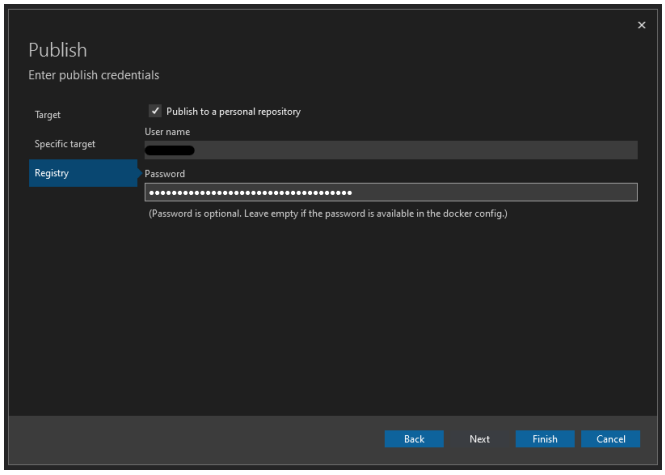


Then, choose the Docker Container Registry option, followed by Docker Hub itself.

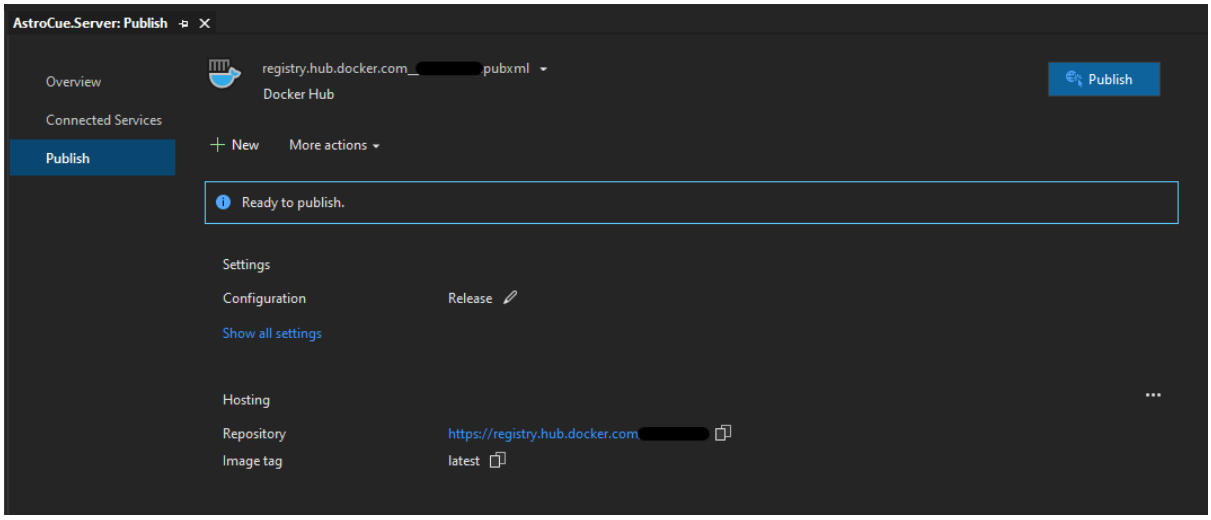




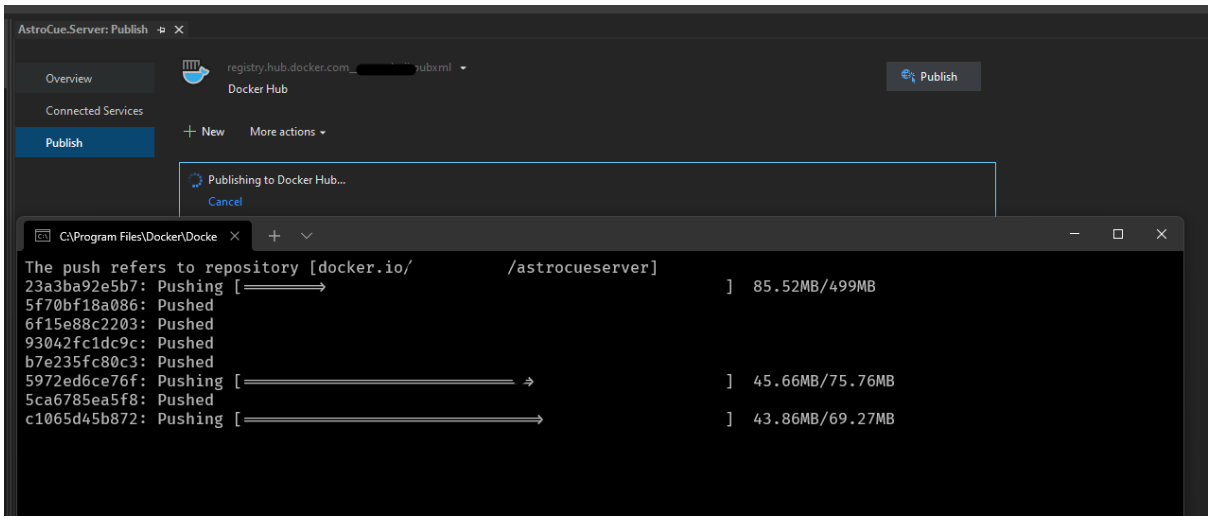
Enter your Docker Hub credentials. If you have 2 factor authentication, the password will be replaced by a personal access token that you can create on the Docker Hub website.



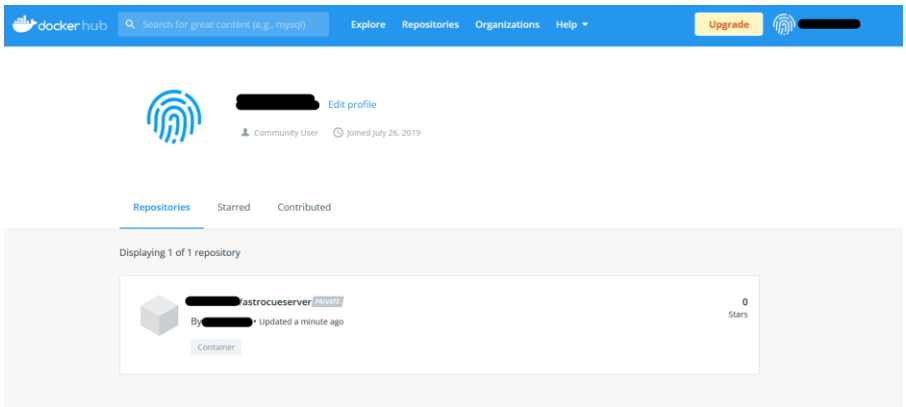
After this step, the deploy profile should be complete. Go ahead a click 'publish'.



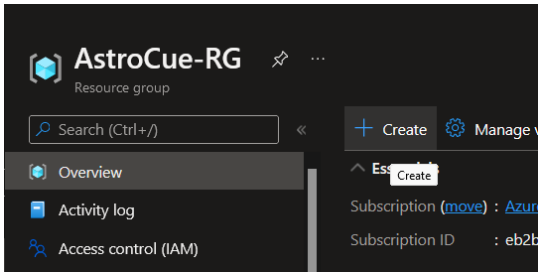
Docker should begin building a release version of the server container and publishing it under the 'latest' tag. As this is the first time doing this, it may take a while.



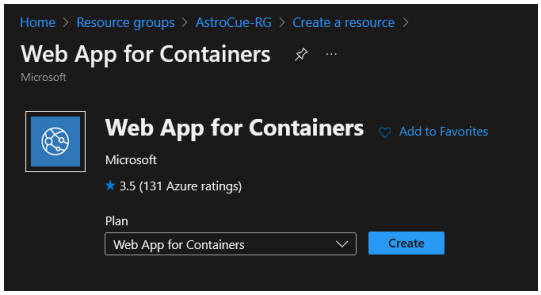
Once completed, it should be visible under your Docker Hub account.



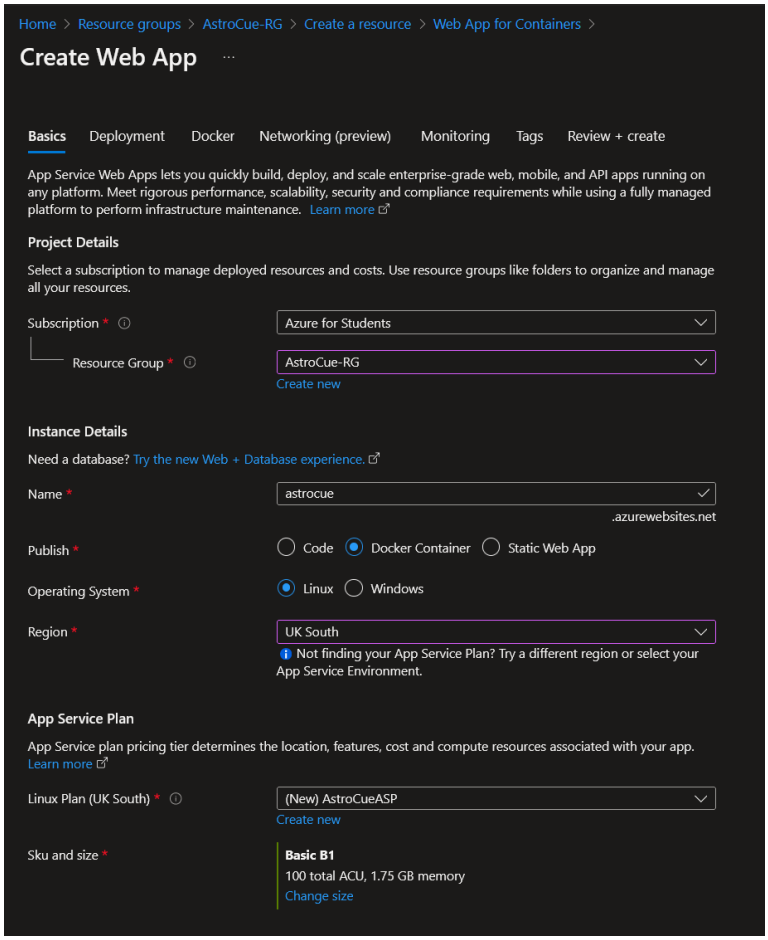
Head back to the Azure Portal, and inside the new Resource Group, go to create a new resource.



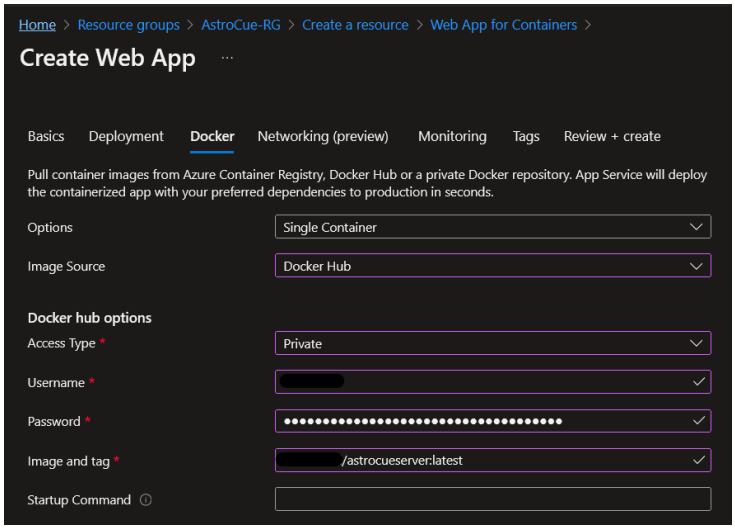
Search for 'Web App for Containers' and click create.



Enter a sensible name and fill out the 'Basics' tab with the following details.

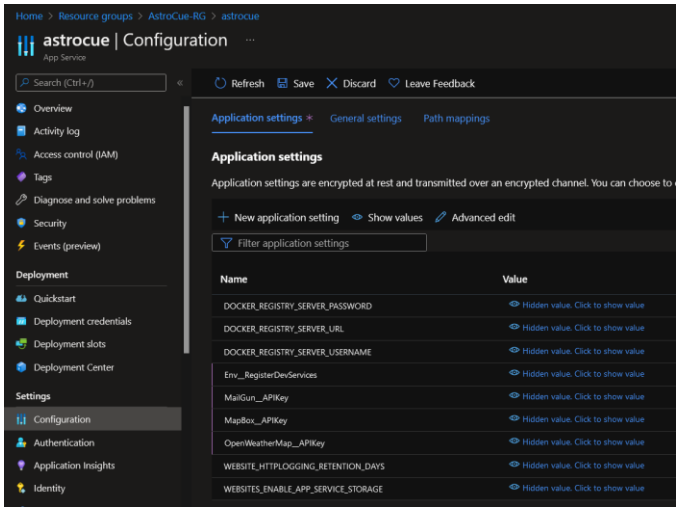


Under the 'Docker' tab, select Docker Hub as your image source. Enter your credentials again and enter the image and tag name in the format `<username/container_name:tag>`.

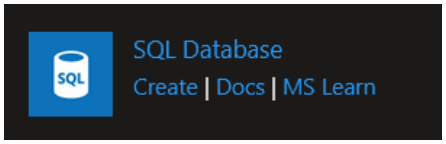


Once the app has been created, head on over to its configuration tab via the new Resource Group entry, and add the following application settings and values to the environment:

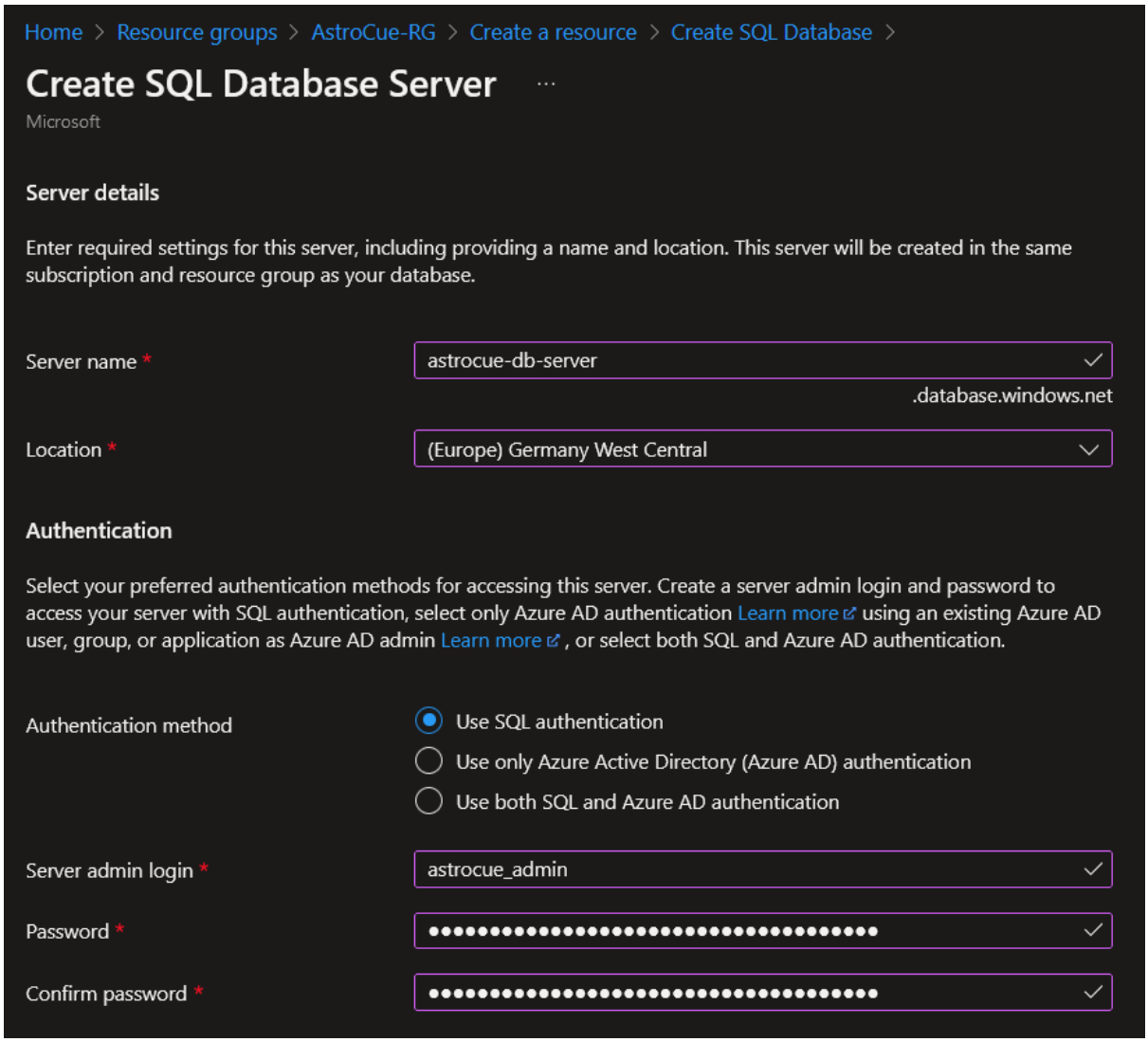
`Env__RegisterDevServices`	`False`
`MailGun__ApiKey`	<a valid MailGun API key>
`MapBox__ApiKey`	<a valid MapBox API key>
`OpenWeatherMap__ApiKey`	<a valid OWM API key>
`WEBSITES_CONTAINER_TIME_LIMIT`	1800
`AppSettings__JsonWebTokenSecret`	<a large (50-100 character) randomly generated string to use as a server-side JWT encryption secret>



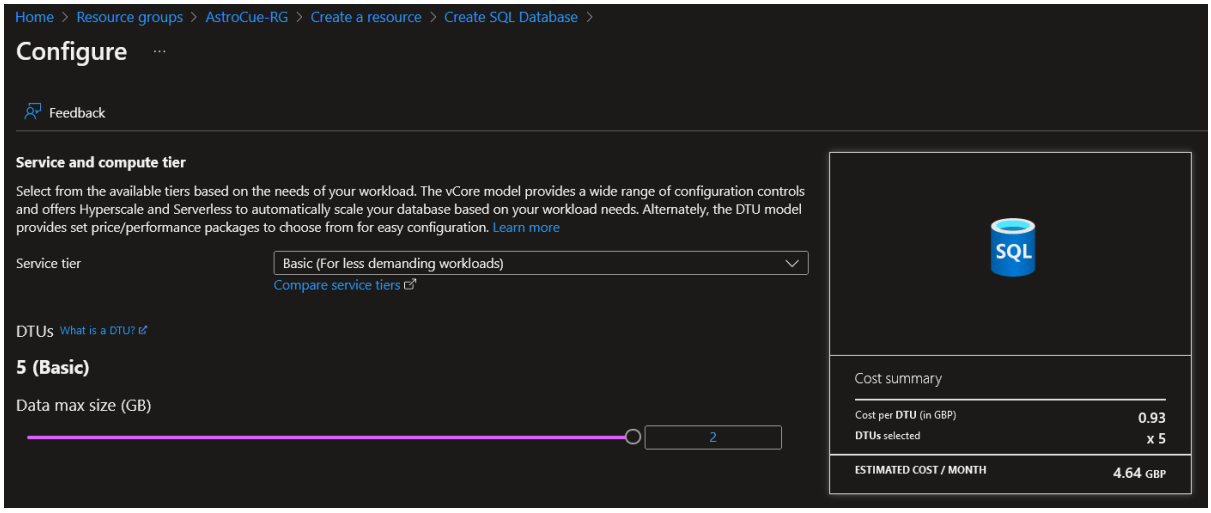
Head back to the Azure Portal and create another new resource. This time, search for 'SQL Database'.



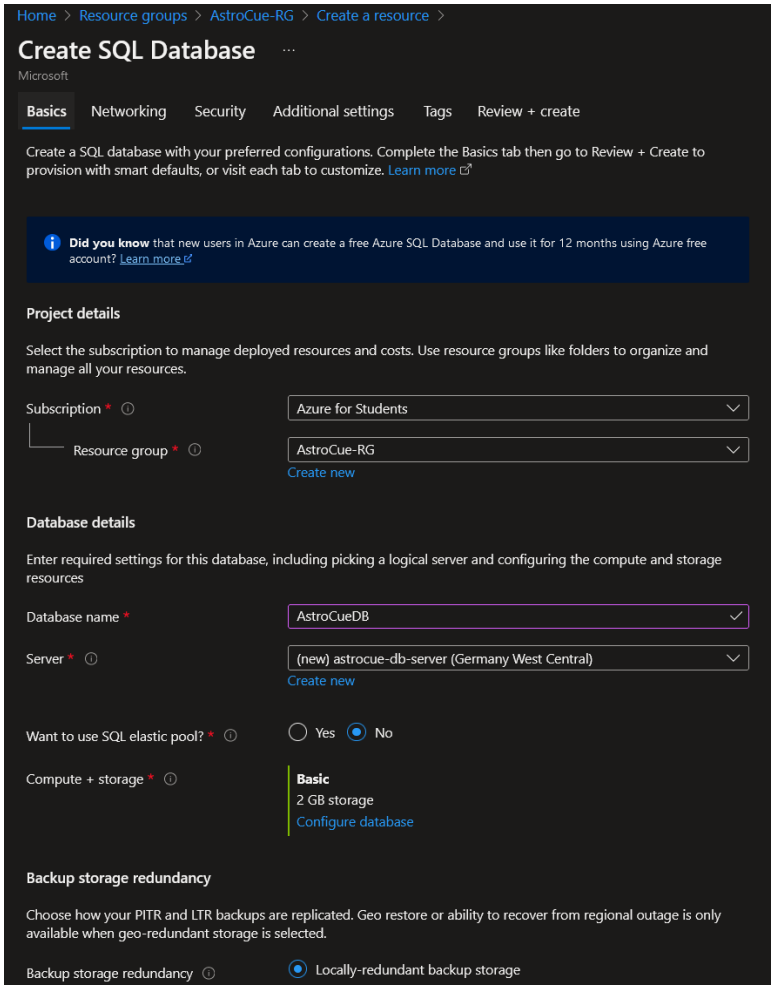
Click create. Give it a sensible name and chose a location. This location has restrictions on it, so choose one closest to the region of your Resource Group. Enter a set of admin credentials. Make sure to note these down somewhere secure for later reference.



Next, you will be asked to configure the resources allocated to the database. Select the basic service tier that comes with 5 DTUs (Database Transaction Units). This should result in a monthly estimated cost of ~£4.60.

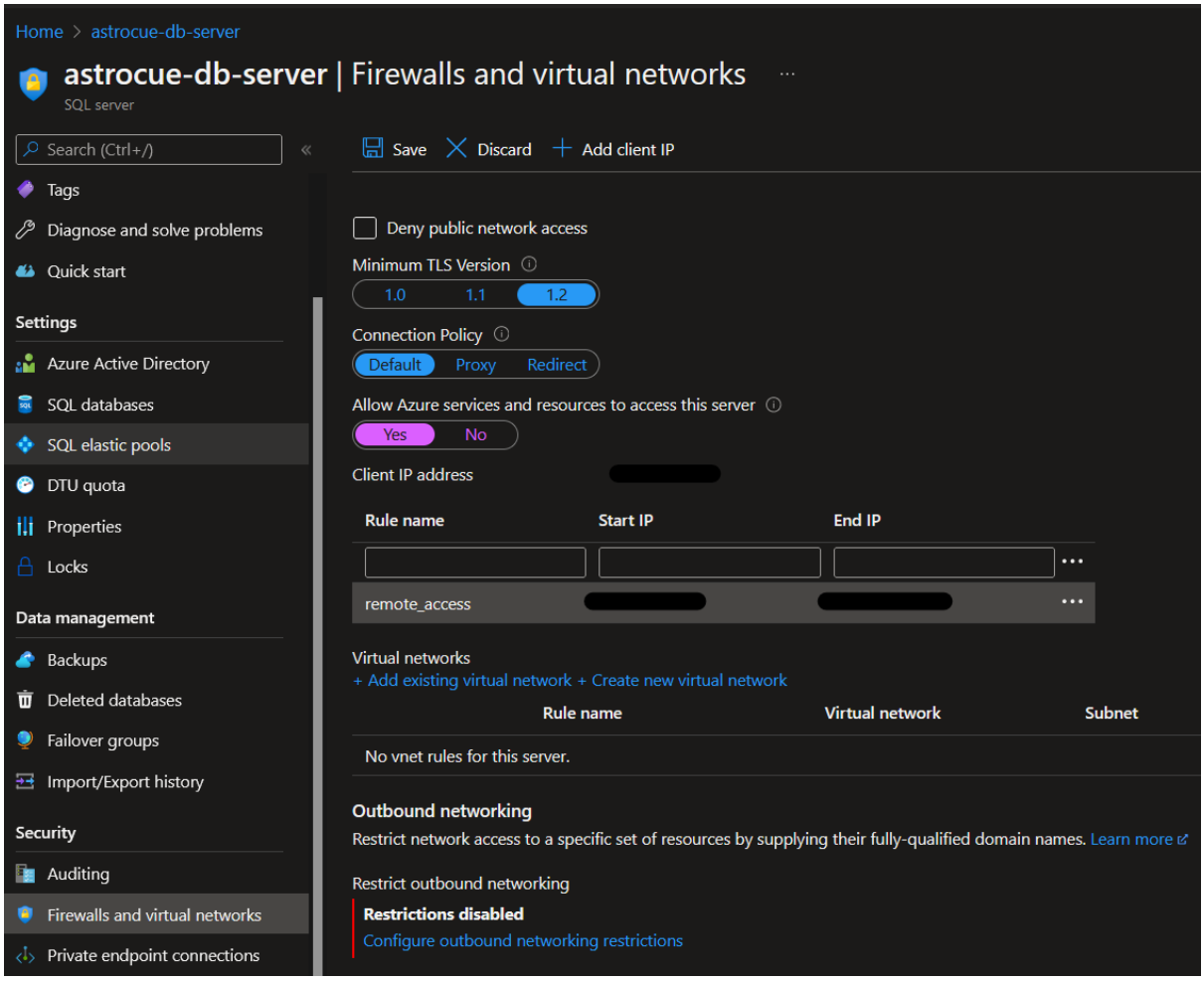


Fill out the remaining options as follows, and then confirm the database creation.

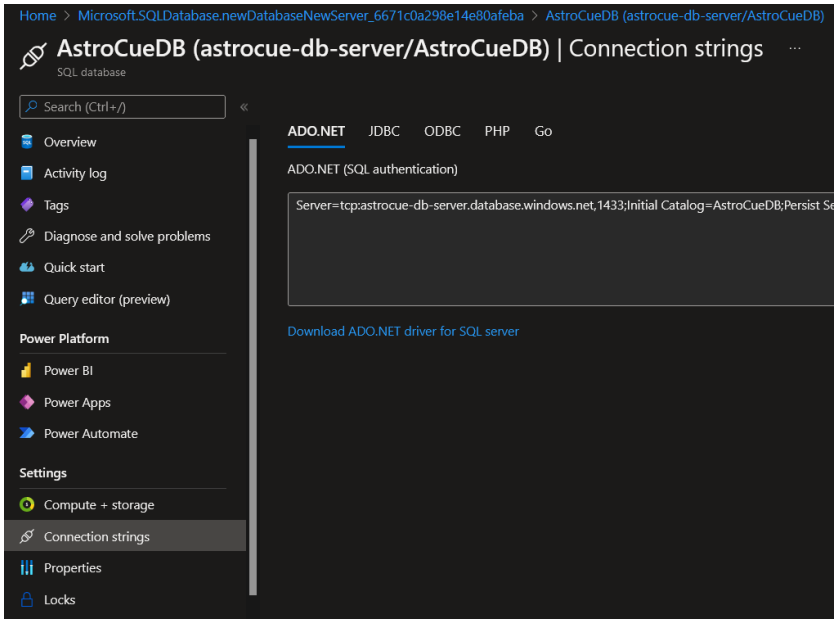


The firewall now needs to be set up. Head over to the new database resource inside the Resource Group and click the 'Firewalls and virtual networks' tab. Make sure that the 'Allow Azure services and resources to access this server' option is set to 'Yes' and add your public IP address to the 'Client IP address' entry field. The **start** and **end** IP should be identical.

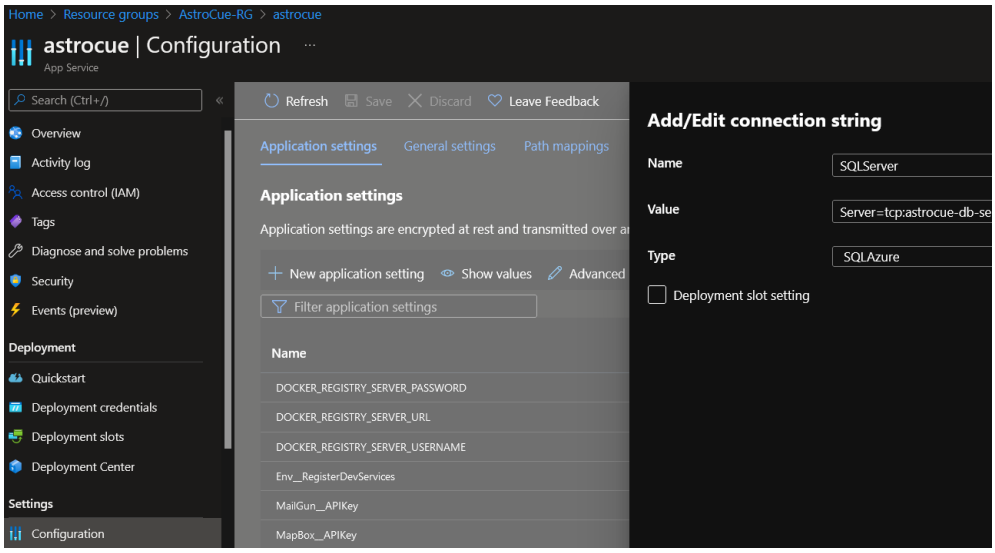
Note that this is not your device's local IP address, but its public IPv4 address. If you do not know this, you can find out at https://whatismyipaddress.com/.



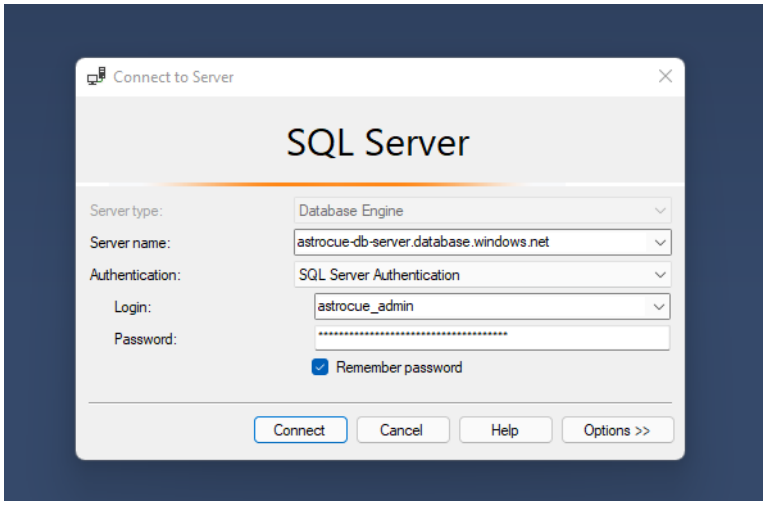
Once you have done this, head to the 'Connection strings' tab of the new database (not the server) – this will provide the string that the server will need to connect to the database. Copy this string to your clipboard.



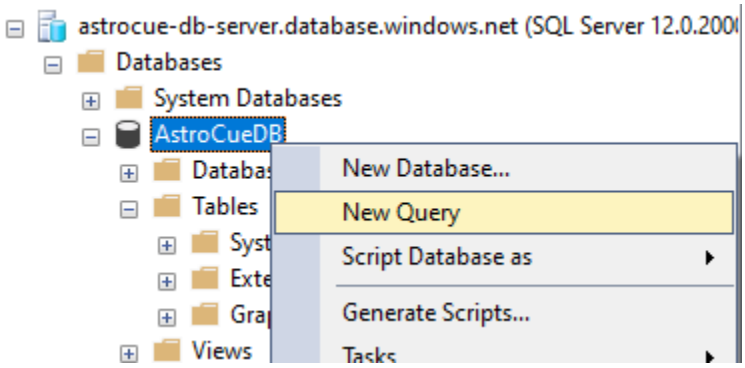
Head back over to the 'Configuration' tab of the App Service instance you created earlier. Just under where the environment variables were defined, there will be a connection strings area. Add a new entry with the name SQLServer. Paste the connection key you copied from the database page. **Inside this string, there will be a template for you to enter your database's password – enter the one you created and noted down earlier before you save your changes. If this password is not present within the string, the server will not be able to connect to the database.**



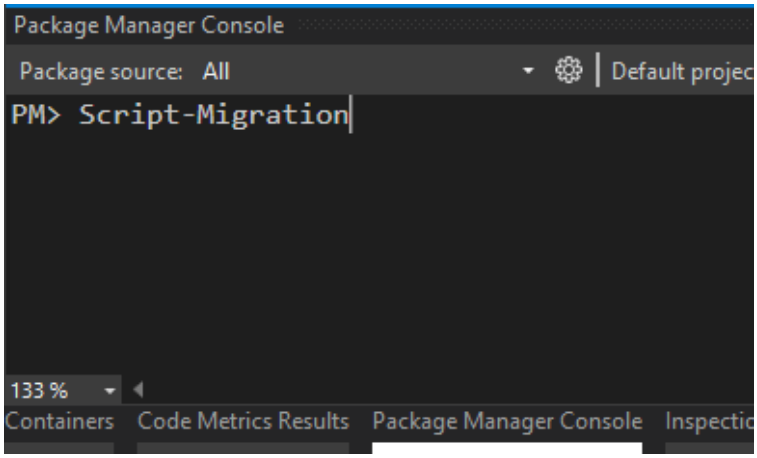
Open SQL Server Management Studio, and enter the database URL, along with the username and password you set up for it earlier. If logging in fails, ensure that the credentials are correct, and that you have whitelisted your public IP on the database server's firewall from Azure.



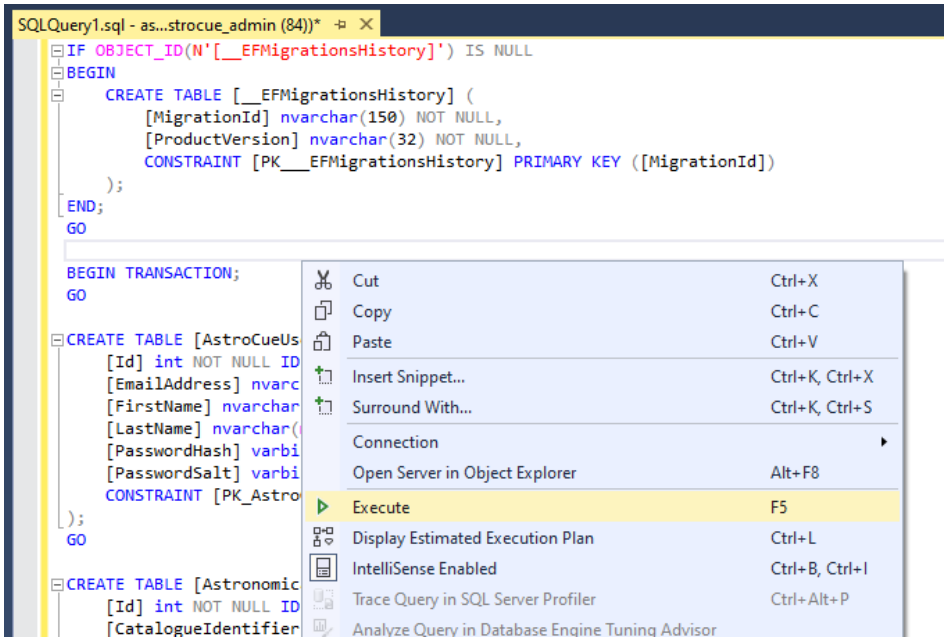
Once logged in, right click the database, and begin executing a new query. We now need to carry out the database migrations to create all the tables and their relationships.



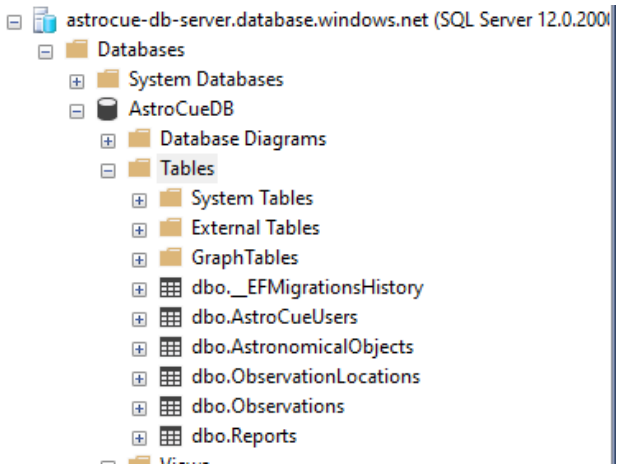
All the migrations can be built into a single SQL script from Visual Studio. Head back over to VS and open the Package Manager Console. Run the command: `Script-Migration`.



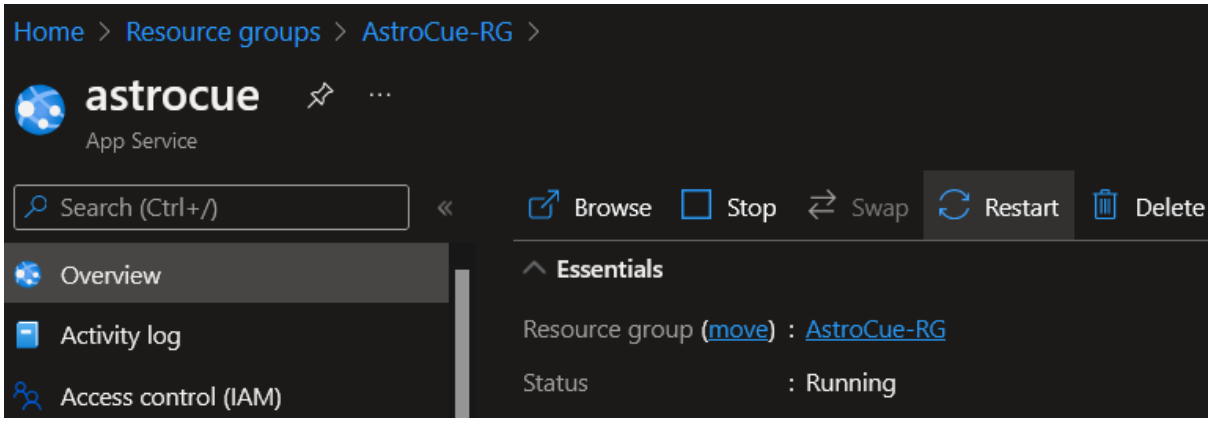
Visual Studio will then use the entity classes in the codebase, along with the auto generated migrations in the `Migrations` folder to build a single SQL script to bring the database schema in line with where it should be. Copy this entire script, paste it into the query window in SQL Server Management Studio. Right click anywhere inside the script and then 'Execute'. Alternatively, press the F5 key.



This script should execute with no errors, and if you right click the 'Tables' folder underneath the database in the left pane and refresh it, all the newly created tables should be visible.



To kickstart everything into functioning, head back over to the Azure Portal, click on the App, and restart it. This will cause it to reboot entirely, which reloads all the environment variables (including API keys and the database connection string). As this is the first time the server would have had all the prerequisites to carry out its database seeding process, it may take a few minutes to become responsive. You can check on this by viewing its container logs.



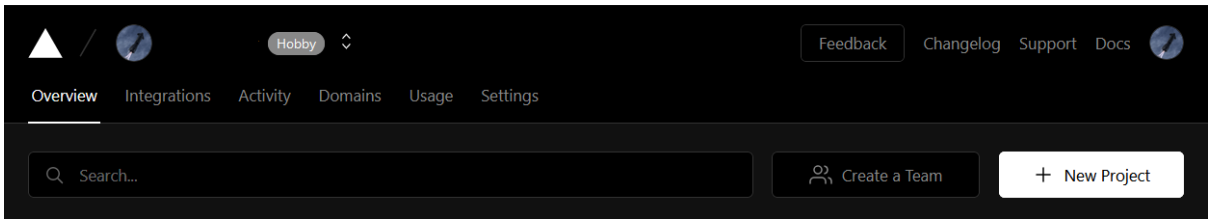
That's it! The server, database, and the blob storage instance are all fully deployed and ready for operation.

9.6.3 FRONTEND DEPLOYMENT TO VERCEL

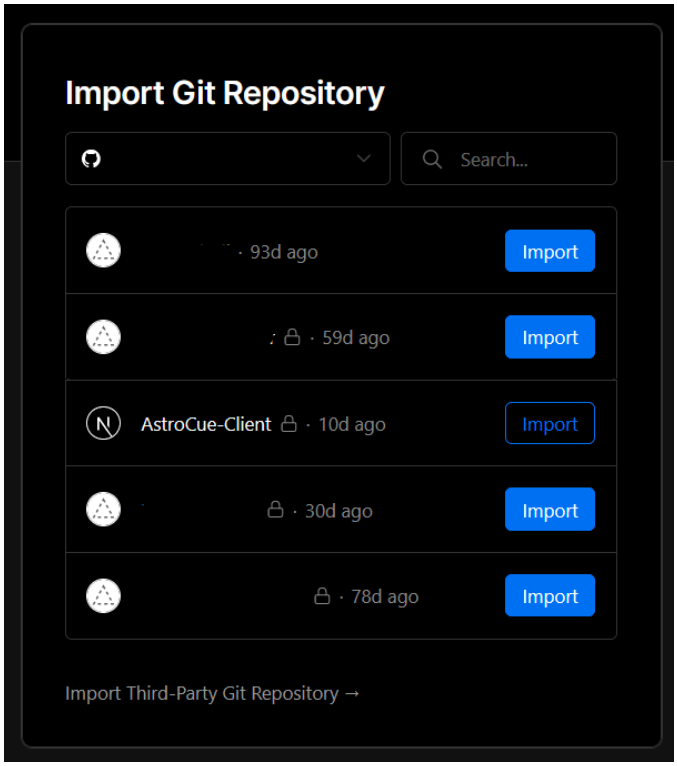
The client side, or 'frontend', section of AstroCue is deployed to the Vercel platform. This deployment is far simpler than the server and can be completed in a few steps.

The only 2 prerequisites here are that you have an account on Vercel (https://vercel.com) and the frontend code is in a GitHub repository.

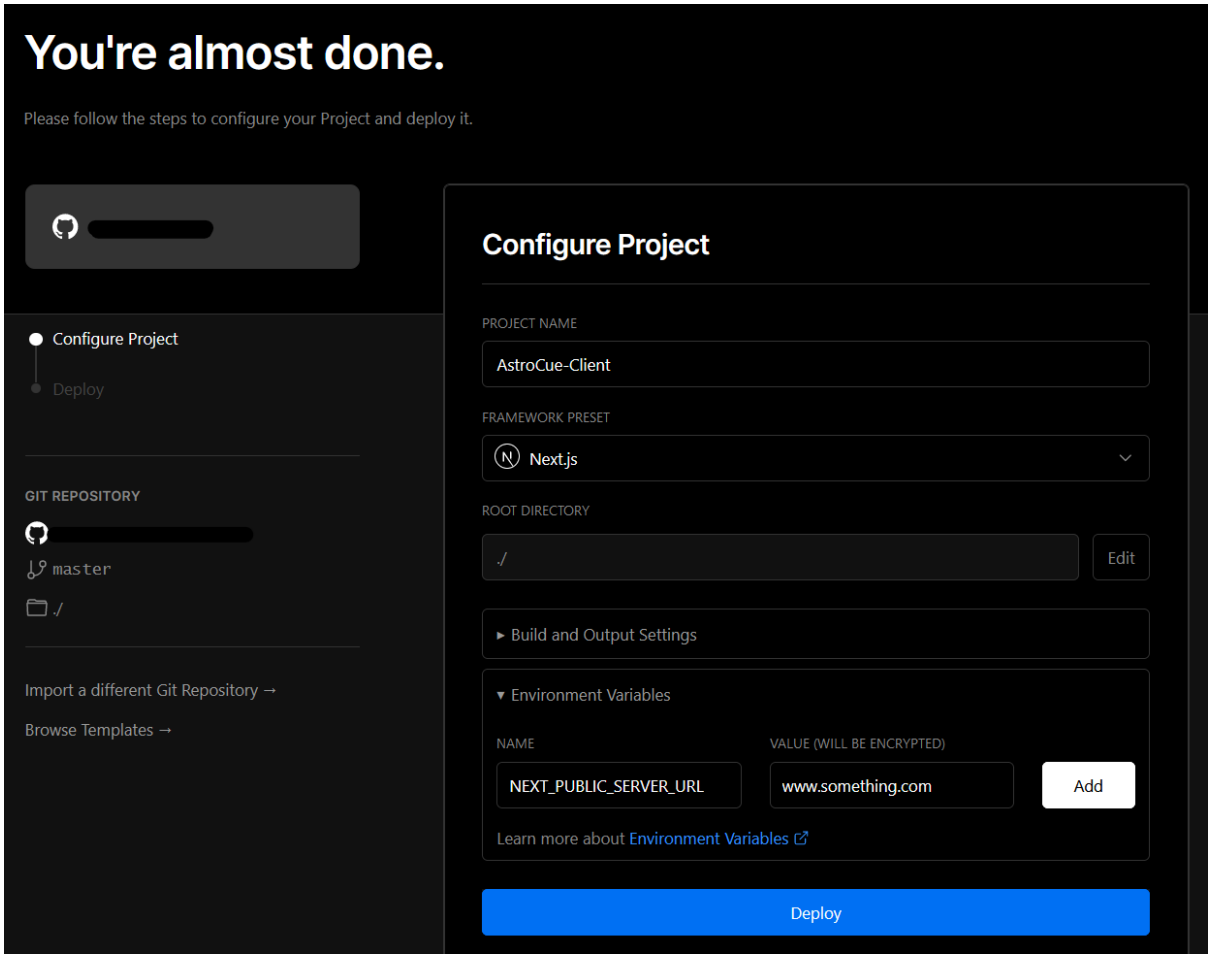
To begin, head to the Vercel Dashboard and click the 'New Project' button.

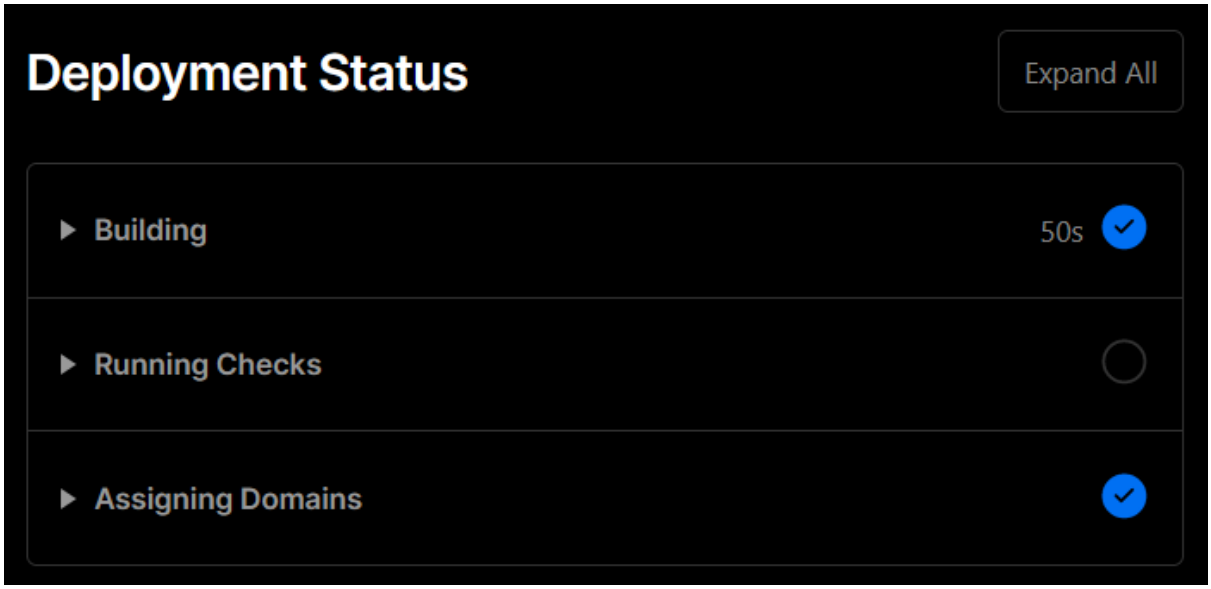


On the next screen, select the repository that the client code is located in. If you have many repositories, you may need to search for it. Vercel should recognise that the repository houses a Next.js app and display the Next.js logo alongside the repository name.

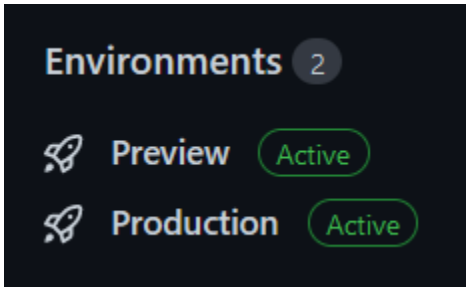


On the next screen, the only setting that you need to change is the environment variables. Add a single variable with the key `NEXT_PUBLIC_SERVER_URL`. This URL, if you do not know it already, is the public URL of the Azure App Service instance deployed from the previous steps. If you do not know it, head to the App within the Azure portal and you can find it out there. Click deploy, and the frontend should deploy successfully.





Vercel should also set up 2 environments on the GitHub repository, `Preview` and `Production`. These are continuous integration/deployment (CI/CD) deployments that automatically issue new builds to Vercel on repository pushes and branch merges. This means that the public URL of the Vercel project is always serving the most up-to-date successful builds directly from source control.

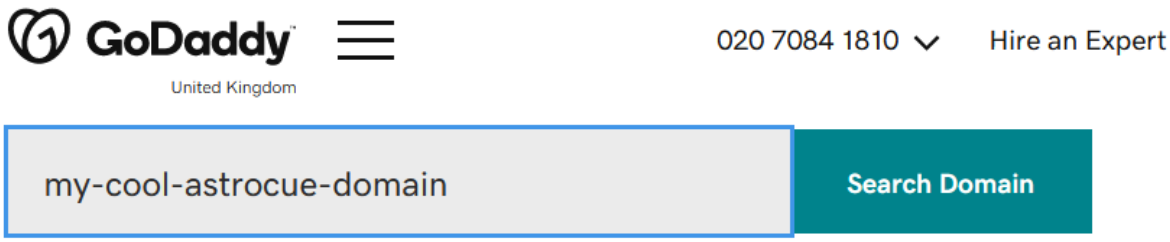


9.6.4 Domain and DNS

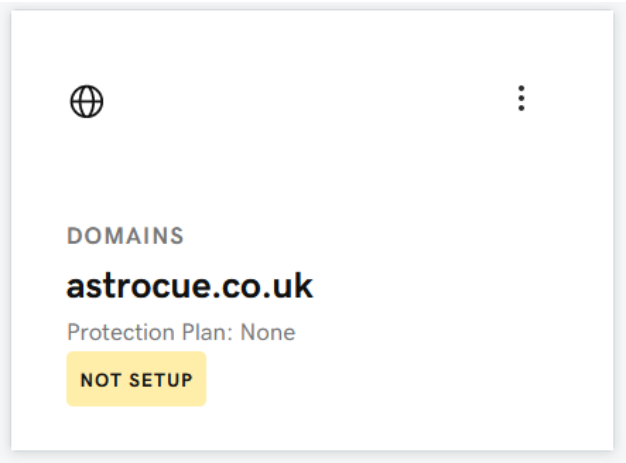
AstroCue used a custom domain for its frontend and emails (astrocue.co.uk), and the following steps show how to assign any valid domain to both resources.

9.6.4.1 Registering a new domain

Firstly, head over to your favourite registrar and search for a domain. The author used GoDaddy (https://www.godaddy.com/en-uk) due to familiarity and cost (and that's what the following instructions refer to), however, any other registrar should remain similar in the steps required.



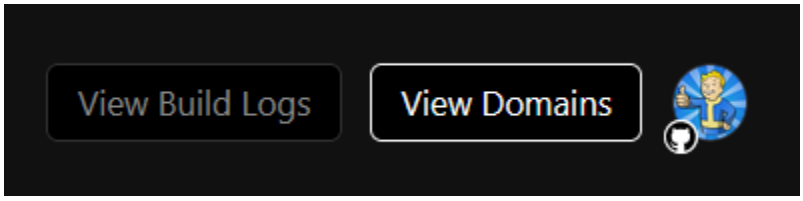
Once you have found a domain, purchase it, and it should soon appear under your account.



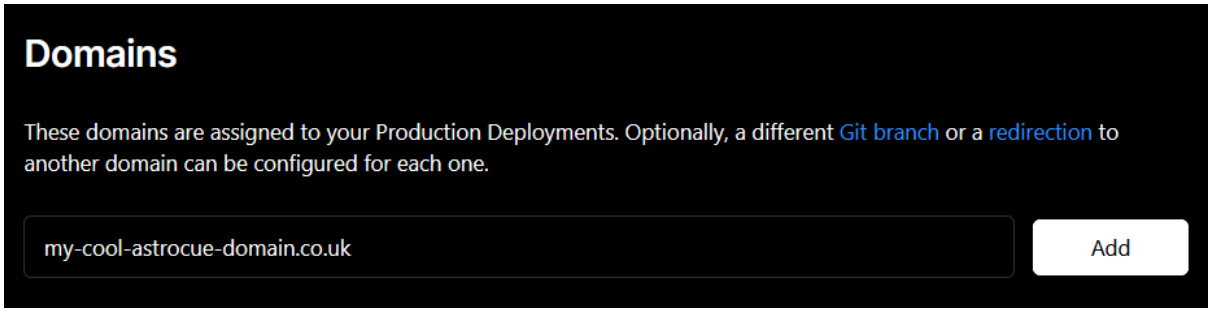
9.6.4.2 LINKING DOMAIN TO VERCEL DEPLOYMENT

The first thing we need to do domain wise, is get the frontend using it. This is very straightforward.

Head over to the Vercel Dashboard and click on the project that contains the new AstroCue deployment. On the right, there will be a 'View Domains' button – click it.

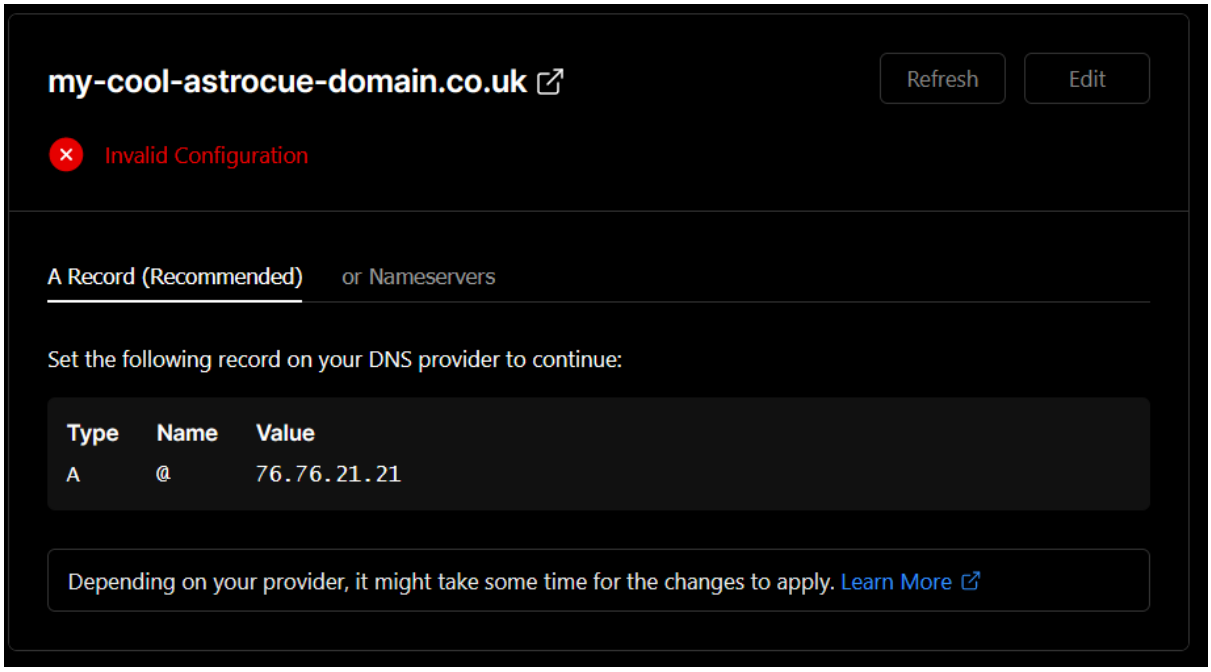


On the next page, enter your registered URL into the domain box, and click 'Add'.

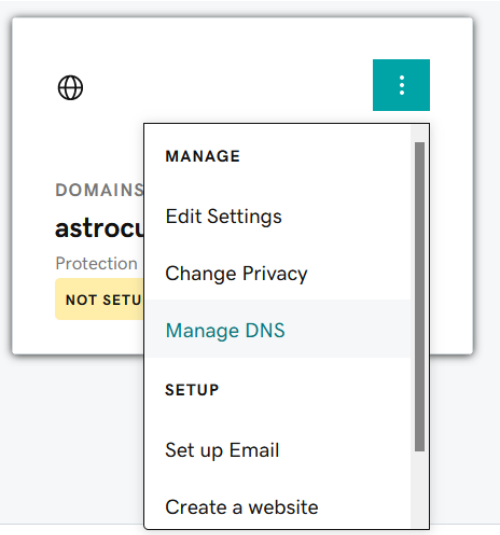


There will be a redirect popup that appears, just select the recommended option.

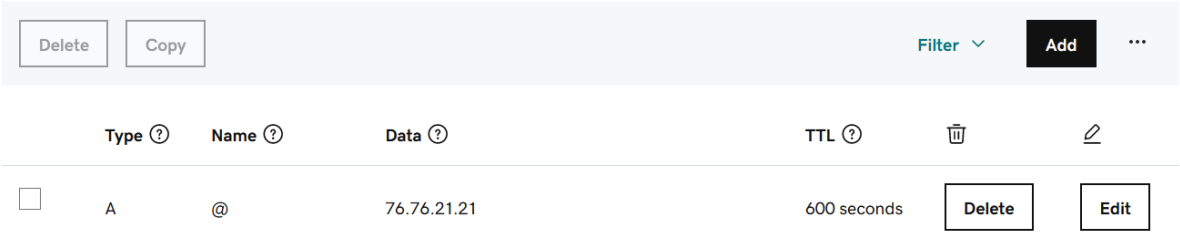
Initially, the configuration will be invalid as an 'A' record needs to be added to the domain's DNS. Vercel will tell you what this is.



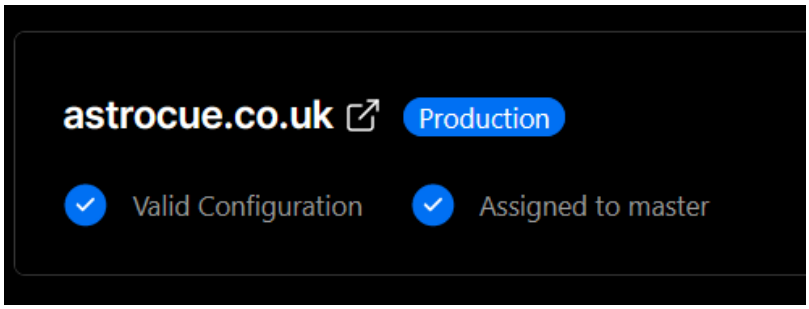
Back on your registrar's page, navigate to the DNS settings page.



Add a new record that matches the one Vercel displayed and wait for the DNS settings to propagate – this can take anywhere from minutes to hours.



You will know when this process has completed because Vercel will show your domain as having a valid configuration.

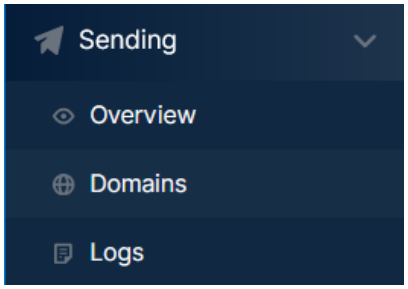


You should now be able to visit this domain in a browser and see the frontend.

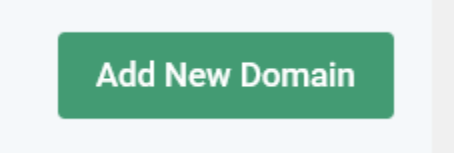
9.6.4.3 LINKING DOMAIN TO MAILGUN

To send emails from the domain, it is also necessary to add DNS records for MailGun. First, head over to the MailGun dashboard.

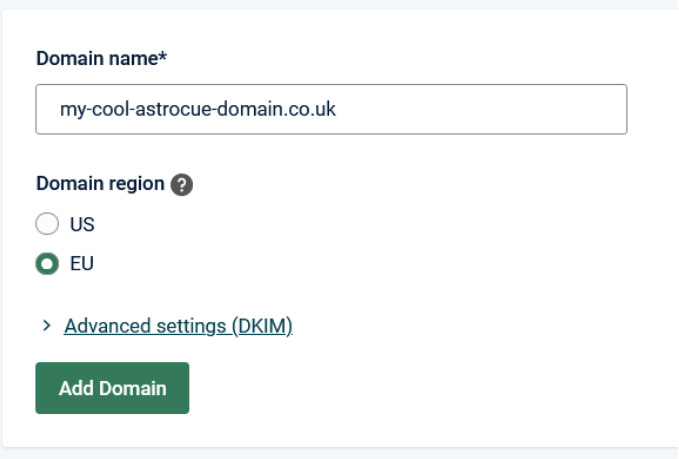
Under the 'Sending' menu, click the 'Domains' button.



And then the 'Add New Domain' button.



Enter the domain, and then add it.



MailGun will then present the DNS records it requires you to add. These include cryptography keys for validating sender information, as well as records pointing the domain to MailGun's servers.

2. Add DNS records for sending Collapse

	Type	Hostname	Enter This Value	Current Value
✖	TXT	my-cool-astrocue-domain.co.uk	v=spf1 include:mailgun.org ~all	*None found*
✖	TXT	**s1._domainkey**.my-cool-astrocue-domain.co.uk	k=rsa; p=MI[redacted]	*None found*

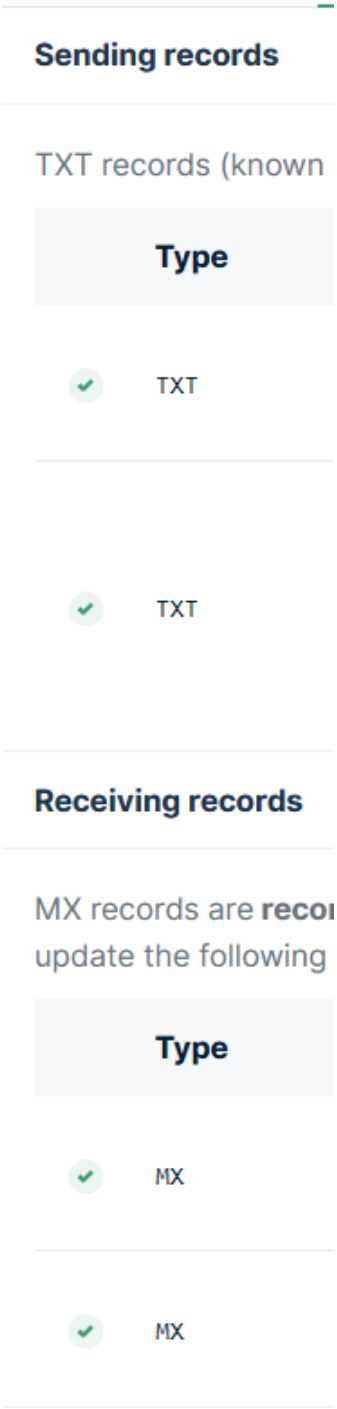
MX records are recommended for all domains, even if you are only sending messages. Unless you already have MX records for this domain pointing to another email provider (e.g. Gmail), you should update the following records. More info on MX records.

	Type	Hostname	Priority	Enter This Value	Current Value
?	MX	my-cool-astrocue-domain.co.uk	10	mxa.eu.mailgun.org	*None found*
?	MX	my-cool-astrocue-domain.co.uk	10	mxb.eu.mailgun.org	*None found*

Add these records to the domain on the registrar's settings page, the same way the records for Vercel were added.

☐	TXT	email._domainkey	k=rsa; p=MI[redacted]	1 Hour	**Delete**	**Edit**
☐	TXT	no-reply	v=spf1 include:mailgun.org ~all	1 Hour	**Delete**	**Edit**
☐	MX	@	mxa.eu.mailgun.org. (Priority: 10)	1 Hour	**Delete**	**Edit**
☐	MX	@	mxb.eu.mailgun.org. (Priority: 10)	1 Hour	**Delete**	**Edit**

Once the DNS settings have propagated (again, a few minutes to a few hours), MailGun should show ticks next to all the records, meaning that it has validated the DNS settings correctly. This domain can now be used to send emails.



9.7 APPENDIX G: EF CORE DATABASE MIGRATION SCRIPT

See Moodle submission.

9.8 APPENDIX H: INTRODUCTION TO CELESTIAL COORDINATE SYSTEMS

Introduction to celestial coordinate systems

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

Consultant: [REDACTED]

9.8.1 CELESTIAL COORDINATE SYSTEMS

To calculate the positions of astronomical objects in the sky from different positions on Earth at different times, AstroCue needed to implement various computational astronomy algorithms and work in multiple coordinate systems.

The two different celestial coordinate systems supported by AstroCue were:

1. Equatorial
2. Horizontal

9.8.2 EQUATORIAL COORDINATE SYSTEM

Much like positions on the surface of the Earth are uniquely identified and located using lines of longitude and latitude, the mapping of stars and deep sky objects is achieved using a similar system.

It appears, from Earth, that astronomical objects (excluding the Moon, planets, and artificial satellites) are stationary relative to one another. The distances to and between them are so incomprehensibly large that just by looking, we cannot discern any movement or relative distances. By extension, one can then imagine a model in which all distant objects seem to be *projected* onto the inside of a large sphere within which the Earth exists – this is concept is called the **celestial sphere**:

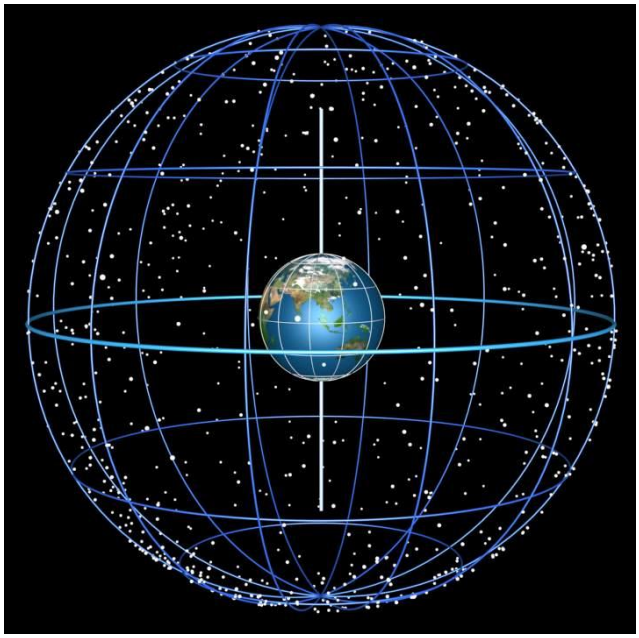


Figure 44: The 'celestial sphere' with stars (Perkic, n.d.)

As can be seen in the above diagram, the Earth's equator is then projected onto this sphere, and we can draw gridlines on the inside of it, just as we can draw longitude and latitude lines on the outside of Earth. The projected equator divides the celestial sphere into northern and southern hemispheres.

Now, units are required – how do we define a specific point in the sky? Right ascension (α/RA) and Declination (δ/Dec) are the units of the equatorial system.

- RA can be thought of as the celestial longitude – ranging from $0°$ to $360°$ starting from the vernal equinox in March.
- Dec is the celestial latitude, ranging from $-90°$ (south celestial pole) to $90°$ (north celestial pole).

However, RA is usually measured in terms of hours/minutes/seconds of Earth's rotation, so for example, $107.145°$ would be represented as $07^h08^m35^s$ (Meeus, 1998, p. 8). Similarly, Dec is represented in degrees/arcminutes/arcseconds, so -33.6019 would be $-33°36'07''$. These all signify the same values but are just representational differences.

A combination of RA and Dec can allow you to describe any position on the celestial sphere, and every catalogued astronomical object has its location given in these coordinates.

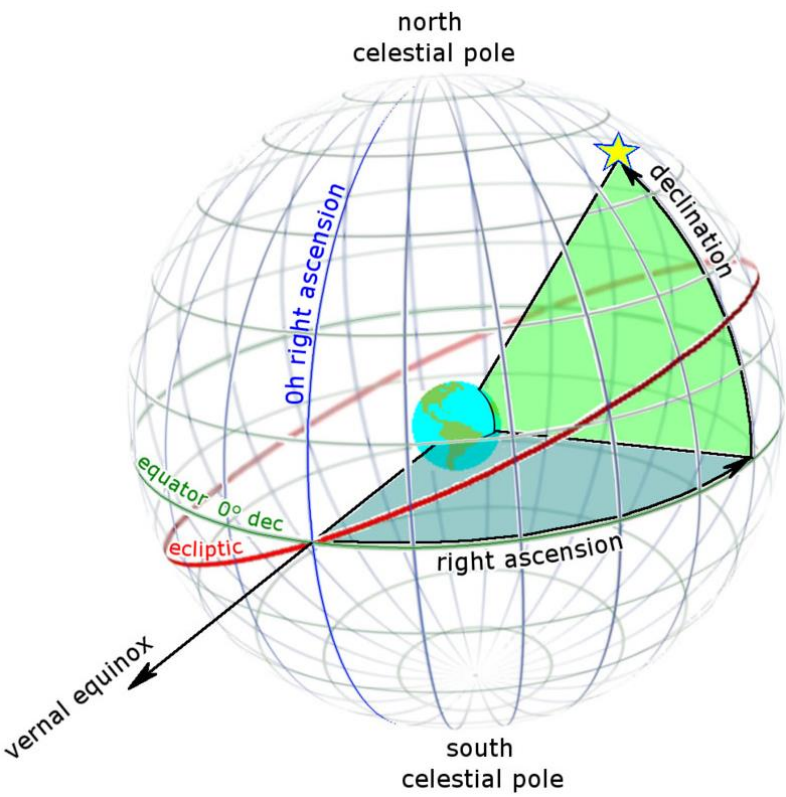


Figure 45: RA and Dec allow location of objects in the celestial sphere (Sky & Telescope, 2019).

However, despite these coordinates being 'fixed', they slowly become more and more inaccurate over time. Why is that? It all comes down to the fact that Earth's movement isn't quite as simple as it first seems. While everyone is aware that Earth spins on its axis, many are unaware that it also slowly 'wobbles' around its axis in a process called '**axial precession**'; caused by tidal forces between the Earth-Moon-Sun system. The completion of a full cycle of precession takes around 26,000 years (Meeus, 1998, p. 131).

Due to this slow drift, the equatorial coordinates of an astronomical object (α, δ) are inherently relative to the instant in which they were measured. The increase in inaccuracy is gradual, but nonetheless present, and needs to be considered in high precision calculations.

However, for the purpose of AstroCue, this inaccuracy doesn't hinder the results in any meaningful way and can be safely ignored. If needed, it can be accounted for algorithmically (Meeus, 1998, p. 132).

To make this measurement relativity clear, equatorial coordinates are often prefixed or suffixed with their 'epoch', or the year in which they were taken. I.e., it is common to see measurements such as:

$$\alpha_{1950} = 07^h 08^m 35^s \textbf{ or } \alpha_{2000} = 07^h 11^m 06^s$$

with the latter clearly being more accurate today if taken at face value without any adjustments.

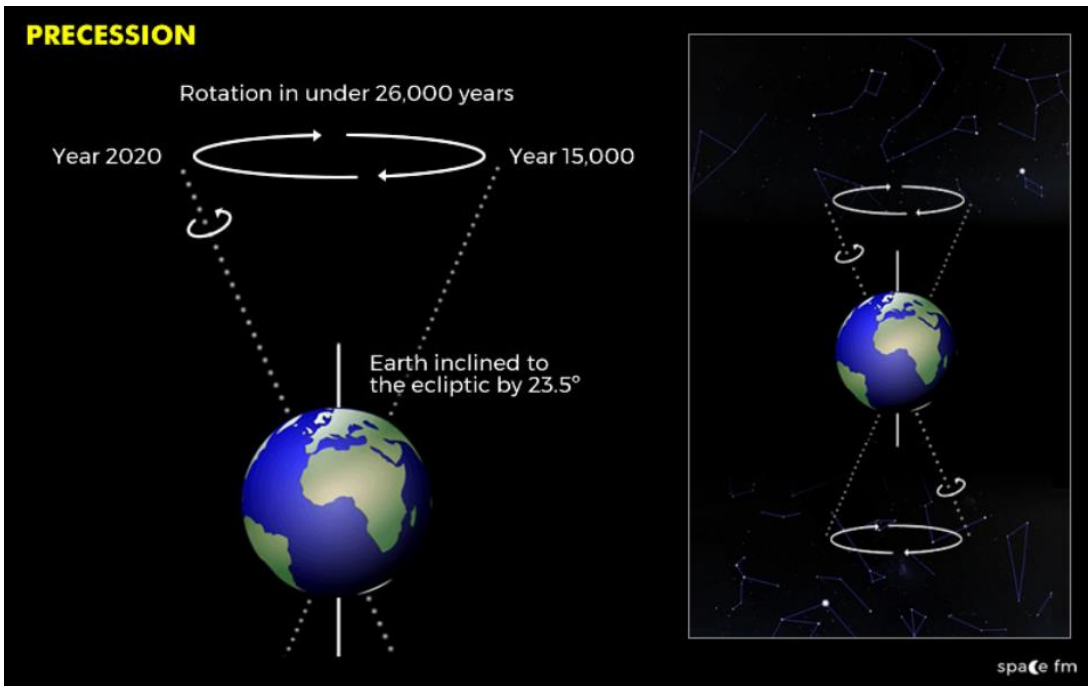


Figure 46: Earth's Precession (space.fm, n.d.).

9.8.3 HORIZONTAL COORDINATE SYSTEM

While the equatorial system is great for positions of objects on the fixed celestial sphere – it isn't particularly useful to observers looking to find objects from locations on Earth. For this purpose, the horizontal coordinate system exists.

The horizontal coordinate system uses an observer's local horizon as the zero point for both of its units, which are altitude and azimuth. Imagine you, the observer, are in a flat field that extends out to the observable horizon all around you, and you are facing South.

Azimuth is measured from $0°$ to $360°$ moving westwards from South (or North in some examples – if you pick one and stick to it for consistency, your calculations will be correct. Astronomers typically measures from the South, however) (Meeus, 1998, pp. 91-92).

Altitude, on the other hand, is measured from $0°$ at the horizon, to $90°$ directly above your head (called the zenith). It then holds that any altitude $< 0°$ is below the horizon.

A combination of altitude and azimuth (alt/az) allows an observer, from any position, to locate an object relative to themselves and their local horizon.

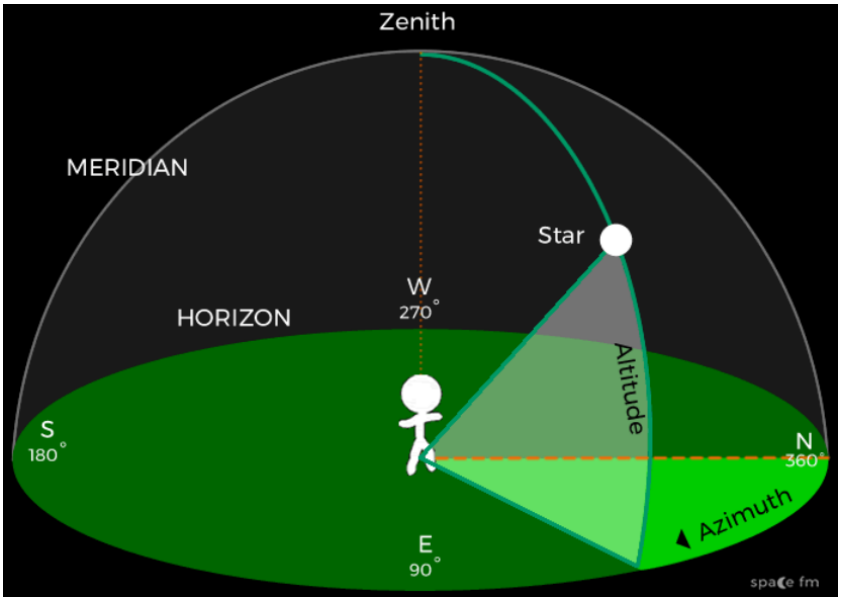


Figure 47: Horizontal coordinate system (space.fm, n.d.).

AstroCue leans on the fact that there are calculations that can transform equatorial coordinates into horizontal coordinates once an observer's longitude, latitude and local time are known.

9.9 APPENDIX I: TESTING

Testing

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

Consultant: [REDACTED]

9.9.1 INTRODUCTION

This document covers the testing that was carried out against the AstroCue platform. The author conducted both use case testing as well as unit testing of the code itself. An element of non-functional testing, maintainability, was also tested.

Testing results begin overleaf. The tests themselves can be seen in the sections following these results.

9.9.2 TESTING RESULTS

9.9.2.1 USE CASE TESTING RESULTS

Test (clickable for easy navigation)	Pass	Fail
[Test 1] - Use cases 1 and 3		
[Test 2] - Extension case 1.1		
[Test 3] - Extension case 2.1		
[Test 4] - Use case 2		
[Test 5] - Use cases 4, 6, 7 and 8		
[Test 6] - Extension case 4.1		
[Test 7] - Use case 5		
[Test 8] - Use cases 9, 11, and 19		
[Test 9] - Extension case 19.1		
[Test 10] - Use cases 20 and 21		
[Test 11] - Use cases 12, 13, 14, 15, 16 and 18		
[Test 12] - Use case 17		
[Test 13] - Use case 10		

9.9.2.2 UNIT TEST RESULTS

All unit tests passed. See section 9.9.4.1.

9.9.2.3 MAINTAINABILITY TESTING RESULTS

The final code maintainability score was 81/100. See section 9.9.5.

9.9.3 USE CASE TESTING

The following are use case tests are based on the use case specifications that arose as part of the requirements gathering stage of AstroCue. Many of the test cases, due to their nature, cover multiple use cases together. Where this is the case, it will be made clear in the test itself. These specifications themselves can be viewed in section 9.4.5.

These tests build upon one another and are designed to be ran in sequence in a single sitting.

9.9.3.1 [TEST 1] - USE CASES 1 AND 3

Test no.: 1	**Covers use case(s) ID:** 1 (Register), 3 (SendWelcomeEmail)		
Preconditions	User has internet access, and an email account with access to the inbox		
Steps	1. Open a web browser and navigate to astrocue.co.uk 2. You will be on the login screen, click the 'Register' link at the bottom 3. Enter a valid email address whose inbox you have access to 4. Enter your first name and last name in the corresponding boxes 5. Enter a password (note this down somewhere for later tests) 6. Click the 'Register' button		
Expected result	• You are redirected to the login screen, and a notification appears thanking you for registering • You have received a welcome email to the email address you used to register ○ If this is not present, check your junk folder		
Result	Pass ☒ Fail ☐		
Tested by	Author	**Date**	12/05/22

9.9.3.2 [TEST 2] - EXTENSION CASE 1.1

Test no.: 2	**Covers use case(s) ID:** 1.1 (Extension: EmailAlreadyExists)		
Preconditions	Test no. 1 has been completed and the user is viewing the login page after having been redirected		
Steps	1. Click the 'Register' link at the bottom 2. In the email field, enter the same email you used in test no. 1 3. Enter any first name, last name, and password 4. Click the 'Register' button		
Expected result	An error is displayed telling you that the email is already taken		
Result	Pass ☒ Fail ☐		
Tested by	Author	**Date**	12/05/22

9.9.3.3 [TEST 3] - EXTENSION CASE 2.1

Test no.: 3	**Covers use case(s) ID:** 2.1 (Extension: IncorrectCredentials)		
Preconditions	Test no. 2 has been completed and the user is viewing the registration page		
Steps	1. Click the 'Login' button up the top right of the navigation bar 2. In the email field, enter the email you used in test no. 1 3. In the password field, enter a password that doesn't match the one you used in test 1 4. Click the 'Login' button		
Expected result	An error message is displayed telling you that either your email or password is incorrect.		
Extra notes	An extra sub-test can be conducted by flipping which one of the credentials is incorrect. Try making the password correct but then entering an invalid email. The expected result stays the same.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.4 [TEST 4] - USE CASE 2

Test no.: 4	**Covers use case(s) ID:** 2 (Login)		
Preconditions	Test no. 3 has been completed and the user is viewing the login page.		
Steps	1. In the email field, enter the email you used in test no. 1 2. In the password field, enter the password you used in test 1 3. Click the 'Login' button		
Expected result	The login should succeed, and you should be now inside AstroCue, viewing the default tab: 'Observation Locations'.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.5 [TEST 5] - USE CASES 4, 6, 7 AND 8

Test no.: 5	**Covers use case(s) ID:** 4 (AddNewObservationLocation), 6 (GetAllObservationLocations), 7 (GetWeatherForecast), 8 (GetStaticMapImage)		
Preconditions	Test no. 4 has been completed and the user is viewing the observation locations tab.		
Steps	1. Click the 'Add Location' button 2. The 'New observation location' dialog opens 3. Enter location name 'City, University of London' 4. Leave the latitude and longitude as their pre-filled defaults 5. Click the 'Confirm' button		
Expected result	The dialog should close and the newly added observation location, along with its name, coordinates, light polluting level, map image, and a weather forecast should appear on the screen.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.6 [TEST 6] - EXTENSION CASE 4.1

Test no.: 6	**Covers use case(s) ID:** 4.1 (Extension: ForwardGeocode)		
Preconditions	Test no. 5 has been completed and the user is viewing the observation locations tab. A single observation location is present.		
Steps	1. Click the 'Add Location' button 2. The 'New observation location' dialog opens 3. Click the magnifying glass icon to the right of the 'Location name' entry field 4. The 'Location search' dialog opens 5. Search for 'Fort William' 6. The top result should be 'Fort William, Highland, Scotland, United Kingdom'. Click it. 7. The 'Location search' dialog closes. The act of closing the dialog has autocompleted the 'New observation location' dialog that was open beneath. 8. Click the 'Confirm' button		
Expected result	The dialog should close and the newly added observation location, along with its name, coordinates, light polluting level, map image, and a weather forecast should appear on the screen. There should now be two observation locations visible.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.7 [TEST 7] - USE CASE 5

Test no.: 7	**Covers use case(s) ID:** 5 (DeleteObservationLocation)		
Preconditions	Test no. 6 has been completed and the user is viewing the observation locations tab. 2 observation locations are present. One of them has the name 'City, University of London'.		
Steps	1. On the observation location named 'City, University of London', click the bin icon. 2. The 'Delete observation location' dialog opens 3. Click the 'Delete' button		
Expected result	A notification is displayed confirming that the observation location was deleted and the location itself is removed from the list. Only one observation location with the name 'Fort William' remains.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.8 [TEST 8] - USE CASES 9, 11, AND 19

Test no.: 8	**Covers use case(s) ID:** 9 (CreateNewObservation), 11 (GetAllObservations), 19 (SearchAstronomicalObjects)		
Preconditions	Test no. 7 has been completed and the user is viewing the observation locations tab. 1 observation location is present.		
Steps	1. Navigate to the 'Observations' tab. 2. Click the 'New Observation button'. 3. The observation location should default to 'Fort William'. Leave the filter options as their defaults. 4. Search for 'Polaris' (the north star) in the 'Astronomical object search' entry field. 5. 2 results should be returned, one of which being just 'Polaris'. It should be bright and visible. Select it. 6. The 'Confirm' button should unlock. Click it.		
Expected result	A notification is displayed confirming that the observation has been created. The observation will appear – showing Polaris being observed from Fort William. 'Polaris' itself should be clickable, opening a new tab on the CDS portal where more detailed information about the object can be seen.		
Extra notes	If you try to repeat the steps again, you should receive an error that an observation between Fort William and Polaris already exists.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.9 [TEST 9] - EXTENSION CASE 19.1

Test no.: 9	**Covers use case(s) ID:** 19.1 (Extension: FilterByBrightOrPopular)		
Preconditions	Test no. 8 has been completed and the user is viewing the observations tab. 1 observation is present.		
Steps	1. Click the 'New Observation button'. 2. The observation location should default to 'Fort William'. 3. Tick the 'Popular' checkbox 4. Select the 'Deep sky' radio button 5. Search for the 'Eagle Nebula' It should appear in the results (as 'Eagle Nebula, M 16') and be bright and visible. 6. Clear the search box, now search for 'NGC 4008'.		
Expected result	The search query 'NGC 4008' should return no results as you are filtering by popular objects, and NGC 4008 (a galaxy) has no widely known designation.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22

9.9.3.10 [TEST 10] - USE CASES 20 AND 21

Test no.: 10	**Covers use case(s) ID:** 20 (GenerateReports), 21 (SendReportEmails)		
Preconditions	Test no. 9 has been completed and the user is viewing the observations tab. 1 observation is present.		
Steps	1. Navigate to the 'Reports' tab 2. Either: a. Click the 'Generate reports' button b. The 'Generate new reports' dialog opens c. Click the 'Generate' button 3. Or: a. Wait ~3 days b. Refresh AstroCue, and return to the 'Reports' tab. 4. Click the 'Add to calendar' button inside the email that arrives in your inbox		
Expected result	IF reports were generated manually: A notification is displayed confirming that the reports have been generated and that an email has been sent. A single report, showing the best time to observe Polaris over the next few days should be visible under the 'Fort William' location. In both cases: In the inbox (or possibly the junk) of the email account, registered in test 1, you should have received an email with the contents of the report. The email's information should match the report on the web page. Once you click the 'Add to calendar' button, your operating system's default browser should open on https://calendar.google.com, with the details of the observation time prefilled ready for you to save. If you are not signed in, it will first ask you to do so.		
Extra notes	Use case 20 states that the main flow involves waiting for the report generation to be triggered by 3 days of time. Due to this test possibly taking 2 flows (either the user produces the reports manually by using the generate button, or by waiting), the steps and results reflect that by providing 2 branches that can be followed independently. *Note that the user's manual generation of reports was not represented as a use case during the initial analysis/requirements stage of this project, hence it's missing from the use cases. The author felt that here would be the best location to retroactively implement a test for it.*		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	12/05/22 (manual) 16/05/22 (automatic)

9.9.3.11 [TEST 11] - USE CASES 12, 13, 14, 15, 16 AND 18

Test no.: 11	**Covers use case(s) ID:** 12 (TakeNewLog), 13 (AddTextualDescription), 14 (AddWeatherConditions), 15 (AddObserver), 16 (AddObservationLocation), 18 (GetAllObservationLogs)		
Preconditions	Test no. 10 has been completed and the user is on the 'Reports' tab. 1 report is present.		
Steps	1. On the report for Polaris from Fort William, click the book icon (referencing a log) 2. The 'Log this observation' dialog opens 3. Enter a description in the 'Description' field. As you likely didn't do this observation, you can just enter anything. 4. Optionally, enter your name into the 'Observer(s)' field 5. Choose an observation type from the 'Observation Type' dropdown. 6. Click the 'Log observations' button		
Expected result	A notification is displayed confirming that the log has been recorded successfully. If you visit the 'Logs' tab, there should be a single log present. If you click anywhere inside the card (except the buttons at the bottom), you should see: • The object that was observed • The observation location (use case 16) • The coordinates that were calculated in the report • The textual description that was recorded in step 3 (use case 13) • The observer(s) that you recorded in step 4 (use case 15) • The type of observation that was entered in step 5 • The weather that the report stated was forecast at the calculated time of observation (use case 14)		
Extra notes	During the analysis and requirements stage, the author made the oversight of imaging users entering *all* of the log's details manually. Of course, this a huge waste of time when many of the details are already stored in the database and can be copied over. Therefore, the use cases refer to the user selecting observation locations and weather conditions etc… In the production software, this is simply pulled in from the report being logged against automatically on the server and users do not need to manually duplicate it.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	13/05/22

9.9.3.12 [TEST 12] - USE CASE 17

Test no.: 12	**Covers use case(s) ID:** 17 (DeleteLog)		
Preconditions	Test no. 11 has been completed and the user is viewing the 'Logs' tab. 1 log is present.		
Steps	1. Click the bin icon on the Polaris and Fort William log 2. The 'Delete log' dialog opens 3. Click the 'Delete' button		
Expected result	A notification is displayed confirming that the log has been deleted. The log is removed from the account, and there are now 0 logs present.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	13/05/22

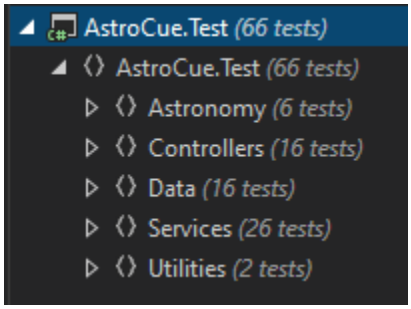
9.9.3.13 [TEST 13] - USE CASE 10

Test no.: 13	**Covers use case(s) ID:** 10 (DeleteObservation)		
Preconditions	Test no. 12 has been completed and the user is viewing the 'Logs' tab. No logs are present.		
Steps	1. Click the bin icon on the Polaris from Fort William observation 2. The 'Delete observation' dialog opens 3. Click the 'Delete' button		
Expected result	A notification is displayed confirming that the observation has been deleted. The observation is removed from the account, and there are now 0 observations present.		
Result	Pass☒ Fail☐		
Tested by	Author	**Date**	13/05/22

9.9.4 UNIT TESTING

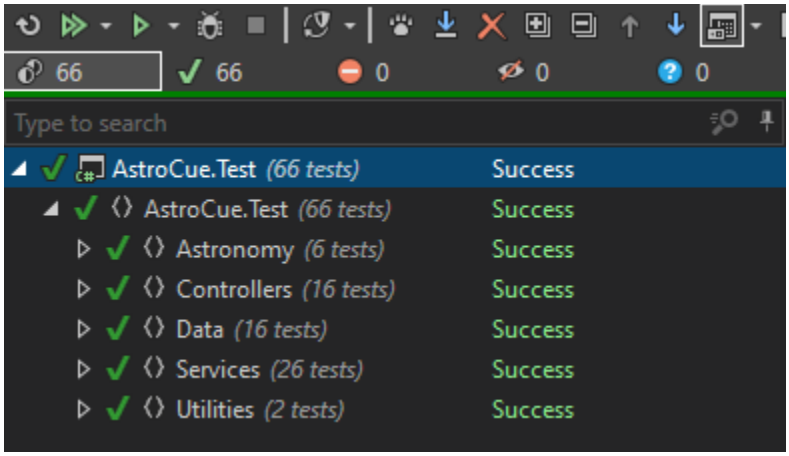
Overall, the author wrote 66 total unit tests for AstroCue. While not entirely comprehensive in either a coverage or use case basis, the unit tests covered several key controllers, many service methods, all the computational astronomy calculations, the catalogue parsers, and some utility classes.

Below is the test breakdown per package, shown by Visual Studio:



9.9.4.1 RESULTS

Unsurprisingly, all unit tests passed. Where a unit test failed, the tested code in question would be fixed so that the test passed with the expected behaviour.



9.9.4.2 IMPORTANT TESTS FOR OBJECTIVES

Some tests were in place to make sure the author was meeting objectives set out at the start of the project.

9.9.4.2.1 SUB OBJECTIVE 3

Sub objective 3, from the PDD and section 1, was *"AstroCue shall be able to calculate positions of astronomical objects in the horizontal coordinate system with an accuracy of ± 10°."* (See - Sub objectives).

The unit test to confirm this objective was in the `CoordinateTransformationsTests` class, specifically the `EquatorialToHorizontalTest` test. This test added and subtracted 10 degrees from known correct coordinate transform test cases and then ensured that AstroCue's calculated values fell within those thresholds. For example:

```
betelgeusePositition.Altitude.Should()
    .BeInRange(betelgeuseExpectedAltitude - 10, betelgeuseExpectedAltitude + 10);
betelgeusePositition.Azimuth.Should()
    .BeInRange(betelgeuseExpectedAzimuith - 10, betelgeuseExpectedAzimuith + 10);
```

As this unit test passed along with all the others, it confirms the success of sub objective 3.

9.9.4.2.2 SUB OBJECTIVE 4

Sub objective 4, from the PDD and section 1, was *"AstroCue shall allow users to register observation locations with any valid latitude and longitude".* (See - Sub objectives).

This test was slightly interesting, as the dataset used for light pollution (discussed in literature review section 3.3.2 and results section 5.4.7), did not cover the full extent of Earth's coordinates. According to the dataset, its extents are: **-180.00**, **-59.99** : **179.99**, **85.05** (longitude, latitude) (Falchi, et al., 2016b). These values are rounded to 2 s.f.

This missing southern ~30° of latitude removes Antarctica from the dataset, which doesn't concern AstroCue as it is almost uninhabited. Meanwhile the missing ~5° of northern latitude falls entirely in the Arctic Ocean so excludes no potential users.

To test that users *within* the bounds of the dataset could have their light pollution levels retrieved, the `ReadLightPollutionForLocation` test was created. The GDAL CLI tools were used to read known correct values from the dataset beforehand so that the unit tests had cases to test against:

```
const float testLongitude1 = -0.0774f;
const float testLatitude1 = 51.5265f;
const float expectedMillicandella1 = 9.6913f;
const int testExpectedBortle1 = 8;

const float testLongitude2 = -2.744f;
const float testLatitude2 = 56.867f;
const float expectedMillicandella2 = 0.0280839f;
const int testExpectedBortle2 = 1;

test1.BortleValue.Should().Be(testExpectedBortle1);
test1.BortleDesc.Should().Be(BortleScale.ScaleToDescription(testExpectedBortle1));
Math.Round(test1.RawMilicandella, 3).Should().Be(Math.Round(expectedMillicandella1,
3));

test2.BortleValue.Should().Be(testExpectedBortle2);
test2.BortleDesc.Should().Be(BortleScale.ScaleToDescription(testExpectedBortle2));
Math.Round(test2.RawMilicandella, 3).Should().Be(Math.Round(expectedMillicandella2,
3));
```

To test the light pollution retrieval for areas *outside* of the dataset failed, the `ReadLightPollutionOutsideOfDatasetTest` test was created:

```
Assert.ThrowsException<ArgumentOutOfRangeException>(() =>
    this._sut.GetLightPollutionForCoords(testLongitude1, testLatitude1));
Assert.ThrowsException<ArgumentOutOfRangeException>(() =>
    this._sut.GetLightPollutionForCoords(testLongitude2, testLatitude2));
Assert.ThrowsException<ArgumentOutOfRangeException>(() =>
    this._sut.GetLightPollutionForCoords(testLongitude3, testLatitude3));
Assert.ThrowsException<ArgumentOutOfRangeException>(() =>
    this._sut.GetLightPollutionForCoords(testLongitude4, testLatitude4));
```

Both unit tests passed as expected.

9.9.4.2.3 SUB OBJECTIVE 7

Sub objective 7, from the PDD and section 1, was "*AstroCue shall be able to retrieve light pollution levels for arbitrary observation locations.*" (See - Sub objectives).

This unit test was in the `LightPollutionServiceTests` class, specifically the `ReadLightPollutionForLocation` test. Again, this test relied on pre-queried values using GDAL. The test then used the service class to query the dataset using the same values to ensure that the calculated values were the same. An example:

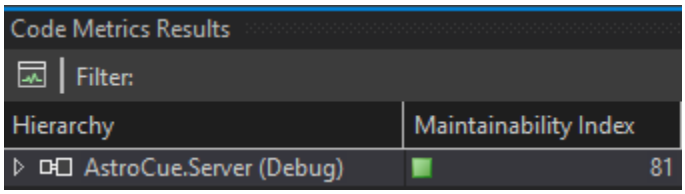
```
test1.BortleValue.Should().Be(testExpectedBortle1);
test1.BortleDesc.Should().Be(BortleScale.ScaleToDescription(testExpectedBortle1));
Math.Round(test1.RawMilicandella, 3).Should().Be(Math.Round(expectedMillicandella1,
3));

test2.BortleValue.Should().Be(testExpectedBortle2);
test2.BortleDesc.Should().Be(BortleScale.ScaleToDescription(testExpectedBortle2));
Math.Round(test2.RawMilicandella, 3).Should().Be(Math.Round(expectedMillicandella2,
3));
```

This unit test also passed, confirming the success of sub objective 7.

9.9.5 MAINTAINABILITY TESTING

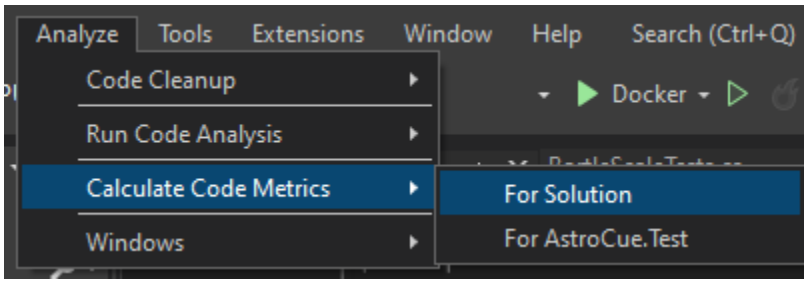
As mentioned in section 4.4.2, the author used the Maintainability Index feature in Visual Studio to test code maintainability levels.

Throughout the implementation of AstroCue, this metric was calculated and any large trouble spots in the code were addressed accordingly. At the end of the implementation, the Author ran a final Maintainability Index calculation and received this result:



81/100 is an acceptable result.

This was calculated in Visual Studio by going to Analyse -> Calculate Code Metrics -> For Solution:



9.10 APPENDIX J: SOFTWARE BUILD GUIDE

Software build guide

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

Consultant: [REDACTED]

9.10.1 INTRODUCTION

This document covers all steps required to run the software locally on your own system, along with all the software needed as a prerequisite.

9.10.2 IMPORTANT NOTE

While you can build and run AstroCue locally, for testing purposes it is easier to visit the live domain where it is hosted, astrocue.co.uk. If you do wish to run it on your own machine however, please read ahead.

Also, the following steps are specific to Windows. AstroCue can be built run on Windows, Mac, and Linux, but the steps will differ.

9.10.3 PREREQUISITES

On top of having the client and server codebases, the following software needs to be installed on any machine that wishes to run AstroCue locally:

1. Visual Studio 2022 (https://visualstudio.microsoft.com/vs/)
2. Docker Desktop (https://www.docker.com/products/docker-desktop/)
3. Microsoft SQL Server Developer Edition (https://www.microsoft.com/en-gb/sql-server/sql-server-downloads)
4. Microsoft SQL Server Management Studio (https://aka.ms/ssmsfullsetup)
5. NodeJS (LTS) (https://nodejs.org/en/)
6. API keys for OpenWeatherMap, MapBox, and, optionally, MailGun.

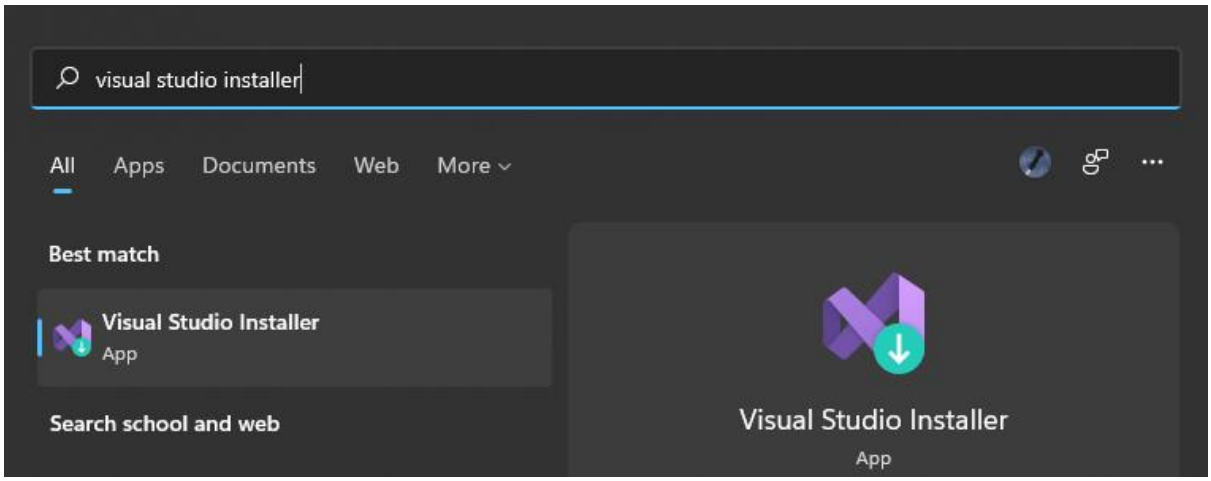
9.10.3.1 EXTRA NICE-TO-HAVES

1. If you are using Windows, install Windows Subsystem for Linux (WSL) version 2 (https://docs.microsoft.com/en-us/windows/wsl/install). This requires virtualization to be enabled on your CPU via your BIOS. The steps to do this will be specific to your machine's hardware. AstroCue can still be ran locally on a Windows machine without WSL, it will just be slower.

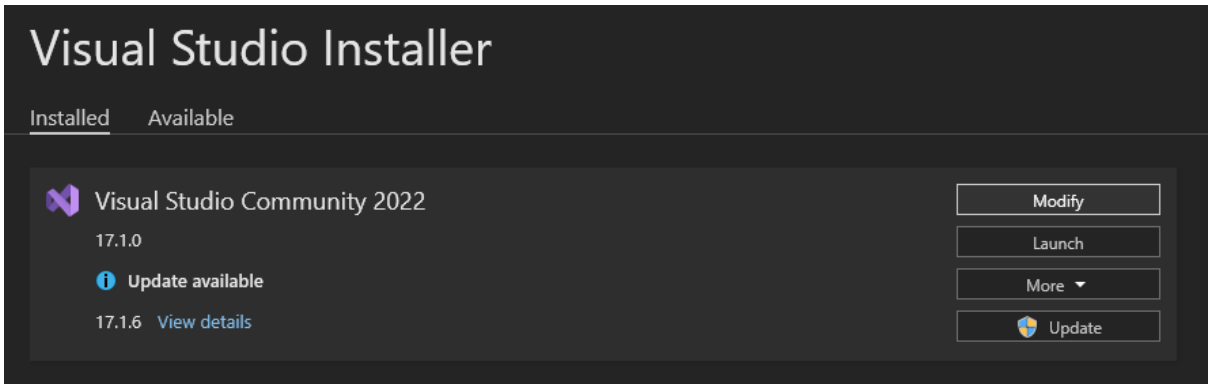
9.10.4 STEPS

9.10.4.1 CONFIGURING VISUAL STUDIO

Once VS 2022 has installed, open `Visual Studio Installer`:



VS 2022 should be present, click the 'Modify' button:



In the next window, you should be on the 'Workloads' tab. Tick and install the following workloads:

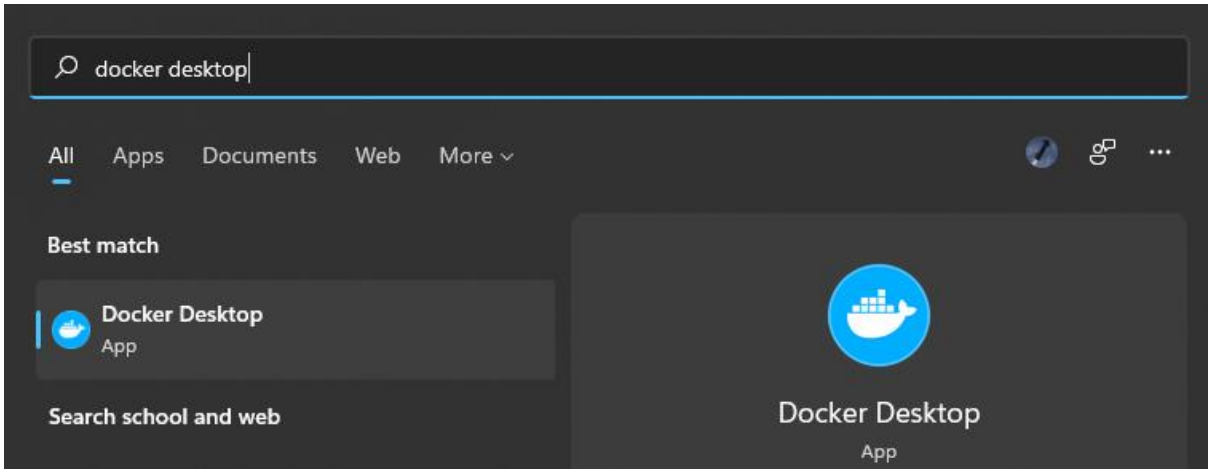
1. ASP.NET and web development
2. Azure development
3. Data storage and processing

In the 'Individual components' tab, make sure that '.NET 5.0 Runtime' is ticked. Then, down the bottom right, click the 'Install while downloading button'. Wait for this to complete.

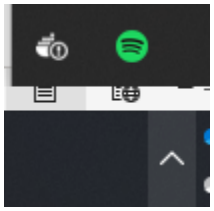
9.10.4.2 Configuring Docker Desktop (optional)

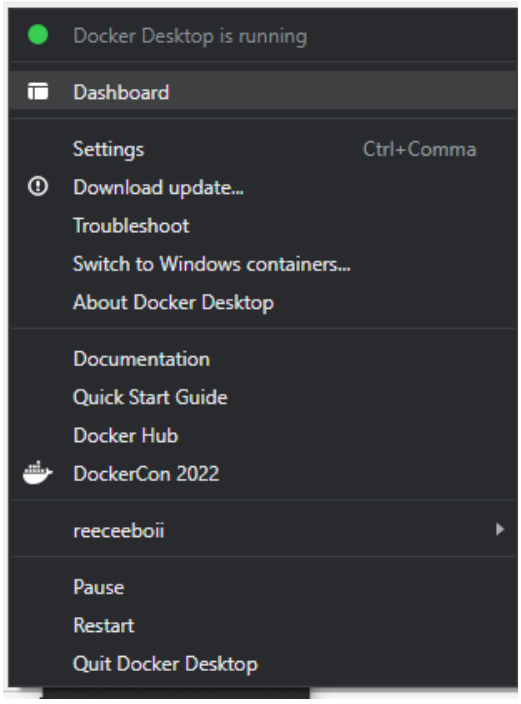
This step only applies if you are running Windows and have WSL2 installed and setup.

Open Docker Desktop:



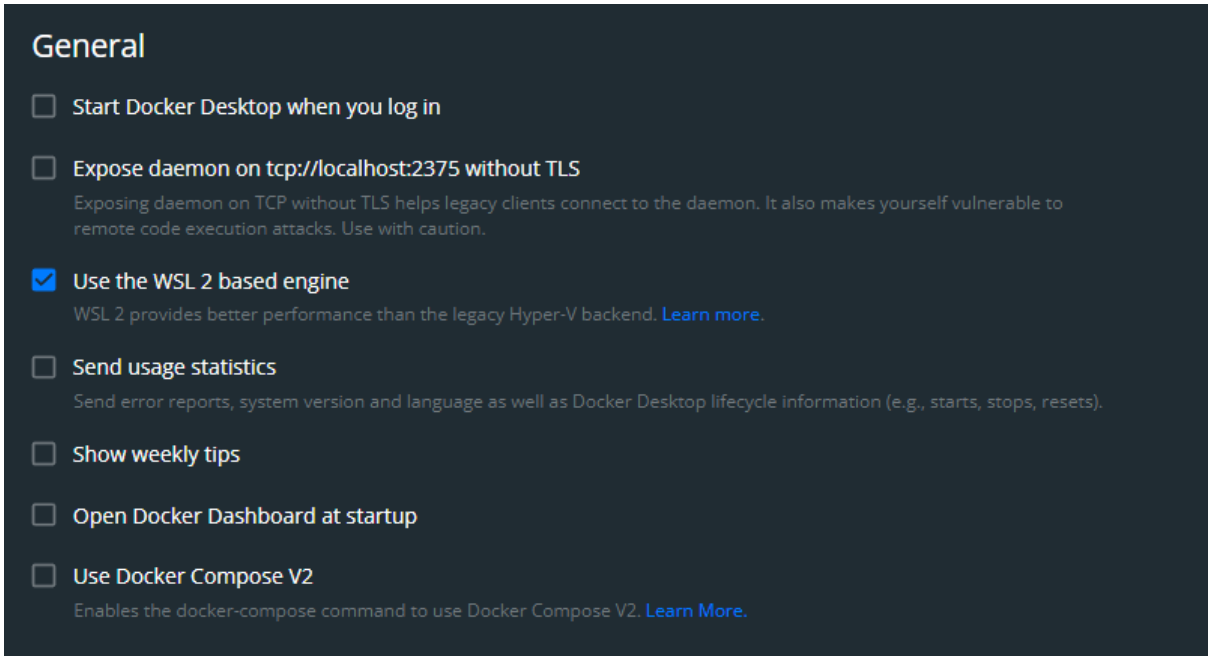
Once you started, it will be hidden in the taskbar overflow, right click it from there and select the desktop option:





On the top bar of Docker, click the settings (cog) icon:

In 'General', make sure the 'Use the WSL 2 based engine' option is checked. You may need to restart Docker and/or your machine for this setting to apply.



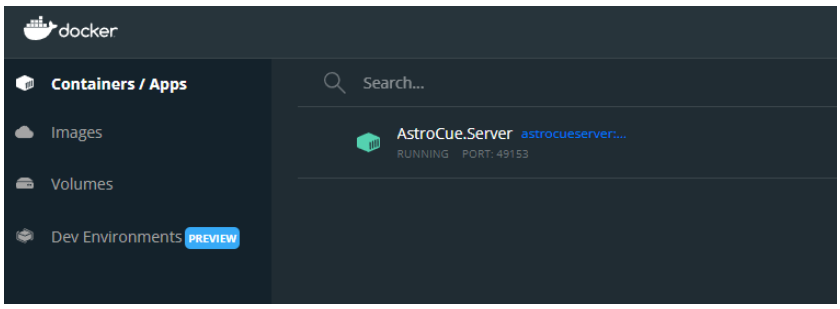
9.10.4.3 CONFIGURING THE SERVER

Navigate to the server codebase on your system. In that folder, there will be a file called '`AstroCue.Server.sln`'. Double click that file and it should open in VS 2022.

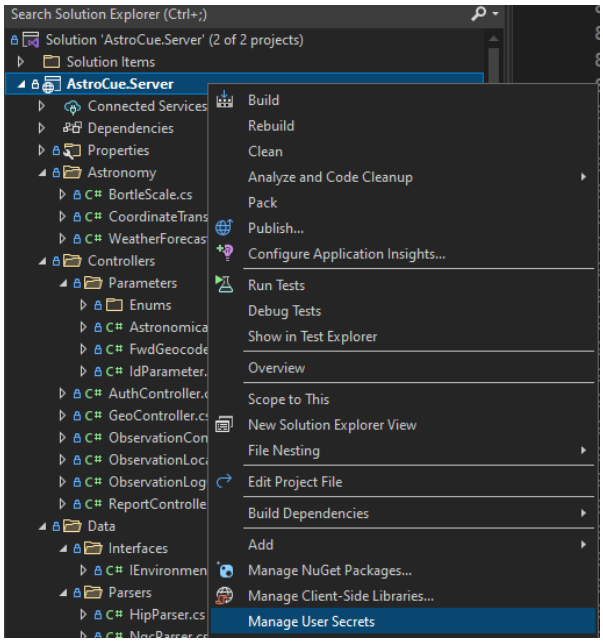
If everything worked up to this point, VS 2022 should have successfully opened, indexed, and configured the solution. You will know if this is the case if:

1. VS has 'Docker' in the execution button:

2. An 'AstroCue.Server' container is running inside Docker:



Next, right click the server project inside the VS Solution Explorer, and click 'Manage User Secrets':



This will open a text file, in this file, there may already be entries, but you need to add the following entries on top of whatever may already be in there:

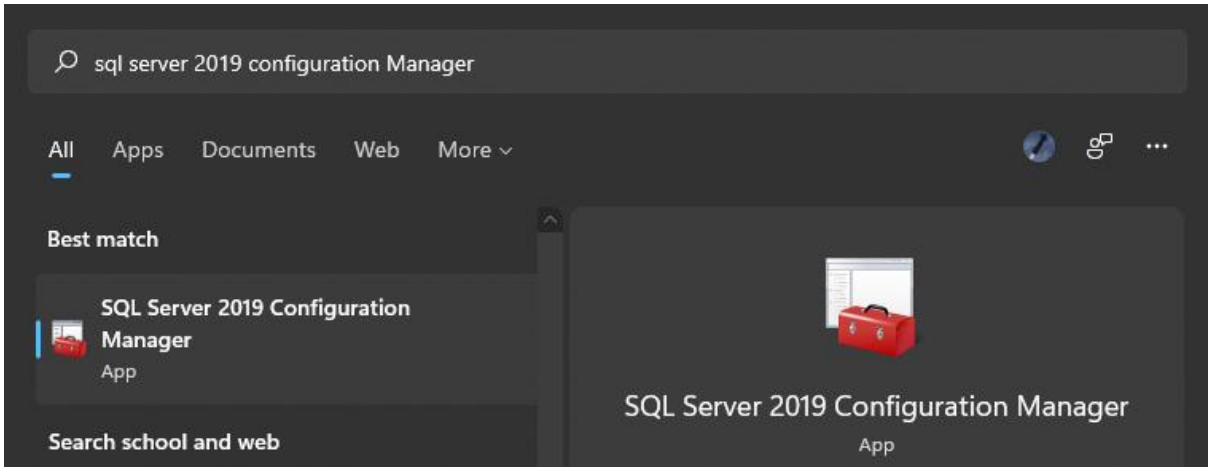
`OpenWeatherMap:APIKey`	`<A valid OWM API key>`
`MapBox:APIKey`	`<A valid MapBox API key>`
`MailGun:APIKey`	`<A valid MailGun API key>`, or an empty string if you plan to set `RegisterDevServices` to `True`. In this case, the server will output email actions to the console rather than contacting the MailGun API.
`Env:RegisterDevServices`	`True` or `False` (see above)

```
1  {
2    "OpenWeatherMap:APIKey": "[redacted],
3    "MapBox:APIKey": "[redacted]",
4    "MailGun:APIKey": [redacted],
5    "Kestrel:Certificates:Development:Password": "6c523655-1249-4910-b127-518b7ab338df",
6    "Env:RegisterDevServices": "True"
7  }
```

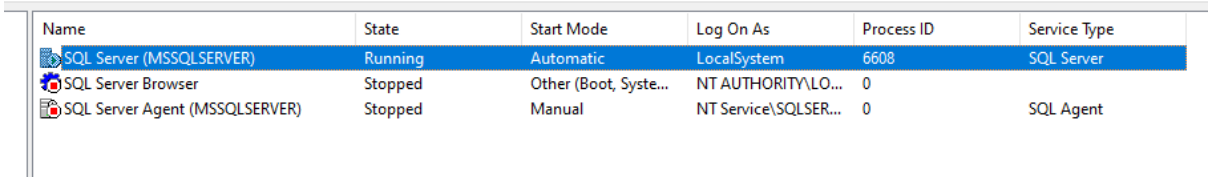
Ensure to save this file before closing it.

9.10.4.4 CONFIGURING THE DATABASE

Open SQL Server 2019 Configuration Manager:

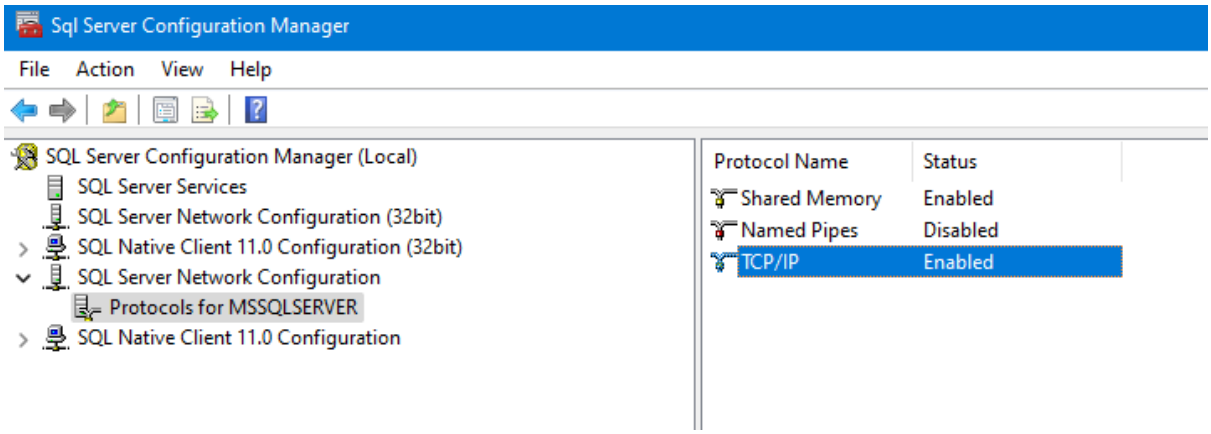


Under 'SQL Server Services', right click 'SQL Server' and select 'Start'. This will start the SQL Server Windows service and it should look like this:

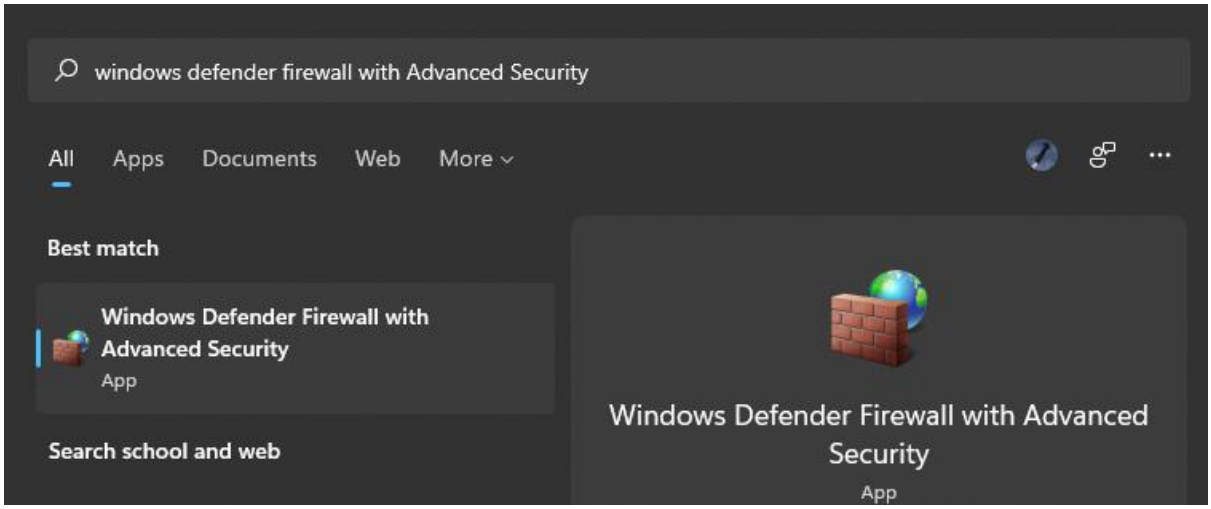


Name	State	Start Mode	Log On As	Process ID	Service Type
SQL Server (MSSQLSERVER)	Running	Automatic	LocalSystem	6608	SQL Server
SQL Server Browser	Stopped	Other (Boot, Syste...	NT AUTHORITY\LO...	0	
SQL Server Agent (MSSQLSERVER)	Stopped	Manual	NT Service\SQLSER...	0	SQL Agent

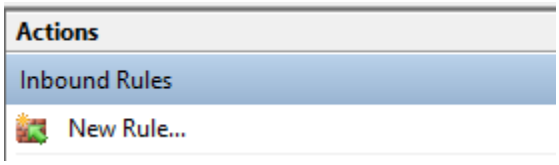
Next, under 'SQL Server Network Configuration', set 'TCP/IP' to 'Enabled':



Open Windows Defender Firewall with Advanced Security:



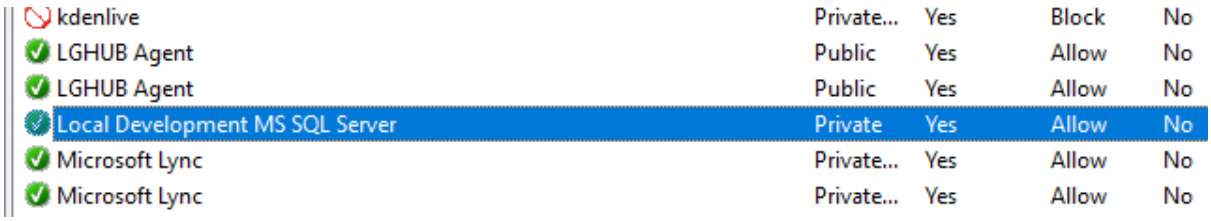
Under 'Inbound Rules', make a new rule:



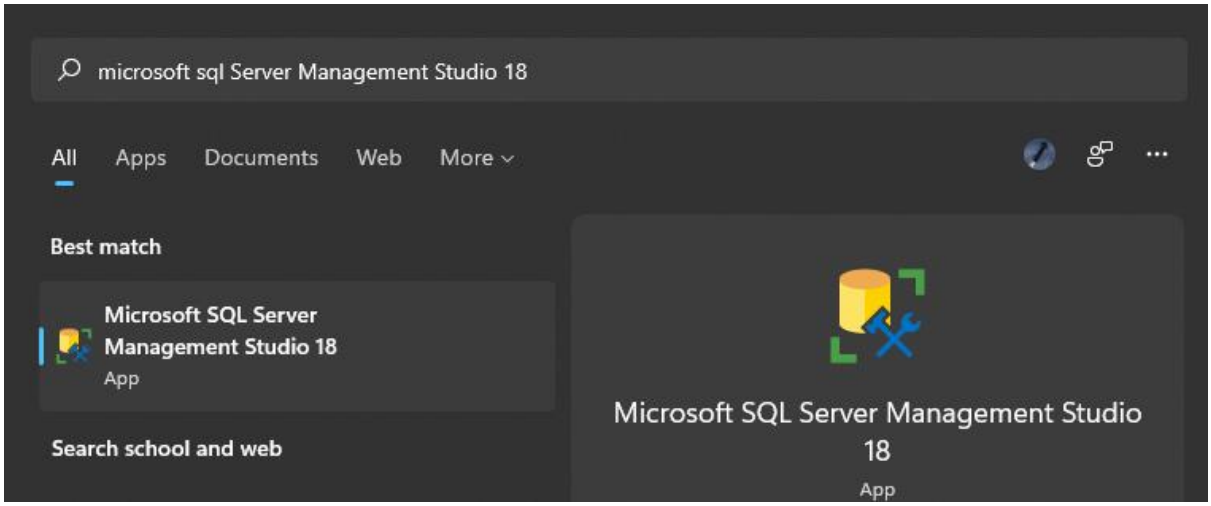
In the popup window, in the corresponding steps: create the following firewall rule:

Step	Action
Rule Type	Port
Protocol and Ports	• TCP • Specific local port 1433
Action	Allow the connection
Profile	Tick all 3 boxes (Domain, Private and Public)
Name	Name the rule something descriptive

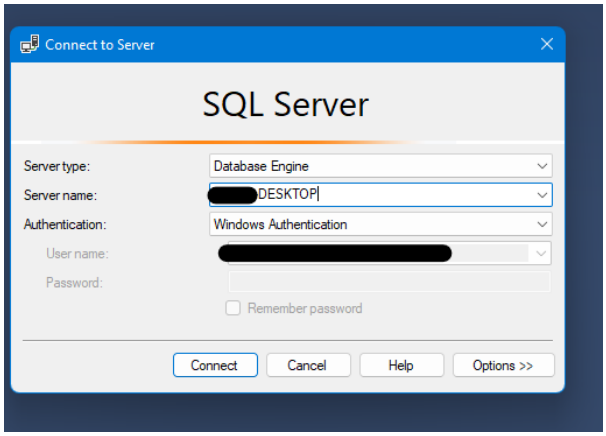
This rule should then be present under the firewall's inbound rules:



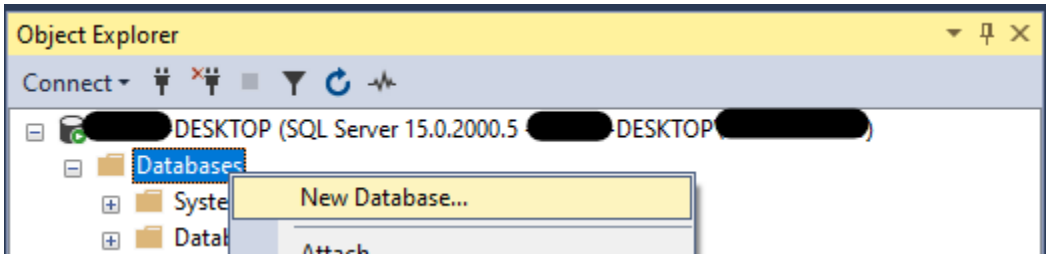
Open Microsoft SQL Server Management Studio:



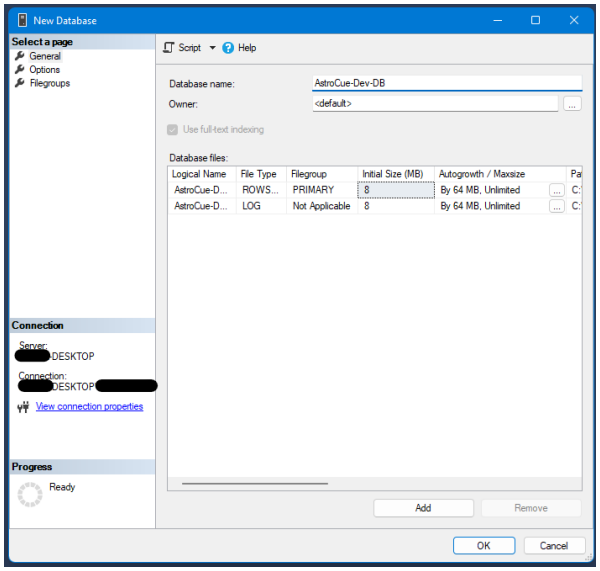
Log in to your local server using the 'Windows Authentication' scheme. The details for this should be automatically filled in as you are already logged in to your system:



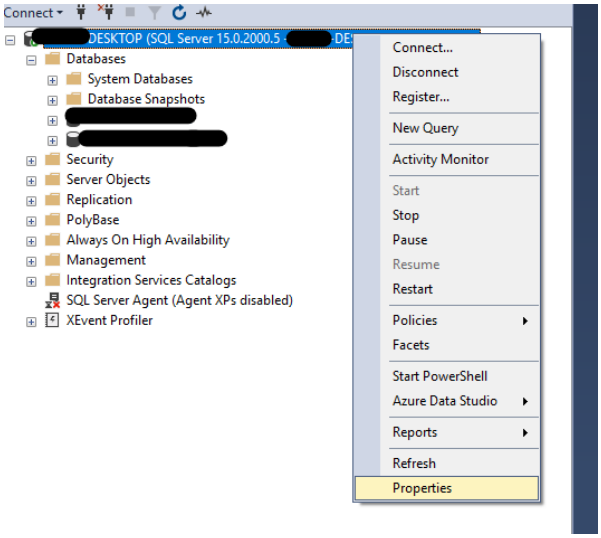
In the object explorer, expand your local server, and right click the 'Databases' folder. In the context menu, click 'New Database…':



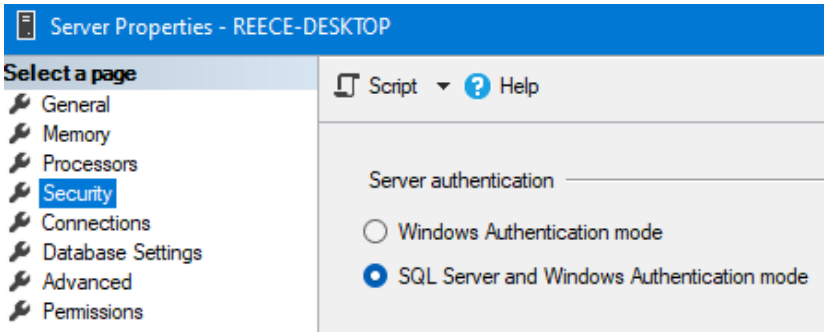
This will be the local database that the AstroCue server will use. Give it a name that makes sense such as 'AstroCue-Dev-DB'. Leave the other options as their defaults and click 'OK':



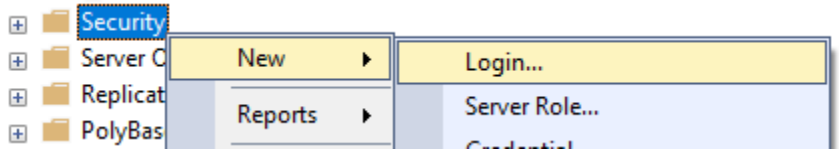
In the object explorer, right click your local server, and then enter the 'Properties' menu:



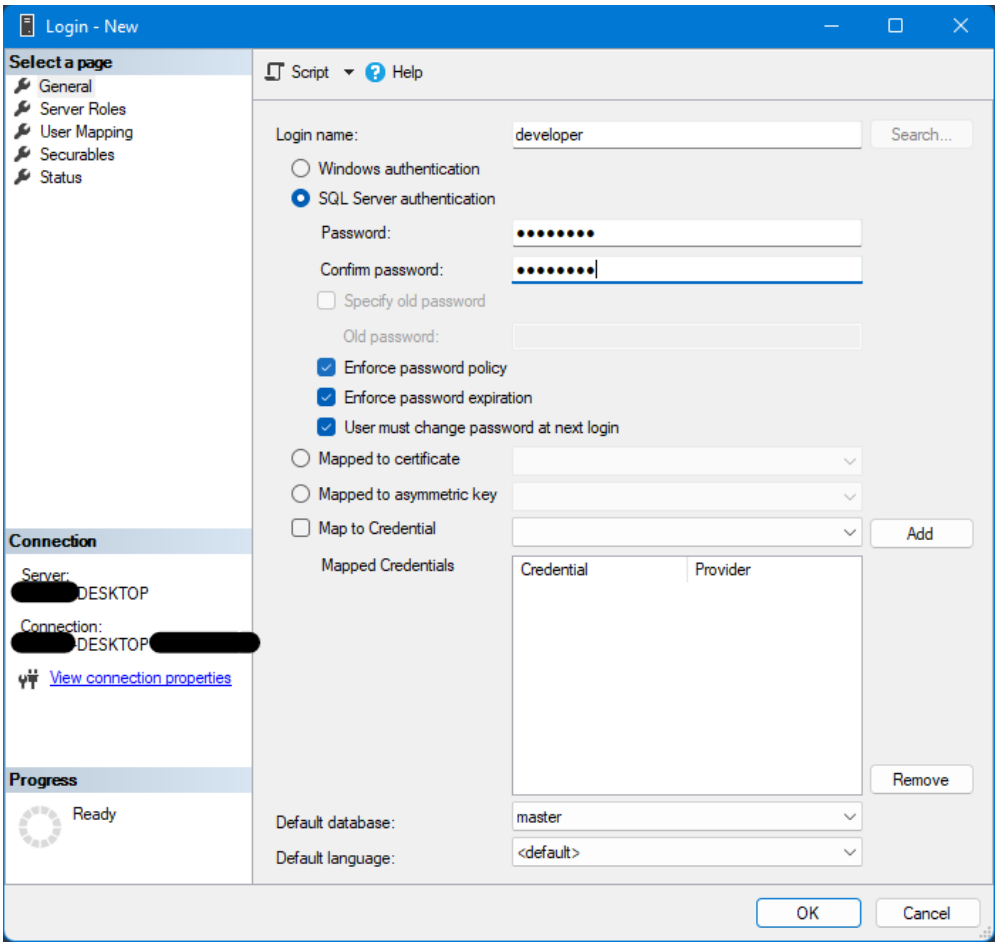
Under 'Security', set 'Server authentication' to 'SQL Server and Windows Authentication mode':



In the object explorer, right click the 'Security' folder under your database and then 'New' -> 'Login':



Create a user with the login name 'developer' and password 'password'. You will need to uncheck the 'Enforce password policy' option. Leave the rest of the settings as their defaults, and click 'OK':



Now that the database is fully configured, we need to create the connection string – essentially a combination of a username and password that the server can use to authenticate with the database server. The connection string is as follows:

```
Server=1.1.1.1.,1433;Database=AstroCue-Dev-DB;User
Id=developer;Password=password;MultipleActiveResultSets=True
```

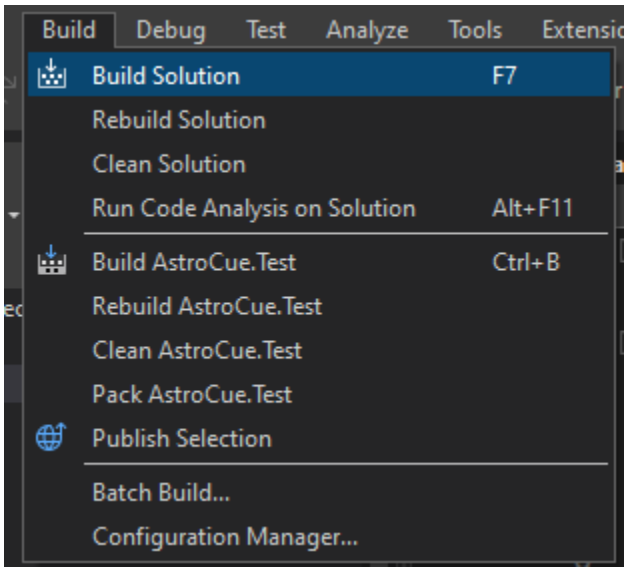
Replace **1.1.1.1** with your machine's local LAN IP address, and the database name with the name you chose earlier.

In the solution explorer in Visual Studio, expand the ‘`appsettings.json`’ file to find a file called ‘`appsettings.Development.json`’. Open this file and add the following two sections:

```
{
  "AppSettings": {
    "JsonWebTokenSecret": "some_very_secret_string"
  },
  "ConnectionStrings": {
    "SQLServer": "Server=192.168.1.2,1433;Database=AstroCue-Dev-DB;User Id=developer;Password=password;MultipleActiveResultSets=True"
  },
  "Logging": {
```

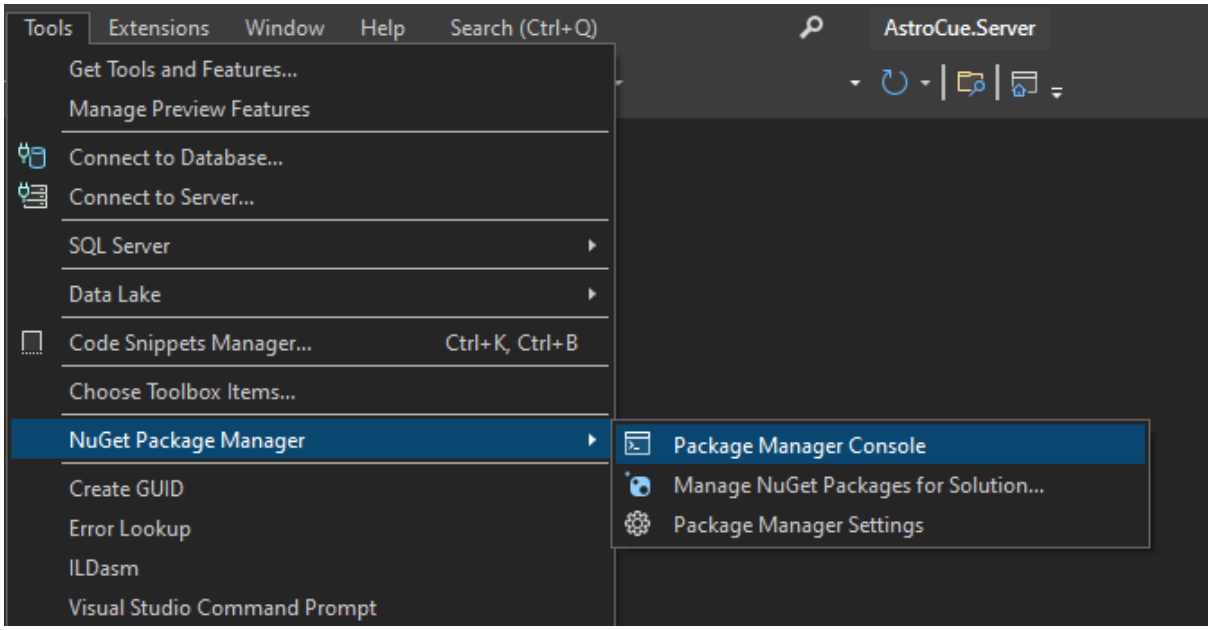
The `JsonWebTokenSecret` is used by the server as a server side JWT encryption secret, and `SQLServer` is the connection string that the server will read.

We will now build the code and allow Visual Studio to install all the required Nuget packages. Open the Build menu, and select ‘Build Solution’:

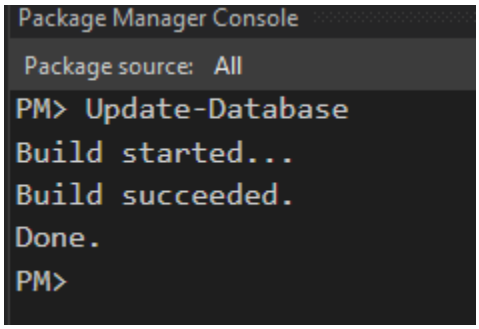


This should complete with zero errors.

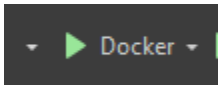
Now, still in Visual Studio, open the Package Manager Console:

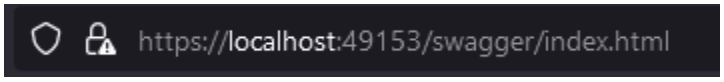


In this console, type the command '`Update-Database`' – this will make EF Core run the database migrations to bring the database in line with the models:



The server is now ready. Run it using the Docker button and wait for the server to open its documentation in your browser. This may take a few minutes on the first boot as it needs to seed the database:





Make a note of this address and port, it will be needed in a minute.

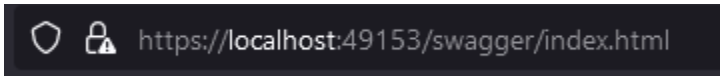
9.10.4.5 CONFIGURING THE CLIENT

Navigate to where the client code is stored on your development machine. In its root folder, create a new file called `.env.local` – the dot before this file is important – the name must exactly match.

In this file, write a single line that matches the following:

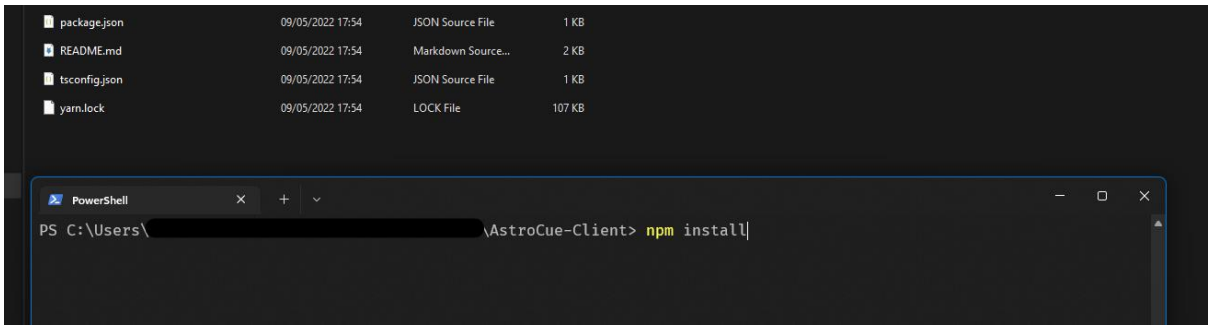
`NEXT_PUBLIC_SERVER_URL=https://localhost:49153`

The address and port on this line must match the one that the server opened in your browser. This is so that the client knows where the backend is:



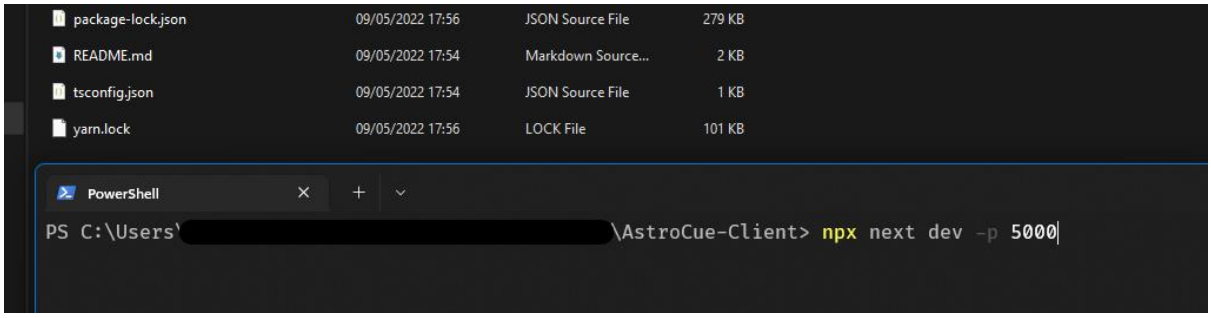
Next, open a terminal and set its working directory to the root of the client code. This can be done by opening a terminal directly inside the folder using the context menu (Windows 11) or simply navigating there manually via the `cd` command.

Once there, run the command `npm install` to download and configure all required dependencies from NPM:

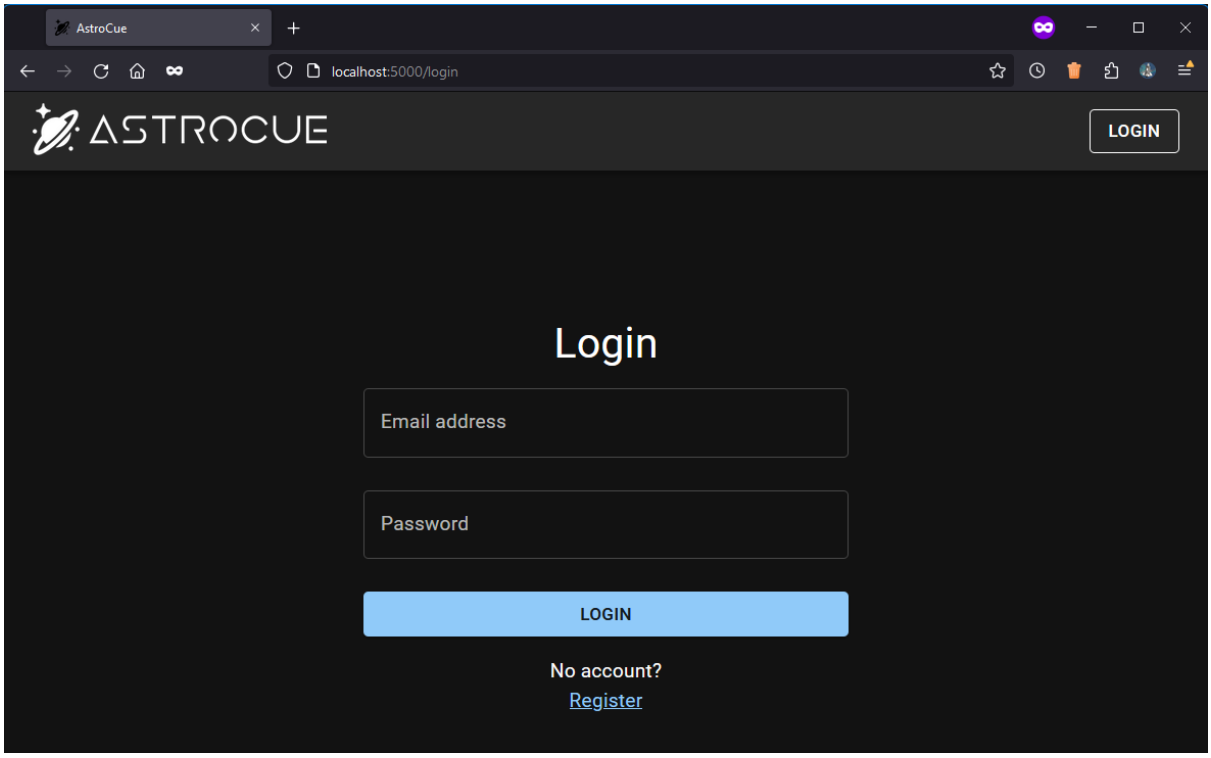


This should complete with no errors.

Finally, run the command `npx next dev` to start up the development server for the client. If you receive any errors regarding ports, try a different port via the `-p` switch:



Once this command succeeds, navigate to the relevant address and port in your browser and you should be greeted with the AstroCue frontend:



You are now done, AstroCue is fully running on your system!

9.11 APPENDIX K: CODE SNIPPETS

Code snippets

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

Consultant: [REDACTED]

9.11.1 INTRODUCTION

This document includes code snippets that were too large to be included in the results section.

9.11.2 Full OpenWeatherMap request with JSON parsing

```
public async Task<SingleForecast> GetCurrentWeatherAsync(float longitude, float
latitude)
{
    RestRequest req = new(this._environmentManager.BaseOwmCurrentWeatherUrl);

    req.AddParameter("lat", latitude);
    req.AddParameter("lon", longitude);
    req.AddParameter("units", "metric");
    req.AddParameter("appid", this._environmentManager.OpenWeatherMapApiKey);

    RestResponse res = await this._client.GetAsync(req);

    if (res.Content == null)
    {
        return new SingleForecast
        {
            Description = "Data could not be retrieved"
        };
    }

    JObject jObject = JObject.Parse(res.Content);

    return new SingleForecast
    {
        Description =
StringUtilities.TrimToUpperFirstChar((string)jObject["weather"]![0]!["description"]),
        CloudCoveragePercent = (int)jObject["clouds"]!["all"],
        TemperatureCelcius = Convert.ToInt32((int)jObject["main"]!["temp"]),
        RetrievedAt = DateTime.Now.ToString("s")
    };
}
```

9.11.3 FULL MAILGUN API REQUEST USING TEMPLATES

```
public async Task<RestResponse> SendWelcomeEmail(AstroCueUser user)
{
    string templateData =
JsonConvert.SerializeObject(this._mapper.Map<WelcomeEmailModel>(user));

    RestRequest req = new();
    req.AddParameter("domain", "astrocue.co.uk", ParameterType.UrlSegment);
    req.Resource = "{domain}/messages";
    req.AddParameter("from", "AstroCue <no-reply@astrocue.co.uk>");
    req.AddParameter("to", user.EmailAddress);
    req.AddParameter("subject", $"Welcome, {user.FirstName}!");
    req.AddParameter("template", WelcomeEmailTemplate);
    req.AddParameter("h:X-Mailgun-Variables", templateData);
    req.Method = Method.Post;

    return await this._client.ExecuteAsync(req);
}
```

9.11.4 LightPollutionService constructor

```
public LightPollutionService(IFileSystem fileSystem)
{
    GdalBase.ConfigureAll();

    string dataFilePath = fileSystem.Path.Join("./Res", DataFileName);
    if (!fileSystem.File.Exists(dataFilePath))
    {
        throw new FileNotFoundException(nameof(DataFileName));
    }

    this._worldAtlasDataset = Gdal.Open(dataFilePath, Access.GA_ReadOnly);

    if (this._worldAtlasDataset == null)
    {
        throw new Exception("World Atlas dataset loaded via GDAL is null");
    }

    if (this._worldAtlasDataset.GetDriver() == null)
    {
        throw new Exception("Driver is null");
    }

    // extract the dataset's geo transform matrix
    this._geoTransformMatrix = new double[6];
    this._worldAtlasDataset.GetGeoTransform(this._geoTransformMatrix);
}
```

9.11.5 LightPollutionService retrieving light pollution levels for coordinates

```
public LightPollution GetLightPollutionForCoords(float longitude, float latitude)
{
    Band band = this._worldAtlasDataset.GetRasterBand(1);

    // calculate pixel offsets
    // adapted from https://gis.stackexchange.com/questions/245472/convert-latitude-
longitude-pair-to-pixels-in-geotiff
    int xOffset = (int)((longitude - this._geoTransformMatrix[0]) /
this._geoTransformMatrix[1]);
    int yOffset = (int)((latitude - this._geoTransformMatrix[3]) /
this._geoTransformMatrix[5]);

    // check that calculated offsets are in bounds
    if (OffsetsOutOfBounds(band, xOffset, yOffset))
    {
        throw new ArgumentOutOfRangeException(nameof(band), "Given coordinates lie
outside of dataset");
    }

    // read band 1 data at x and y offsets [mcd/m^2]
    float[] milicandellaPerM2 = new float[1];
    band.ReadRaster(xOffset, yOffset, 1, 1, milicandellaPerM2, 1, 1, 0, 0);

    int bortle = BortleScale.McdM2ToBortle(milicandellaPerM2[0]);

    return new LightPollution()
    {
        BortleValue = bortle,
        BortleDesc = BortleScale.ScaleToDescription(bortle),
        RawMilicandella = milicandellaPerM2[0]
    };
}
```

9.11.5.1 Bounds check method

```
private static bool OffsetsOutOfBounds(Band band, int xOffset, int yOffset)
{
    if (xOffset < 0 || xOffset > band.XSize)
    {
        return true;
    }

    return yOffset < 0 || yOffset > band.YSize;
}
```

9.11.6 BORTLE SCALE METHODS

```
public static int McdM2ToBortle(float mcdM2)
{
    return mcdM2 switch
    {
        <= 0.25f => 1,
        > 0.25f and <= 0.275f => 2,
        > 0.275f and <= 0.325f => 3,
        > 0.325f and <= 0.5f => 4,
        > 0.5f and <= 2.5f => 5,
        > 2.5f and <= 4f => 6,
        > 4f and <= 6.85f => 7,
        _ => 8
    };
}

public static string ScaleToDescription(int bortle)
{
    return bortle switch
    {
        1 => "Excellent Dark-sky Site",
        2 => "Typical Truly Dark Site",
        3 => "Rural Sky",
        4 => "Rural/Suburban Transition",
        5 => "Suburban Sky",
        6 => "Bright Suburban Sky",
        7 => "Suburban/Urban Transition",
        8 => "City or inner city sky",
        _ => throw new ArgumentOutOfRangeException(nameof(bortle), "Argument out of
range")
    };
}

public static float ScaleToNakedEyeLimitingMagnitude(int bortle)
{
    return bortle switch
    {
        1 => 7.8f,
        2 => 7.3f,
        3 => 6.8f,
        4 => 6.3f,
        5 => 5.8f,
        6 => 5.5f,
        7 => 5.0f,
        8 => 4.25f,
        _ => throw new ArgumentOutOfRangeException(nameof(bortle), "Argument out of
range")
    };
}
```

9.11.7 COMPUTATIONAL ASTRONOMY CALCULATIONS

9.11.7.1 CONVERTING UTC DATE TO JULIAN DAY

```
public static double DateToJulianDay(DateTime instant)
{
    if (instant.Kind != DateTimeKind.Utc)
    {
        throw new ArgumentException("Date needed in UTC", nameof(instant));
    }

    int year = instant.Year;
    int month = instant.Month;
    double dayFrac = instant.Hour / 24.0 + instant.Minute / 1440.0 + instant.Day;

    if (!(month > 2))
    {
        year -= 1;
        month += 12;
    }

    int a = (int)Math.Floor(year / 100.0);
    int b = 2 - a + (int)Math.Floor(a / 4.0);

    return Math.Floor(365.25 * (year + 4716)) + Math.Floor(30.6001 * (month + 1)) +
dayFrac + b - 1524.5;
}
```

9.11.7.2 SIDEREAL TIME

```
public static double MeanSiderealTimeAtInstant(DateTime instant)
{
    if (instant.Kind != DateTimeKind.Utc)
    {
        throw new ArgumentException("Date needed in UTC", nameof(instant));
    }

    double jd = DateToJulianDay(instant);
    double t = (jd - 2451545.0) / 36525;

    double mst = 280.46061837 + 360.98564736629 * (jd - 2451545.0) + 0.000387933 * (t
* t) - t * t * t / 38710000;

    // bring in range 0-360
    mst = BringInRange360(mst);

    return mst;
}

private static double BringInRange360(double val)
{
    if (val < 360)
    {
        while (val < 0)
        {
            val += 360;
        }
    }

    if (val > 360)
    {
        while (val > 360)
        {
            val -= 360;
        }
    }

    return val;
}
```

9.11.7.3 EQUATORIAL COORDINATE TO HORIZONTAL COORDINATE TRANSFORM

```
public static AltAz EquatorialToHorizontal(AstronomicalObject obj, DateTime instant,
float longitude, float latitude)
{
    if (instant.Kind != DateTimeKind.Utc)
    {
        throw new ArgumentException("Date needed in UTC", nameof(instant));
    }

    // first, convert the object's Right Ascension and Declination values to degrees
    // (Meeus, 1998, p. 8)
    double raDeg = (obj.RightAscension.Hours + obj.RightAscension.Minutes / 60.0 +
                    obj.RightAscension.Seconds / 3600.0) * 15;

    // if declination's degrees are negative, all components need to be negated
    double decDeg;
    if (obj.Declination.Degrees < 0)
    {
        int m = -obj.Declination.Minutes;
        double s = -obj.Declination.Seconds;
        decDeg = obj.Declination.Degrees + m / 60.0 + s / 3600.0;
    }
    else
    {
        int m = obj.Declination.Minutes;
        double s = obj.Declination.Seconds;
        decDeg = obj.Declination.Degrees + m / 60.0 + s / 3600.0;
    }

    // and then convert to radians
    double decRad = DegToRad(decDeg);

    // convert observation latitude to radians
    double latitudeRad = DegToRad(Convert.ToDouble(latitude));

    // calculate hour angle (Meeus, 1998, p. 92)
    double hourAngleDeg = MeanSiderealTimeAtInstant(instant) - longitude - raDeg;
    hourAngleDeg = BringInRange360(hourAngleDeg);

    double hourAngleRad = DegToRad(hourAngleDeg);

    // calculate local apparent altitude and azimuth (Meeus 1998, p. 93) equations
13.5 & 13.6
    double azimuthRad = Math.Atan2(Math.Sin(hourAngleRad),
        Math.Cos(hourAngleRad) * Math.Sin(latitudeRad) - Math.Tan(decRad) *
Math.Cos(latitudeRad));

    double altitudeRad = Math.Sin(latitudeRad) * Math.Sin(decRad) +
                         Math.Cos(latitudeRad) * Math.Cos(decRad) *
Math.Cos(hourAngleRad);

    float az = (float)RadToDeg(azimuthRad);
    float alt = (float)RadToDeg(Math.Asin(altitudeRad));

    AltAz coordinates = new()
    {
        // derive Azimuth from North instead of South by adding 180 degrees
        Azimuth = az + 180,
        Altitude = alt
    };

    return coordinates;
}
```

9.11.8 WEATHER INDEX SORTING

```
HourlyForecast best = null;
DateTime timeOfBest = DateTime.Now.ToUniversalTime();
float bestIndex = 0f;

DateTime astronomicalSunset =
forecastForLocation.Sunset.AddMinutes(90).ToUniversalTime();
DateTime astronomicalSunrise = forecastForLocation.Sunrise.AddMinutes(-
90).ToUniversalTime();

DateTime endOfWindow =
timeOfReportGenUtc.AddHours(forecastForLocation.Forecasts.Count);
for (DateTime i = timeOfReportGenUtc; i <= endOfWindow; i = i.AddHours(1))
{
    // if iterated hour does not fall into nighttime
    if (!(i.Hour > astronomicalSunset.Hour || i.Hour < astronomicalSunrise.Hour))
    {
        continue;
    }

    // construct a new EqTimeRecord instance for easy indexing into forecast list
    EqTimeRecord eq = new()
    {
        Day = i.Day,
        Hour = i.Hour
    };

    // if there's no weather forecast instance available (there should be, this is
just a sanity check)
    if (!forecastForLocation.Forecasts.TryGetValue(eq, out HourlyForecast
forecastInstance))
    {
        continue;
    }

    if (best == null)
    {
        best = forecastInstance;
        bestIndex = WeatherForecastRating.CalculateIndex(forecastInstance);
        timeOfBest = i;
    }
    else
    {
        float index = WeatherForecastRating.CalculateIndex(forecastInstance);

        if (!(index < bestIndex))
        {
            continue;
        }

        best = forecastInstance;
        bestIndex = index;
        timeOfBest = i;
    }
}
```

9.12 APPENDIX L: FULL CODEBASE

To see the code in action, AstroCue is hosted live at astrocue.co.uk.

For the raw and full code, see the Moodle submission.

9.13 APPENDIX M: USER INTERFACE / ALL WEB PAGES

User interface / all web pages

City, University of London

AstroCue: planning and logging astronomical observations via computational astronomy, open data, and public APIs

Proposed by: Reece Mercer [REDACTED]

BSc Computer Science

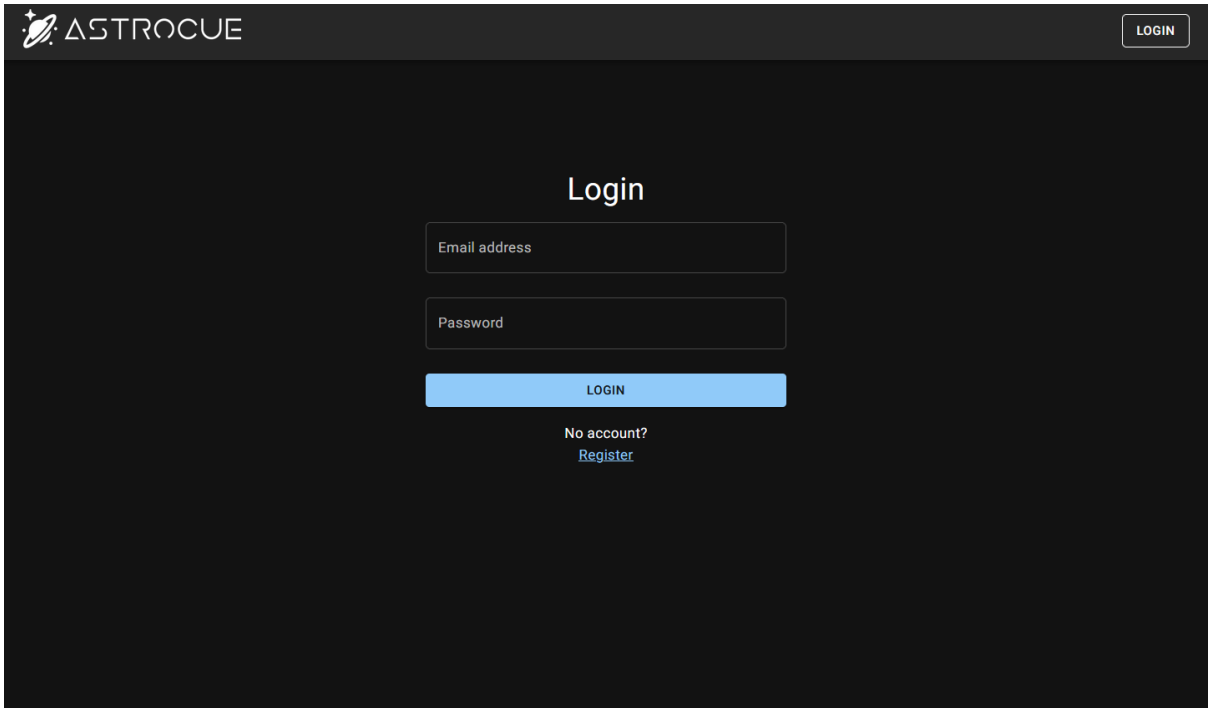
Consultant: [REDACTED]

9.13.1 INTRODUCTION

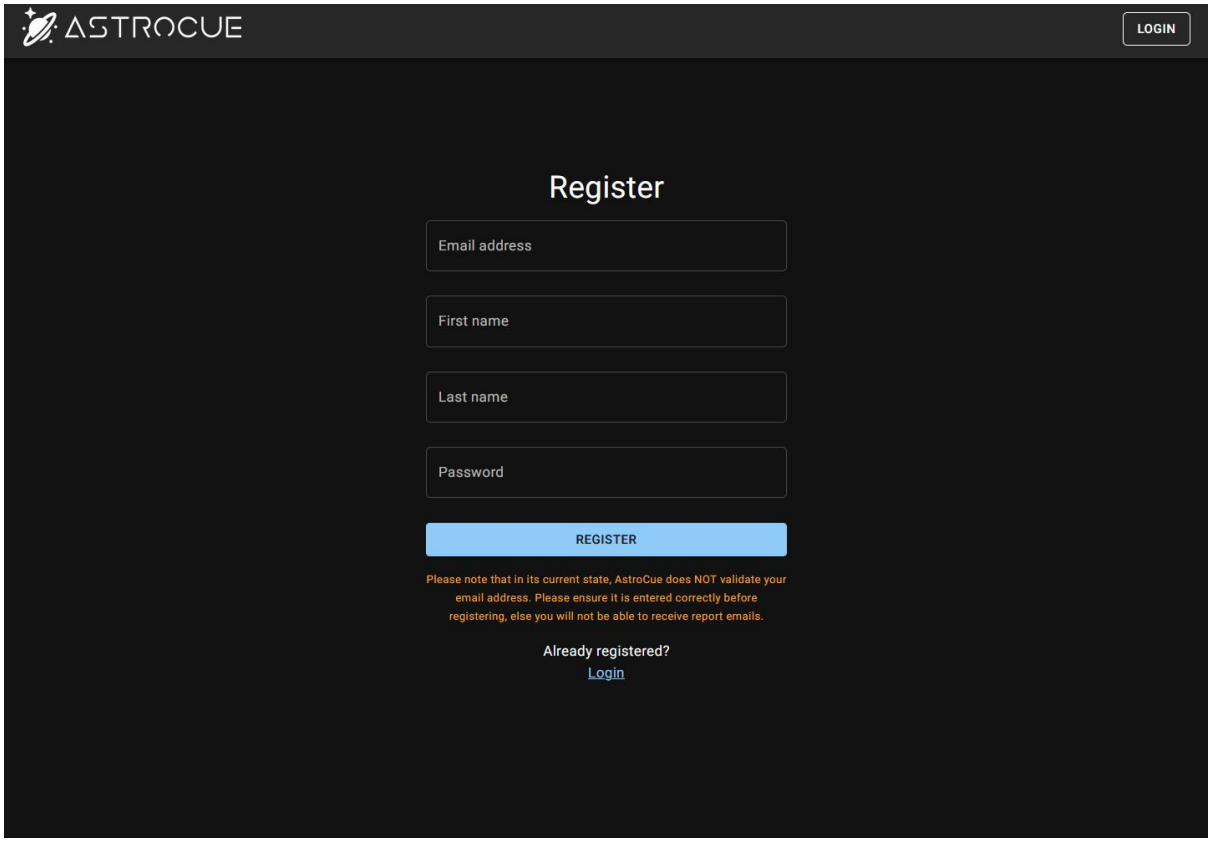
This appendix entry houses all the user interface elements produced for AstroCue.

9.13.2 AUTHENTICATION

9.13.2.1 LOGIN PAGE

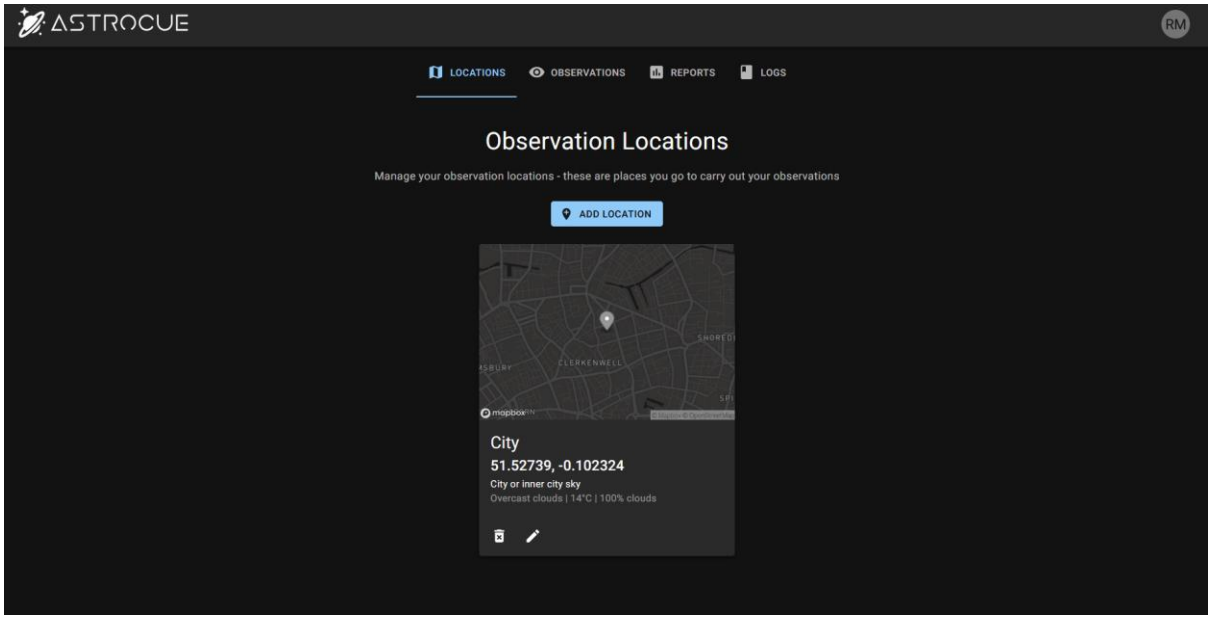


9.13.2.2 REGISTRATION PAGE

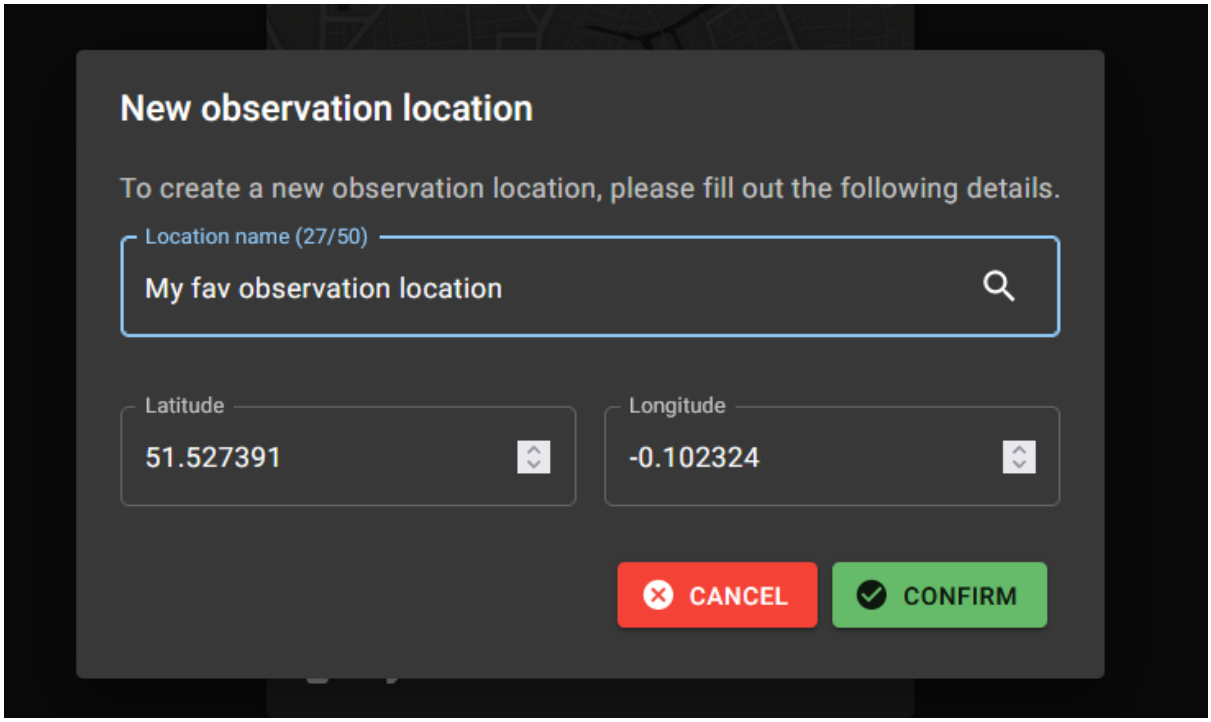


9.13.3 OBSERVATION LOCATIONS

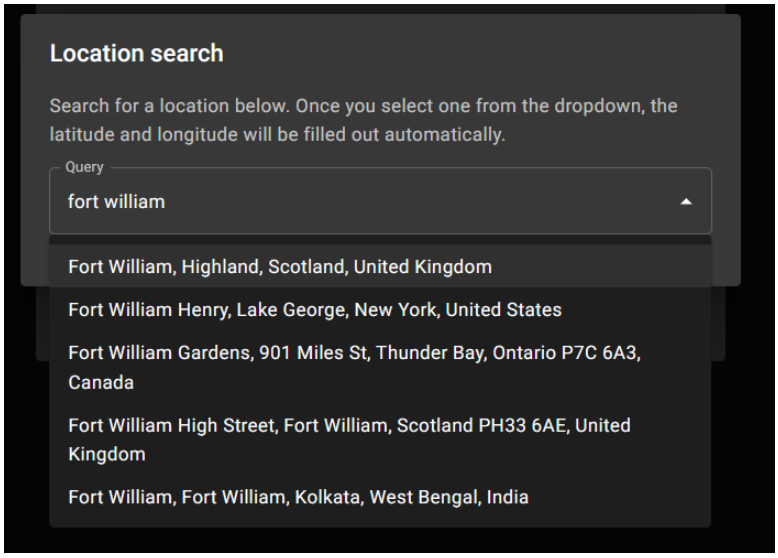
9.13.3.1 OBSERVATION LOCATIONS TAB



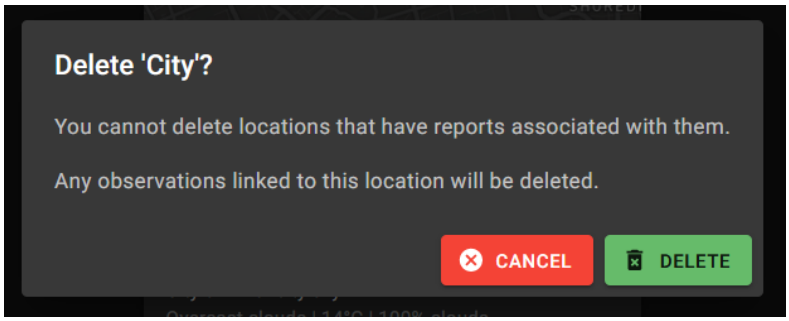
9.13.3.2 NEW OBSERVATION LOCATION DIALOG



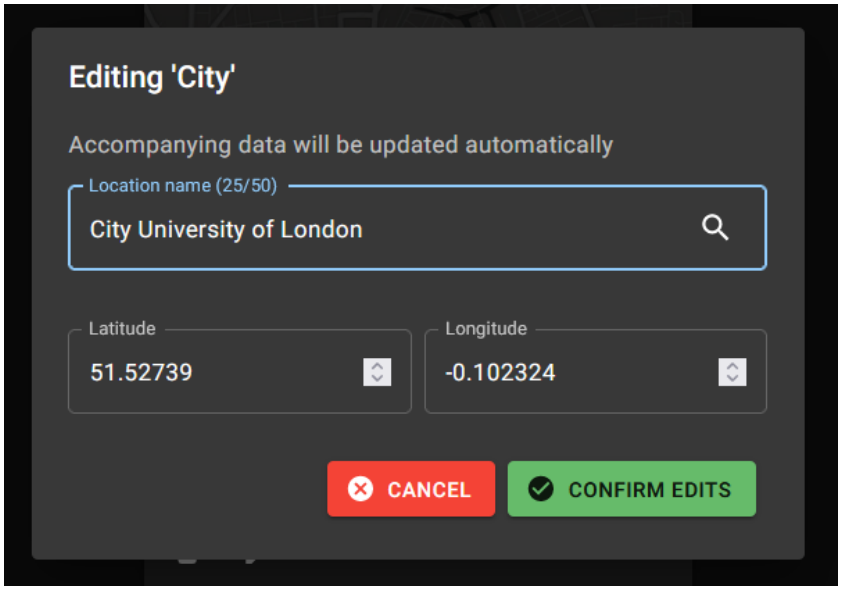
9.13.3.3 FORWARD GEOCODE DIALOG



9.13.3.4 DELETE OBSERVATION LOCATION DIALOG

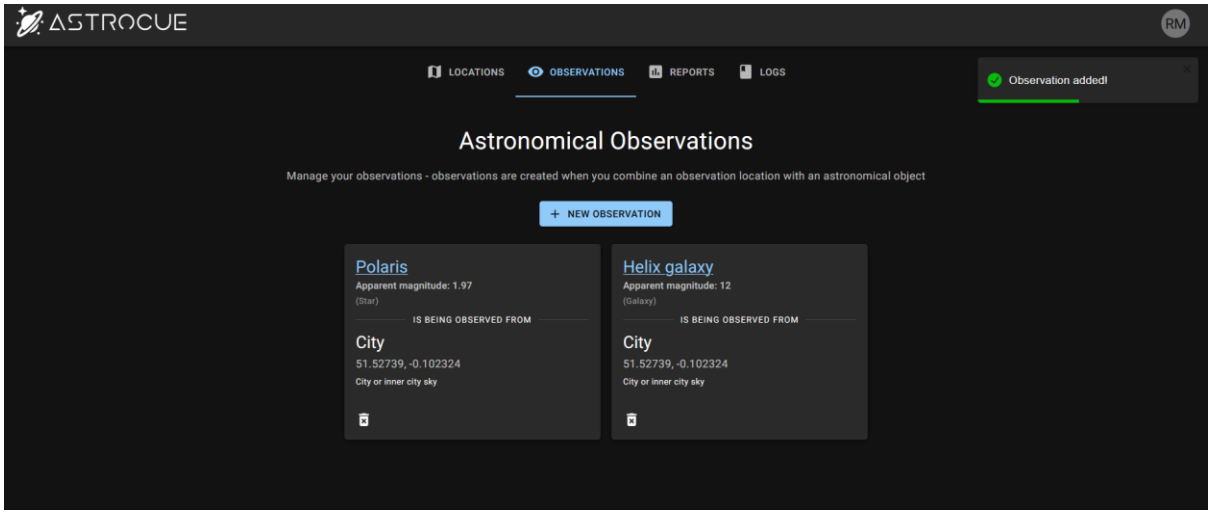


9.13.3.5 EDIT OBSERVATION LOCATION DIALOG

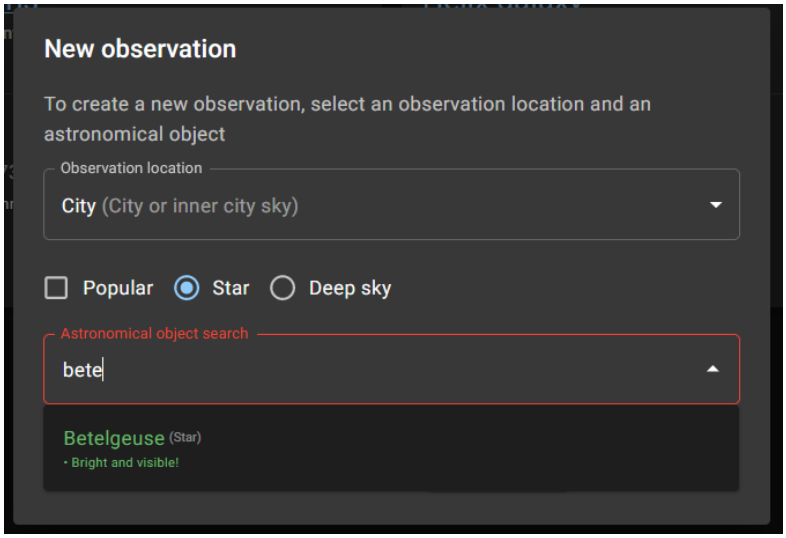


9.13.4 Observations

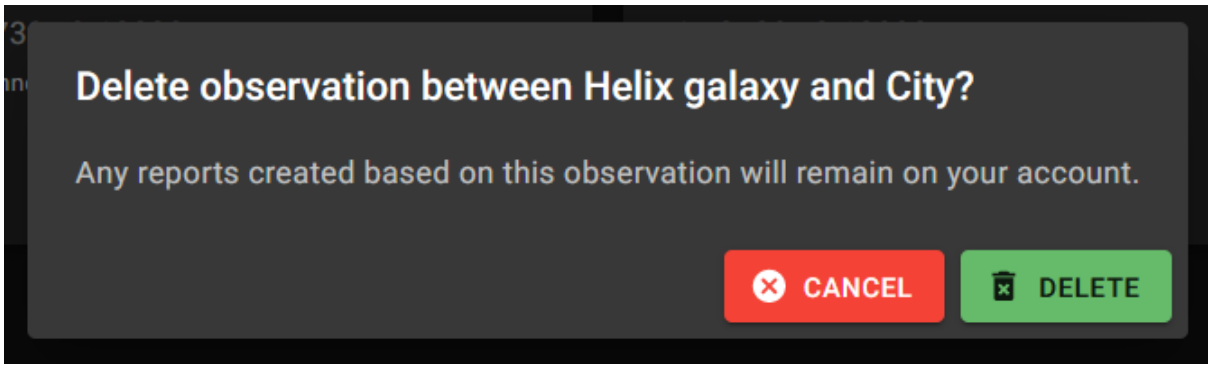
9.13.4.1 Observations tab



9.13.4.2 New observation dialog

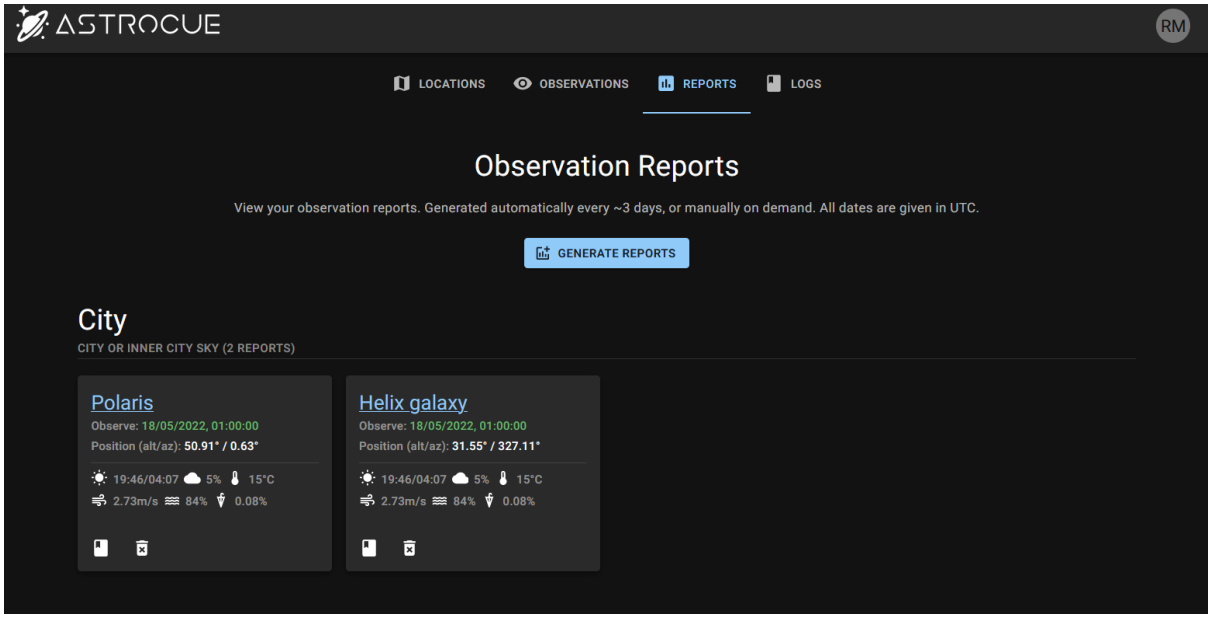


9.13.4.3 Delete observation dialog

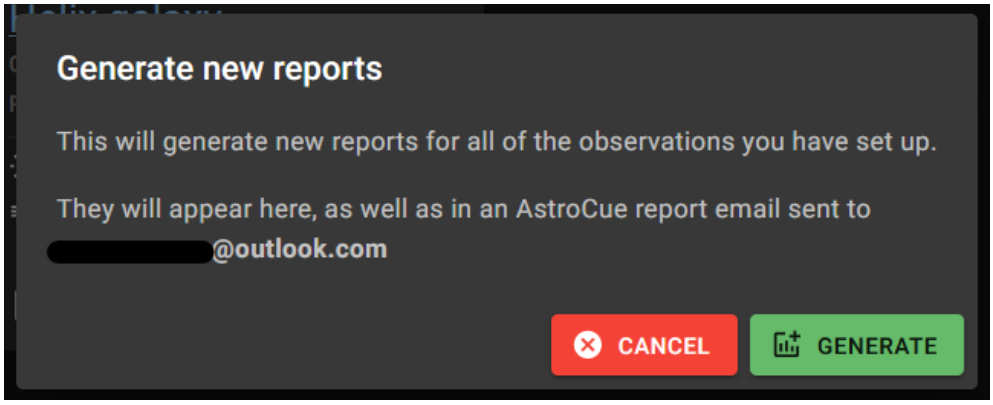


9.13.5 REPORTS

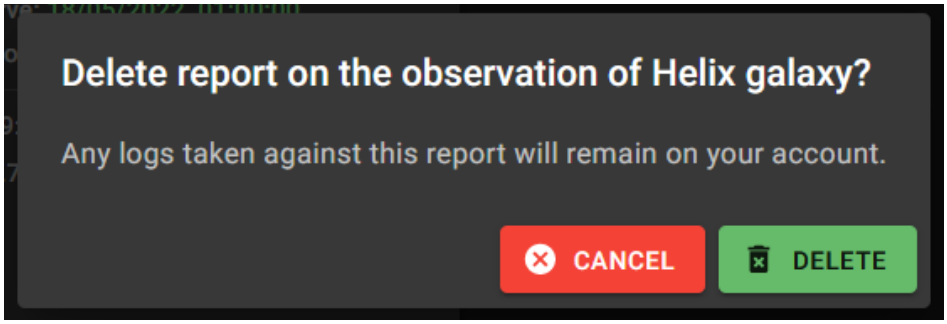
9.13.5.1 REPORTS TAB



9.13.5.2 GENERATE REPORTS DIALOG

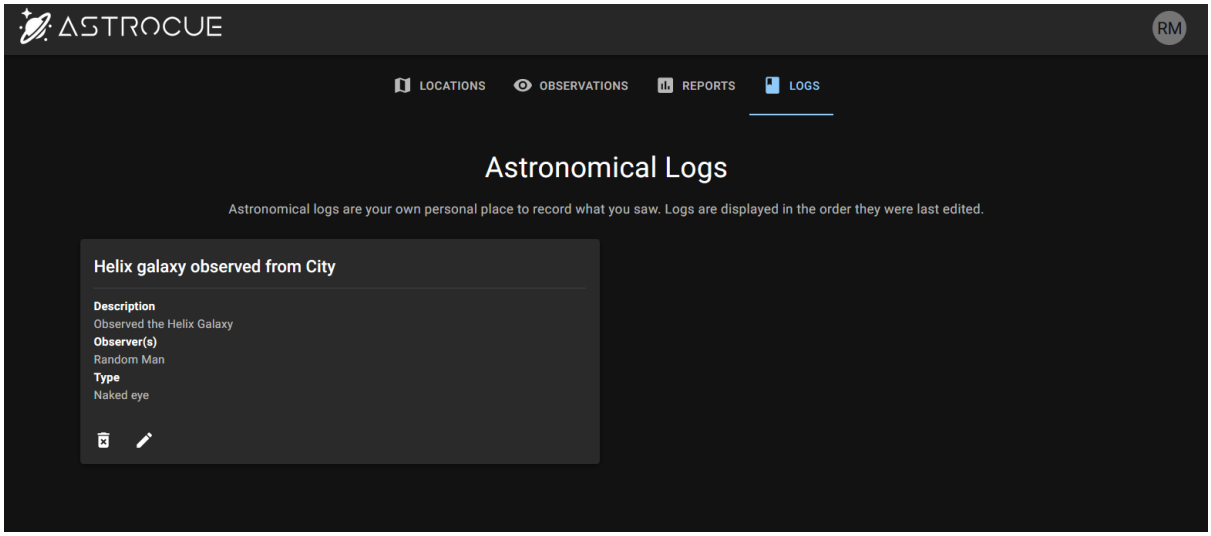


9.13.5.3 DELETE REPORTS DIALOG

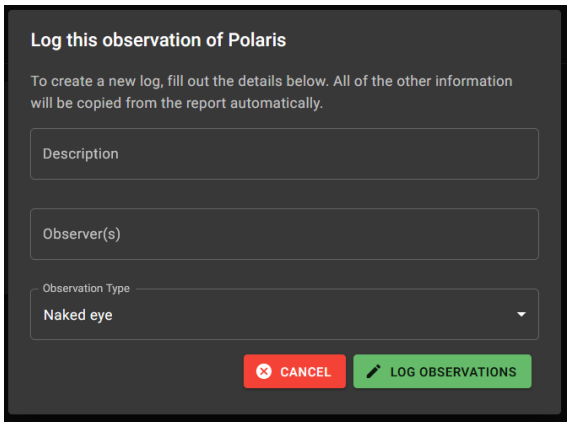


9.13.6 OBSERVATION LOGS

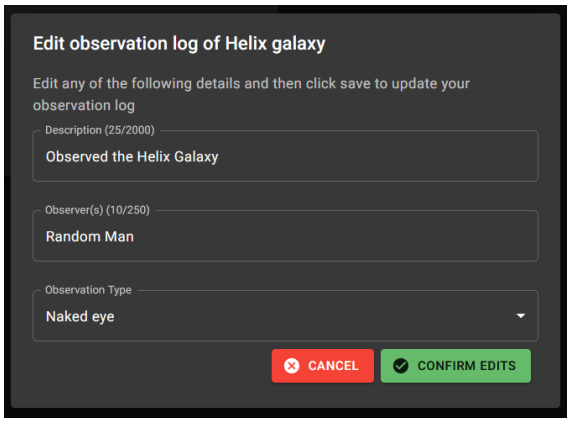
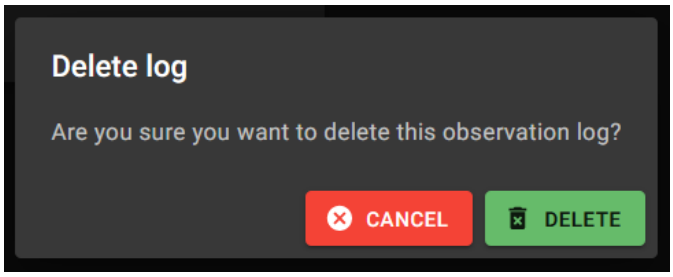
9.13.6.1 OBSERVATION LOGS TAB



9.13.6.2 NEW OBSERVATION LOG DIALOG



9.13.6.3 EDIT OBSERVATION LOG DIALOG



9.13.6.4 DELETE OBSERVATION LOG DIALOG