Early Earthquake and Tsunami Warning Viewer

AQA A-Level Computing NEA Report

Yicheng Shao (Eason)

Candidate Number: 7616

20th March 2025

Abstract

Give a brief summary outline of your project.

This work is formatted using XeLaTeX.
The references are managed using JabRef [37] and compiled using BibLaTeX.
The source code is available at EasonSYC/nea-report .
The relevant program source code is available at EasonSYC/EEETWV , licensed under the MIT Licence.

Contents

1 Analysis **9**
- 1.1 Background Information . . . 9
  - 1.1.1 The Early Earthquake Warning System . . . 9
  - 1.1.2 Earthquake Terminology . . . 9
- 1.2 Problem Area . . . 10
- 1.3 Client and End User . . . 10
- 1.4 Research Methodology . . . 11
  - 1.4.1 Client Interview . . . 11
  - 1.4.2 Existing Applications and Solutions . . . 12
- 1.5 Features of proposed solution . . . 17
- 1.6 Critical Path . . . 19
  - 1.6.1 Part 1. NIED and DM-D.S.S. API/WebSocket Connection . . . 19
  - 1.6.2 Part 2a. Real-time Monitoring Map . . . 19
  - 1.6.3 Part 2b. Past-earthquake Viewing . . . 20
  - 1.6.4 Part 3. Joint functionality . . . 20
  - 1.6.5 Part 4. Setting page for customisation . . . 20
  - 1.6.6 Part 5. Additional functionalities . . . 20
- 1.7 Requirements Specification . . . 21
- 1.8 Initial Modelling . . . 21
  - 1.8.1 Colour Scheme to Use . . . 23
  - 1.8.2 Details to Display . . . 23

2 Design **26**
- 2.1 Hierarchy Chart . . . 26
- 2.2 External Data Sources . . . 26
  - 2.2.1 NIED Data Source . . . 26
  - 2.2.2 DM-D.S.S. Data Source . . . 38
- 2.3 Algorithms . . . 62
  - 2.3.1 Wavefront Calculation . . . 62
  - 2.3.2 Shake Detection . . . 65
- 2.4 User Interface . . . 65
  - 2.4.1 Map . . . 67
  - 2.4.2 Real-Time Monitoring Screen . . . 67
  - 2.4.3 Past Earthquakes . . . 69
  - 2.4.4 Settings Page . . . 69
  - 2.4.5 Joint Functionalities and Technicalities . . . 70
- 2.5 OOP Model . . . 71
  - 2.5.1 Design Patterns . . . 71
  - 2.5.2 Abstractions (Interfaces), Options and Services (Classes) . . . 73
  - 2.5.3 Design of UI Classes with MVVM Pattern . . . 82
  - 2.5.4 Design of DTOs (Record Classes) . . . 95
- 2.6 Hardware and Software Requirements . . . 98

3 Technical Implementation **100**
3.1 Key Code Segments . . . 100
3.1.1 Data structures . . . 100
3.1.2 Modularity . . . 100
3.1.3 Defensive Programming/Robustness . . . 100

4 Testing **101**
4.1 Test Strategy . . . 101
4.2 Testing Video . . . 101
4.3 System Tests (against original requirements' specification) . . . 101

5 Evaluation **102**
5.1 Requirements Specification Evaluation . . . 102
5.2 Independent End-User Feedback . . . 102
5.3 Improvements . . . 102

A Code Listing **104**

B Use of AI **111**

C Bibliography **114**

List of Tables

1.1 Comparison of target users and clients . . . 10
1.2 Supported platforms of existing solutions . . . 12
1.3 Feature comparison in monitoring of existing solutions . . . 12
1.4 Feature comparison in configuration and customisability of existing solutions . . . 14
1.5 Measurement methods . . . 21
1.6 Requirements for Part 1 . . . 21
1.7 Requirements for Part 2(a) . . . 22
1.8 Requirements for Part 2(b) . . . 22
1.9 Requirements for Part 3 . . . 22
1.10 Requirements for Part 4 and 5 . . . 23
1.11 Colour Scheme for Application . . . 24

2.1 Data available in 'Kyoshin' monitor . . . 30
2.2 Sensors available in 'Kyoshin' monitor . . . 30
2.3 Initial values for f_H . . . 33
2.4 Initial values for f_S . . . 34
2.5 Initial values for f_V . . . 35
2.6 Error codes for authorisation code step in OAuth2 . . . 41
2.7 Error codes for access token step in OAuth2 . . . 42
2.8 Necessary access permissions for DM-D.S.S. . . . 45
2.9 Standard errors for API. . . . 46
2.10 Errors for `socket.start`. . . . 48
2.11 Errors for WebSocket connection. . . . 56
2.12 Telegrams used in the application . . . 59
2.13 Distance intervals in JMA 2001. . . . 63
2.14 Hardware Specification for App . . . 98

4.1 Table of Tests. . . . 101

5.1 Table of Evaluation. . . . 102
5.2 Table of Feedback. . . . 102

List of Figures

1.1 Feature introduction of JQuake . . . 13
1.2 Feature introduction of Quarog . . . 13
1.3 Customisable colour scheme of KEVI . . . 14
1.4 Customisability of Quarog . . . 14
1.5 Past earthquake pages for KEVI and SREV . . . 15
1.6 Display of observation stations and regions of KEVI . . . 15
1.7 Intensity tree on side panel of KEVI . . . 16
1.8 Sidebar options of JQuake . . . 16
1.9 Real-time and tsunami pages of KEVI . . . 17
1.10 Real-time and tsunami pages of SREV . . . 18
1.11 Main page for JQuake . . . 18
1.12 JMA Colour Scheme for Measured Intensities . . . 23
1.13 JMA Colour Scheme for Tsunamis . . . 24

2.1 Hierarchy chart of the whole application . . . 27
2.2 Hierarchy chart of the GUI Component . . . 28
2.3 A comparison of the K-NET, F-net and Hi-net. . . . 29
2.4 Distribution of the sensors of different nets. . . . 29
2.5 Sample GIF image achieved from 'Kyoshin' Monitor . . . 30
2.6 Scale colours of different measurements . . . 31
2.7 The hue scale in HSL/HSV encoding . . . 32
2.8 The values of (H, S, V) against pixel row r . . . 32
2.9 The values of (H, S, V) against normalised height h . . . 33
2.10 The fit result for $f_H : h \mapsto H$ . . . 34
2.11 The fit result for $f_S : h \mapsto S$ . . . 35
2.12 The fit result for $f_V : h \mapsto V$ . . . 36
2.13 Flow of data in NIED data sources . . . 38
2.14 Relation between abstract variables . . . 39
2.15 Colour generated using fitted functions . . . 39
2.16 Control panel for API Keys . . . 39
2.17 OAuth Procedure Outline . . . 40
2.18 Flowchart for OAuth 2.0 Authorisation Code and New Refresh Token . . . 43
2.19 Flowchart for OAuth 2.0 Acquire Access Token using Refresh Token . . . 44
2.20 Flowchart of WebSocket Wrapper . . . 58
2.21 Flowchart of dealing with WebSocket Response . . . 59
2.22 Shake detection flowchart and diagram . . . 66
2.23 Design of GUI for Real-Time Page . . . 68
2.24 Design of GUI for Past Earthquakes Page . . . 69
2.25 Design of GUI for Settings Page . . . 70
2.26 Class Diagram for JMA Time Table . . . 74
2.27 Class Diagram for Kmoni . . . 76
2.28 Class Diagram for Extra Services of Kmoni . . . 76
2.29 Class Diagram for DM-D.S.S. Authentication . . . 78
2.30 Class Diagram for DM-D.S.S. API Calls . . . 79
2.31 Class Diagram for DM-D.S.S Telegram Fetching and Parsing . . . 80
2.32 Class Diagram for DM-D.S.S WebSocket . . . 81

2.33 Class Diagram for DM-D.S.S WebSocket DTOs 82
2.34 Class Diagram for Logging Services 83
2.35 Class Diagram for Map Resources 84
2.36 Class Diagram for Time Abstraction 84
2.37 Class Diagram for Kmoni Settings Manager 85
2.38 Class Diagram for View Models 86
2.39 Class Diagram for Window 88
2.40 Class Diagram for Settings 89
2.41 Class Diagram for Past Earthquake 91
2.42 Class Diagram for Real-Time Page 93
2.43 Class Diagram for DTOs for API Calls 97
2.44 Class Diagram for DTOs for Telegrams 98

B.1 Prompt to ChatGPT in [59], Part 1 111
B.2 Prompt to ChatGPT in [59], Part 2 112
B.3 Prompt to ChatGPT in [59], Part 3 113

List of Listings

2.1 Response for OAuth Access Token Request . . . 42
2.2 Error for OAuth Access Token Request . . . 42
2.3 `ok` status for API . . . 45
2.4 `error` status for API . . . 45
2.5 Cursor token sample JSON. . . . 46
2.6 Contract list sample JSON. . . . 47
2.7 Socket start sample request JSON. . . . 48
2.8 Socket start sample response JSON. . . . 49
2.9 Socket list sample response JSON. . . . 51
2.10 Earthquake parameter sample response JSON. . . . 51
2.11 Past earthquake list sample response JSON. . . . 53
2.12 Past earthquake event sample response JSON. . . . 54
2.13 WebSocket Ping JSON. . . . 55
2.14 WebSocket Pong JSON. . . . 55
2.15 WebSocket Error JSON. . . . 56
2.16 WebSocket Data JSON. . . . 57
2.17 Head of JSON Schema . . . 60
2.18 Comments in JSON Schema . . . 61
2.19 JMA 2001 Wave Travel Tables . . . 62
A.1 Code for Polynomial Fit of Colour . . . 109
A.2 Code generated by ChatGPT in [59] . . . 110

List of Algorithms

1 Algorithm for f^{-1} 37
2 Algorithm for Finding Distance based on Time 64

Chapter 1

Analysis

Overview

This section introduces the background for the EEW system in place in Japan, consisting of EEW Forecasts and EEW Warnings, and together with the tsunami warnings. Concepts such as intensities, magnitude and epicentres are defined. Two key users and targets are identified, specifically passionate geologists and earthquake-sensitive industries, each having different requirements. A client in the former was interviewed, together with a thorough analysis and comparison of some existing solutions such as SREV, JQuake, KEVI and Quarog. A detailed requirement specification and the critical path is also outlined, consisting of DM-D.S.S. parsing, real-time monitoring, past-earthquake viewing and joint functionalities.

1.1 Background Information

1.1.1 The Early Earthquake Warning System

Earthquake is one of the most common natural disasters in the whole world, and direct consequences of earthquakes include tsunamis which could be catastrophic.

Japan, sitting on the intersection of the Eurasian, the Philippine and the North-American plates, is the countries with most earthquakes. Historically, the Great Kantō Earthquake (関東大震災) in 1923, the Great East Japan Earthquake (東日本大震災) in 2011 (a.k.a. the Tōhoku Earthquake) and the recent 2024 Noto Peninsula Earthquake (能登半島地震) all caused hundreds of deaths, both due to the result of the earthquake(s) and the resulting tsunami.

To provide protection to its residents, the Japan Meteorological Agency (JMA, 気象庁), together with the National Research Institute for Earth Science and Disaster Resilience (NIED, 防災科研) placed thousands of **earthquake sensors** across Japan (the Hi-net, the K-NET, the KiK-net and the F-NET), with several of them lying deep in the sea bed, measuring displacement, velocity and acceleration, which are connected to multiple servers, including two located in Ōsaka and Tōkyo.

Using data obtained from the sensors, computers do some complicated algorithms (mentioned below) to send out **early earthquake warnings (EEWs, 緊急地震速報)** automatically within milliseconds. There are two types of EEWs:

1. **EEW (Forecast, 予報).** Sent out to **highly-dependent industries** (e.g. rail industry, power plants) and **subscribed users**, when maximum intensity level of more than 3, or a magnitude of more than 3.5 is expected.
2. **EEW (Warning, 警報).** Sent out to **everyone** via TV, Radio, Mobile Phone, SMS, etc., when a maximum intensity level of more than 4 is expected.

After the earthquake, JMA staff will determine the location and severity of tsunami warnings to be issued, if necessary.

1.1.2 Earthquake Terminology

- **Intensity (震度).** The intensity describes the intensity vibration of a point due to an earthquake. It is not unique to an earthquake - **different places can have different intensities** due to the

distance to the epicentre, and intensity will also change over time. JMA measures intensity using **9 levels: 1, 2, 3, 4, 5–, 5+, 6–, 6+ and 7** in increasing order.

- **Magnitude/Scale.** The magnitude of an earthquake describes the energy released in the earthquake in a logarithmic scale. **It is unique to an earthquake.**
- **Epicentre/Hypocentre.** The epicentre is the surface point directly above the true centre of the earthquake.
- **Focal Depth.** The focal depth is the depth of the true centre of the earthquake.
- **P-Wave and S-Wave.** These are seismic waves, sourced from the true centre of the earthquake, travelling at different speeds, with Primary (P)-Wave travelling faster and Secondary (S)-Wave travelling slower.

1.2 Problem Area

The main goal of this application is to provide a visualisation of the earthquake/tsunami related data feed(s) provided by JMA's affiliated institution, Disaster Mitigation Data Send Service (DM-D.S.S), and some other data feeds as well. There are numerous apps providing a list of recent earthquakes, the real-time data measured by the sensors, and the real time earthquake warning displayed on a map, but rarely are there good apps that combine all those features together satisfyingly, with just the necessary features the author needs.

Some applications are no longer being updated due to change in the user's policy of the related data feed. Furthermore, most of the apps available are only in Japanese, not in English or my home language Chinese, which can create trouble for the author to understand.

1.3 Client and End User

The primary target of this application will be passionate geographers and geologists who are interested in the study of earthquake observations and predictions. The age group of this vary all the way from primary-school students to adults, including the author who has been amazed by the technology since the age of 12. They could take any employment, ranging from students to full-time jobs. Their proficiency usually varies, since there are people new to this field who probably does not have much knowledge, so the interface of the application should be relatively user-friendly and understandable, hiding unnecessary technical complexities.

Another target client could be industries which highly rely on earthquake predictions due to the risk imposed by earthquakes. High-speed railway and nuclear power plants are good examples of this. Therefore, the staff in charge monitoring will usually have higher proficiency and would like more detailed data of the earthquake. However, they will only need the necessary data from earthquakes happening close to them and only require intensity data of the point in interest (e.g. the power plant). To put this into content, an earthquake happening 1000 km away from them does not need to be fed into their system, while they would like to see the intensity of the shock and the arrival time of the seismic waves to decide the actions. In fact, the author really likes investigating on the rail industry, whose infrastructure could be greatly affected by earthquakes.

Table 1.1 compares relative features of these two target users/clients.

Feature	Primary Target	Secondary Target
Description	Passionate Geologists	Earthquake-Sensitive Industries
Age	Varies (Middle School – Adults)	Work Age
Reason	Monitor Live & Latest Earthquakes	Monitor Risks to Infrastructure
Proficiency	Varies (Beginner – Amateur)	Trained Professional
General Requirements	Monitor Overall Movement	Alert intensity and arrival time at certain location

Table 1.1: Comparison of target users and clients

1.4 Research Methodology

1.4.1 Client Interview

The author interviewed my friend Wesley Ma, who is a passionate geologist on earthquake studies and also monitor earthquakes regularly.

1. *Which earthquake monitoring apps do you use?*

   Response: JQuake, SREV, Quarog, KEVI, Kyoshin-Monitor (Support discontinued), Kiwi Monitor (Support discontinued)

2. *Do you subscribe/pay to services such as DM-D.S.S. to use earthquake monitoring apps, and do you think it is worth the price?*

   Response: Yes. However, DM-D.S.S. is a little expensive. However, the price becomes more affordable considering the information provided by the subscription.

3. *Do you watch YouTube livestreams on earthquake monitoring?*

   Response: I do not usually watch the live streams, as almost no one who has already has a monitoring app will use the stream. They provide mostly the same information as the applications, just real-time streaming the windows.

4. *Why do you use earthquake monitoring apps?*

   Response: To monitor the earthquake. This is derived from my interest in broadcasting culture in Japan. This eventually led me to be intrigued with the development of earthquake monitoring technologies and theories in Japan.

5. *How often do you use earthquake monitoring apps (e.g. all the time/after school/only after big earthquakes)?*

   Response: After big earthquakes. But I usually open one or two apps for all-day monitoring to catch potential major (or medium) earthquakes.

6. *Describe the advantages and disadvantages of each of them, mentioning the specific features.*

   Response:

   - SREV is a good one, but it is only available in a browser with no app.
   - Kyoshin-Monitor and Kiwi Monitor are relatively more stable, but the source is not the same as that from the former two apps, and its support is also discontinued.
   - Quarog has weak response time and interacting interface.
   - JQuake is the most developed app, which includes nearly all functions that can be thought of. However sometimes the connection of WebSocket is unstable.
   - KEVI does not have the sound files configured by default. Some information are not displayed clearly enough.

7. *What features do you use the most/least?*

   Response: Basic earthquake notifications, and should be including sufficient and prompt information.

8. *What features are redundant in the earthquake monitoring apps you use?*

   Response: KEVI has a weather monitoring function, which could be redundant. But that could be caused by the different purposes of the app. Hence, no further comments. It is not a bad thing.

9. *What are the critical features of an earthquake monitoring app?*

   Response: To provide accurate, prompt and detailed information according to the source released by the JMA, the information display interface should be easy to read and understand. This is not only helping the people who like to monitor, but more importantly, provides the easiest way to the people who really need to seek information to minimise the harm brought by earthquakes and successive disaster.

10. *What additional features would you like to have in those existing apps?*

    Response: Summarise all the useful features from different apps into one app. Stable connection. Nothing else.

1.4.2 Existing Applications and Solutions

Based on the applications the author uses and the feedback from interviewee, there are the following commonly-used applications:

- JQuake
- Scratch Real-time Earthquake Viewer (SREV), available in compiled form at kotoho7/scratch-realtime-earthquake-viewer-page
- Kyoshin EEW Viewer for Ingen (KEVI), available at ingen084/KyoshinEewViewerIngen
- Quarog

1.4.2.1 Supported Platforms

Supported platforms of those apps are listed in Table 1.2. In particular, note that SREV is a web-based and GitHub Pages-hosted application therefore supporting all platforms. KEVI is written in .NET Framework and supports the second most platforms, with JQuake not supporting Linux and Quarog only supporting Windows.

Platform	JQuake	SREV	KEVI	Quarog
Windows	✓	✓	✓	✓
macOS	✓	✓	✓	
Linux	✓	✓	✓	
Android/iOS		✓		

Table 1.2: Supported platforms of existing solutions

1.4.2.2 Earthquake Monitoring

An overview feature table of monitoring is included in Table 1.3. In particular, note that due to the nature of SREV being Scratch-programmed and web-based, it reached a special agreement with DM-D.S.S. to use the API without the need of all users paying for this, since it is hard to integrate such function into a web application.

Quarog is a relatively new application and only supports basic functionalities of EEW Viewing and past earthquake listing, as shown in Figure 1.2. JQuake is solely dedicated to earthquake and tsunami monitoring as shown in Figure 1.1, while KEVI has features like rain clouds map and natural disaster warning which is beyond the scope of this analysis (which is a burden to users only requiring earthquake monitoring and a waste of storage space).

It is worth noting that for SREV, DM-D.S.S. is integrated with special permission with no login required. For JQuake, although it has tsunami warnings/special warnings, the tsunami forecast is not available, since they use a different data source.

Feature	JQuake	SREV	KEVI	Quarog
NIED Real-time Shake Support	✓	✓	✓	
DM-D.S.S. WebSocket Support	✓	✓	✓	✓
Real-time Sensor Data	✓	✓	✓	
Vibration Alert	✓	✓	✓	
Past Earthquake List	✓	✓	✓	✓
Past Earthquake Details		✓	✓	✓
Tsunami Warning	✓	✓	✓	
Real-time EEW	✓	✓	✓	✓
Calculated Seismic Wavefronts	✓	✓	✓	✓
User-Defined Key Monitor Point	✓		✓	
Sub-Map for the Okinawa Area	✓		✓	
Replay	✓		✓	

Table 1.3: Feature comparison in monitoring of existing solutions

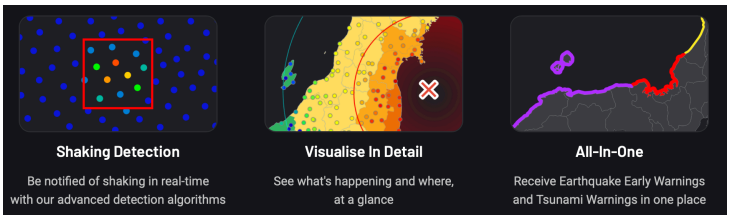


Figure 1.1: Feature introduction of JQuake, screenshot from website

Figure 1.2: Feature introduction of Quarog, screenshot from website
Top-left: Past earthquake information; Top-right: Real-time EEW;
Bottom-Left: Past earthquake list; Bottom-Right: Details of EEW

1.4.2.3 Configuration Options

There are also a variety of configuration options available for all apps, as listed in Table 1.4. Both KEVI and Quarog supports the adjustment of the colour theme, and Quarog even supports changing the style of how blocks are displayed and coloured as shown in Figure 1.3 and 1.4. Playing a sound on the speaker is also common among the apps to remind the user of earthquakes.

Feature	JQuake	SREV	KEVI	Quarog
DM-D.S.S. Login	✓		✓	✓
Sound Alert	✓	✓	✓	✓
System Notification			✓	
Colour Theme			✓	✓
Map Colouring Style		✓		✓

Table 1.4: Feature comparison in configuration and customisability of existing solutions

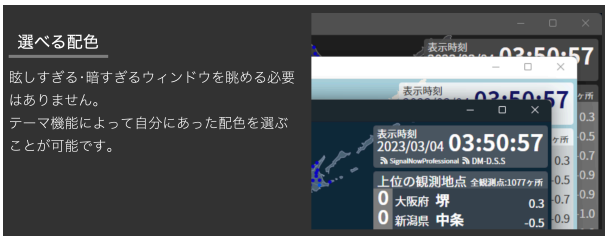


Figure 1.3: Customisable colour scheme of KEVI, screenshot from website

Figure 1.4: Customisability of Quarog, screenshot from website

1.4.2.4 GUI Analysis

All applications are similar in a way, that their past earthquakes list is displayed on a sidebar.

For Quarog and KEVI, an earthquake can be selected from the sidebar to view the details of, as in Figure 1.5. They will both colour on the map the maximum observed intensities by regions. While KEVI chooses to display the regions as colours and also provide an overlay of the observed intensities by observation stations (which will only appear after zoomed in enough, as in Figure 1.6), Quarog chooses to have a separate button to switch between display of regions, and the display of observation stations.

In particular for KEVI, there is also a tree structure on a panel displaying the details of the observed intensities by hierarchy structure, consisting of prefectures, skipping regions, then districts, then the exact observation stations, as shown in Figure 1.7.

As for SREV, the selected earthquake will be displayed on the map and coloured by region, but no information on observation stations will be provided. For JQuake, it is impossible to select a particular earthquake, but only possible to replay the past earthquake, and jump to an external weather service to view the details, as shown in Figure 1.8.

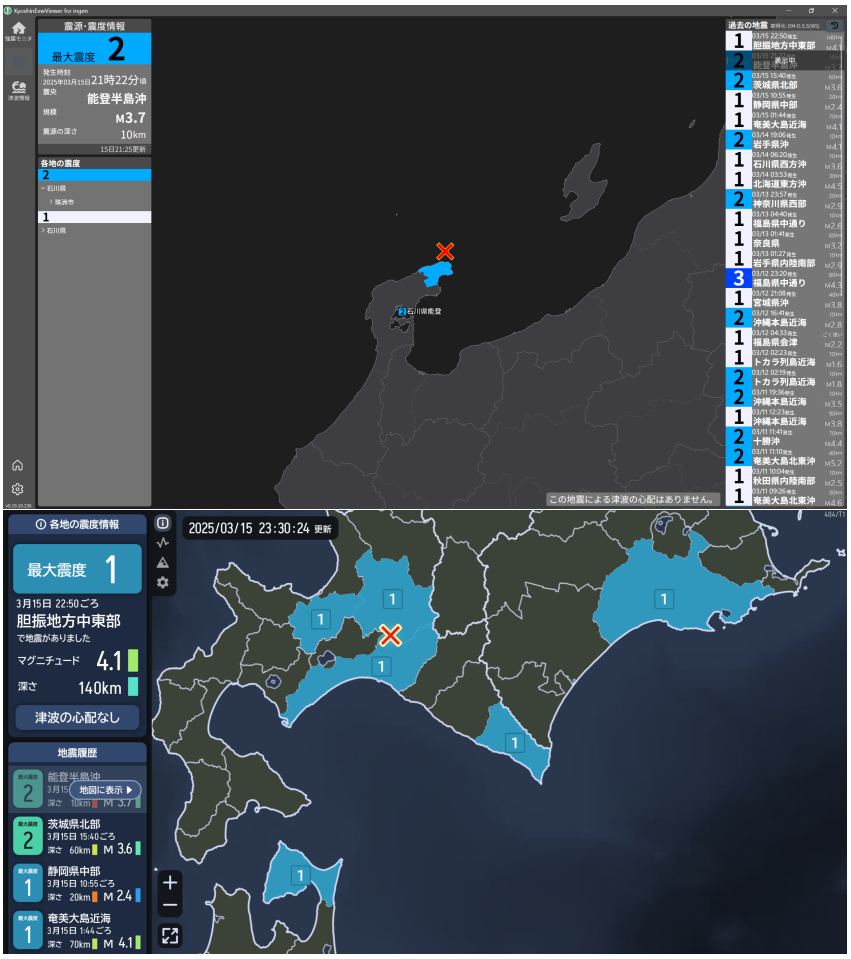


Figure 1.5: Past earthquake pages
Top: KEVI; Bottom: SREV

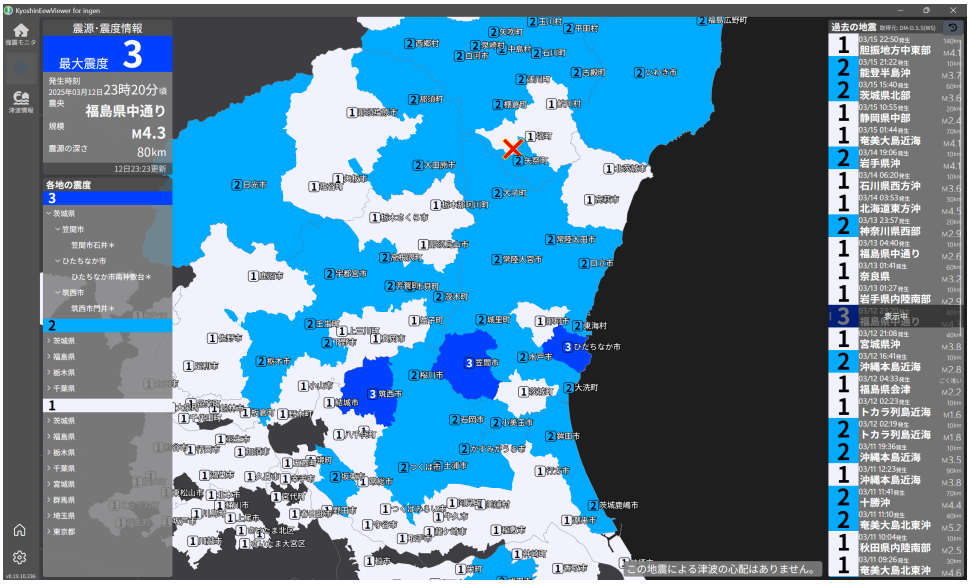


Figure 1.6: Display of observation stations and regions of KEVI

Figure 1.7: Intensity tree on side panel of KEVI

Figure 1.8: Sidebar options of JQuake
Globe: Link to Yahoo! weather service page for earthquake;
Time-lapse: Replay of earthquake

A significant difference between the applications is how they split the different functionalities. For KEVI and SREV, there are separate pages for real-time monitoring, past-earthquake viewing and tsunami warnings, as shown in Figure 1.9 for KEVI, and Figure 1.10 for SREV.

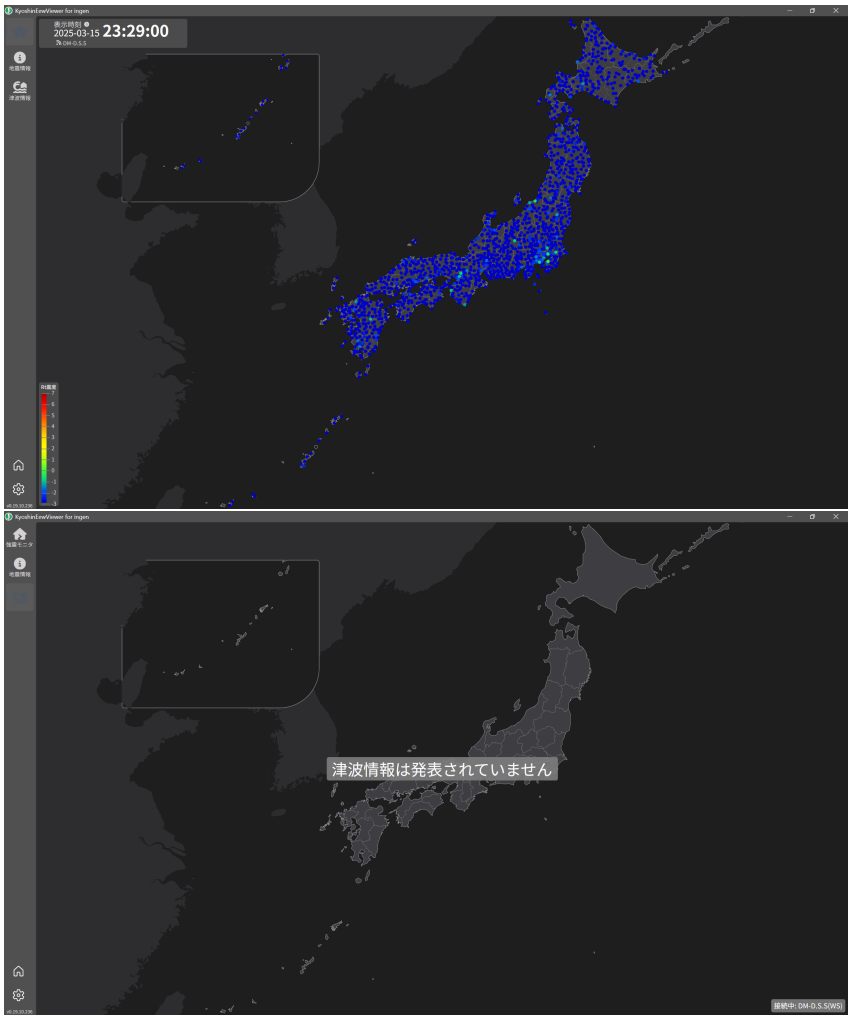


Figure 1.9: Real-time and tsunami pages of KEVI

On the contrary, since JQuake's map only supports replay and real-time functionality (EEW and tsunami), there is only one page, as shown in Figure 1.11. Quarog will automatically switch the map to display EEWs if EEWs are issued, and change the sidebar to display the predicted intensities of EEWs, as shown in Figure 1.2.

1.5 Features of proposed solution

Based on the potential user/client interview results, and the research into the existing solutions, the following key features should appear in the solution, since they are essential to earthquake monitoring applications:

1. DM-D.S.S. Login functionality (for the data source)
2. Real-Time Sensor Shake Intensity Data
3. EEW Visualisation (w/ Calculated Seismic Wavefronts)
4. Past Earthquake List (w/ Option to review details)
5. Tsunami Warning Visualisation

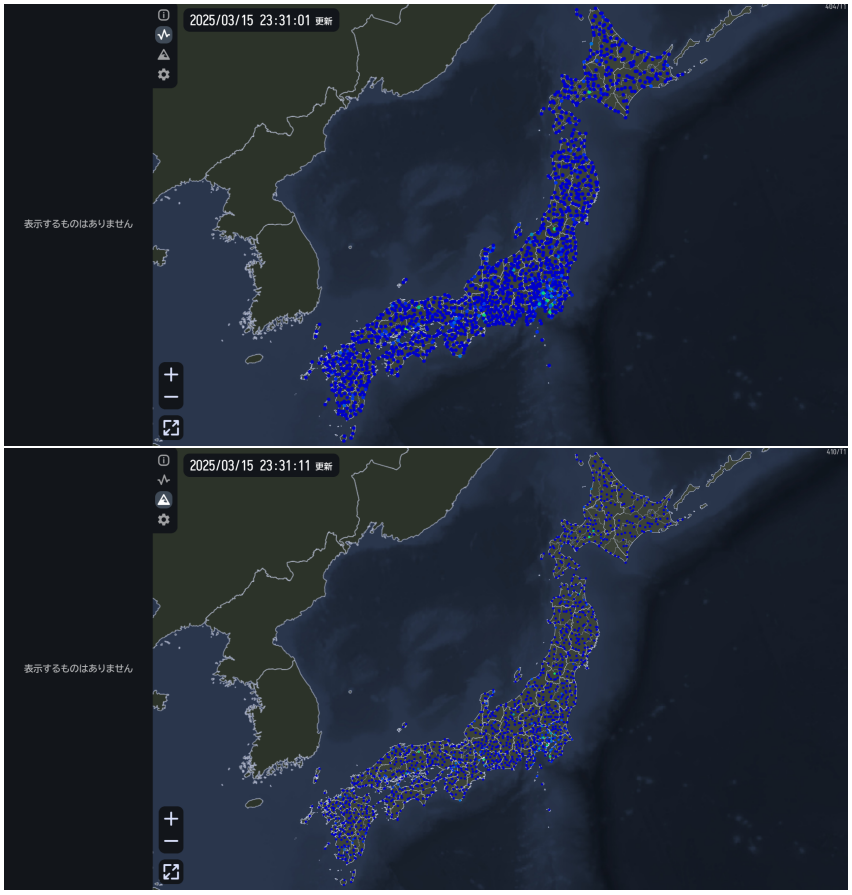


Figure 1.10: Real-time and tsunami pages of SREV

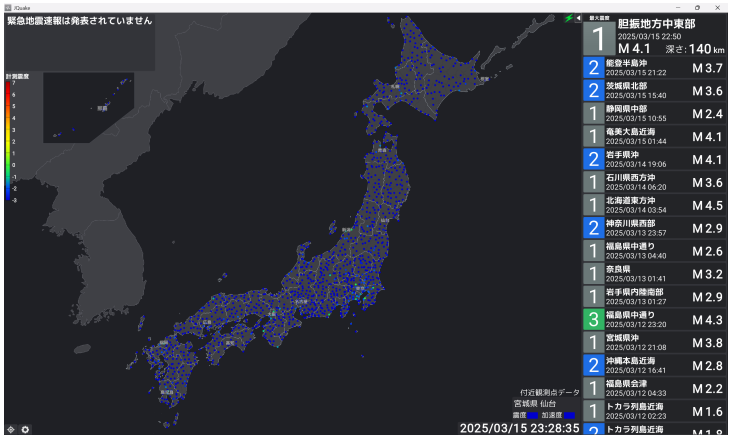


Figure 1.11: Main page for JQuake

However, due to the limitation of time, the following functionalities are excluded from the scope of this project, which include mostly the customisation parts, ranked in decreasing importance:

1. Shake detection algorithm (based on real-time sensor data)
2. User-Defined Key Monitor Point
3. (Customisable) Sound Alert
4. Customisable Colour Theme

The following features are considered too complicated for the scope for this project and the time constraints, hence not considered:

1. Replay (due to a server needing to store past data)
2. Sub-Map for the Okinawa Area (due to the difficulty in implementation)

With those key features implemented, the application should mirror all essential functionalities of an earthquake monitoring application, as compared to the four existing solutions above.

1.6 Critical Path

The functionality of this application can essentially be divided into $1+2+1+a+b$ steps, where the 1 is to receive information from the API/WebSocket provided by DM-D.S.S. and NIED, and 2 is to, briefly saying:

1. Real-time Monitoring: Produce a real-time map to monitor real time vibration and plot real-time EEW/tsunami warnings
2. Past-earthquake Viewing: Produce a menu to select an earthquake to display information on the map.

The next 1 is to merge these two functions, specifically the functionality to switch back to the real-time monitoring option immediately when a new EEW is released (to make sure the user does not miss any information on real-time EEWs while looking at past earthquake information).

The a is to implement a setting page for customisation and some additional features, and the b is to implement additional functionalities that are excluded from this project, but could be completed if there is time.

1.6.1 Part 1. NIED and DM-D.S.S. API/WebSocket Connection

1. Investigate into the list of APIs/WebSockets that should be used by the application.
2. Investigate into patterns of the response data, and design the classes/DTOs using correct OOP patterns such as inheritance and composition.
3. Implement classes/DTOs to convert the information into C# objects.
4. Implement classes/DTOs to convert telegrams into C# objects.
5. Use API Key to test the functionality of this system.
6. Implement OAuth 2 login functionality to reduce complexity and increase security.

1.6.2 Part 2a. Real-time Monitoring Map

1. Display a map in the application.
2. Extract the data from the real-time measured intensities of observation stations from the data source.
3. Display the data on the map using a default colour scheme at the correct positions.
4. Colour the monitoring points using a colour scheme on the map.

5. Investigate into and implement an algorithm to calculate seismic wave fronts, using necessary data provided by the JMA.
6. Plot and colour real time EEWs, considering special cases such as:
    - Cancellation of EEW;
    - Upgrade from EEW Forecast to EEW Warning;
    - Multiple Earthquakes (hence displaying multiple epicentres, and switching between earthquakes in details display);
    - Assumed Hypocentre (hence using a different hypocentre symbol to display).
7. Colour areas on the map with the maximum expected intensity.
8. Plot on the map the hypocentre of the earthquake, using the correct symbol.
9. Plot and colour real time tsunami warnings.

1.6.3 Part 2b. Past-earthquake Viewing

1. Plot a sidebar of a list to display path earthquakes.
2. Provide a button to refresh the sidebar, and another one to load more earthquakes.
3. Provide a functionality to choose an earthquake from the sidebar to view the detailed information on the earthquake (e.g. magnitude, epicentre, time, additional message).
4. Plot the hypocentre of the earthquake.
5. Plot the observed maximum intensities on the map by regions.
6. Plot the intensities measured by each observation station on the map.
7. Provide external link to view earthquake in the weather service provider website.

1.6.4 Part 3. Joint functionality

1. Provide sidebar to switch between real-time monitoring and past-earthquake viewing, and settings options.
2. Implement automatic functionality to switch back to real-time earthquake when an EEW/Tsunami Warning is being published.

1.6.5 Part 4. Setting page for customisation

1. Provide a text box to input the API Key, and a button to confirm the API Key.
2. Provide a button to connect and authorise with OAuth 2.0.
3. Provide a button to connect to the WebSocket.
4. Display a list of current WebSocket connections, and provide a button to refresh such list.
5. Provide two dropdown menus, to choose the sensors to display on the real-time map (all sensors, or borehole only), and the data to display (real time intensity, PGA, PGV, PGD, etc.).

1.6.6 Part 5. Additional functionalities

1. Investigate into and implement an algorithm to detect shakes based on the observed data.
2. Provide functionality to define a specific point on the map, and display the measured intensity at that point.
3. Play sound alerts when EEW/Tsunami warnings are received.
4. Provide custom sound alert settings to specify the file to the sound alert.
5. Provide custom colour theme settings to specify the colour scheme of the colouring on the map.

1.7 Requirements Specification

A detailed specification is defined here that is to be aimed to be fulfilled at the end of the project, split into four sections:

1. **NIED and DM-D.S.S. Functionality** – Corresponding to **Part 1**, in Table 1.6
2. **Real-Time Earthquake Monitoring** – Corresponding to **Part 2a**, in Table 1.7
3. **Past-earthquake Viewing** – Corresponding to **Part 2b**, in Table 1.8
4. **Joint functionalities** – Corresponding to **Part 3**, in Table 1.9
5. **Settings Page** – Corresponding to **Part 4**, in Table 1.10
6. **Additional functionalities** – Corresponding to **Part 5**, also in Table 1.10, marked with a *

Those objectives marked with an * are optional depending on time, since they are not core functionalities for the project.

The objectives here are SMART, meaning they are specific, measurable, achievable, realistic and timely.

Table 1.5 gives abbreviations for the types of testing.

Test Method	Abbr.
Unit Testing	UT
Integrated Testing	IT

Table 1.5: Measurement methods

Req. №	Description	Success Criteria	Test
1(i)	Connect to DM-D.S.S. using an API Key	Included in header for requests	UT
1(ii)	Call HTTP-Based APIs	Calls to correct URLs with specified parameters where applicable	UT
1(iii)	Connect to WebSocket Data Feed	Successful connection with regular pings/pongs	IT
1(iv)	Obtain stable connection on WebSocket	Connected for 1 hour under reliable internet connection	IT
1(v)	Parse API calls JSON into C# objects	Information parsed with correct values	UT
1(vi)	Recognise schema for JSON Telegrams	Correct JSON schema recognised	UT
1(vii)	Parse JSON Telegrams into C# objects	Parsed into correct type with correct values	UT
1(viii)	Exception handling for incorrect JSON	Exceptions thrown	UT
1(ix)	Exception handling for failed connections	Exceptions thrown	UT
1(x)	Integrated DM-D.S.S. functionalities	Correct objects created	IT
1(xi)	Connect to DM-D.S.S. using OAuth2	Access token included in header for requests	IT
1(xii)	Refresh tokens when expired using OAuth2	New access token successfully obtained with the refresh token	IT

Table 1.6: Requirements for Part 1

1.8 Initial Modelling

Some initial ideas on the applications are considered in this section, but more detailed design will be reflected in the GUI design section in the next chapter.

Req. №	Description	Success Criteria	Test
2a(i)	Display real-time observed data on map	Coloured points at correct locations	IT
2a(ii)	Display current time at the corner	Time displayed and refresh every second	IT
2a(iii)	Display details of current EEW	Details displaying correctly	IT
2a(iv)	Plot epicentre of EEW on map	Epicentre at correct position with correct symbol	IT
2a(v)	Calculate seismic wavefronts position	Correct result from dataset	UT
2a(vi)	Plot seismic wavefronts	Correct wavefronts at correct positions	IT
2a(vii)	Colour expected max. intensity of regions	Correctly coloured	IT
2a(viii)	Update/Cancel EEW when appropriate	Update reflecting on both the details and the map	IT
2a(ix)	Handle multiple simultaneous EEWs	Switch between details of EEWs at regular intervals	IT
2a(x)	Display tsunami warnings when issued	Details displaying correctly	IT
2a(xi)	Colour shorelines from tsunami warnings	Coloured at correct location	IT

Table 1.7: Requirements for Part 2(a)

Req. №	Description	Success Criteria	Test
2b(i)	Display past-earthquake side list using data from DM-D.S.S.	Displayed and updated	IT
2b(ii)	Refresh button to refresh the latest earthquakes	Loaded and displayed	IT
2b(iii)	Load more button to load extra earthquakes	Loaded and displayed	IT
2b(iv)	Display the time, location, depth, intensity and magnitude of earthquake on the side list	Correctly formatted and displayed	IT
2b(v)	Select an earthquake item from list	Selection functioning	UT
2b(vi)	Display details of selected earthquake on sidebar	Correctly formatted and displayed	IT
2b(vii)	Colour map by the maximum intensity of regions of selected earthquake	Correctly coloured	IT
2b(viii)	Plot hypocentre of selected earthquake	Correctly plotted	IT
2b(ix)	Plot observed maximum intensities of observation stations of selected earthquake	Correctly plotted and coloured	IT
2b(x)	Provide link to external weather services for details	Linked to correct website and earthquake	IT

Table 1.8: Requirements for Part 2(b)

Req. №	Description	Success Criteria	Test
3(i)	Provide sidebar to switch between real-time, past earthquake and settings	Sidebar functioning and able to switch pages	IT
3(ii)	Sidebar has an expansion/hide button to show/hide names	Button working as described	IT
3(iii)	Switch to real-time monitoring map when EEWs and tsunami warnings are issued	Page switched	IT

Table 1.9: Requirements for Part 3

Req. №	Description	Success Criteria	Test
4(i)	Provide with functionality to set API Key or connect to OAuth2 to authenticate	Authentication functioning	IT
4(ii)	Provide button to connect and disconnect from WebSocket	Button functioning	IT
4(iii)	Provide button to view a list of WebSocket connections	List displayed	IT
4(iv)	Provide button in the list to disconnect a particular WebSocket	Button functioning to disconnect	IT
4(v)	Provide dropdown menus to select sensors and data to display on the map	Correct data displayed on the real-time map	IT
4(vi)	Provide an easy-to-use GUI	Potential user gives rating of more than 7 out of 10	
4(vii)*	Provide page to design colour scheme	Functional design page and applied to the whole UI	UT
4(viii)*	Play sound when events happen and provide option to customise sound files	Correct sound played when required	IT
4(ix)*	Provide option to define a point on the map to monitor	Functioning as specified in Part 2a	IT
4(x)*	Display real-time shake data at a user-defined point	Display the name of the point with acceleration and intensity	IT
4(xi)*	Provide sub-map to display Okinawa Area	Map functioning as specified in Part 2	IT

Table 1.10: Requirements for Part 4 and 5

1.8.1 Colour Scheme to Use

Although most built-in applications tend to introduce their own colour schemes, the author decided to follow the standard guidance provided by the JMA [73] and use the official colour scheme, to prevent any confusion and make the application generally accessible.

Figure 1.12 shows the JMA Colour Scheme for measured intensities, and Figure 1.13 shows the JMA Colour Scheme for different levels of tsunami warnings (the row for the tsunami is highlighted).

表２－２　数値が高くなるほど危険度が増す情報の配色（震度）

使用する基本配色 (RGB 値)	180 ,0 ,104	165,0,33	255,40,0	255,153,0	255,230,0	250,230,150	0,65,255	0,170,255	242,242,255
震度	震度7	震度6強	震度6弱	震度5強	震度5弱	震度4	震度3	震度2	震度1

Figure 1.12: JMA Colour Scheme for Measured Intensities

In addition to these colours, for tsunami informational advisory, the colour `80FFFF` is used, in accordance with the colour used for no release of river flooding warning, and for endings of tsunamis warnings, the colour `F2F2FF` is used, in accordance with the minimal level (the extremely light grey) in Figure 1.13.

The colour `F0F0F0` is used for intensity zero (i.e. any intensity less than intensity 1), and `C8C8CB` is used for unknown intensities.

A similar colour scheme is used for EEW warnings. Specifically, the same colour as tsunami warning is used for EEW warnings, and the same colour as endings of tsunami warnings is used for EEW cancellation. For EEW forecasts, the colour `FFAA00` (in accordance with the orange colour in Figure 1.13) is used, and for EEW final reports, the same colour is used as unknown intensities.

A summary of the colours used is in the table below.

1.8.2 Details to Display

Comparing the information provided by the different applications, the following information about an earthquake will be displayed in the sidebar, as consistent with most applications analysed:

- Maximum observed intensity of earthquake;

表１－２　警報・注意報及びこれに類する情報の配色（台風・津波・噴火等の情報）

	← 警戒度 大			警戒度 小 →		発表なし
使用する基本配色 （RGB 値）	200,0,255	255,40,0	255,170,0	250,245,0	242,242,255	200,200,203
台風情報（経路図）		暴風域		強風域		
台風情報（暴風域に入る確率） 時間変化	70～100%	30～70%		5～30%		
台風情報（暴風域に入る確率） 分布表示				発表中		発表なし
海上警報	台風警報	暴風警報	強風警報	風警報		
大津波警報・津波警報・津波注意報	大津波警報 （幅2倍）	津波警報		津波注意報		
噴火警報・予報 （噴火警戒レベル導入火山）	レベル 5 避難	レベル 4 避難準備	レベル 3 入山規制	レベル 2 火口周辺規制	レベル 1 活火山であることに留意	
噴火警報・予報 （噴火警戒レベル未導入火山）	居住地域 厳重警戒		入山危険	火口周辺危険	活火山であることに留意	
噴火警報・予報 （海底火山）			周辺海域警戒		活火山であることに留意	
レーダー・ナウキャスト（雷）	活動度4	活動度3	活動度2	活動度1		
レーダー・ナウキャスト（竜巻）		発生確度2		発生確度1		

Figure 1.13: JMA Colour Scheme for Tsunamis
Purple: Special Warning; Red: Tsunami Warning; Yellow: Tsunami Caution

Intensity	Colour
?	C8C8CB
0	F0F0F0
1	F2F2FF
2	00AAFF
3	0041FF
4	FAE696
5-	FFE600
5+	FF9900
6-	FF2800
6+	A50021
7	B40068

EEW Level	Colour
Forecast	FFAA00
Warning	FF2800
Final	C8C8CB
Cancelled	F2F2FF

Tsunami Warning Level	Colour
Information	80FFFF
Caution	FAF500
Warning	FF2800
Special Warning	C800FF
None	F2F2FF

Table 1.11: Colour Scheme for Application

- Time of the earthquake;
- Name of position of hypocentre;
- Depth of hypocentre;
- Magnitude of the earthquake.

In addition to these, on the side panel, the following information will be displayed in addition to the previous ones:

- Last updated time of information;
- The tree of intensities measured by observation stations, in hierarchy structure of:
    1. Intensity; then by
    2. Prefecture; then by
    3. Regions; then by
    4. Cities/Districts; finally
    5. Observation Stations.
- Any additional text provided by the JMA on the report, where applicable.

Note there is an extra level of regions in the hierarchy tree, compared to KEVI.

As for the EEW warning, there are slightly different information displayed, due to different information provided, and the nature of the EEW. The following information will be displayed:

- The level of the EEW warning;
- The serial number of the EEW warning,
- Maximum predicted intensity of earthquake;
- Origin time of the earthquake;
- Name of position of hypocentre;
- Warning text if the hypocentre is assumed (as a JMA requirement);
- Depth of hypocentre;
- Magnitude of the earthquake;
- Precision/Accuracy of information on hypocentre, depth and magnitude;
- Warning text if the earthquake has low accuracy (as a JMA requirement);
- Last updated time of the information,
- Any additional text provided by the JMA on the report, where applicable.

Note that (as a JMA requirement), if the hypocentre is assumed for algorithmic reasons, then the hypocentre details and the magnitude have no seismological meaning, and the information on origin time, the position of hypocentre, the depth of hypocentre and the magnitude of the earthquake will be hidden

The information displayed for a tsunami warning will be relatively simple:

- The level of the tsunami warning;
- Any additional text provided by the JMA on the report, where applicable.

Chapter 2

Design

Overview

This section includes a breakdown of the application into sections of data processing, GUI functionalities and joint functionalities. The use of external sources including DM-D.S.S. and NIED data sources is discussed, together with the relevant formats (JSON) and the objects related. UML class diagrams are included to discuss OOP relations of classes, records (record classes) and enums including the use of inheritance, composition, association and aggregation. They also implement different interfaces. Design patterns that should be included in the application are also introduced and outlined. An outline of the design of the user interface is included. The expected hardware requirements of the systems are also listed, but any laptop with an up-to-date operating system (running Windows or macOS) should be able to run the program.

2.1 Hierarchy Chart

As discussed in the analysis section, the program consists of three parts: data-parsing from external data sources, GUI functionalities and joint functionalities, where the GUI part will be divided into two parts focusing on real-time monitoring and past-earthquake information, respectively. Here, functionalities for each particular module is further split up. Figure 2.1 is a hierarchy diagram for the whole application. This shows how **decomposition** technique is applied to reduce a sophisticated problem into more attackable problems.

Since the GUI part is quite complicated, it is included in this separate diagram, in Figure 2.2.

The whole application (apart from the polynomial fitting part which was done in Python) will be written in C# using the .NET Core Version 9.0, since the .NET Core is designed oriented around OOP techniques, and some JSON properties (e.g. the `JsonStringEnumMemberName` to customise the parsing of enums) will require the use of .NET 9.

2.2 External Data Sources

There are two data sources this program will use: the NIED and the DM-D.S.S. Specifically, the former one is used to achieve the real-time shake data of the sensor points which were set up by the government (whose data is free to use), and the latter one is used to achieve past earthquake information and EEW information sent out by the JMA (which is pay-to-use). Note that DM-D.S.S. does also provide the real-time intensity data of the observation points, however it is pay-to-use only for companies and institutions on request. Therefore, it will not be feasible to use this data source in the program since one of and the principle target users is people passionate in monitoring earthquakes.

2.2.1 NIED Data Source

As mentioned before, NIED has numerous 'earthquake observation nets' across Japan. Specifically, there is the K-NET and KiK-net [80], which is dedicated to the observation of strong seismic motion. The K-NET consists of approximately 1000 sensors located across Japan, while the KiK-net also includes some sensors which are located within the earth, which will often have different readings compared to those

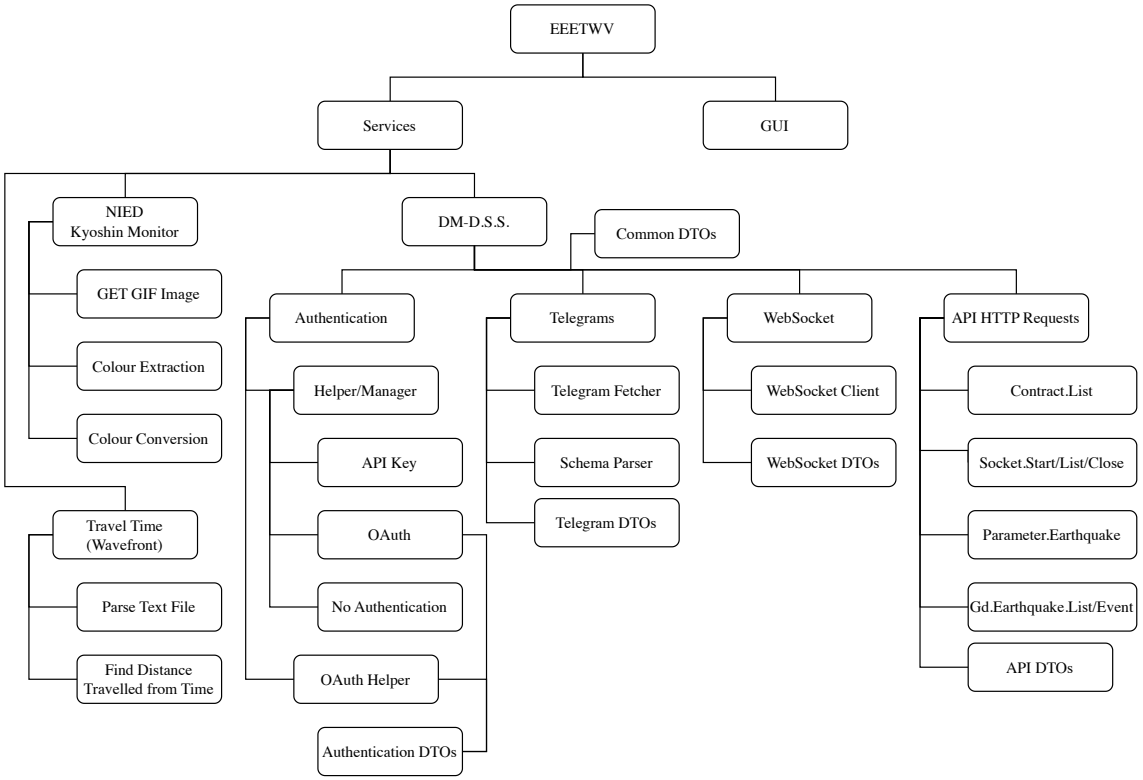


Figure 2.1: Hierarchy chart of the whole application

located on the surface. They are extremely capable of detecting strong motion of ground. Furthermore, the K-NET and the KiK-net provides real-time intensity data webpage of two types, the 'Kyoshin' (強震) monitor and the long-period ground motion (LPGM, 長周期地震動) monitor (not working at the time of investigation). The Hi-net [79] stands for high-sensitivity seismograph network, and it is dedicated for observation of minor motions of the ground. They release the waveforms to those who are researching seismic movements. As for the F-net [78] which stands for the Full Range Seismograph Network of Japan, which is used to analyse the mechanism of a certain earthquake by analysing movements. None of the three nets provide a real-time API data feed.

Having compared the functionalities described above of the K/KiK-net, Hi-net and F-net and how they feed the data sources, the most suitable data source to reflect real-time motion of ground movements will be the **K/KiK-net**'s data feed, since it detects strong ground movements and is available real-time for the purpose of the application. (This is also the data source that JQuake and KEVI use in fact.)

In fact, in addition to these three networks, there are also the S-net, the DONET and the N-net, which detects the ground seismic movements in the sea. These data were adapted by SREV, but this is beyond the scope of this NEA analysis.

A comparison from the official website of MOWLAS (Monitoring of Waves on Land and Seafloor) [81] of the three nets are included in Figure 2.3 and a map of the distribution of the sensors are included in Figure 2.4.

2.2.1.1 Achieving image format data source

The data source fed by the 'Kyoshin' Monitor is split into 8 types (detailed below in Table 2.1), and each type split into 2 types of data sources, surface sensors and borehole (earth) sensors, with codes in Table 2.2. The link to the GIF image is in the following format:

`http://www.kmoni.bosai.go.jp/data/map_img/RealTimeImg/[#1]_[#2]/[yyyyMMdd]/[yyyyMMdd][hhmmss].[#1]_[#2].gif`

In the link, the `[yyyyMMdd]` and the `[hhmmss]` part should be replaced with the date and time respectively (in JST, UTC+8), and the `#1` replaced with the codes detailed below for the data types,

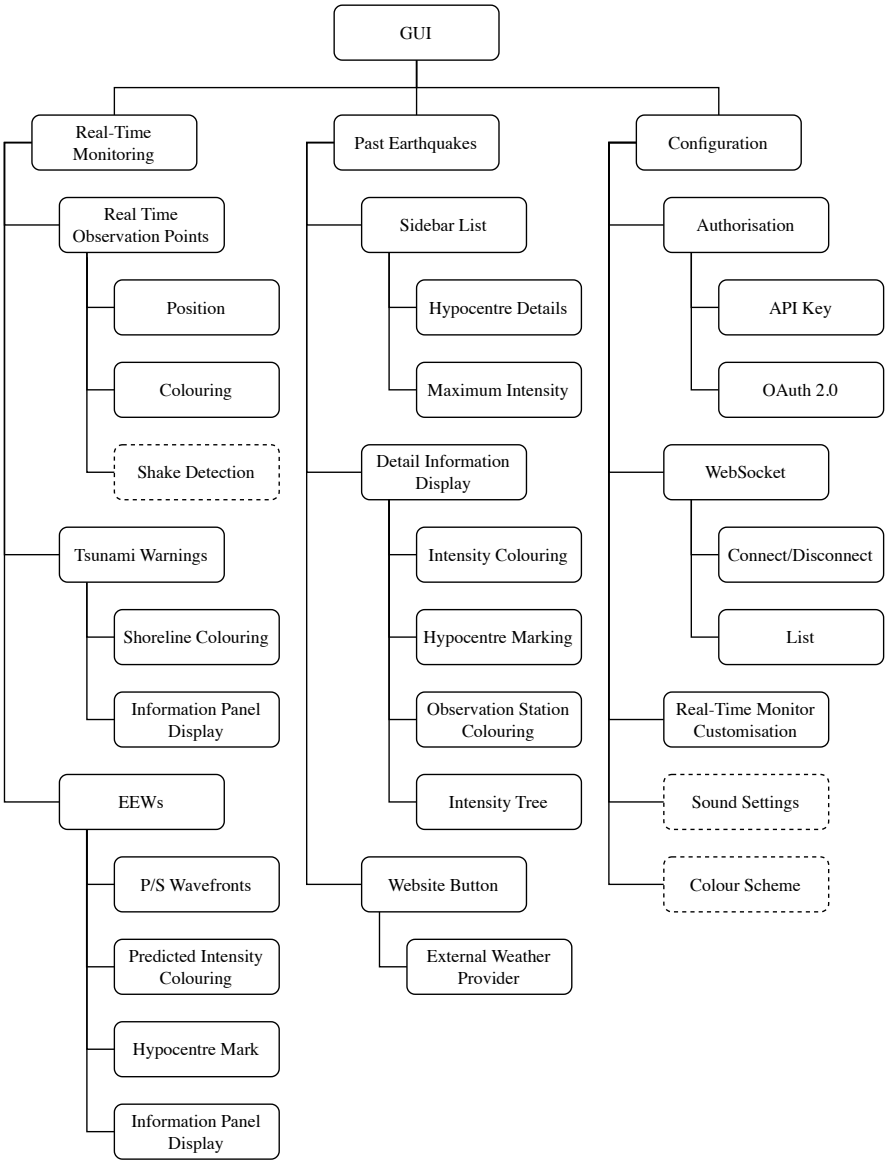


Figure 2.2: Hierarchy chart of the GUI Component

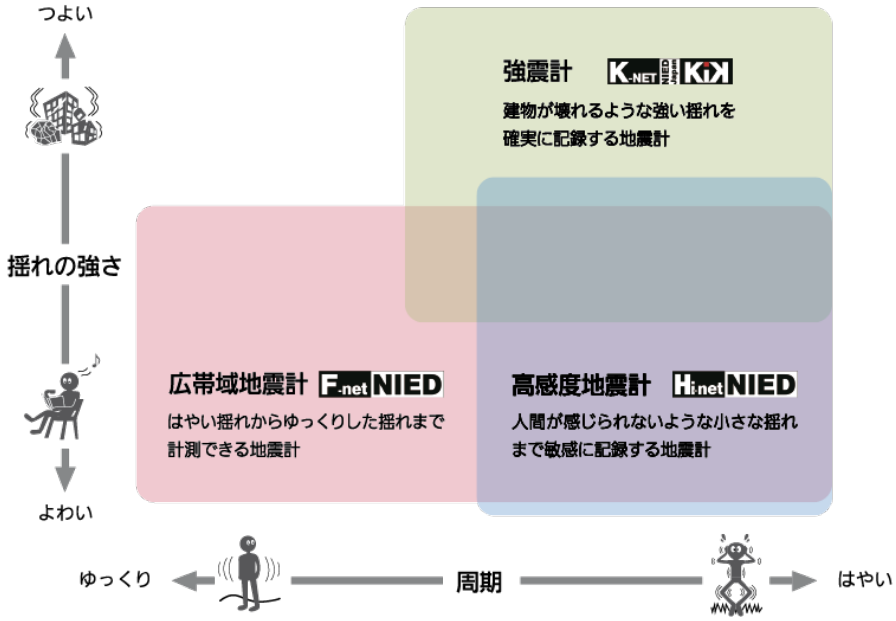


Figure 2.3: A comparison of the K-NET, F-net and Hi-net.

Figure 2.4: Distribution of the sensors of different nets.

and `#2` replaced with the codes detailed below for data sources. An example of the imaged achieved is in Figure 2.5.

Data Type	Description/Meaning	Code in `#1`
Real-time Shindo	Real-time Measured Intensity	`jma`
PGA	Peak (Maximal) Ground Acceleration	`acmap`
PGV	Peak (Maximal) Ground Velocity	`vcmap`
PGD	Peak (Maximal) Ground Displacement	`dcmap`
Response 0.125Hz	Response spectrum for 0.125Hz PGV	`rsp0125`
Response 0.250Hz	Response spectrum for 0.250Hz PGV	`rsp0250`
Response 0.500Hz	Response spectrum for 0.500Hz PGV	`rsp0500`
Response 1.000Hz	Response spectrum for 1.000Hz PGV	`rsp1000`
Response 2.000Hz	Response spectrum for 2.000Hz PGV	`rsp2000`
Response 4.000Hz	Response spectrum for 4.000Hz PGV	`rsp4000`

Table 2.1: Data available in 'Kyoshin' monitor

Sensor Type	Description/Meaning	Code in `#2`
Surface	K-NET and KiK-net sensors	`s`
Borehole	KiK-net sensors within earth	`b`

Table 2.2: Sensors available in 'Kyoshin' monitor

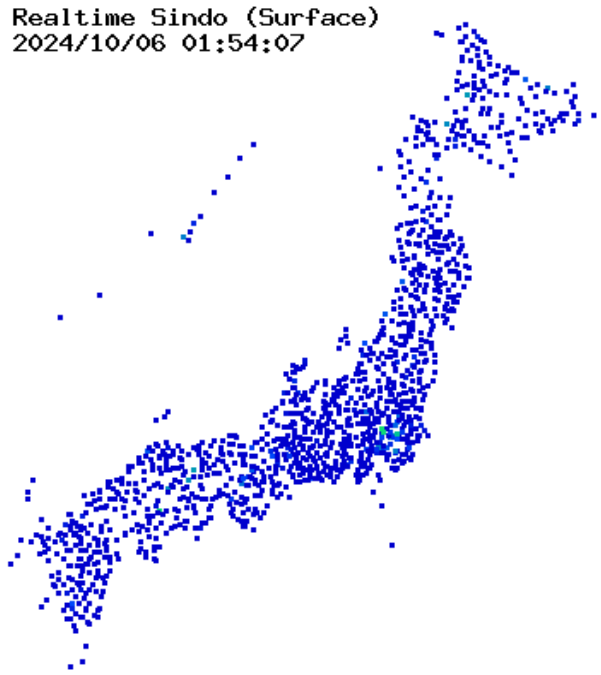


Figure 2.5: Sample GIF image achieved from 'Kyoshin' Monitor

2.2.1.2 Extracting colour for each observation point

Unfortunately, it seems to the author (and is widely accepted in the EEW monitoring app development society) that the position of the points (squares) on the image does not follow any significant pattern of position, i.e. there is no obvious conversion of coordinates to us from the official longitude/latitude locations to the positions on the image. Therefore, a manual conversion one-to-one mapping has to be developed.

NIED does have an official released list of observation points, which include their names and positions. This list has around 1700 of those observation points. However, in the actual image (like those in Figure

2.5), there are only 1000 of those in use in real time, consistent with K-NET's official introduction, and the rest 700 of those are invalid observation points. Therefore, it will be worth removing them from the list of earthquake monitoring points, before attempting to make the dictionary.

Unfortunately, 1000 is still quite a lot for us to deal with. Luckily, Ingen who used a similar approach to develop the KEVI application has already made such a mapping inside his open-source application in the file ShindoObsPoints.mpk.lz4 within ingen084/KyoshinEewViewerIngen , and even developed an editor for this at ingen084/KyoshinShindoPlaceEditor .

Due to the limited time for this NEA, the author will primarily use the pre-determined observation points for the K-NET and the KiK-net by Ingen in JSON format.

This paragraph referred to [35].

2.2.1.3 Converting colour to number format for further processing

The true numerical data does not seem fully necessary at the first glance (since we might just as well just achieve the colour from the image and just plot them on the map, without the need to convert to a colour and back). However, for us to detect the shake in certain regions, it is necessary for us to achieve the numerical value to run the algorithm on it. Nevertheless, it is just good to have the number for us to have the numerical value for potential future developments.

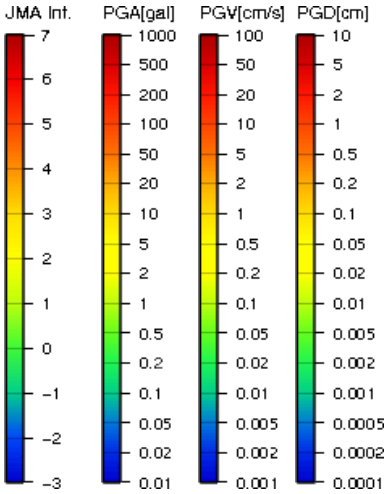


Figure 2.6: Scale colours of different measurements

As shown in Figure 2.6, the NIED 'Kyoshin' Monitor does indeed provide a scale of colours and reference to numerical values. However, there is a chance that a certain colour is not 'exactly' mapped on the scale, and further concerning that it is very slow and difficult to 'loop over' a colour legend, it is necessary to have an algorithmic-approach (numerical mapping-based approach) to map the colours in the colour space to numerical values (and back) is necessary.

Abstraction of colour scale Notice that the scale for PGA/PGV/PGD follow a logarithmic scale, while measured intensity follows a linear scale (though noting that the way intensity and magnitude is calculated is logarithmic as well). Therefore, if we normalise the vertical distance from the bottom of the axis h to $0 \leq h \leq 1$ (i.e. $h = 0$ at the bottom of the scale, $h = 1$ at the top of the scale), and if we denote intensity using I in JMA scale, PGA as a in gal, PGV as v in cm per second, and PGD as s in cm, from the scale, the following transforming formulae obviously hold:

$$
\begin{aligned}
I = 10h - 3 &\iff h = \frac{I+3}{10}, \\
a = 10^{5h-2} &\iff h = \frac{\lg a + 2}{5}, \\
v = 10^{5h-3} &\iff h = \frac{\lg v + 3}{5}, \\
x = 10^{5h-4} &\iff h = \frac{\lg x + 4}{5}.
\end{aligned}
$$

However, it is worth noting that NIED did use $1, 2, 5, 10$ on the logarithmic scale at equal intervals, so it is not a perfect logarithmic scale. The author is unsure why they designed the scale like this, nor if it's an intended approximation. Nevertheless, the logarithmic scale is a good enough approximation.

The next step is to develop a mapping from this colour space $\mathcal{C}$ to h, which of course should be invertible. Denote this as $f : [0, 1] \to \mathcal{C}$.

Describing colour numerically We consider using a suitable base to decompose $\mathcal{C}$. The colour of the given scale is an immediate suggestion to use a base containing **hue**, which in fact is designed to describe how human perceive colour, and unlike RGB and CMYK which uses principle colours to describe colour. A hue scale is shown in Figure 2.7.

Figure 2.7: The hue scale in HSL/HSV encoding

Therefore, a colour in the colour space $\mathcal{C}$ can be represented as a 3-D vector $\mathcal{C} \ni C = (H, S, V)$, where $H \in [0, 360)$ in degrees is the hue value, $S \in [0, 1]$ stands for the saturation, and $V \in [0, 1]$ stands for the value (a brightness). And hence we will be able to decompose f into three components $f = (f_H, f_S, f_V)$.

Figure 2.8 plots the values of H, S and V against h (this is the graph of f and its components) of discrete values of h, and depending on the result we will attempt some fit/regression to a suitable function.

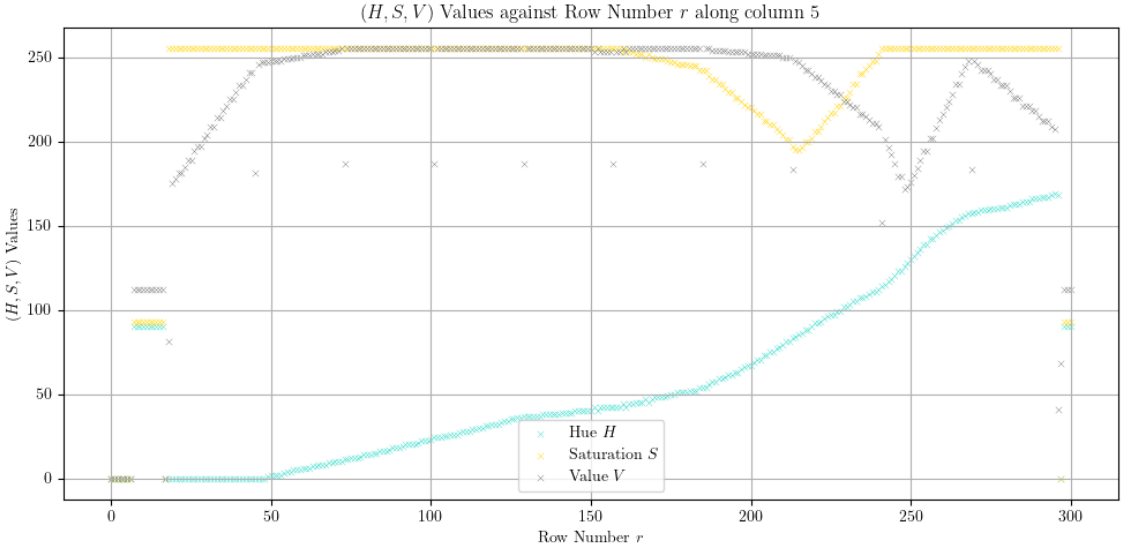


Figure 2.8: The values of (H, S, V) against pixel row r

Notice that in this plot, all values of (H, S, V) in fact range from 0 to 255, as in an 8-bit binary.

It is worth noting that the scale has some space on the top (to show the type), and some space at the bottom. Notice that when the row $r = 17$ and $r = 297$ have values significantly different, so we extract the rows $r = 18$ and $r = 296$ to correspond (linearly) to $h = 1$ and $h = 0$, i.e.,

$$h = 1 - \frac{r - 18}{278}.$$

Figure 2.9 shows the result of this transformation being applied.

From here onwards, all values of (H, S, V) will be adjusted to be within the range which they should be in, i.e. $H \in [0, 360), S \in [0, 1], V \in [0, 1]$.

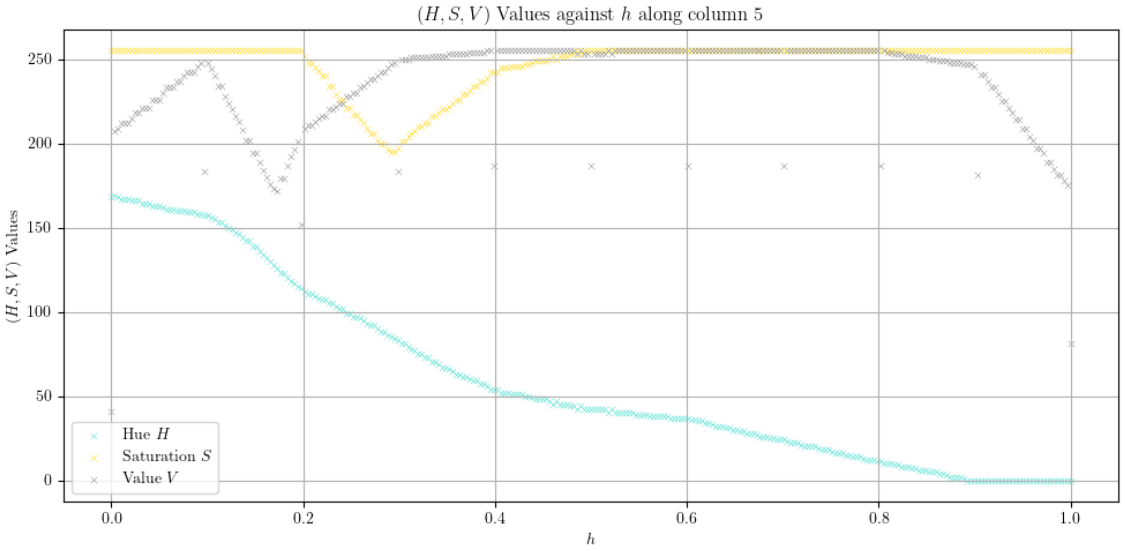


Figure 2.9: The values of (H, S, V) against normalised height h

Finding f_H in terms of h We consider finding f_H first, which is the cyan line. Notice that its trend can be split into 4 parts:

- $h \in [0, 0.1]$: linear;
- $h \in [0.1, 0.6]$: curving, ideally a cubic;
- $h \in [0.6, 0.9]$: linear;
- $h \in [0.9, 1]$: constant (0).

h	$H = f_H(h)$
0	237
0.1	222
0.6	51
0.9	0
1	0

Table 2.3: Initial values for f_H

Furthermore, boundary conditions in Table 2.3 are applied to ensure that the function is continuous and nicely-behaving while matching the existing data. We use the following function to apply the fit:

$$f_H(h) = \begin{cases} -150h + 237, & h \in [0, 0.1], \\ \odot, & h \in [0.1, 0.6], \\ -170h + 153, & h \in [0.6, 0.9], \\ 0, & h \in [0.9, 1]. \end{cases}$$

Here,

$$\begin{aligned} \odot = &\frac{222 \cdot (h - 0.3) \cdot (h - 0.4) \cdot (h - 0.6)}{(0.1 - 0.3) \cdot (0.1 - 0.4) \cdot (0.1 - 0.6)} \\ &+ \frac{y_1 \cdot (h - 0.1) \cdot (h - 0.4) \cdot (h - 0.6)}{(0.3 - 0.1) \cdot (0.3 - 0.4) \cdot (0.3 - 0.6)} \\ &+ \frac{y_2 \cdot (h - 0.1) \cdot (h - 0.3) \cdot (h - 0.6)}{(0.4 - 0.1) \cdot (0.4 - 0.3) \cdot (0.4 - 0.6)} \\ &+ \frac{51 \cdot (h - 0.1) \cdot (h - 0.3) \cdot (h - 0.4)}{(0.6 - 0.1) \cdot (0.6 - 0.3) \cdot (0.6 - 0.4)}. \end{aligned}$$

Here, m_1 is the gradient of the line for $h \in [0, 0.1]$, $y_1 = f_H(0.3), y_2 = f_H(0.4)$ for $h \in [0.1, 0.6]$ (using Lagrange Polynomial), and the equation between $h \in [0.6, 0.9]$ is in fact fixed due to the initial conditions.

By applying a curve fit to the original data, the following results are obtained:

$$(y_1, y_2) = (115, 79.5).$$

Plotting H and $f_H(h)$ against h gives us Figure 2.10, which is decent.

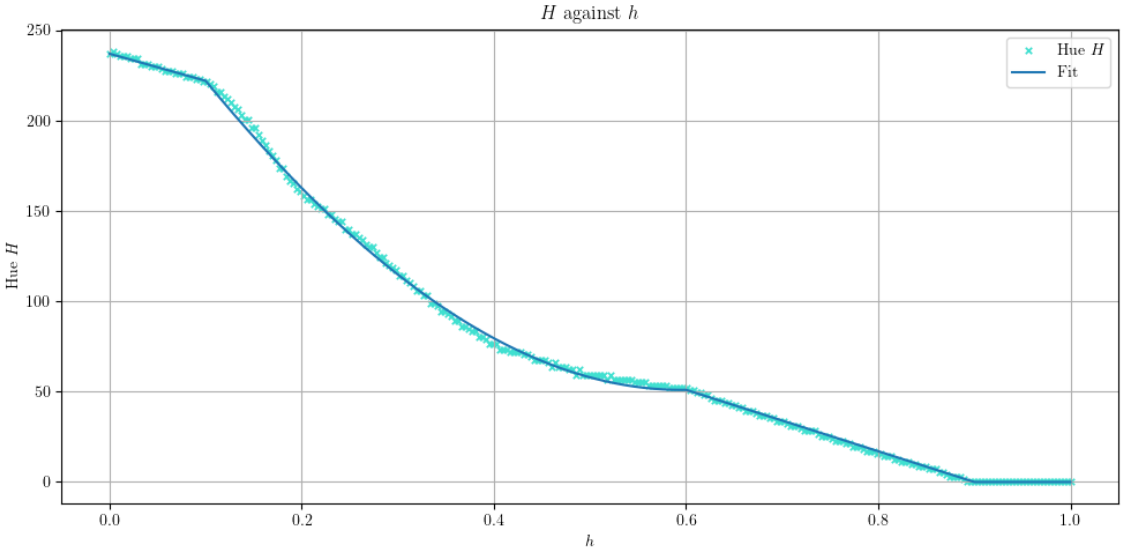


Figure 2.10: The fit result for $f_H : h \mapsto H$

Finding f_S in terms of h As for $f_S(h)$, the obvious thing to do is to split it into 5 (4) piecewise functions, specifically $f_S = 1$ for $h \in [0, 0.2] \cup [0.5, 1]$, and three linear functions for $h \in [0.2, 0.29], h \in [0.29, 0.4]$ and $h \in [0.4, 0.5]$. Initial values are included in Table 2.4.

h	$S = f_S(h)$
0	1
0.2	1
0.29	0.765
0.4	0.95
0.5	1
1	1

Table 2.4: Initial values for f_S

This gives us that

$$f_S(h) = \begin{cases} 1, & h \in [0, 0.2], \\ -2.611h + 1.522 & h \in [0.2, 0.29], \\ 1.682h + 0.277, & h \in [0.29, 0.4], \\ 0.5h + 0.75 & h \in [0.4, 0.5], \\ 1, & h \in [0.5, 1]. \end{cases}$$

Plotting this out gives Figure 2.11.

Finding f_V in terms of h As for $f_V(h)$, we shall divide it into even more piecewise linear functions. Specifically, I chose to divide the interval $[0, 1]$ at $0.1, 0.172, 0.2, 0.3, 0.4, 0.8$ and 0.9. Initial values are included in Table 2.5.

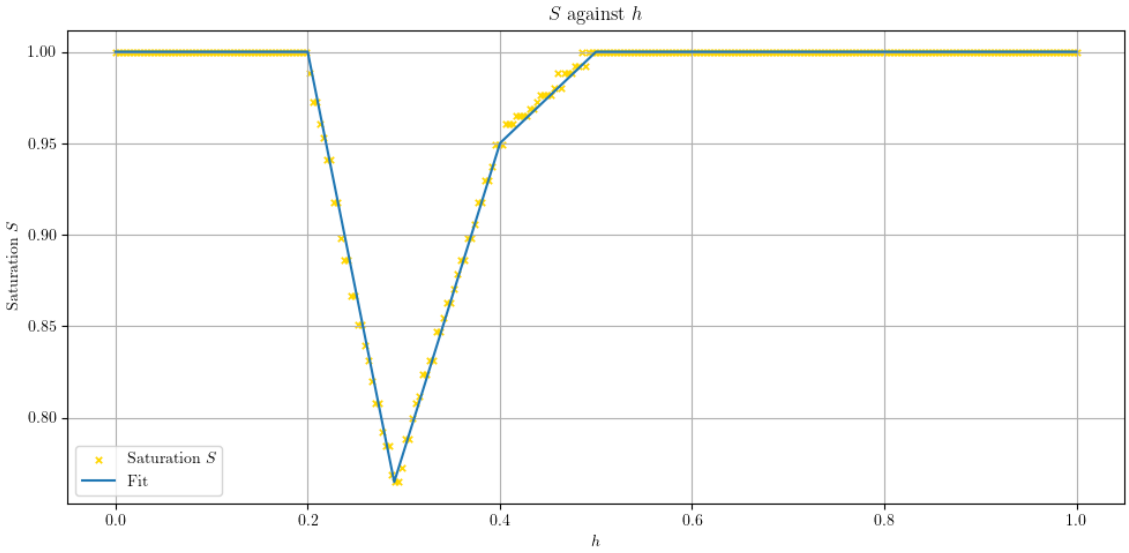


Figure 2.11: The fit result for $f_S : h \mapsto S$

h	$V = f_V(h)$
0	0.8
0.1	0.98
0.172	0.66
0.2	0.82
0.3	0.98
0.4	1
0.8	1
0.9	0.97
1	0.68

Table 2.5: Initial values for f_V

This gives us the piecewise function

$$f_V(h) = \begin{cases} 1.8h + 0.8, & h \in [0, 0.1], \\ -4.444h + 1.424, & h \in [0.1, 0.172], \\ 5.714h - 0.323, & h \in [0.172, 0.2], \\ 1.6h + 0.5, & h \in [0.2, 0.3], \\ 0.2h + 0.92, & h \in [0.3, 0.4], \\ 1, & h \in [0.4, 0.8], \\ -0.3h + 1.24, & h \in [0.8, 0.9], \\ -2.9h + 3.58, & h \in [0.9, 1]. \end{cases}$$

Plotting this out gives us Figure 2.12.

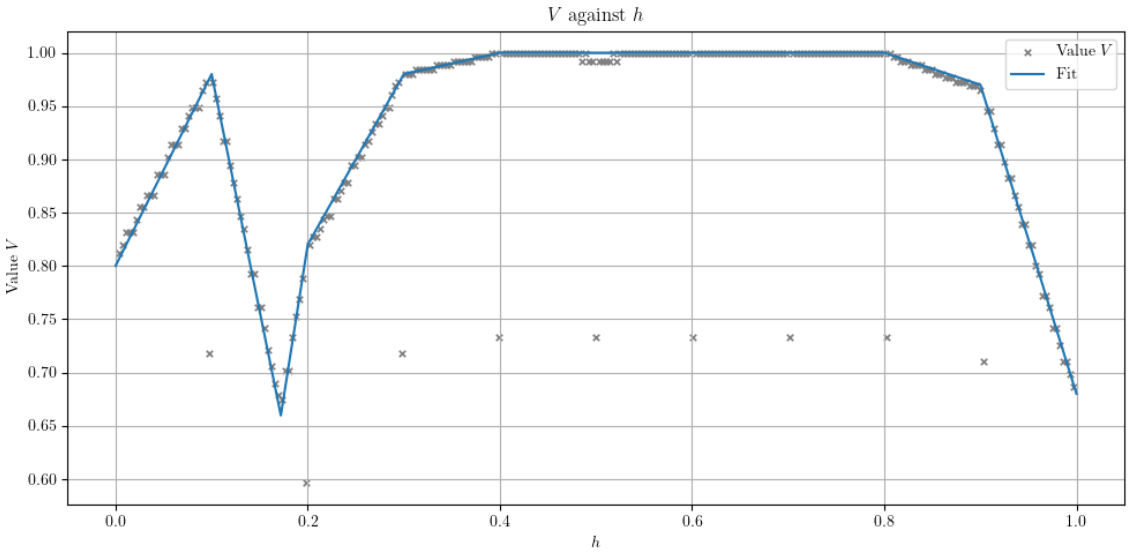


Figure 2.12: The fit result for $f_V : h \mapsto V$

Note that in this plot, V when $h = 0$ or $h = 1$ is excluded, since just like every $h = 0.1k$ for some $k \in \mathbb{N}$, they are anomalies created by the horizontal black line in the scale.

Finding f^{-1} To find $f^{-1} : \mathcal{C} \to [0, 1]$, we do not need necessarily to find an expression of h in terms of (H, V, S). If we notice that f_H is one-to-one on $h \in [0, 0.9]$, and f_V is one-to-one on $h \in [0.9, 1]$, we can use f_H^{-1} to determine h from H only if H is non-zero, and use f_V^{-1} otherwise.

$$f^{-1}(H, S, V) = \begin{cases} f_H^{-1}(H), & H \neq 0, \\ f_V^{-1}(V), & H = 0. \end{cases}$$

Notice that for $h \in [0.1, 0.6]$, f_H is a cubic and is not easily invertible. However, it would be plausible to use a binary-search algorithm to find h based on H since it is monotonic, and it is within a reasonable amount of time, to relatively good precision. Otherwise, on the linear parts, it is fine to simply mathematically invert it.

Algorithm 1 describes the logic. Note that, this algorithm uses different V ranges from before (which is consistent with SkiaSharp library in C#), and the ranges is $V \in [0, 100]$. Notice that, $\epsilon(h)n$ and $\epsilon(H)$ are two constant values that determines the precision of such binary search, and they are chosen to be 0.01 and 0.5 in the implementation.

2.2.1.4 Flowchart of data and sidenotes

To summarise, we discussed the mapping from the colour space$\mathcal{C}$ to the normalised height h, and back, and we also discussed how h is related with the measured intensity I, the PGA a, the PGV v, and the

Algorithm 1 Algorithm for f^{-1}

Require: $H \in [0, 360)$
Require: $S \in [0, 100]$
Require: $V \in [0, 100]$
Ensure: $h \in [0, 1]$
  $V \leftarrow V/100$ ▷ *Normalise V*
  if V is 0 **then** ▷ *Use V*
    return $(V - 3.50)/(-2.9)$
  else▷ *Use H*
    ▷ *Deal with out-of-range Hs, and use inverse linear functions* ◁
    if $H \geq 222$ **then**
      return $(H - 237)/(-150)$
    else if $H \leq 51$ **then**
      return $(H - 153)/(-170)$
      ▷ *Inverse-Cubic by Binary Search, with set errors* ◁
    else
      $l \leftarrow 0.1$
      $r \leftarrow 0.6$
      ▷ *Checks if the normalised height is already precise enough* ◁
      while $r - l \geq \epsilon(h)$ **do**
        $m \leftarrow (l + r)/2$
        $c \leftarrow$ Height to Hue(m)
        ▷ *Checks if the calculated hue is already precise enough* ◁
        if $|c - H| \leq \epsilon(H)$ **then**
          return m
        else if $c > H$ **then**
          $l \leftarrow m$
        else if $c < H$ **then**
          $r \leftarrow m$
        end if
      end while
      return $(l + r)/2$
    end if
  end if

PGD x. They can be transformed forwards and backwards using simple mathematical explicit relations, and specifically for $f_H^{-1}(H)$ will use a binary search algorithm.

Figure 2.13 shows the data flow, Figure 2.14 shows the relation between abstract variables, and 2.15 shows the result of colour generated compared with the original.

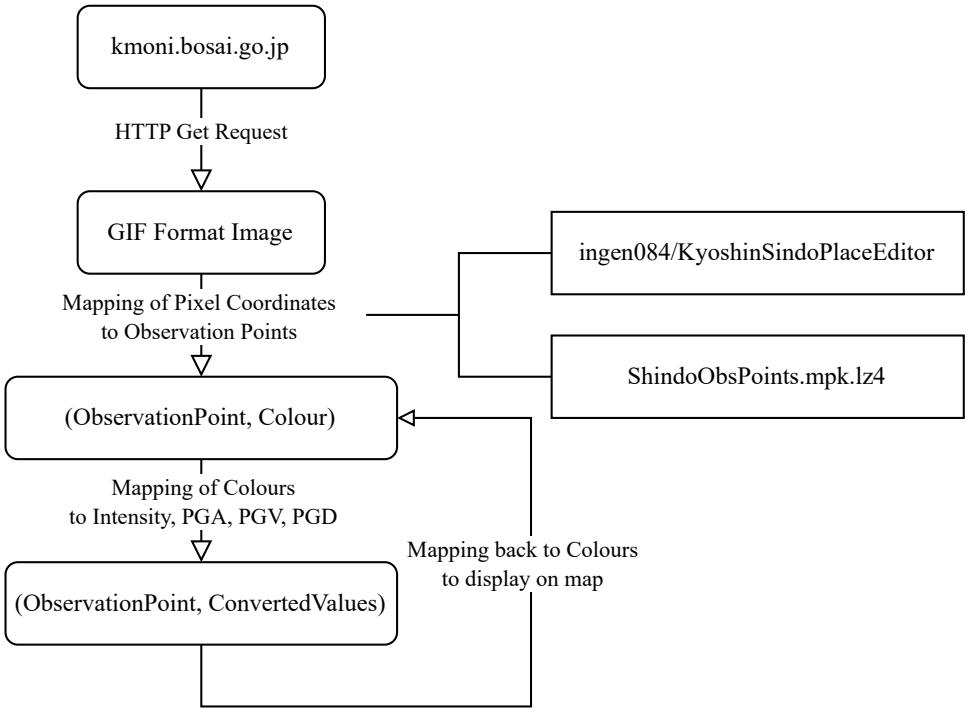


Figure 2.13: Flow of data in NIED data sources

It is worth noting that an existing NuGet Library, ingen084/KyoshinMonitorLib and introduced in [31] which is designated to manage intensities, as well as extracting intensities from the 'Kyoshin' monitor. This NEA did refer to this for some guidance but is not dependent on this library, and its necessary functionalities within the scope of this NEA is realised again using the author's own code. The developer of this library did also mention that it is quite purpose-built so might not be suitable for general use.

It is also worth noting that, technically, scraping the data from the 'Kyoshin' monitor page of NIED is not explicitly allowed, but not explicitly banned either. However, extracting and displaying numerical data in the application is strictly banned by the NIED, and therefore the numerical values will only serve as internal values of the application and will not be displayed to the users in any way.

This paragraph referred to [58]. The code used for this section is in Listing A.1.

2.2.2 DM-D.S.S. Data Source

DM-D.S.S. is a well-structured official data source with low latency and reliable information and services. This is going to be the primary data source for most part of the application.

Their APIs are split into two types: HTTP based requests and WebSocket based connections. HTTP based requests are typically for more static information, while WebSocket connections are for live time-essential data feeds, such as the EEW warnings, tsunami warnings and latest earthquake information.

2.2.2.1 Authorisation

There are two types of authorisation that DM-D.S.S. supports, API Keys and OAuth2 Access Tokens.

API Keys access tokens are extremely easy to program, since it simply uses Basic Authorisation in the header, and uses the key as the username (without a password). It uses the Basic BasicBase64 Authorisation encoding. However, this introduces an extra layer of complexity for the users, since they would have to go to the settings of the DM-D.S.S. webpage and achieve an API Key to paste into the application, as shown in Figure 2.16.

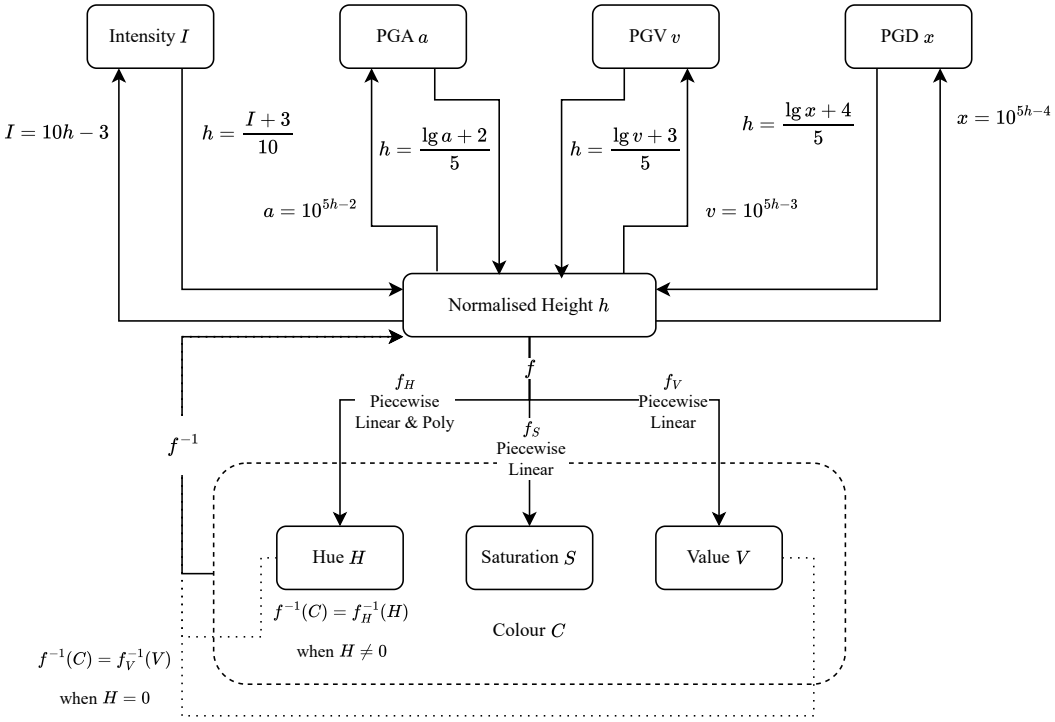


Figure 2.14: Relation between abstract variables

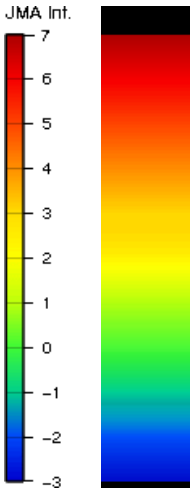


Figure 2.15: Colour generated using fitted functions

Figure 2.16: Control panel for API Keys

As for OAuth2, it will be much simpler for the users, since it will provide the user with a login interface on the website, and ask them to give the program certain permissions, which is just a few simple clicks. Rather than using the user as a bridge for sharing the credentials with the application, they are shared between the authorisation server (DM-D.S.S.) and the application directly, without the need for the user to deal with such human-unreadable codes.

OAuth 2.0 OAuth 2.0 is a standard protocol that allows a user to authorise an application without sharing any of their credentials. Figure 2.17 gives a brief outline of the procedure involved.

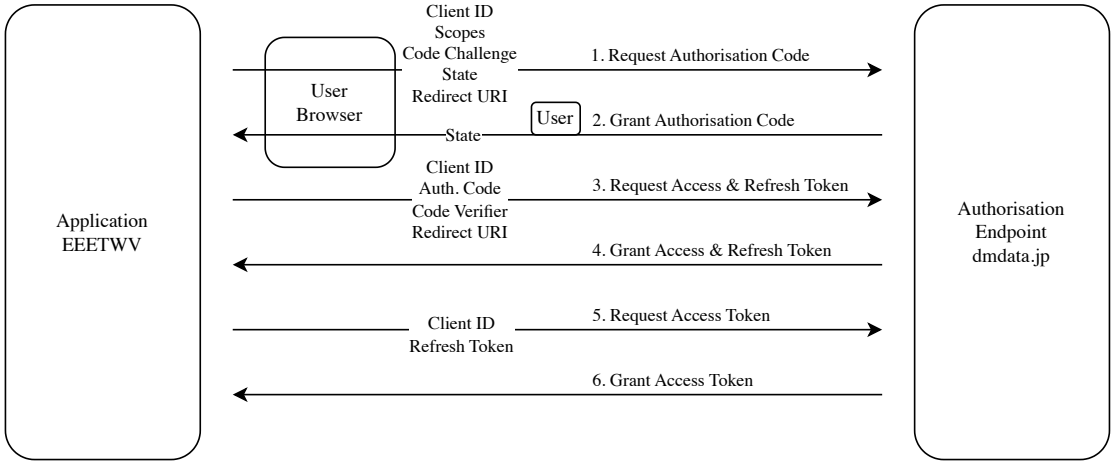


Figure 2.17: OAuth Procedure Outline

Authorisation Code The authorisation begins with the application requesting an **authorisation token** from the authorisation endpoint, via the user's browser. This is the only step where the user has to be involved, but all they have to do is log-in to their DM-D.S.S. account and click the 'grant' button on the page.

First, the user launches the browser to the authorisation endpoint (Step 1)

`https://manager.dmdata.jp/account/oauth2/v1/auth`

with the following parameters:

- `client_id`: the client ID for the application to grant (which is public and safe to be seen by the user);
- `redirect_uri`: redirect URL for the endpoint to redirect to, to inform the application of the authorisation results;
- `response_type`: `code`
- `response_mode`: the way to response to the application (we will only use the response type `query`);
- `scope`: the string containing the permissions the application is requesting, separated by a space;
- `state`: the state which will be returned by the return query to prevent CSRF attacks;
- `code_challenge`: the code challenge for PKCE;
- `code_challenge_method`: the way to encode the code challenge (we will use `S256`).

At the same time as launching the browser, the program should start a listener on the local IP address and port specified in the redirect URL, and wait until it receives a request from the browser. In the response (Step 2), it will contain two parameters: `code` which contains the authorisation code (which is only valid for 10 minutes), and `state` which should be the same as the state sent in the request.

The reason why the **state** is essential is to prevent CSRF (Cross-Site Request Forgery) attacks, where the application can make sure the response is truly corresponding to the request the application

Error Code	Description
`invalid_request`	The parameters are invalid.
`invalid_client`	The Client ID is invalid.
`invalid_redirect_url`	The redirect URL is not supported by the Client ID.
`invalid_scope`	The scope string is invalid.
`unauthorized_client`	The client to authorise using this method is invalid.
`access_denied`	The access is denied.
`recaptcha_verification_failed`	The reCAPTCHA verification failed.
`unsupported_response_type`	The response type is not recognised.
`unsupported_code_challenge_method`	The code challenge method is not recognised.
`no_signin`	The user is not signed in (usually should not happen).
`server_error`	The internal server encountered an error.

Table 2.6: Error codes for authorisation code step in OAuth2

made (not by any other applications), and no one forged a response to the application (pretended to be the browser acting as authorisation endpoint). Verifying the state is essential for the security of the application.

If the request fails, the return query will return the `error`, and with the `state` as well. Table 2.6 includes a table of errors that could occur in the authorisation code request.

In the application, only the errors `access_denied`, `recaptcha_verification_failed`, `no_signin` and `server_error` should appear (and the final two shouldn't appear usually), since all other errors are due to issue with design of the application.

Refresh and Access Token After this step, the program will use the authorisation token to request an **access token** which comes with a **request token** (Step 3). The program will send an HTTP POST request to the token endpoint

`https://manager.dmdata.jp/account/oauth2/v1/token`

with the following query parameters, in URL Encoded form:

- `client_id`: the client ID for the application to grant (which is public and safe to be seen by the user);
- `grant_type`: `authorization_code`;
- `code`: the authorisation code received from the previous step;
- `redirect_uri`: the redirect URL used in the previous step;
- `code_verifier`: the original version of the code challenge used in the previous step.

Now the reason of the code challenge and code verifier is apparent (they are part of PKCE, Proof Key for Code Exchange): since the code challenge is a hashed version of the code verifier and is exchanged in the first step, the code verifier (the plain version) is used to prove that the application is the same as the one that requested the authorisation code. This means, if a man-in-the-middle acquired the authorisation code and the code challenge, they would not be able to use it, since it requires them to reverse hash the code challenge to get the code verifier, which is impossible in a reasonable amount of time. To ensure the security, the code verifier should be generated randomly and be very long, and disposed after each use.

The response (Step 4) will be JSON format, as shown in Listing 2.1.

The properties `token_type`, and `expires_in` are constants, the `scope` are the scopes which are authorised for, and the most important two are the `access_token` and `refresh_token`.

The refresh token is a long-living token, and it has lifetime 183 days in DM-D.S.S., resetting every time a new access token is acquired (see below). On the other hand, the access token is a short-living token, and it has lifetime 6 hours only. This does not mean that the access token should be disposed per-use – it should be reused within the application, and only when it expires, a new access token should be requested using the refresh token. However, when the application is closed, only the refresh token should be stored, and the access token revoked – next time the application launches, the access token should be acquired again.

```
{
    "access_token": "ATn.TTTTTTTTTTTTTTTTTTTTTTTTT",
    "token_type": "Bearer",
    "expires_in": 21600,
    "refresh_token": "ARh.RRRRRRRRRRRRRRRRRRRRRRRRR",
    "scope": "telegram.list telegram.get.earthquake telegram.data"
}
```

Listing 2.1: Response for OAuth Access Token Request

```
{
    "error": "invalid_request",
    "error_description": "The client_id is missing."
}
```

Listing 2.2: Error for OAuth Access Token Request

If the request failed, then an error will be returned, also in JSON format, as shown in Listing 2.2. Table 2.7 includes a table of errors that could occur in the authorisation code request. The only error that should occur here is `server_error`.

Refreshing Access Token Finally, Step 5 and 6 will be repeated multiple times in the application, to request a new access token using the provided refresh token. Similarly, the request will be a post request, but only requiring the `client_id`, the `grant_type` (as `refresh_token`) and `refresh_token` in the URL Encoded form, to the same endpoint.

The response JSON will be exactly the same, except the refresh token is not included again. It includes the newly acquired access token. In case of an error, all errors in Table 2.7 apart from `invalid_redirect_url` and `invalid_code_verifier` could occur. In this case however, there could be a chance of `invalid_grant`, if the application is closed for a very long period of time and the refresh token expired. In this case, the user should be asked to re-authorise the application.

Revoking Tokens Tokens should be revoked whenever they are no longer used. The revoke is done by sending a POST request to the revoke endpoint, which is

`https://manager.dmdata.jp/account/oauth2/v1/revoke`,

with the following parameters:

- `client_id`, the client ID;
- `token`, the token to be revoked.

The API will not return anything if successful, and if the token is already invalid, it will still be successful.

Error Code	Description
`invalid_request`	The parameters are invalid.
`invalid_client`	The Client ID is invalid.
`invalid_redirect_url`	The redirect URL is not supported by the Client ID.
`invalid_grant`	The authorisation code is invalid.
`invalid_code_verifier`	The PKCE verification failed.
`unauthorized_client`	The client to authorise using this method is invalid.
`unsupported_grant_type`	The grant type is not recognised.
`server_error`	The internal server encountered an error.

Table 2.7: Error codes for access token step in OAuth2

An error will be similar to that in Table 2.7, but only `invalid_request`, `invalid_client` and `server_error` could occur. Nonetheless, only `server_error` should occur in the application.

Flowchart In terms of the design of classes to support OAuth 2.0, there should be two separate classes responsible for this: one for acquiring the authorisation code and the refresh token, and the other for acquiring the access token using the refresh token.

Figure 2.18 shows the flowchart for the former, and 2.19 shows the flowchart for the latter.

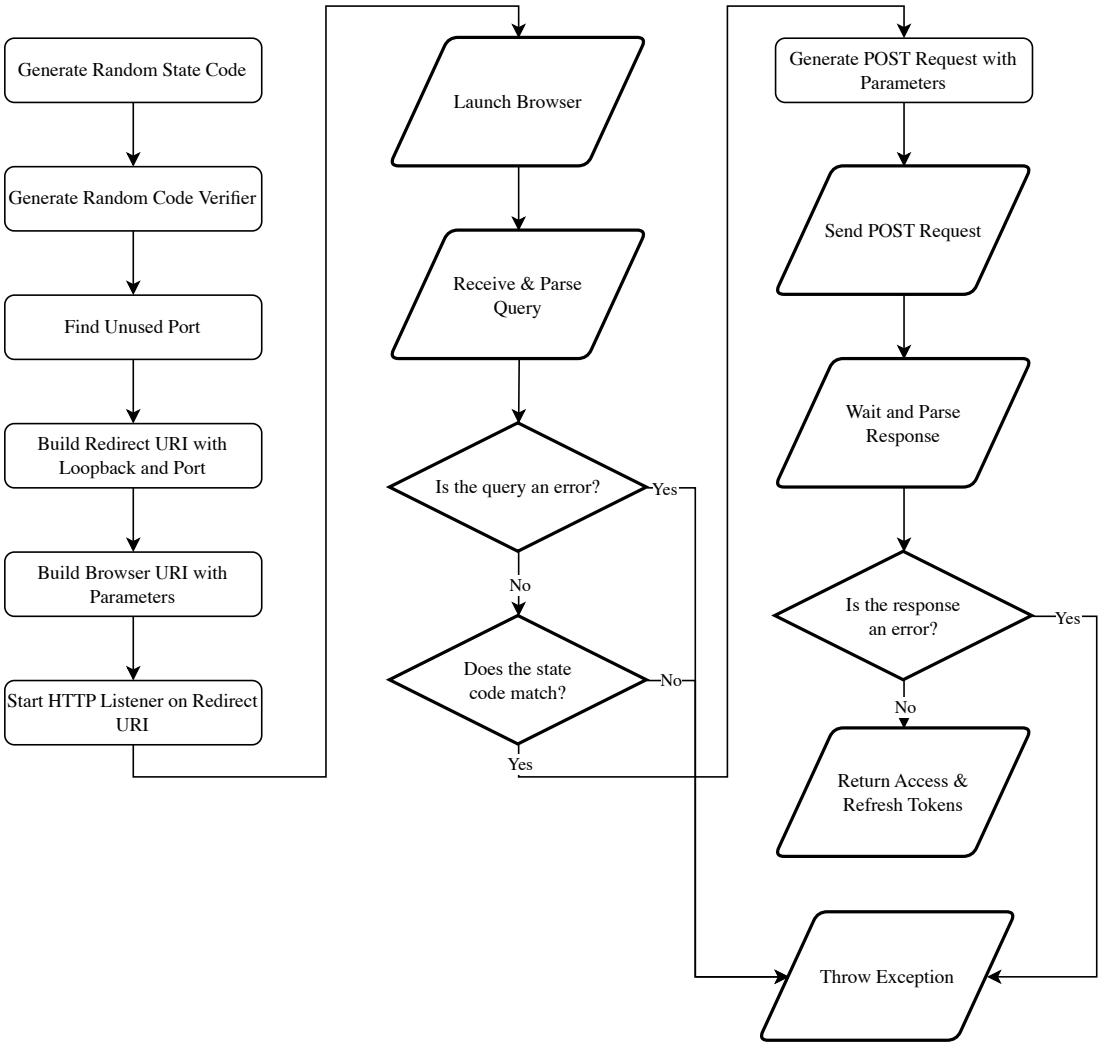


Figure 2.18: Flowchart for OAuth 2.0 Authorisation Code and New Refresh Token

Figure 2.19 represents the behaviour of the OAuth2 main class for authentication, while the previous figure 2.18 represents the behaviour of the helper.

Notes For the purposes of this NEA, we will primarily use the API Key to test the application for accessing way of accessing DM-D.S.S. since it will be easier to code and debug, and OAuth2 will introduce quite a lot of complexity to the program. However, the program should be designed to be able to modify to OAuth2 authentication without much modification, and if time permits OAuth2 will be implemented in the application.

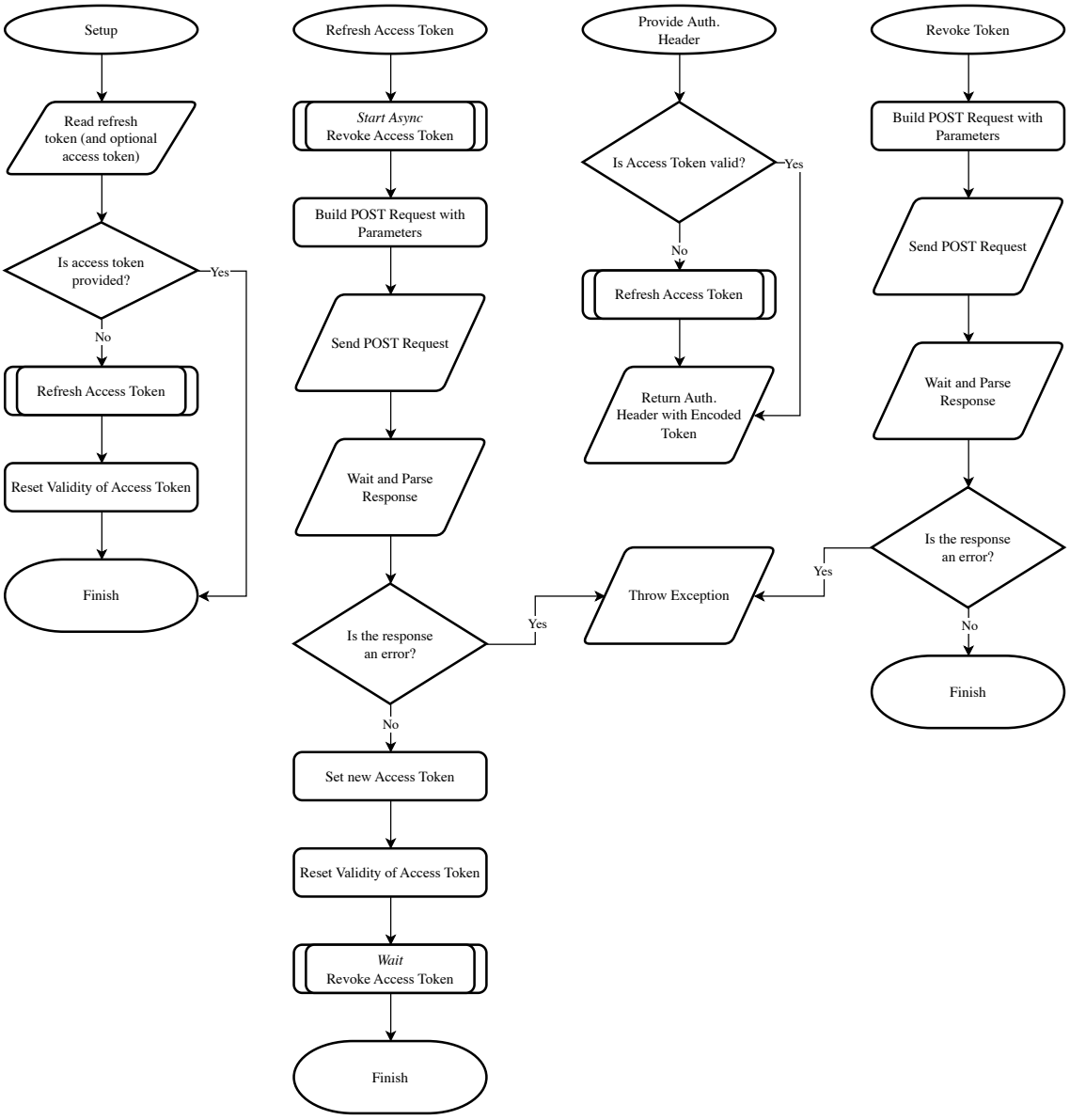


Figure 2.19: Flowchart for OAuth 2.0 Acquire Access Token using Refresh Token

2.2.2.2 HTTP Based API Requests

The base URL of all requests is `https://api.dmdata.jp/v2/` which will be indicated as `base://` from now on.

Table 2.8 shows the necessary permissions would be necessary for the application to function. How each of them functions will be discussed below.

Permission Code	Permission Details
`contract.list`	Get the list of subscriptions the user has.
`gd.earthquake`	Get list of past earthquakes.
`parameter.earthquake`	Get details of observation points for earthquake intensities.
`socket.start`	Start a new WebSocket connection.
`socket.list`	Get list of existing WebSocket connections.
`socket.close`	Close an existing WebSocket connection.
`telegram.data`	Get specific telegram released by the JMA.
`telegram.get.earthquake`	Allows program to access telegrams on earthquake information.
`eew.get.forecast`	Allows program to access telegrams on EEW forecasts (including warnings).

Table 2.8: Necessary access permissions for DM-D.S.S.

Standard Return Information and Errors There are two status that an API call could return: a successful `ok` status, or an unsuccessful `error` status.

Listing 2.3 shows the JSON for a successful `ok` response, and Listing 2.4 shows the JSON for a `error` response.

```
{
    "responseId": "66d23c0cede77d82",
    "responseTime": "2021-04-01T00:00:00.000Z",
    "status": "ok"
}
```

Listing 2.3: `ok` status for API

```
{
    "responseId": "66d23c0cede77d82",
    "responseTime": "2021-04-01T00:00:00.000Z",
    "status": "error",
    "error": {
        "message": "Authentication required.",
        "code": 401
    }
}
```

Listing 2.4: `error` status for API

In both responses, there is an attribute `responseId` which gives the unique ID for each response as a `string`, and an attribute `responseTime` which gives the date and time of response in `ISO8601Time` format.

The `status` attribute in `string` indicates whether it is a successful response (in which case it will be `"ok"`) and if it is an error response (in which case it will be `"error"` and gives the relevant HTTP error as well). The error will be indicated in the object `error` and will give the relevant message and code.

The list of standard errors together with their meanings is discussed in Table 2.9.

The only error that might occur in the application is `401` authorisation required, in the case when the OAuth 2 token expired, or the user inputted an incorrect API Key.

Error Code	Error Message
400	The query parameters are required.
400	The post parameters are required.
400	Unexpected data of search query `cursorToken`.
401	Authorisation required.
403	Insufficient scope for ....
403	Requests are not allowed.

Table 2.9: Standard errors for API.

From here onwards, the `responseId`, `responseTime` and `status` will be removed from the sample response by default.

Cursor Token For some API data calls, and specifically for `telegram.list` that we are going to use, there is too much data to be returned in one API call. Therefore, in a response, there will be a specified attribute named `nextToken` indicated with type `string`, sometimes as well as `nextPooloing` and `nextPoolingInterval`, as shown in Listing 2.5 as an example call of

`base://telegram?type=VXSE53`.

```
{
    "responseId": "8359288359fc5bb9",
    "responseTime": "2021-04-01T00:00:00.000Z",
    "status": "ok",
    "items": [
        {},
        {},
        {}
    ],
    "nextToken": "bmV4dCAgICAgICAgNTc0MzI",
    "nextPooling": "cG9sbGluZyAgICAgNzM0MzE",
    "nextPoolingInterval": 1600
}
```

Listing 2.5: Cursor token sample JSON.

Specifically, the calls for `socket.list` and `gd.eew` will only return a `nextToken` for the next call, while for `telegram.list` and `gd.earthquake` it will return a `nextPooling` as well.

If one would like to do the next call to find the next few items of the list, the `cursorToken` parameter should be specified as the same as the `nextToken` or `nextPooling`, with all the rest of the parameters identical to the previous request, i.e.,

`base://telegram?type=VXSE53&cursorToken=bmV4dCAgICAgICAgNTc0MzI`

if the `nextToken` is used, and

`base://telegram?type=VXSE53&cursorToken=cG9sbGluZyAgICAgNzM0MzE`

if `nextPooling` is used (which should be after `nextPoolingInterval` in milliseconds).

The document suggested that, due to performance issues, the `nextPooling` should be used wherever possible. Note that the `nextToken` and `nextPooling` will change for every single call of the API.

From here onwards, in the API sample results, `nextToken`, `nextPooling` and `nextPoolingInterval` will not be included.

Contract API There is only one contract-related API, which is `contract.list`. It is an HTTP GET request, on `base://contract` with no parameters.

It will return a list of contracts (subscriptions) of DM-D.S.S., whether or not the user has subscribed to it.

```
{
    "items": [
        {
            "id": 92,
            "planId": 1,
            "planName": " 地震・津波関連",
            "classification": "telegram.earthquake",
            "price": {
                "day": 15,
                "month": 350
            },
            "start": "2021-01-01T01:01:00.000Z",
            "isValid": true,
            "connectionCounts": 1
        }
    ]
}
```

Listing 2.6: Contract list sample JSON.

Listing 2.6 shows a sample return of this API call.

Here, `items` is the list of contracts (subscription plans), with each of its element being an object representing a plan. For each of them, there are the properties:

- `id` (nullable) which stands for the subscription ID (could be `null` if the user is not subscribed);
- `planId` which stands for the subscription plan ID, which is unique for each plan;
- `planName` which stands for the name of the subscription plan;
- `classifications` which stands for the classification code of the subscription plan;
- `price.day` and `price.month` which stands for the price of the plan per day/per month;
- `start` (nullable) which stands for the start of the subscription;
- `isValid` which is a `boolean` to represent whether the plan is valid; and
- `connectionCounts` which stands for the number of extra WebSocket connections this subscription plan provides.

Note that how `id` and `start` can be `null` since it lists all the subscriptions, whether or not the user has been subscribed to it.

This API will be used to calculate the number of WebSocket connections, to display empty WebSocket slots in the listing WebSocket functionality.

WebSocket APIs There are three WebSocket-Related APIs, specifically, `socket.start`, to start a socket, `socket.list` to list all sockets, and `socket.close`, to close a socket. They are all called on `base://socket`.

Socket Start `socket.start` is a POST method which has a request body in JSON, as shown in Listing 2.7.

The attributes are:

- `classifications`, which is a list of classifications of the subscription plans (which include all the types below);
- `types` (optional), which is a list of telegram wished to be transmitted through the WebSocket;
- `test` (optional, default to `"no"`), which indicates whether test telegrams should be received (`"including"`) or not (`"no"`);

```
{
    "classifications": [
        "eew.forecast",
        "telegram.earthquake"
    ],
    "types": [
        "VXSE45",
        "VXSE51",
        "VXSE52",
        "VXSE53"
    ],
    "test": "including",
    "appName": "Application Test",
    "formatMode": "json"
}
```

Listing 2.7: Socket start sample request JSON.

- `appName` (optional), which is the name of the application to be recorded with the WebSocket; and
- `formatMode` (optional, default to `"raw"`), which is either `"json"` or `"raw"` depending on the desired return type.

The detailed classifications and types will be discussed in the next section on WebSocket connections. When the `types` is excluded, the WebSocket will return all the telegrams included in the classification.

It will have a response as in Listing 2.8.

Some attributes are exactly the same as the JSON sent has header, `classifications`, `types` (nullable), `test` and `appName`. The additional attributes are:

- `ticket`, which is a code in `string` for the connection. This is also included in the `websocket` object.
- `websocket`, which is the object containing details for the connections. Its attribute `id` is unique for each connection and `url` is the WebSocket (`wss://`) URL to use to connect to. `protocal` being a constant array containing `"dmdata.v2"` indicating that this is the 2nd version of the protocol. `expiration` is in seconds the time the connection will expire if no transmission handshakes are made (see below), and is always `300` as constant.
- `formats`, which is a list of formats used to encode the data transmitted in the WebSocket. This is a constant list including `"xml"`, `"a/n"` and `"binary"` if `formatMode` is set to `"raw"`, and a list with only `"json"` if set to `"json"` in the request JSON.

Apart from the standard errors, it will also output errors shown in Table 2.10

Error Code	Error Message
400	The body of the request is not JSON.
400	At least one element of `classifications` is required.
400	The `types` is not a string or has more than 30 elements.
400	The `appName` is up to 24 bytes.
400	You have entered a string that is not defined in `formatMode`.
402	No contract.
409	The maximum number of simultaneous connections is exceeded.

Table 2.10: Errors for `socket.start`.

The only error that should occur in the application is the final one, where the maximum number of simultaneous connections is exceeded. To allow the user to deal with this situation, a disconnect button for each connection in the current WebSocket list is provided.

```
{
    "ticket": "Tik....",
    "websocket": {
        "id": 0,
        "url": "wss://ws003.api.dmdata.jp/v2/websocket?ticket=Tik....",
        "protocol": [
            "dmdata.v2"
        ],
        "expiration": 300
    },
    "classifications": [
        "eew.forecast",
        "telegram.earthquake"
    ],
    "test": "including",
    "types": [
        "VXSE45",
        "VXSE51",
        "VXSE52",
        "VXSE53"
    ],
    "formats": [
        "xml",
        "a/n",
        "binary"
    ],
    "appName": "Application Test"
}
```


Listing 2.8: Socket start sample response JSON.

Socket Close `socket.close` is a DELETE method, which specifies the ID of the WebSocket port to be closed in the URL parameter `:id`, e.g., `base://socket/30`. If the result is successful, the request will not return anything. If the status is error, it will return a standard error (detailed above), or a `404` error, which indicates that the specified WebSocket `id` is not found.

There is a chance of a `404` error for WebSocket ID not found, if the list has not been refreshed, and the user attempts to disconnect to the WebSocket which was already disconnected. Due to this, the refresh WebSocket list button is provided should such error occur.

Socket List `socket.list` is a GET method which supports `cursorToken` (optional) as discussed above. There are other parameters in the query:

- `id` (optional), which stands for the ID of the WebSocket connection that details is wished to be retrieved;
- `status` (optional), a `string` indicating the status of the WebSocket, including `open` which stands for operating live, `waiting` which means idle, and `closed` standing for closed connections;
- `limit` (optional, default 20), of type `integer` with a maximum of 100, indicating the number of WebSockets listed in a single response.

Listing 2.9 shows a response from the call. The `items` is a list of objects, each representing a WebSocket, with the following properties:

- `id`, `ticket`, `classifications`, `test`, `types`, `formats`, `appName`, identical to described above;
- `start`, a `ISO0601Time` representing the time of the start of the connection;
- `end`, a nullable `ISO0601Time` representing the time of the end of the connection;
- `ping`, a nullable `ISO0601Time` representing the previous time of ping-pong;
- `ipAddress`, a nullable `string` representing the IP address of the source of connection;
- `server`, a nullable `string` representing the connected WebSocket server; and
- `status`, a `string` representing the connection status.

However, it is worth noting that even though the document claims that `ticket` is not `null`, it will only be not null when the WebSocket is waiting connection, and null otherwise. Furthermore, `formats` never appears in the result, even though the document claims that it should always appear.

These API calls will be used to manage (open new, list existing and close) WebSockets in the application.

Parameter API The program would need to have a list of observation points of earthquakes to plot on the map the observed stations' intensity on the past-page map. Therefore, we would need to call `parameter.earthquake`, which is available at `base://parameter/earthquake/station`.

This is a GET request with no parameters to specify, and will return all stations in one go (which is time-costly). Listing 2.10 shows a sample response.

The `changeTime` and `version` of this list of parameters is often the same for calls since the list is rarely updated, and they give the version of the list and the last time of the update.

For each object in `items`, they represent an earthquake observation point. The attributes are:

- `region` and `city`, both containing `code`, `name` and `kana` (Kana (カナ, 仮名) represents the pronunciation), which is unique for each region and city;
- `noCode`, the unique code, and `code`, the code used in XML (WebSocket data feeds);
- the `name` and `kana`, which are not necessarily unique for each observation point;
- `status` standing for whether the station is in use. `" 現"` (now) means the data hasn't changed in the update, `" 変更"` (change) means the data has been altered, `" 新規"` (new) stands for a new observation point, and `" 廃止"` (abolished) means the observation point has been discontinued;
- the `owner`, representing the owner of the observation point; and

```
{
    "items": [
        {
            "id": 0,
            "ticket": "Tik....",
            "types": [
                "VXSE45",
                "VXSE51",
                "VXSE52",
                "VXSE53"
            ],
            "test": "including",
            "classifications": [
                "eew.forecast",
                "telegram.earthquake"
            ],
            "ipAddress": "192.168.0.0",
            "status": "open",
            "server": "websocket-03",
            "start": "2021-04-01T00:00:00.000Z",
            "end": null,
            "ping": "2021-04-01T00:00:00.000Z",
            "appName": "ApplicationTest"
        }
    ]
}
```


Listing 2.9: Socket list sample response JSON.


```
{
    "changeTime": "2024-07-18T12:00:00+09:00",
    "version": "20240718",
    "items": [
        {
            "region": {
                "code": "100",
                "name": " 石狩地方北部",
                "kana": " イシカリチホウホクブ"
            },
            "city": {
                "code": "0123500",
                "name": " 石狩市",
                "kana": " イシカリシ"
            },
            "noCode": "1000000",
            "code": "0123500",
            "name": " 石狩市花川",
            "kana": " イシカリシハナカワ",
            "status": " 現",
            "owner": " 気象庁",
            "latitude": "43.1714",
            "longitude": "141.3156"
        }
    ]
}
```


Listing 2.10: Earthquake parameter sample response JSON.

- the `latitude` and `longitude`, which as the name suggests, represent the position of the observation point.

Due to the nature of JSON being very long and barely updated (an email will be sent to users of DM-D.S.S. every time its updated in fact), it will be obtained upon application launch/authentication by the user, and stored in the main memory for the lifetime of the application, rather than obtaining such every time when necessary.

Past Earthquake API The API `gd.earthquake` allows us to use a GET request to obtain a series of earthquake on `base://gd/earthquake`, with the option to obtain a list, or to obtain only a certain event.

Past Earthquake List The API call `gd.earthquake.list`, which supports `cursorToken` and `poolingToken` allows us to obtain a list of past earthquakes, with the ability to specify the following parameters:

- `hypocenter`, the 3-digit code to filer the hypocentre position;
- `maxInt`, the lower bound of the filter for the maximum measured intensity of the earthquake;
- `date`, the date when the earthquake is observed;
- `limit`, the number of earthquakes returned in one go, with a maximum of 100.

A sample response is shown in Listing 2.11.
These APIs will be used to display a list of past earthquakes. For each object representing an earthquake, this includes:

- Its unique `id`, `eventId`, `originTime` (if it is not a report on the intensity), and `arrivalTime`;
- its `type` (which is `normal` for earthquakes in Japan, and `distant` for earthquake reported by other countries);
- its `hypocenter`, including the `code`, `name`, the `coordinate` and the `depth`;
- its `magnitude` (an object representing units and values); and
- its `maxInt`, the maximum intensity observed.

It might also include LPGM information, in `maxLgint` and `lgCategory`, if such information is released, usually only for huge earthquakes.
The data used will be used to create the sidebar (list of past earthquakes) of the past page in the application, and will be called upon the click of refresh button/load more button.

Past Earthquake Event If the call `base://gd/earthquake/:eventId` is used, it will return the object representing the earthquake in the `event` attribute (most attributes removed since identical to Listing 2.11), as well as a list of telegrams in the `telegrams` attribute, which was detailed above in the `telegram.list` section, as shown in Listing 2.12.
In the `telegrams` list, each object represents a telegram, with certain properties:

- `serial` stands for the serial ID of this transmission, while `id` is a unique ID for the JSON telegram (non-serial);
- `originalId` gives the original ID of the telegram in XML format;
- `classification` gives the classification of this telegram. For the purpose of this application, all classification will be of type `telegram.earthquake`;
- `head` gives the head information of this telegram, including the `type`, the `author`, the `time`, and whether it is `test`;
- `receivedTime` gives the time of this telegram being released,
- `xmlReport` details parts of information encoded in the XML;

```
{
    "items": [
        {
            "id": 1584,
            "type": "normal",
            "eventId": "20210808085414",
            "originTime": "2021-08-08T08:54:00+09:00",
            "arrivalTime": "2021-08-08T08:54:00+09:00",
            "hypocenter": {
                "code": "787",
                "name": " 鹿児島湾",
                "coordinate": {
                    "latitude": {
                        "text": "31.3°N",
                        "value": "31.3000"
                    },
                    "longitude": {
                        "text": "130.6°E",
                        "value": "130.6000"
                    },
                    "height": {
                        "type": " 高さ",
                        "unit": "m",
                        "value": "0"
                    },
                    "geodeticSystem": " 日本測地系"
                },
                "depth": {
                    "type": " 深さ",
                    "unit": "km",
                    "value": "0",
                    "condition": " ごく浅い"
                }
            },
            "magnitude": {
                "type": " マグニチュード",
                "unit": "Mj",
                "value": "2.6"
            },
            "maxInt": "2"
        }
    ]
}
```


Listing 2.11: Past earthquake list sample response JSON.

```
{
    "event": {
        "telegrams": [
            {
                "serial": 0,
                "id": "...",
                "originalId": "...",
                "classification": "telegram.earthquake",
                "head": {
                    "type": "VXSE53",
                    "author": "RJTD",
                    "time": "2021-08-07T23:58:00.000Z",
                    "designation": null,
                    "test": false
                },
                "receivedTime": "2021-08-07T23:58:10.311Z",
                "xmlReport": {
                    "head": {
                        "title": " 震源・震度情報",
                        "serial": "1",
                        "eventId": "20210808085414",
                        "headline": "　８日０８時５４分ころ、地震がありました。",
                        "infoKind": " 地震情報",
                        "infoType": " 発表",
                        "reportDateTime": "2021-08-08T08:58:00+09:00",
                        "targetDateTime": "2021-08-08T08:58:00+09:00",
                        "infoKindVersion": "1.0_1"
                    },
                    "control": {
                        "title": " 震源・震度に関する情報",
                        "status": " 通常",
                        "dateTime": "2021-08-07T23:58:08Z",
                        "editorialOffice": " 気象庁本庁",
                        "publishingOffice": " 気象庁"
                    }
                },
                "schema": {
                    "type": "earthquake-information",
                    "version": "1.0.0"
                },
                "format": "json",
                "url": "https://data.api.dmdata.jp/v1/..."
            }
        ]
    }
}
```


Listing 2.12: Past earthquake event sample response JSON.

- `schema` details the JSON schema used for the telegram, including the `type` and `version` of the schema used;
- `url` gives the URL for the detailed information of this telegram (that can be retrieved using another GET request), and `format` gives its format.

All telegrams provided in this list are in JSON format.
This API will be used to load the details for a certain earthquake to display on the sidebar panel.

2.2.2.3 WebSocket Connections

WebSocket connections are essential to real-time applications, since it provides a real-time feed from the server to the client on latest data, without the need of the client to query (which consumes a lot of resources). DM-D.S.S. provides WebSockets to initiate real-time connection to the server to achieve the latest telegrams and earthquake warnings.

The URL of the WebSocket will be `wss://{#1}.api.dmdata.jp/v2/websocket`, where the `#1` part could be `ws001` to `ws004`, but it is optional. If not included, the server will simply distribute the load.

WebSocket Responses There are four types of responses that a WebSocket could return, indicated in the `type` attribute:

- a `start`. This only happens when the WebSocket is started, and this includes the exact same information in the `socket.start` call in Listing 2.8, with an additional `time` to indicate the time of the response;
- a (server-side initiated) `ping`. This will happen regularly, and the server will send a JSON in the format in Listing 2.13.

```
{
    "type": "ping",
    "pingId": "012345"
}
```

Listing 2.13: WebSocket Ping JSON.

Upon receiving, the client should return a `pong` JSON in the same format, with the same `pingId`, as in Listing 2.14.

```
{
    "type": "pong",
    "pingId": "012345"
}
```

Listing 2.14: WebSocket Pong JSON.

This is in place to ensure that there are no 'dead' WebSocket connections, and the server will close the WebSocket connection after 120s of no response in such ping-pong calls, to not take up the limited number of WebSocket connection slots;

- a client-side initiated `ping` as in Listing 2.13, which the server will return a `pong` as in Listing 2.14. This works almost identically as before, apart from that such `pingId` is optional. This is in place to ensure that the application would be able to know if the connection is broken, for example due to unstable internet connections, and would be able to restart another WebSocket connection;
- a `error`, which will be the format in Listing 2.15. There will also be a boolean, `close`, to indicate whether the WebSocket connection is closed or not.

  A table of possible WebSocket errors is included in 2.11, with the relevant WebSocket error codes.

```
{
    "type": "error",
    "error": "Server error.",
    "code": 4503,
    "close": true
}
```

Listing 2.15: WebSocket Error JSON.

Error Code	Error Details
4400	Compulsory parameters are not specified when starting WebSocket.
4404	Ticket is not specified when starting WebSocket.
4409	Number of connection limit is reached.
4503	Internal server error.
4640	`pingId` does not match for a server-initiated `ping`.
4641	Incorrect data (non-JSON) received from client.
4808	Client requested closure of WebSocket.
4807	Contracts are discontinued in connection.

Table 2.11: Errors for WebSocket connection.

In the application, only `4409` (when the connection number exceeded the maximum), `4503` (when the server has an internal error), `4808` (when the client requested closure of WebSocket), and `4807` (contract discontinued) should occur in the application.

The program should be designed to be able to handle these errors, and to close the connection if necessary;

- a `data`, an example shown in Listing 2.16.

  This will include a `classification`, a version `version`, and a unique `id`. A list named `passing` gives the `name` and `time` of each passing location of this data piece inside the server.

  Furthermore, the `head` and `xmlReport` gives details of the telegram received, similar to in Listing 2.12.

  The `format`, `compression` (`gzip`, `zip` or `null`) and `encoding` (`base64` or `utf-8`) gives information on how the `body` should be processed and interpreted.

  Note that the body should always be dealt with in the following sequence: decode; decompress; and deserialise.

  This gives the key piece of information that will be passed between the data component of the software and the UI component of the software.

JSON v.s. XML Most data nowadays on the internet are serialised in JSON, but the JMA provides its data in XML format. DM-D.S.S. does also provide JSON formatted schemas for most of the XML data files which are also processed real time. Even though there is a high chance of delay (up to 1 second) if the JSON format is used, not to say the potential of an error in processing which lead to undefined behaviour in the JSON format, the JSON format is much easier to parse and deserialise into C# DTO objects, and the schemas are very well-defined on the DM-D.S.S. references [16], in contrast to the JMA documentation [74] which is in Japanese (which the author is not an expert in). Therefore, JSON format telegrams and data will be used in the application.

Design of WebSocket Wrapper .NET has a built-in sealed `ClientWebSocket` class, and therefore to use it, we would have to wrap a class around it (i.e. a class that composites `ClientWebSocket` class), since it is responsible for the life-cycle of the WebSocket as well. The wrapper is re-used over multiple connections, while the backing `ClientWebSocket` class is disposed per-use.

Events are used here (Data Received and Status Changed) to use callbacks to send back the data to the necessary component, such as the View Model for the GUI (introduced later). Furthermore, the FIFO data structure queue is used to store pings sent by the client, since a later ping received would

```
{
    "type": "data",
    "version": "2.0",
    "classification": "telegram.weather",
    "id": "123456789abcdef...",
    "passing": [
        {
            "name": "websocket-01",
            "time": "2020-01-01T00:00:00.120Z"
        }
    ],
    "head": {
        "type": "VPWW54",
        "author": "JPTD",
        "time": "2020-01-01T00:00:00.000Z",
        "test": false,
        "xml": true
    },
    "xmlReport": {
        "control": {
            "title": " 気象警報・注意報（Ｈ２７）",
            "dateTime": "2020-02-27T00:00:00Z",
            "status": " 通常",
            "editorialOffice": " 気象庁本庁",
            "publishingOffice": " 気象庁予報部"
        },
        "head": {
            "title": " 東京都気象警報・注意報",
            "reportDateTime": "2020-02-27T09:00:00+09:00",
            "targetDateTime": "2020-02-27T09:00:00+09:00",
            "eventId": null,
            "serial": null,
            "infoType": " 発表",
            "infoKind": " 気象警報・注意報",
            "infoKindVersion": "1.2_1",
            "headline": " 注意報を解除します。"
        }
    },
    "format": "xml",
    "compression": "gzip",
    "encoding": "base64",
    "body": "H4sIAAAAAAAAA..."
}
```


Listing 2.16: WebSocket Data JSON.

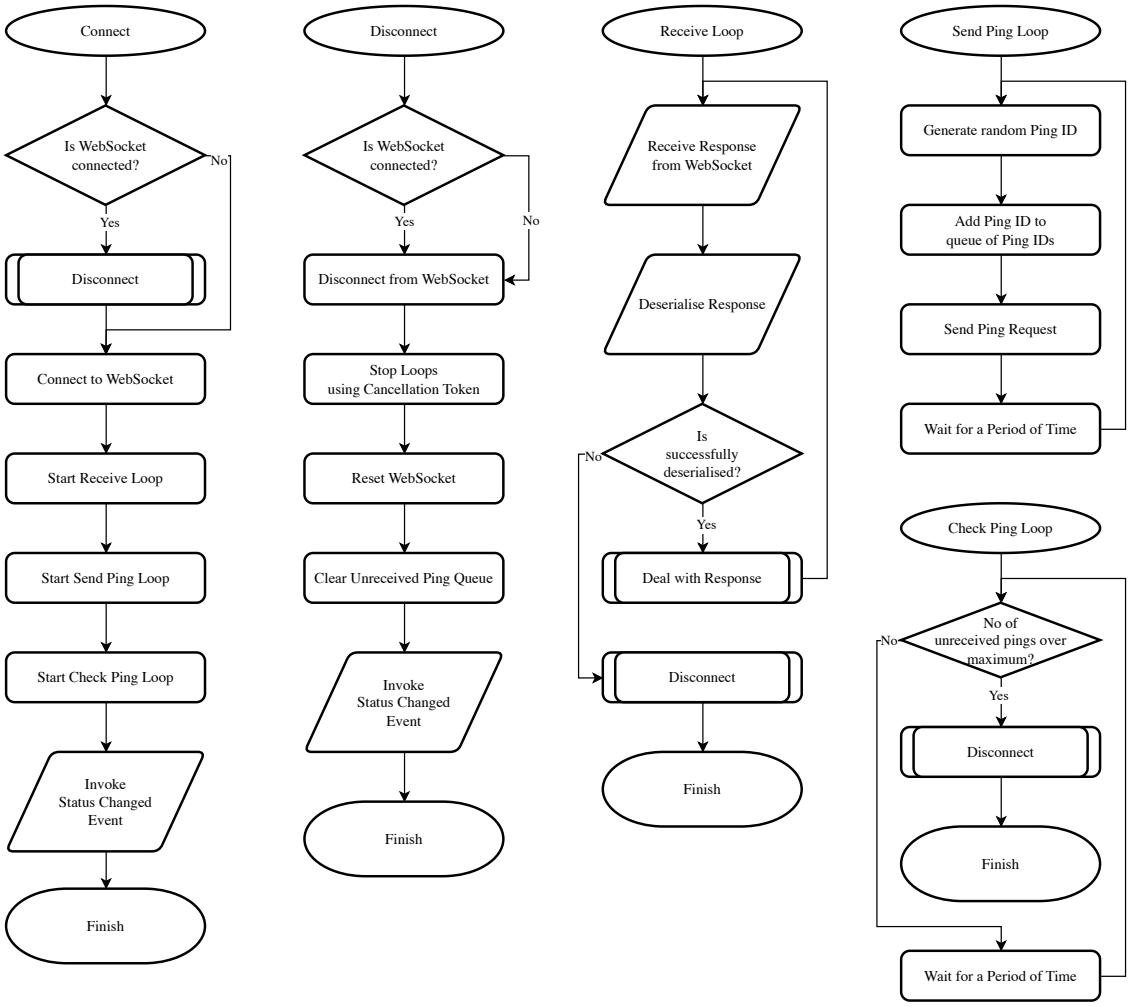


Figure 2.20: Flowchart of WebSocket Wrapper

indicate that the connection is still alive, and the ping, together with all pings before that ping (and including that ping itself), could be removed from the queue.

The overall flow structure is shown in Figure 2.20 for the overall functionality of the WebSocket flowchart, and in Figure 2.21 for dealing with the received response.

2.2.2.4 Telegrams

As discussed in the previous section, JSON telegrams is going to be used in the application, and there is no particular reason why telegrams in general (i.e. for the earthquake telegrams in `gd.earthquake.event` for the past-page) should be retrieved in XML format compared to JSON format. In this section we will discuss the telegrams used, and the structure of a telegram in DM-D.S.S.

Telegrams Used Considering the uses of the application, Table 2.12 gives an overview of the telegrams used in the application.

In the real-time page, VXSE45 and VTSE41 telegrams are fed via the WebSocket connection detailed above to be displayed. On the past page, VXSE51 53 telegrams are used to display the past earthquake information. In specific, the following logic will be used to determine which telegram to use: (in the following, $x = 1, 2, 3$)

- Filter the telegram to only include VXSE5x;
- Group telegrams by the telegram code;

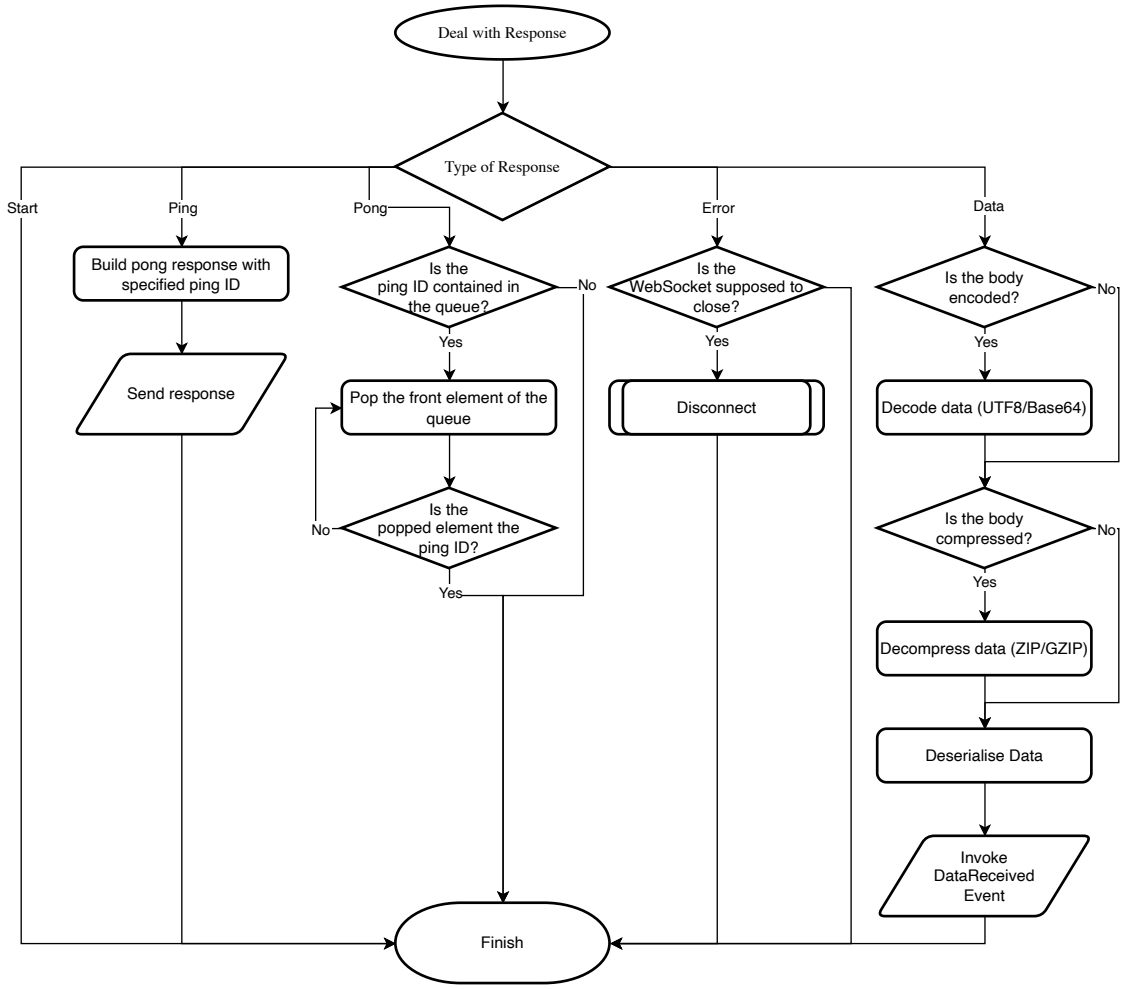


Figure 2.21: Flowchart of dealing with WebSocket Response

Code	Telegram Classification	JSON Schema	Description
VXSE45	`eew.forecast`	`eew-information`	All EEW telegrams including forecasts and warnings.
VXSE51	`telegram.earthquake`	`earthquake-information`	Preliminary report on the measured intensity of earthquake.
VXSE52	`telegram.earthquake`	`earthquake-information`	Preliminary report on the hypocentre of earthquake.
VXSE53	`telegram.earthquake`	`earthquake-information`	Full report on intensity and hypocentre of earthquake.
VTSE41	`telegram.earthquake`	`tsunami-information`	Tsunami warnings of all types.

Table 2.12: Telegrams used in the application

```
{
    "_originalId": "...",
    "_schema": {
        "type": "earthquake-information",
        "version": "1.0.0"
    },
    "type": " 震源・震度に関する情報",
    "title": " 震源・震度情報",
    "status": " 通常",
    "infoType": " 発表",
    "editorialOffice": " 気象庁本庁",
    "publishingOffice": [
        " 気象庁"
    ],
    "pressDateTime": "2021-08-12T06:05:36Z",
    "reportDateTime": "2021-08-12T15:05:00+09:00",
    "targetDateTime": "2021-08-12T15:05:00+09:00",
    "eventId": "20210812150301",
    "serialNo": "1",
    "infoKind": " 地震情報",
    "infoKindVersion": "1.0_1",
    "headline": " １２日１５時０２分ころ、地震がありました。",
    "body": {}
}
```


Listing 2.17: Head of JSON Schema

- Find the latest telegram (by serial number) in each of the category;
- Use the telegram with the biggest x (i.e. VXSE53 > VXSE52 > VXSE51).

The particular choice for the final one is due to the fact that VXSE53 usually contains the most information, while VXSE51 and VXSE52 usually contains less information than VXSE53. (They are usually released earlier than VXSE53 though).

JSON Schemas All JSON Schemas include a part called the **schema head**, and a sample is shown in Listing 2.17.

There are the following properties:

- `_originalId`, the original ID of the telegram;
- `_schema`, the DM-D.S.S. JSON Schema used for this telegram, including the type and the version;
- `type`, the name of the telegram, and `title`, the title of the telegram;
- `status`, the status of the telegram, whether it is normal (通常), a training telegram (訓練), or a test telegram (試験). Note that telegrams should be filtered in the application to only display normal telegrams to the user;
- `infoType`, the type of information, whether it is a release (発表), a correction (訂正), a delay (遅延), or a cancellation (取消). Note that in EEW telegrams, there is also an indication in the `body` part whether it is a cancellation;
- `editorialOffice`, and a list of `publishingOffice`;
- `pressDateTime` when the telegram is made, `reportDateTime` when the telegram is released;
- `targetDateTime` which is nullable, describes the base time of the report, and `targetDatetimeDubious` giving the error;

- `validDateTime` for which the report is valid until (e.g. for tsunami information, there will not be cancellation reports, only a date time the information is valid until);
- `eventId`, the unique ID for the event that this describes. This should be used to identify earthquakes in the EEW;
- `serialNo`, the number that is released for the specific event. In EEW, if a telegram with lower serial is received after a telegram with a higher serial (e.g. due to delay reasons), then the telegram should not be used and should be disposed of;
- `infoKind` and `infoKindVersion`, the schema used in the XML report;
- `headline`, which includes the summary headline of the information;
- `body`, the body of all the information.

In particular, the information of `_schema`, `status`, `pressDateTime` (for displaying the last updated time of the information), `eventId`, `serialNo` and the `body` will be useful for the application.

The `body` part of the telegram will be omitted for brevity, since they are mostly just convoluted JSON objects. Anything interesting in particular will be mentioned later in the design for the DTOs.

Text and Comments component of the body For all schemas used in the application, in the `body` component, there will be properties `text` (of type string) and `comments` which includes text and comments on the telegram.

In the `comments` property, there could always be a `free` property where any string could be recorded, and some other properties (whose names may vary: `warning` for `tsunami-information` and `eew-information`, and `forecast` and `var` for `earthquake-information`), each containing a list of strings and codes which are paired, where the code is a 3-digit code representing the code of the comment.

An example of this is shown in Listing

```
{
    "body": {
        "comments": {
            "forecast": {
                "text": " 津波警報等（大津波警報・津波警報あるいは津波注意報）を発表中で
                ↪  す。\n この地震について、緊急地震速報を発表しています。",
                "codes": [
                    "0211",
                    "0241"
                ]
            },
            "var": {
                "text": " ＊印は気象庁以外の震度観測点についての情報です。",
                "codes": [
                    "0262"
                ]
            }
        }
    }
}
```

Listing 2.18: Comments in JSON Schema

These should be used, together with the headline property, to create the informational text to display with a telegram.

2.2.2.5 Sidenotes

It is worth noting that an existing NuGet Library, ingen084/DmdataSharp supports dealing with the DM-D.S.S. data flow and converting them to C# objects (and exceptions). However, for the purposes

of this NEA, we will implement our own way to interact with the APIs and the correlated C# DTOs. The developer of this library did also mention that it is quite purpose-built so might not be suitable for general use.

This section referred to [14, 16, 17, 19].

2.3 Algorithms

There are two key 'algorithms' that was designed for this application (in addition to the inverse-cubic for converting colours as described previously): the first one used to calculate the position of the P/S wavefronts, and the other one used to detect shake on the real-time monitoring screen.

2.3.1 Wavefront Calculation

Earthquake waves do not travel at uniform rate as they spread out due to the decrease in energy, and so there is no uniform velocity that the seismic waves travel in. We cannot do a simple equation like $s = vt$ to calculate the distance of the wavefront of a seismic wave from the epicentre.

However, JMA provides tables for the time that seismic waves take to travel a certain distance in [71], together with [75]. It will be possible for us to look up the table and plot the seismic waves at correct distance at a certain time.

Comparing the different wave travel tables that JMA provides, the most suitable one for this application will be the **JMA 2001**, since it is designed for general purpose uses for both land-based and ocean-based earthquakes, and does not take into account the effect of the altitude of the observation points. Listing 2.19 provides a preview of the first 10 lines of this file.

```
P␣␣␣␣0.000␣S␣␣␣␣0.000␣␣␣0␣␣␣␣␣␣0
P␣␣␣␣0.416␣S␣␣␣␣0.703␣␣␣0␣␣␣␣␣␣2
P␣␣␣␣0.831␣S␣␣␣␣1.403␣␣␣0␣␣␣␣␣␣4
P␣␣␣␣1.243␣S␣␣␣␣2.099␣␣␣0␣␣␣␣␣␣6
P␣␣␣␣1.650␣S␣␣␣␣2.786␣␣␣0␣␣␣␣␣␣8
P␣␣␣␣2.052␣S␣␣␣␣3.465␣␣␣0␣␣␣␣␣10
P␣␣␣␣2.448␣S␣␣␣␣4.135␣␣␣0␣␣␣␣␣12
P␣␣␣␣2.838␣S␣␣␣␣4.794␣␣␣0␣␣␣␣␣14
P␣␣␣␣3.221␣S␣␣␣␣5.444␣␣␣0␣␣␣␣␣16
P␣␣␣␣3.600␣S␣␣␣␣6.084␣␣␣0␣␣␣␣␣18
```

Listing 2.19: JMA 2001 Wave Travel Tables

The table is a lookup table from horizontal distance to time in terms of the depth of the epicentre and the type of wave (P or S) to the time needed to travel such distance. Each row represents the P-Wave, time taken for the P-wave to travel (in seconds), S-wave, time taken for the S-wave to travel (in seconds), depth of epicentre (in kilometres), distance from epicentres (in kilometres).

This file will be read by the program upon launch every time and stored in the active memory when the program is running.

2.3.1.1 Format of Rows

The format of the table is as follows:

- **Column 1.** The wave (P);
- **Column 2.** Whitespace;
- **Column 3-10.** The time in seconds, with exactly 4 characters (whitespace padding to the left) before the decimal point, and 3 after;
- **Column 11.** Whitespace;
- **Column 12.** The wave (S);

- **Column 13.** Whitespace;
- **Column 14-21.** The time in seconds, with exactly 4 characters (whitespace padding to the left) before the decimal point, and 3 after;
- **Column 22.** Whitespace;
- **Column 23-25.** The depth of the epicentre in kilometres, with spaces padding to the left;
- **Column 26-27.** Whitespace;
- **Column 28-32.** The distance from the epicentre in kilometres, with spaces padding to the left.

Therefore, the following regular expression is designed to match each row (with the ânchor at the start, and $ anchor at the end):

```
P ((?=.{4}) *\d+)\.\d{3} S ((?=.{4}) *\d+)\.\d{3} ((?=.{3}) *\d+)  ((?=.{5}) *\d+)
```

Most part of this is quite straightforward, but it is worth explaining what `((?=.{x}) *\d+)` means here. The `(?=.{x})` is a positive lookahead looking x digits afterwards, but it is not really part of the match. The pattern `*\d+` is the pattern that this lookahead should match to. Therefore, `((?=.{x}) *\d+)` matches to a number that has left padding to reach exactly x digits, which is exactly what is required for the numbers in the time, depth and distance column, providing the matching for the padding.

2.3.1.2 Format of Table

The depth and the distance from epicentre is provided in the intervals specified in 2.13. They are provided for distances up to 2000 kilometres and depths up to 700 kilometres.

Distance	Interval for Depth	Interval for distance
0 ~ 50	2	2
50 ~ 200	5	5
200 ~ 700	10	10
700 ~ 2000	–	10

Table 2.13: Distance intervals in JMA 2001.

The table is ordered in increasing order of depth, then by distance. This means that there are

$$\frac{50}{2}+\frac{200-50}{5}+\frac{700-200}{10}+\frac{2000-700}{10}+1=25+30+50+130+1=236$$

rows corresponding to each depth.

2.3.1.3 Algorithm of Finding Distance from Time and Depth

However, the table gives times based on distance and depth, but this does not provide us with the distance we would like, which is drawing a circle based on how long it has been since the earthquake has happened – the other way around.

Luckily, for a fixed depth, the list is sorted since the distance is always increasing, and so is the time. Therefore, after fixing the region of rows for the given depth, we could apply a linear search (a binary search would not make much of a difference here since there are only so few rows for each depth), and find the neighbouring two times such that the current time elapsed from the earthquake happening is between them. After doing this, a linear interpolation will be done on the neighbouring time-distance (t, d) pairs to find the accurate distance (radius) corresponding with the time and the depth.

The algorithm is outlined in Algorithm 2.

Algorithm 2 Algorithm for Finding Distance based on Time

Require: $0 \leq d \leq 700, d \in \mathbb{Z}$ ▷ *Depth of earthquake*
Require: $0 < t, t \in \mathbb{R}$ ▷ *Time of travel*
Require: $i \in \{0, 1\}$ ▷ *Index of time for the particular wave (0 for P, 1 for S)*
  $r \leftarrow 236$ ▷ *Number of rows per depth*
  ▷ *Find Starting Line* ◁
  if $d \geq 0$ **and** $d \leq 50$ **then**
    $s \leftarrow d/2 * r$
  else if $d \geq 50$ **and** $d \leq 200$ **then**
    $s \leftarrow 25 + (d - 5)/5 * r$
  else if $d \geq 200$ **and** $d \leq 700$ **then**
    $s \leftarrow 55 + (d - 200)/10 * r$
  else
    return
  end if
  $e \leftarrow s + r$ ▷ *End Line is exclusive*
  $l \leftarrow s$ ▷ *Index of Left Point used for Linear Inter/Extrapolation*
  if $T[s][i] < t$ **then**
    ▷ *Time is shorter than first time (i.e. seismic wave hasn't reached surface)* ◁
    return 0
  else if $T[e - 1][i] > t$ **then**
    ▷ *Time is longer than final time* ◁
    $l \leftarrow e - 2$
  else
    ▷ *Find the time before the given time* ◁
    while $l < e$ **and** $T[l][i] \leq t$ **do**
    end while
    $l \leftarrow l - 1$
  end if
  ▷ *Coordinates for Linear Interpolation* ◁
  $x_1 = T[l][i]$
  $y_1 = T[l].R$
  $x_2 = T[l + 1][i]$
  $y_2 = T[l + 1].R$
  ▷ *Linear Interpolation* ◁
  return $y_1 + (t - x_1) * (y_2 - y_1)/(x_2 - x_1)$

2.3.2 Shake Detection

As discussed in the analysis section, one of the common features of applications with real-time intensity display map is the shake detecting feature. Specifically, if the measured intensities within a certain region shows an increasing tendency, the application will display a box around that area to indicate there is a chance that the earthquake will happen.

However, there is a chance that certain observation points will malfunction and suddenly show a big increase in intensity, which does not indicate an earthquake, and we would like to prevent this from happening. Furthermore, we wish to be able to distinguish between a big earthquake and a small earthquake on the display when such shake event has occurred.

It is worth thinking how human would see an earthquake from the real-time intensity map. First, we would notice a certain observation point has an increase in the intensity for a certain observation point. Then, we will naturally look at the observation points near it, and see whether they also have an increase, or if that was just an anomaly. If the points nearby also has increasing intensity, then we would think it is probably an event, and we will look at the observation points near that, etc. If after a certain time period, that the intensity is no longer increasing, then we may say that the shake event is over.

The algorithm used in KEVI used a similar idea, and this is the algorithm that we are going to implement in our solution. Specifically, it will create a list of observation points that are near each of the observation points (we will refer to them as neighbouring points) upon launch of the program. When there is a significant increase of the measured value of an observation point, the program will look at the neighbouring points, and if they show an increase (not necessarily significant), this will create a new event including the observation point and the neighbouring points. If, further, that the neighbouring points also observe a significant increase in intensity, then they will be added to the event, and their neighbouring points will be added to the list of neighbouring points as well.

An event could be represented by three things: the time it has happened, the observation point in the event, and the neighbouring points of those points. The neighbouring points will become observation points themselves if they have significant increase in intensity, and then their neighbouring points will become some new neighbouring points. If no such increase has been found, then after a certain time period of waiting, this event would end, and it could be seen that the primary motion of the earthquake is over.

Classification of the significance of an event is straightforward – it can be determined by the maximum intensity observed within the event. A suitable scale would be split at intensity 1 (which is which human can feel a shake), and at intensity 3 (which is when earthquake might actually cause damage), and it has to have minimum measured intensity 0.0. Most real-time measurements have negative measured intensity, so this should not lead to detection of anomalies.

Figure 2.22 shows the concept of an observation point and neighbouring points together with a flowchart of how the algorithm operates. The time constants of 10 seconds and 30 seconds are chosen to reflect the behaviour of common applications including JQuake and KEVI, but should be left to be adjustable in the code to ensure the maintainability.

This section referred to [34].

2.4 User Interface

The GUI will be designed using Avalonia, which is designed to develop cross-platform applications in .NET.

The nature of this application decided that the UI will be a graphical interface. Comparing MAUI in .NET with Avalonia, due to the fact that MAUI is more suitable for a mobile touch-based platform (that also works on desktop), while Avalonia is designed to support the desktop paradigm, the latter will be used to implement the graphical interface.

As discussed in the previous sections, there are three main pages that the program will include: the page for real-time observations, the page for viewing past earthquakes, and the page for controls and settings.

A sidebar is used to switch between the three pages.

The mock-ups of GUI Pages are demonstrated using [11].

Note that the following mock-up images are for demonstration purposes only – it is not an actual earthquake. They only include functionalities identified as 'key' in the analysis section, and therefore additional functionalities like shake-detection boxes, as well as customisation features are not included.

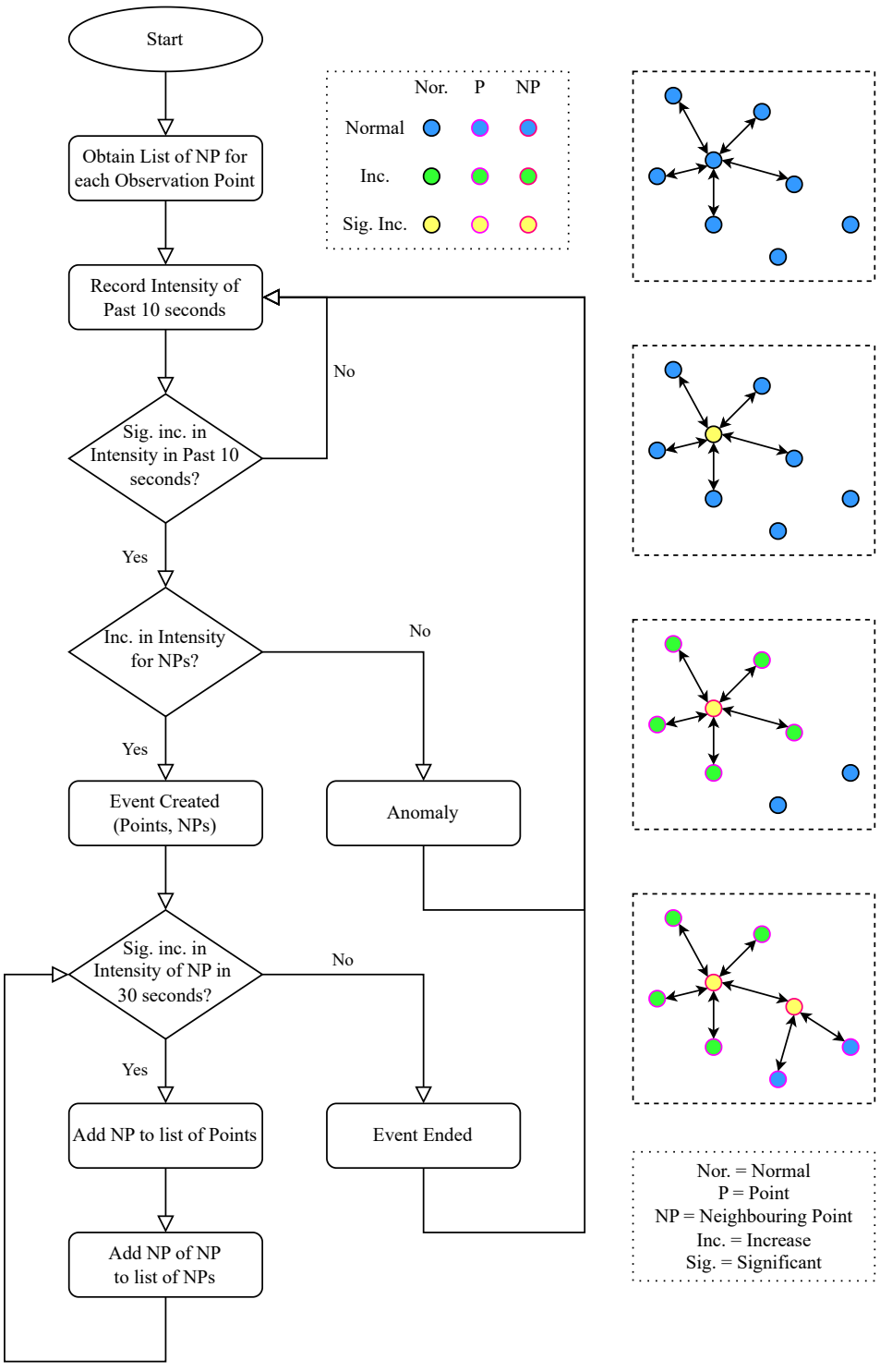


Figure 2.22: Shake detection flowchart and diagram

2.4.1 Map

The package Mapsui [43] will be used to draw the maps in the application, since it is one of the best-supported candidates in .NET.

The OpenStreetMap layer will be used as the background, first because it is free to use and supported very well, also does it allow users to see the actual map of the area of which the earthquake occurred.

To colour the regions and the shorelines, Shape Files (`.shp`) has to be used in the application. The Shape Files are sourced from the JMA webpage [72].

However, the shape files are extremely precise and accurate, tracing each and every coastline and border of the regions. However, this is not necessary for the purposes of the application (it does not make much difference zoomed out), and this will cause performance issues (loading the original shape files will cause a performance issue using more than 4 GB of RAM and will cause the application to halt).

Therefore, using the tool available on [13], the shape files are simplified using the Ramer–Douglas–Peucker algorithm, preserving 5% of the original details (which is still extremely accurate), and the line intersections are also repaired (which is a natural consequence of the algorithm). However, this algorithm is not a key part to this application, so the existing implementation on [13] is used.

2.4.2 Real-Time Monitoring Screen

The real-time monitoring screen holds the following three functionalities:

- real-time intensity display;
- tsunami warning display (including shoreline colouring); and
- EEW display (including hypocentre marking, predicted intensity colouring and real-time wavefront positions).

Figure 2.23 shows a design of this page that should be implemented.

The real-time intensity colour points, the current time and the colour scale will be displayed at all times. On the colour scale, there will be the name of the current data displayed, with the number to indicate the scale of the colour. The colour scale and time are situated in the bottom-left corner of the page.

If a EEW is received, there will be a side panel appearing in the top-left corner of the screen. It should display the level of the EEW, together with the serial of the EEW. The level of the EEW has priority follows: Cancellation > Final > Warning > Forecast. The EEW will take level the first in the list that it is in. The details of the EEW are displayed, together with the informational text in scrollable form.

Upon receiving or updating an EEW, the hypocentre of the earthquake will be marked on the map. If the hypocentre is a seismological hypocentre, a red cross will be used. However, if the hypocentre is assumed and does not seismological meaning, as discussed in the analysis section, a different symbol will be used. The author chose to use a dashed circle.

The predicted maximum intensity of regions will also be coloured, and the wavefronts as well, with the P-Wave being blue (since less destructive) and S-Wave being orange (since more destructive).

If multiple EEWs are in effect simultaneously, then the top-left panel will switch between the earthquakes at a fixed rate. However, the information on the map for both earthquakes will be displayed simultaneously.

The EEW shall be removed (and the layers on the map as well) after a certain amount of time.

The tsunami warning will be displayed only if it is released, or updated. The shorelines will be coloured in accordance of the predicted warning level of the tsunami on that particular shore line region. In the bottom-right corner, the level of the tsunami warning (which is the maximum of the tsunami warnings for each shoreline) will be displayed including its colour, and the last update time. There will be a scrollable bar for the informational text, since it is usually very long.

There are no buttons for the user to interact with. However, there will be scroll bars for users to view the informational text for the particular earthquake, and the user will be able to zoom/move the map (as supported by Mapsui).

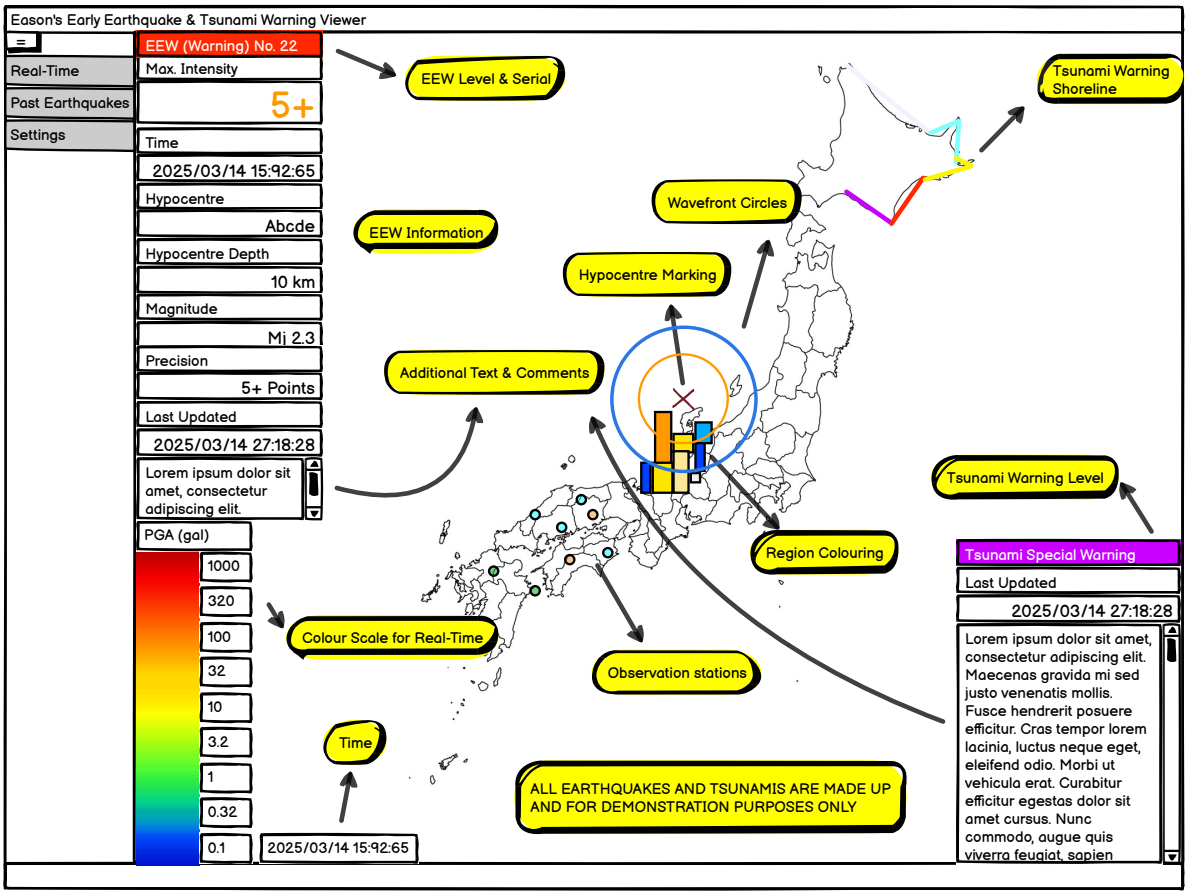


Figure 2.23: Design of GUI for Real-Time Page

2.4.3 Past Earthquakes

The past-earthquake screen regularly displays a sidebar to select the earthquake from, with a map in the middle, and on the left a panel displaying the detailed information of the selected earthquake. Figure 2.24 shows a design of this page.

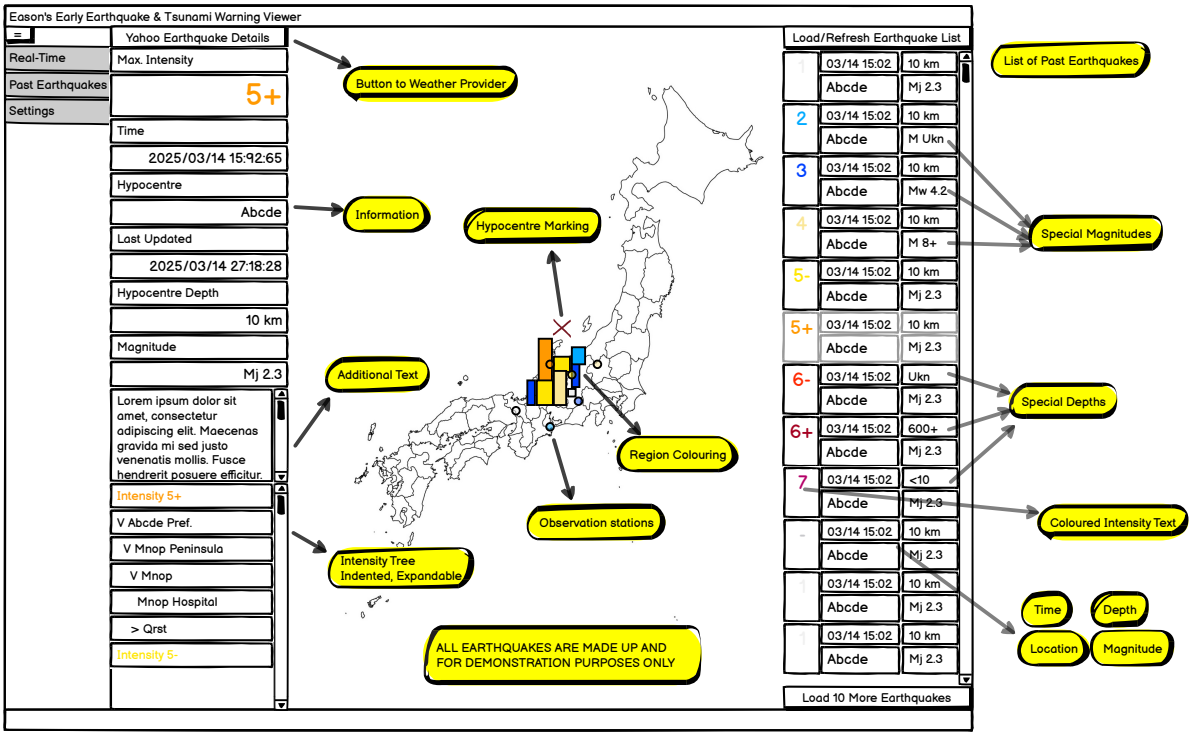


Figure 2.24: Design of GUI for Past Earthquakes Page

On the sidebar, for each earthquake, the intensity will be displayed, coloured as defined in the colour scheme. The name of the hypocentre's location, the depth, the magnitude, and the time of the earthquake is also included.

Notice that there are special cases of magnitudes, such as in different units, huge earthquakes or unknown, and special cases of depths as well, such as very deep, shallow or unknown.

Above the sidebar there is a load/refresh button, and below a button to load earthquakes. The sidebar is scrollable.

In the centre of the map, there is a red cross indicating the position of the hypocentre. Regions are coloured based on the maximum observed intensity in the region, and there are dots on the map indicating the observed intensity at that particular observation station.

On the left, the details of the selected earthquake will be displayed. On the top there is the button to link to Yahoo Earthquake Information. The information will be displayed more accurately here, together with the last updated time of the information. Then, there is the additional text and comments in scrollable form (since they could be quite long), and finally there is the intensity tree, first grouping by intensity, then by region, then by city, and finally the exact observation stations. The intensity level should be coloured in correspondence with the colour for that intensity.

The interactions the users is allowed to do on a page is to select an earthquake from the sidebar, and to click the load/refresh button to load the earthquakes. The user is also allowed to click on the Yahoo Earthquake Information button to view the information on the Yahoo Earthquake Information page. They are allowed to interact with the map (zoom/move), as well as using the scrollbars.

2.4.4 Settings Page

Figure 2.25 shows a mock-up of the design of the customisation page.

In the settings page, according to the JMA requirements, there has to be explanatory text explaining the limitations and possible errors of EEWs.

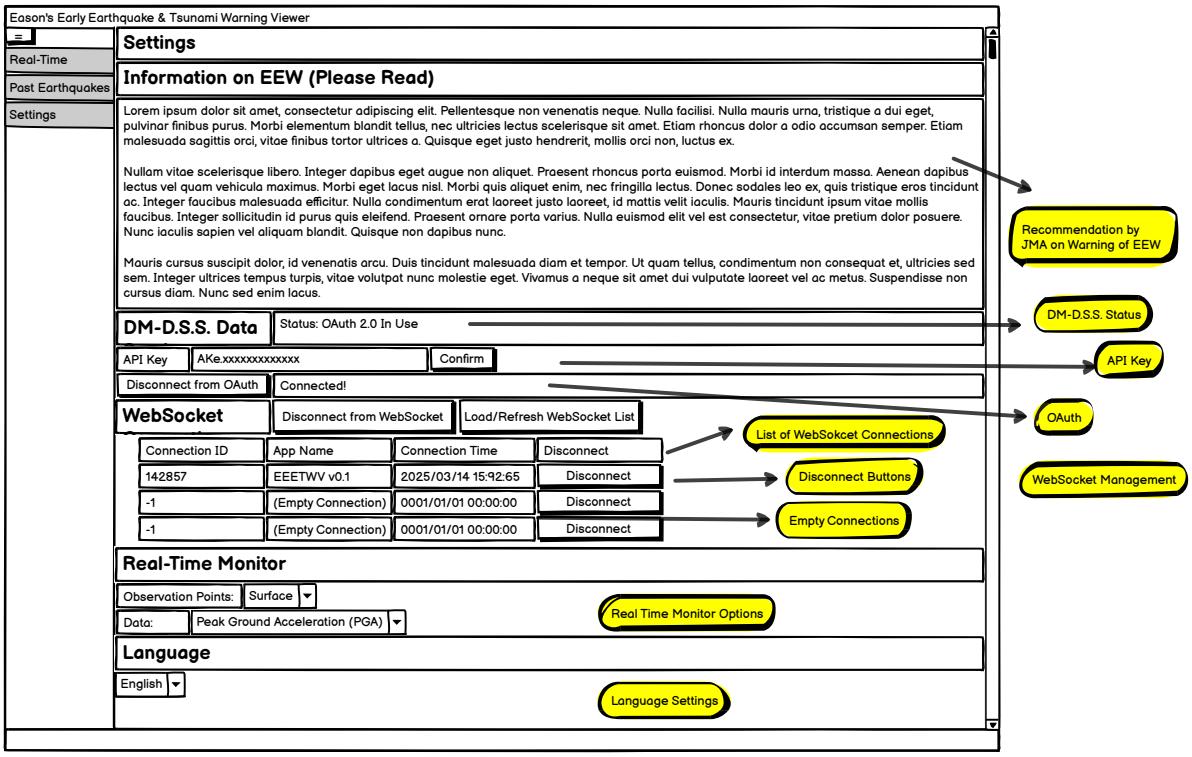


Figure 2.25: Design of GUI for Settings Page

There then should be a section to set up DM-D.S.S., where the user is allowed to enter an API Key and there should be a place to input an API Key from DM-D.S.S., or click a button to connect to OAuth 2.0.

Since OAuth 2.0 takes priority over API Key, the connect/disconnect button for OAuth is enabled all the time, allowing the user to authorise/revoke authorisation of the application.

The button to confirm the API Key should only be enabled if and only if all the following conditions are satisfied:

1. The API Key input box has changed;
2. The API Key has the correct format (begins with `AKe.`);
3. OAuth 2.0 is not in use.

Next up there is the section for the WebSocket, where a button is provided to connect to/disconnect from the WebSocket, and to load/refresh the list of WebSockets. In the list of WebSockets, the ID, App Name, and Connection Name of a connection is displayed, and a disconnect button is provided per line for each line, which will be disabled upon successful disconnection.

Finally, there are two dropdowns corresponding to the sensors and data of the real-time monitoring screen. The user should be able to select the sensors/data that they would like to view in the dropdown, and the data in the real-time page will reflect accordingly to the user's choice, including the text and the numbers on the scale.

At the end there is also a dropdown for the user to select the language of the application, which will change the language of the application, necessarily working upon restart.

2.4.5 Joint Functionalities and Technicalities

There is a joint functionality between the real-time page and the past earthquakes page, that is, when a new EEW/Tsunami Warning is released, the application should automatically jump to the real-time page to display the latest earthquake information to the user.

Since this a GUI-based application, there will be a significant use of asynchronous programming and events, which will be detailed later in the OOP section.

2.5 OOP Model

The program mainly uses the OOP (Object-Oriented Programming) paradigm, since it allows code object to be modelled like real-live objects. This is particularly useful in this application, since each service could be represented by an object (e.g. the API caller, the WebSocket connection), and each operation on it is modelled as a method.

This section will discuss some design patterns to use in the application, and will include class diagrams for modelling of services and DTOs.

2.5.1 Design Patterns

2.5.1.1 Dependency Injection and Singleton Pattern

Dependency injection is an extremely helpful tool dealing with complex classes with multiple dependencies to initialise. It would be a terrible idea for each class to instantiate the dependency themselves (e.g. the View Model for the Real-Time, Past, Setting Page instantiating their own services for API Calls, Authentications, Telegrams, etc., or the telegram fetcher and the WebSocket both each instantiating their own services to parse telegrams), since firstly it breaks the principle of single responsibility, and secondly it is performance-costly (the garbage collector has to collect garbage all the time, heap has to be assigned and de-assigned), and will lead to a lot of code duplication.

Dependency injection is achieved by sharing a **common container** of services, which .NET has a built-in `IServiceCollection` with concrete implementation `ServiceCollection`. The service container will deal with how to provide in the correct dependencies to the class that requires it. This avoids the need of complicated constructors in the code.

Singleton pattern is a pattern commonly used with dependency injection. Restricted to this application, **all services are registered as singletons**, meaning that there will only be **one** instance of each service in the application, and all services which depends on it, will **share this common instance**. Its lifetime will be managed by the container of the dependency injection (and we will not worry about it, it will get dealt with after the application stops).

The only exception to this rule is the loggers, since two loggers are involved, both implementing `ILogger`: the debug logger (by Microsoft) and my custom file logger, having different concrete implementations to the same base interface. This means they are to be registered as **enumerable** in the logger builder.

This section referred to [26], introducing the basics of dependency injection and how to use dependency injection with the Microsoft packages.

2.5.1.2 Logging

Logging is essential to the application, since it allows the developer to see what is happening, and allows the developer to deal with bugs that are reported by the users. It allows for tracking of what is happening with the application at all times.

Microsoft has a built-in `ILogger` and `ILoggerProvider` service. `ILogger` is the actual logger, and `ILoggerProvider` is a provider (factory) of the logger.

When the application is debugging, the built-in .NET Debug Logger will be used, and the logs will be logger to the output in debug. However, when the application is in production, the custom file logger will be used, and the logs will be logged to a file (a `StreamWriter` which is a file stream).

Logging is used in almost every service class of the application, unless the functionality is trivial, or any actions on the class is always visible.

This section referred to [29] introducing the Microsoft `ILogging` interface, and how to use it with dependency injection.

2.5.1.3 Adaptor Pattern

Adaptor pattern is about adapting one interface to another. Here, we will not detail in about the adaptor pattern involved between the results achieved from the API calls and the models that stores the data for the UI (this will be discussed later in the MVVM pattern).

The only two other adaptor patterns involved here, is the adaptor for logging for Avalonia and Mapsui.

Avalonia uses a very similar (but different) logging interface `ILogSink` to log, however we used the more modern `ILogSink` interface, and we will need to adapt the `ILogger` to `ILogSink`. This is achieved by

an adaptor class `AvaloniaToMicrosoftLoggingAdaptor`, which will implement the `ILogSink` interface, and aggregate a `ILogger`. The `Log` method will be implemented to call the `Log` method of the `ILogger`, and `IsEnabled` to call the `IsEnabled` method of the `ILogger` as well. The `ILogger` provided to the adaptor class will be retrieved from the dependency injection container.

On the other hand, Mapsui uses a delegate (a callback function) to achieve logging. This is relatively simpler to achieve, since we could just attach to the `Logger.LogDelegate` method of Mapsui the log delegate we want, in our case the `ILogger.Log` method, where the `ILogger` is retrieved from the dependency injection container.

This section referred to [7, 42] on how logging is achieved in MapsUI and Avalonia.

2.5.1.4 Options Pattern

It is almost certainly a bad idea to hard-encode some configurations (especially strings) in different services, such as the location of the file for the JMA Time Table, the base URI of API Calls, and the location to store the authenticator. Therefore, a commonly-used approach in .NET is to use a file named `appsettings.json`. The bad thing about this approach, is that it is not typed, and it is stored like a dictionary with strings as keys. Luckily, Microsoft has an Options Binding pattern, which allows us to bind sections of the configuration JSON file, to an options class, which could then be injected into the dependency injection container, and used by the services.

The options pattern is used for the configuration for the JMA Time Table (for the location of the file), the configuration for the DM-D.S.S. (for the base URIs, the OAuth 2 setup options, and the start post data for WebSocket), and the configuration for the Kmoni components (for the base URI, relative URI, and the file path to which the observation points are stored). Other options will not be type bound, but retrieved in a similar method as the string key dictionary as described before.

A small caveat here is that the options bound using type `T` will be used as a `IOptions<T>` in the dependency injection container, and the options will be retrieved using the `Value` property of the `IOptions<T>`.

This section referred to [27] on best practices in binding options and how to do this with dependency injection.

2.5.1.5 Factory Pattern

First, there is the `FileLoggerProvider` (which is the provider for my custom file logger), which implements a `ILoggerProvider` to be used in the logger builder in dependency injection. Rather than providing a constructor for the user to directly call, there is a create method in the provider to create an instance, and that is essentially what factory pattern is about.

Some constructors (setting-up) of the services involved is relatively complicated (e.g. involved the base API which is a string, hence cannot be dependency injected), and others might require setting up two classes which is quite complicated (e.g. in OAuth2 services, if a new service is being set-up, then two classes are involved, one is the one getting the authorisation code and the other involves refreshing the token; while if reading refresh token from a file only the latter is needed). In these cases, factory classes or methods will hide this complexity to the users.

These will be introduced in detail later.

2.5.1.6 MVVM Pattern

The MVVM (Model-View-View Model) pattern is a commonly-used pattern in UI design. The main reason it is used is for a clear separation of concerns – the **view** describes the windows, and **binds** to certain properties in the **view model**, which is the model that the view binds to, and actual data is stored in the **models**.

As a brief description of the pattern, the following interfaces will be discussed. The detailed discussion of the view models and the models will be left to later.

`INotifyPropertyChanged` This interface is the most commonly-used interface in MVVM – it allows the object to notify other objects (that subscribes to it) whenever its properties has changed. It is especially useful in MVVM pattern, since it allows the model or the view model to notify the view model or the view that its properties to change, which allows the view model to deal with this accordingly, or let the GUI refresh the data of the view to update the window.

`INotifyPropertyChanged` only defines one particular thing in the interface – the `PropertyChanged` event, which is an event that is raised whenever a property has changed.

This section referred to [9].

`ObservableObject`, `ObservableProperty` and `RelayCommand` It would be painful to implement this interface for every single model/view model, since we would need to implement a backing (private) field, and a public property whose set property raises the `PropertyChanged` event.

`CommunityToolkit.Mvvm` addresses this problem by providing a base class (`ObservableObject`) and source generators `ObservableProperty` and `RelayCommand` to mark as attributes to use on the fields or on the methods, to generate the properties or the commands, respectively. It reduces our workload of implementing the same interface for each property repeatedly, and is very useful in the MVVM pattern. We will use this significantly to implement the view models in the MVVM pattern.

This section referred to [63].

2.5.1.7 Reflection

There are multiple enumerations that we would like to bind to in our view. Therefore, extension methods are provided to convert the enumerations to a readable string, or to a colour string, to bind to the view. However, the implementation of the Avalonia binding system decided that binding will not be possible with the extension methods.

Therefore, **converters** are used here. Specifically, the typed one-way converter `FuncValueConverter` is used. However, it would be ridiculous to write such a converter for each enumeration.

Therefore, the code is designed to put all extension methods for converting to a string in a single class, and all extension methods for converting to a colour in another class.

A converter is then written for the generic `Enum` class to convert to a string. Then, **reflection** will be used to find the list of all the methods provided by the particular extension classes, and filtering the method that takes the specific concrete type of the enumeration as the parameter, and invoke the method on it to convert to string. This is still type-safe, since the type check for parameters is used, but we have achieved 'type-safe dynamic typing' using reflection!

This section referred to [6, 48, 55].

2.5.1.8 Events

Events are raised when something of interest happened, and a **handler** is attached to the event to handle the event (i.e. executes some code). As an example, there are two events for WebSocket, the data received (which invokes the event handler in the real time view model), and status changed (which invokes the event handler in the setting view model).

The detailed use of events will be introduced in the separate components.

This section referred to [28], the .NET guidance on how to raise and handle events.

2.5.1.9 Exceptions

Exceptions are thrown when unexpected things (errors) happen in the application code (e.g. when an API call fails, when a WebSocket disconnects unexpectedly, when an attempt to write to a file fails). Exceptions are to be **handled** by the application (and should not cause the application to crash any way) and logged into the logs.

Custom exceptions are designed in the authentication, telegram, and WebSocket components in the application, and will be demonstrated later within the class diagrams.

This section referred to [22], the .NET guidance on how to create custom exceptions, and best practices on handling exceptions.

2.5.2 Abstractions (Interfaces), Options and Services (Classes)

Interfaces is an abstraction of a service, and is a contract between the service and the user on what the service will be able to do. It defines the functionality of the service, but does not provide the implementation. This is especially useful for mocking the service for unit testing purposes, and provide the possibility for polymorphism, which could be achieved by different implementation of the same service.

Options on the other hand gives the service the necessary setup data. In most cases, it makes the constructor more meaningful, and allows for the options pattern and binding to an external settings file (see later for a detailed description of the settings pattern).

Each section below is the description of one particular responsibility (service) that is necessary for the application, and in separate namespaces.

2.5.2.1 JMA Time Table

For the service of the JMA Time Table, **options pattern** is used to provide the location of the raw text file (the only setup required).

Abstraction is provided for the service that calculates the distance based on the time and depth, and this is the only responsibility of the interface, `ITimeTable`.

The services will be split into two parts: the actual service for the Time Table, `TimeTable`, and the builder of the Time Table Database, `TimeTableBuilder`, which is responsible for reading the file, verifying that it has the correct format via the Regular Expression, and parsing the raw text file into the DTO format to be transferred into the Time Table service provider. This is an example of the **factory pattern**, and it will return an object with the type of the interface, rather than the concrete class itself. This hides the detail implementation from the user, and provides the user with only an interface to interact with.

Logging will be provided for the service class `TimeTable`.

Extension methods on `IServiceCollection` is provided to **inject** a singleton of `ITimeTable` into the service collection, and the options pattern is used to provide the location of the file. This also hides the factory from the user as well.

This functionality does not have any dependencies on other functionalities.

Figure 2.26 is the class diagram for this functionality.

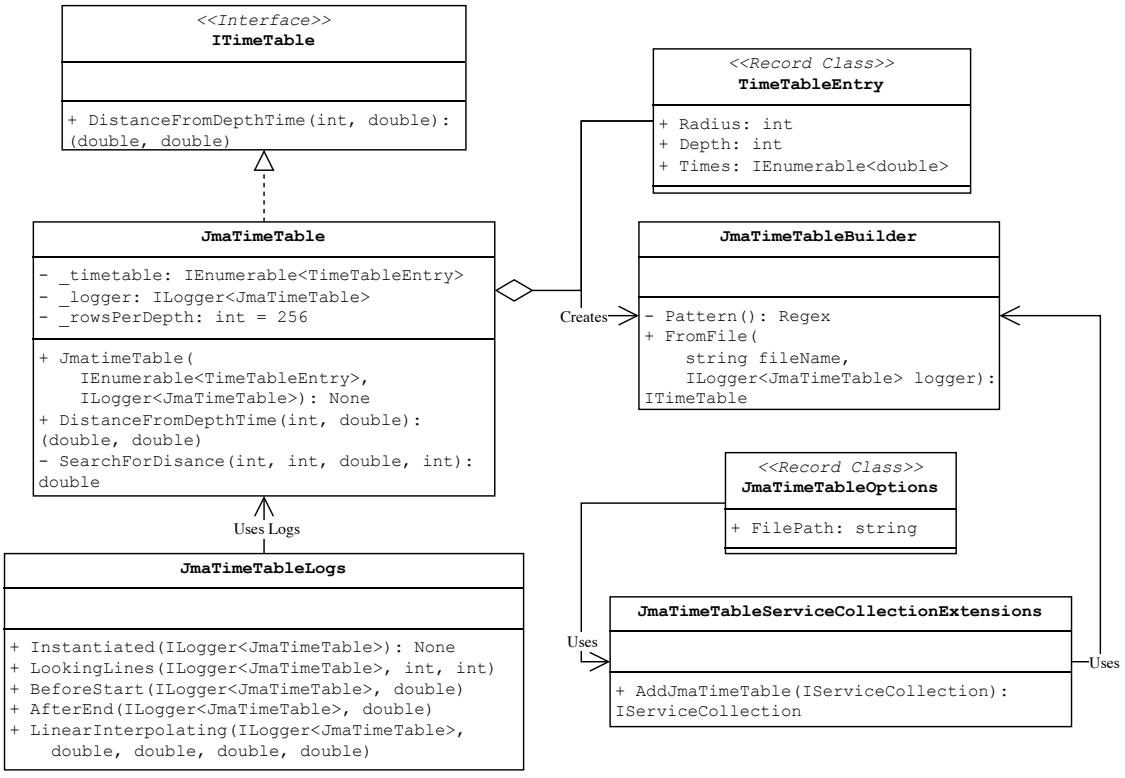


Figure 2.26: Class Diagram for JMA Time Table

2.5.2.2 Kyoshin Monitor

To manipulate with colours and images, the C# library SkiaSharp is used, specifically the classes `SKColor`, `SKBitmap` and `SKImage`.

The functionality is split into two parts, the fetching of the images from the webpage, and the extraction of the points using give data on the extraction of points in JSON format. As for the first functionality, the **abstraction** used is the interface `IImageFetch`, and `ImageFetch` is the concrete implementation of it. They all depend on two enumerations, the `MeasurementType` and the `SensorType`, with an extension class providing conversion of the enumerations to a string in the URI. **Logging** is achieved by `ImageFetchLogs`.

The second functionality is the extraction of the points from the JSON data, and the **abstraction** used is the interface `IPointExtraction`, and `PointExtraction` is the concrete implementation of it.

For the DTOs to deserialise the JSON file which stores the mapping information of pixel position and geographical location, the DTO `ObservationPoint` is used, **composed** in the class `PointExtract` (i.e. they have the same life-cycle). `GeographicCoordinate`, `PixelCoordinate` and the enumeration `PointType` has composition relationship with the `ObservationPoint` class as well, since they are considered part of the observation point.

To achieve **dependency injection**, extension methods on `IServiceCollection` is provided to inject `IImageFetch` and `IPointExtract` into the service collection, using **options pattern** to set up the base URI, relative URI format string, and the file path to the JSON file storing the mapping of pixels to geographical locations.

This functionality does not have any dependencies on other functionalities.

Figure 2.27 is the class diagram for this functionality.

Apart from those basic services, the colour conversion is included as extension methods provided in a static class. Also, an extension method is used in the conversion from the `byte[]` to `SKBitmap` to connect between `IImageFetch` and `IPointExtraction`. Figure 2.28 gives the class diagram for these two Functionalities.

2.5.2.3 DM-D.S.S. Authentication

As briefly discussed previously, there will be three types of authenticators involved here: not authenticated (null authenticator), API key authenticator, and OAuth 2 authenticator. To **abstract** their behaviour, the interface `IAuthenticator` is introduced. To manage the authenticators (to achieve a singleton pattern), the manager which acts as a **factory** is abstracted as `IAuthenticatorHelper`. The enumeration `AuthenticationStatus` is introduced to get the status of the current authenticator.

In terms of the concrete implementations of the interface, `NullAuthenticator` is the easiest. It will throw a custom exception `NullAuthenticatorException` when attempting to get a header from the class. However, since its behaviour is identical, it will be meaningless to create a new instance of it/dispose it when its necessary/unnecessary. Therefore, a common shared static instance is used, as in the `Instance` property of the class. The `ApiKeyAuthenticator` class also has a very simple implementation. It only has one thing, the API Key.

The OAuth 2 classes are slightly more convoluted. `OAuth2Authenticator` is the key part of the functionality, and it implements `IAuthenticator`. However, it is only its responsibility to acquire access token from refresh token, and to revoke tokens. The responsibilities of acquiring an authorisation code and the first refresh token is done by the `OAuth2Helper` class, which shows a clear separate of concern. Since there are multiple parameters necessary to set up OAuth2, the **options pattern** is used and `OAuth2Options` class provide the configuration. To hide this complexity, **factory methods** are also used, as in `OAuth2Provider`. (Shared static methods between OAuth 2 classes are in `OAuth2SharedMethod`).

Very simple DTO objects are used to store the JSON objects returned from the authentication endpoint. The `Error` DTO class represents an error response, `TokenRefresh` represents a token refresh response, and `TokenRequest` represents a token request response. However, since `TokenRequest` response is just basically `TokenRefresh` response plus the refresh token, **inheritance** is used here to reduce redundant code.

All concrete implementations of `IAuthenticator` must override the `ToString` method to provide means of saving the authenticator to a file.

The concrete implementation of `IAuthenticatorHelper` is `AuthennticatorHelper`. It is responsible for creating the authenticator based the new choice of the authenticator (null, API Key, OAuth), and a method is also provided to read and setup from the string representation of the current authenticator from a file.

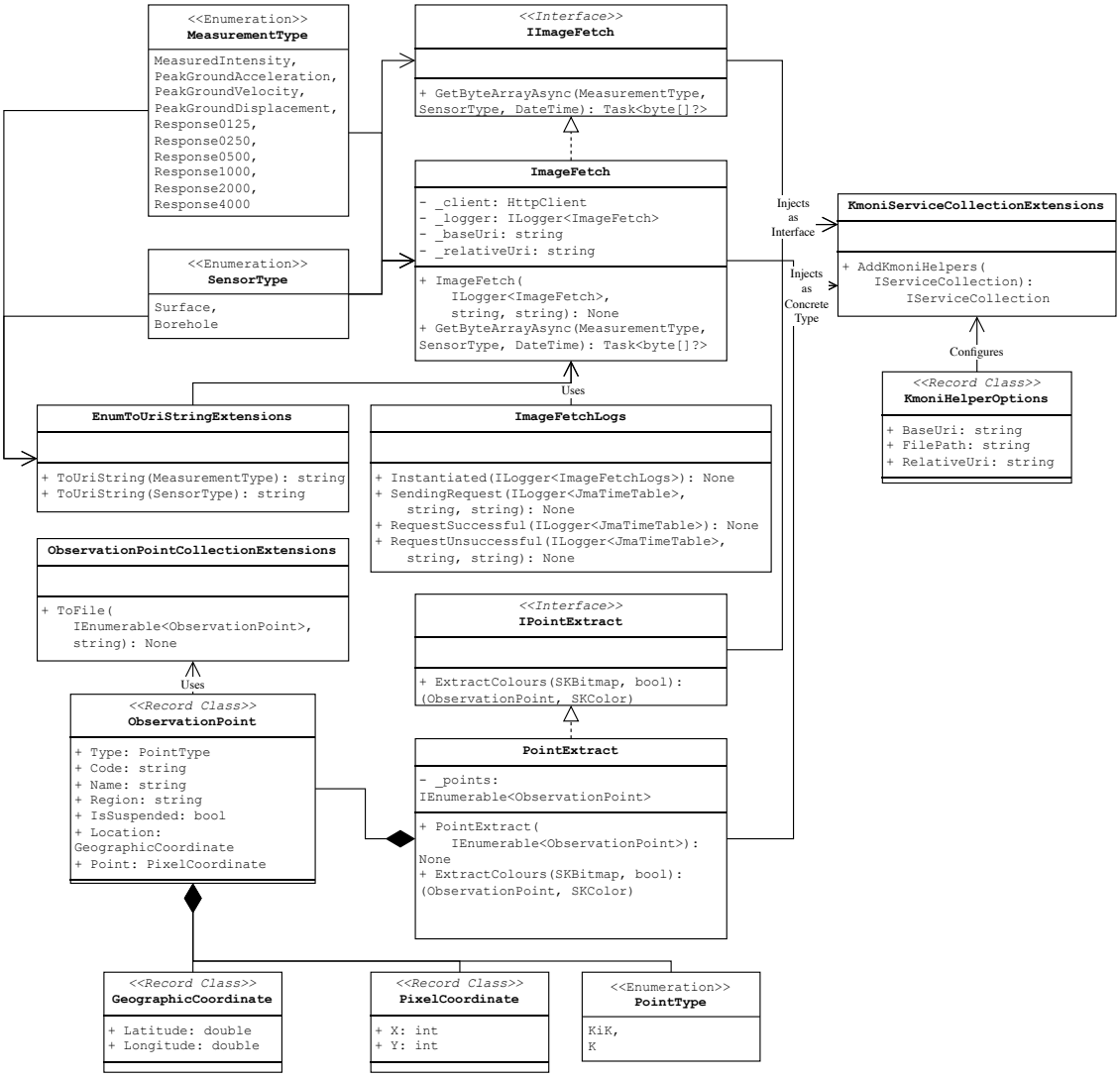


Figure 2.27: Class Diagram for Kmoni

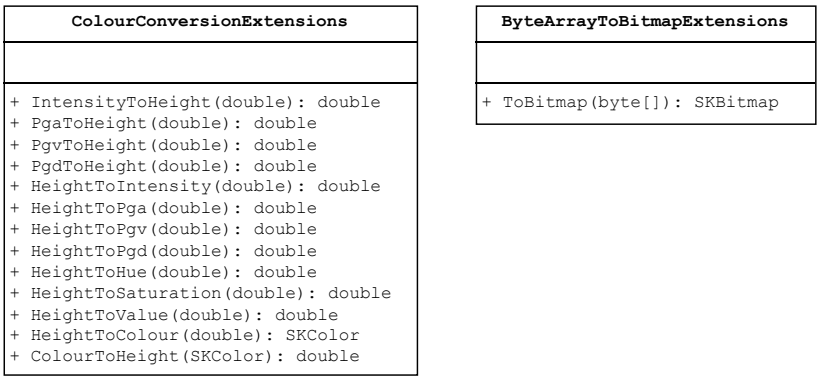


Figure 2.28: Class Diagram for Extra Services of Kmoni

Furthermore, there is an **event** in `IAuthenticatorHelper` which is `AuthennticatonStatusChanged`, which notifies other parts of the code that it has changed. The refresh of the components of GUI Setting Page is achieved by subscribing to this event (adding event handler), and the writing of the current authenticator used to a file is also achieved this way (rather than hard-coding into the code, since that would break single-responsibility).

Extension methods are provided in `AuthenticationHelperServiceCollectionExtension` to inject services into the dependency injection container. Logging is provided for `AuthenticationHelper`, `OAuth2Authenticator` and `OAuth2Helper`, since they have quite convoluted functionalities.

Figure 2.29 gives the class diagram for this functionality.

2.5.2.4 DM-D.S.S. API Calls

The service for API calls is relatively trivial, with `IApiCaller` representing the **abstraction**, `ApiCaller` being the **concrete implementation**, and `ApiCallerLogs` providing the **logs** for API calls. Extension methods for **dependency injection** are provided in `ApiCallerServiceCollectionExtensions`, and some other service extensions are provided.

To improve code reusability (and share error-handling code and behaviour), there are several private methods introduced in `ApiCaller` for this. For example, `CreateHttpRequest` is used to create HTTP Requests, `HandleError` is used to handle errors in HTTP Requests, and `SendRequestAsync` is used to send requests.

Notice that this class should not expose API errors to the external code that is using it – it will just return null if the API call fails.

The class diagram is in Figure 2.30. Notice that this does not include the DTOs, since they are not part of the service. (They will be discussed later.)

This service depends on the authentication service, and the common DTOs.

2.5.2.5 DM-D.S.S. Telegram Fetching and Parsing

The responsibilities of parsing the schema for the JSON telegram and parsing the telegram, is separate responsibilities from fetching the JSON telegram. Therefore, there are two **interfaces** involved here: `ITelegramParser` for parsing the JSON into the DTO of the corresponding schema, and determining the C# type of the object corresponding to a schema; and `ITelegramRetriever` for acquiring the telegram itself from the API.

Their concrete services are `TelegramParser` and `TelegramRetriever` respectively. However, notice that `TelegramRetriever` would aggregate an object of `TelegramParser`. This is an **aggregation** since their life cycle are not fully (though highly) correlated – the parser is provided in the constructor for the retriever, and could as well exist without the retriever.

Both concrete services have **logging**, and extension methods for **dependency injection** is also provided. Similar to previous classes, due to the existence of the dependency injection methods, the service classes are not exposed to external code, and the only way to create an instance of the service is through the dependency injection container.

The class diagram is in Figure 2.31.

This component has dependency on the authentication class, and the shared common DTOs.

2.5.2.6 DM-D.S.S. WebSocket

There is one **interface** for this, `IWebSocketClient`, and the concrete **service** is `WebSocketClient`.

There are two events: data received (with event arguments `DataReceivedEventArgs`, containing the telegram received), and status changed (with event arguments `StatusChangedEventArgs`, containing a boolean indicating if the new connection is connected). **Exceptions** are designed as well, but this is mainly for internal purposes, and they will not be exposed to the users of this (all handled internally).

The class diagram is in Figure 2.32. Note that DTOs are excluded from this.

The modelling of the DTOs for WebSocket is particularly interesting. They are all internal, since they are just designed for internal usage, and not for exposing to external code. First, a **base record class** `ResponseBase` is implemented, which contains the type of the response in an enumeration, `MessageType`. Contra-intuitively, it is **not virtual**, since we would like to first parse the data to this, determine the type, and then parse further to get the concrete type of the response depending on the type received. Different concrete types will have different `MessageType`, which is an example of **polymorphism**.

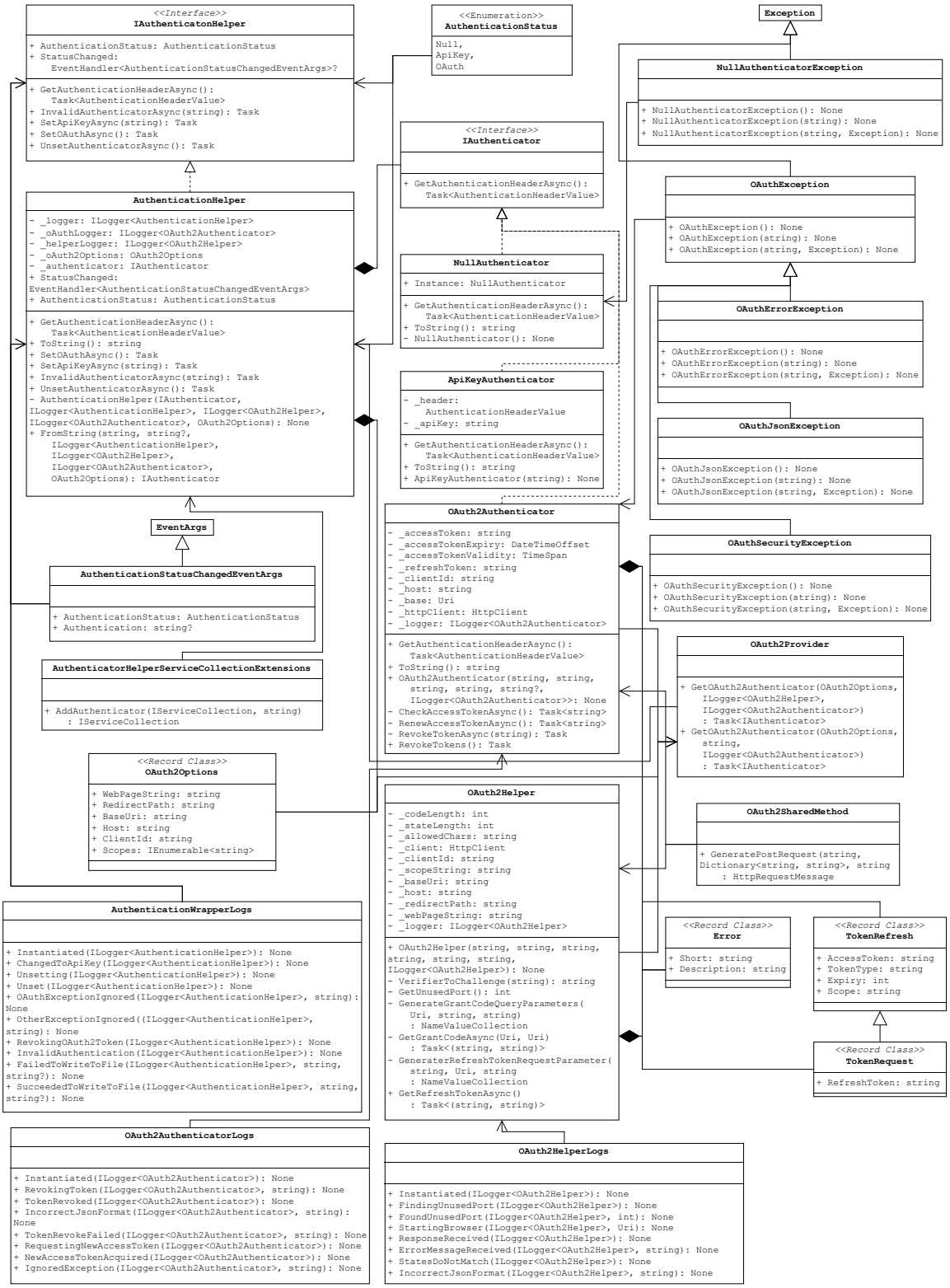


Figure 2.29: Class Diagram for DM-D.S.S. Authentication

Figure 2.30: Class Diagram for DM-D.S.S. API Calls

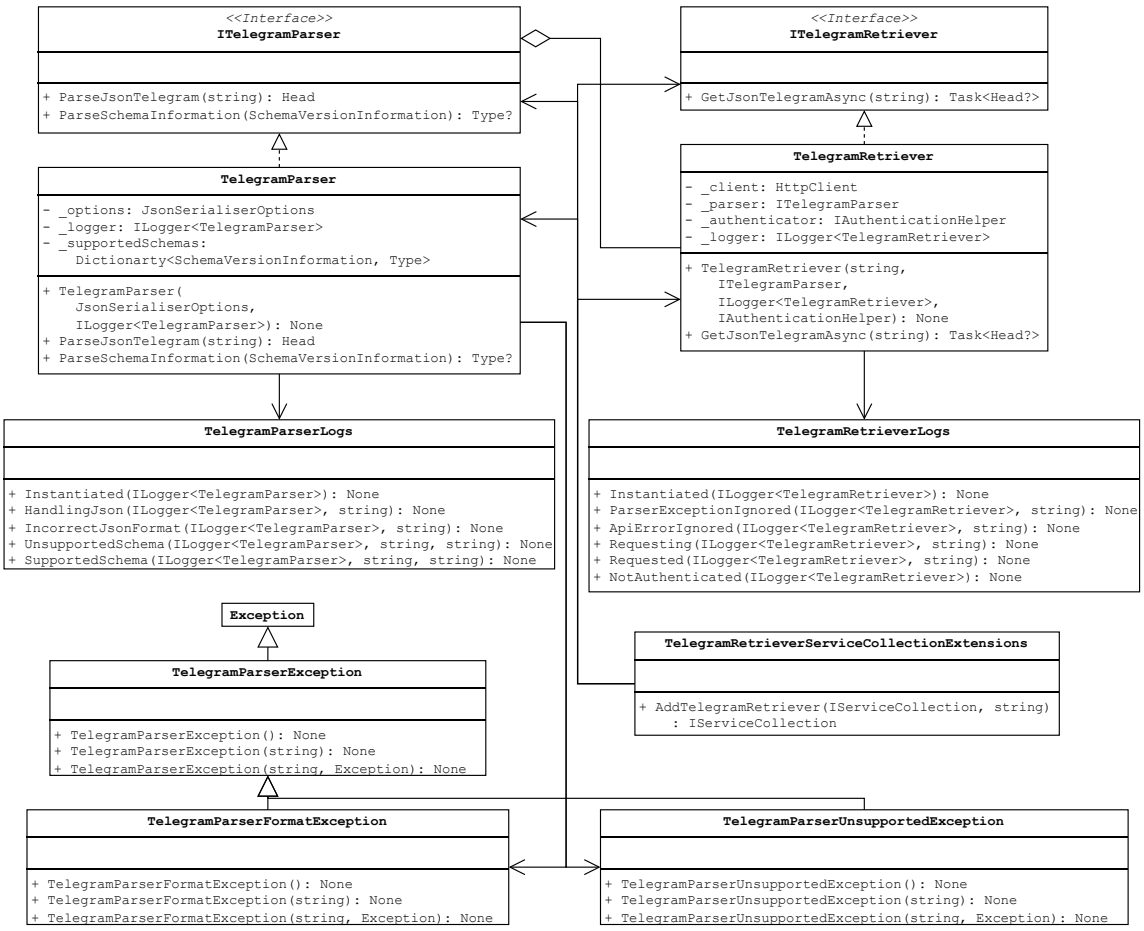


Figure 2.31: Class Diagram for DM-D.S.S Telegram Fetching and Parsing

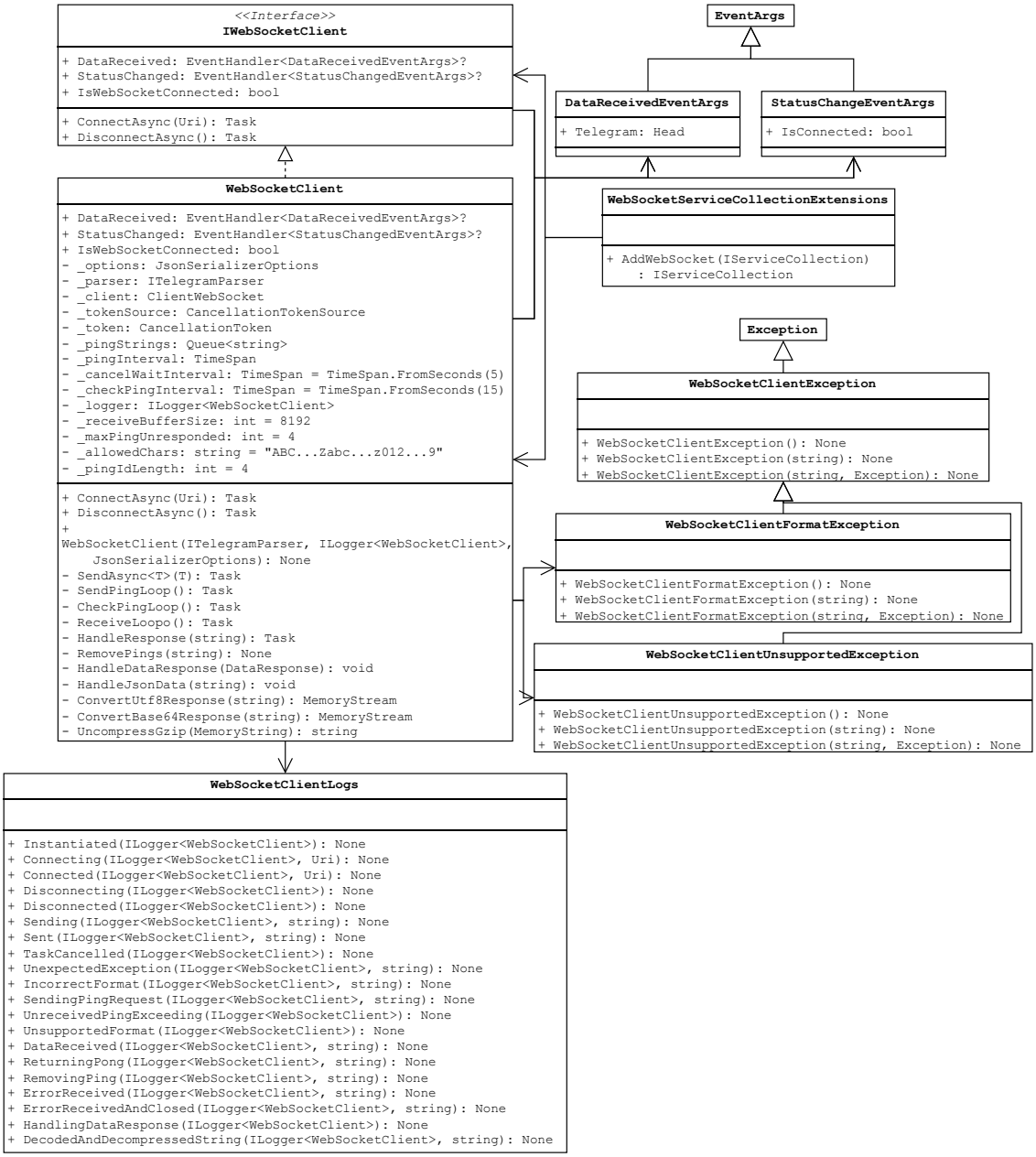


Figure 2.32: Class Diagram for DM-D.S.S WebSocket

The five different types of responses, and including the two requests are all inherited from this. The reason is that they all share the same behaviour with the `MessageType`.

The `CompressionType` and `EncodingType` are described as enumerations rather than strings, since they can only take finitely many values, and this allows for easier comparison, and preserves type safety (unlike strings, which might lead to meaningless results).

Apart from this, other compositions of record classes to represents DTOs is just normal behaviour, same as what the API described. Figure 2.33 gives the class diagram for the DTOs involved. Notice how inheritance and polymorphism is involved.

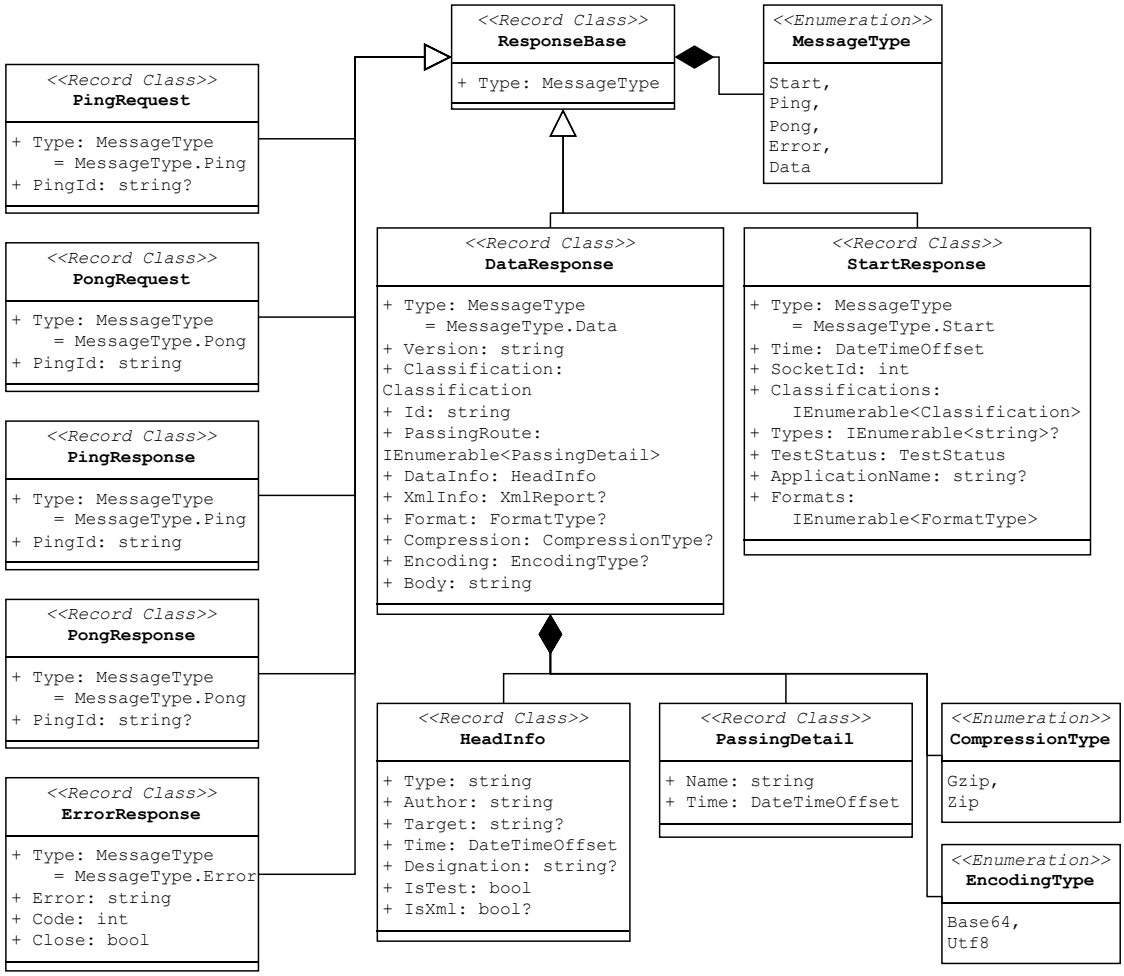


Figure 2.33: Class Diagram for DM-D.S.S WebSocket DTOs

This service depends on the common DTOs, and the telegram parser (but not the telegram fetcher, which releases its dependency on the authentication). This means that `ITelegramParser` is **aggregated** within `IWebSocketClient`, but not **composed**, just like with `ITelegramRetriever`. In the dependency injection, the `ITelegramParser` will be injected as well, but **singleton** pattern ensures there is only one of them in play, commonly shared.

2.5.3 Design of UI Classes with MVVM Pattern

2.5.3.1 Services

Logging Since the use of the **adaptor pattern** and **factory pattern** in logging is described before (for Avalonia and File Logger respectively), a class diagram is included here on how this is in Figure 2.34.

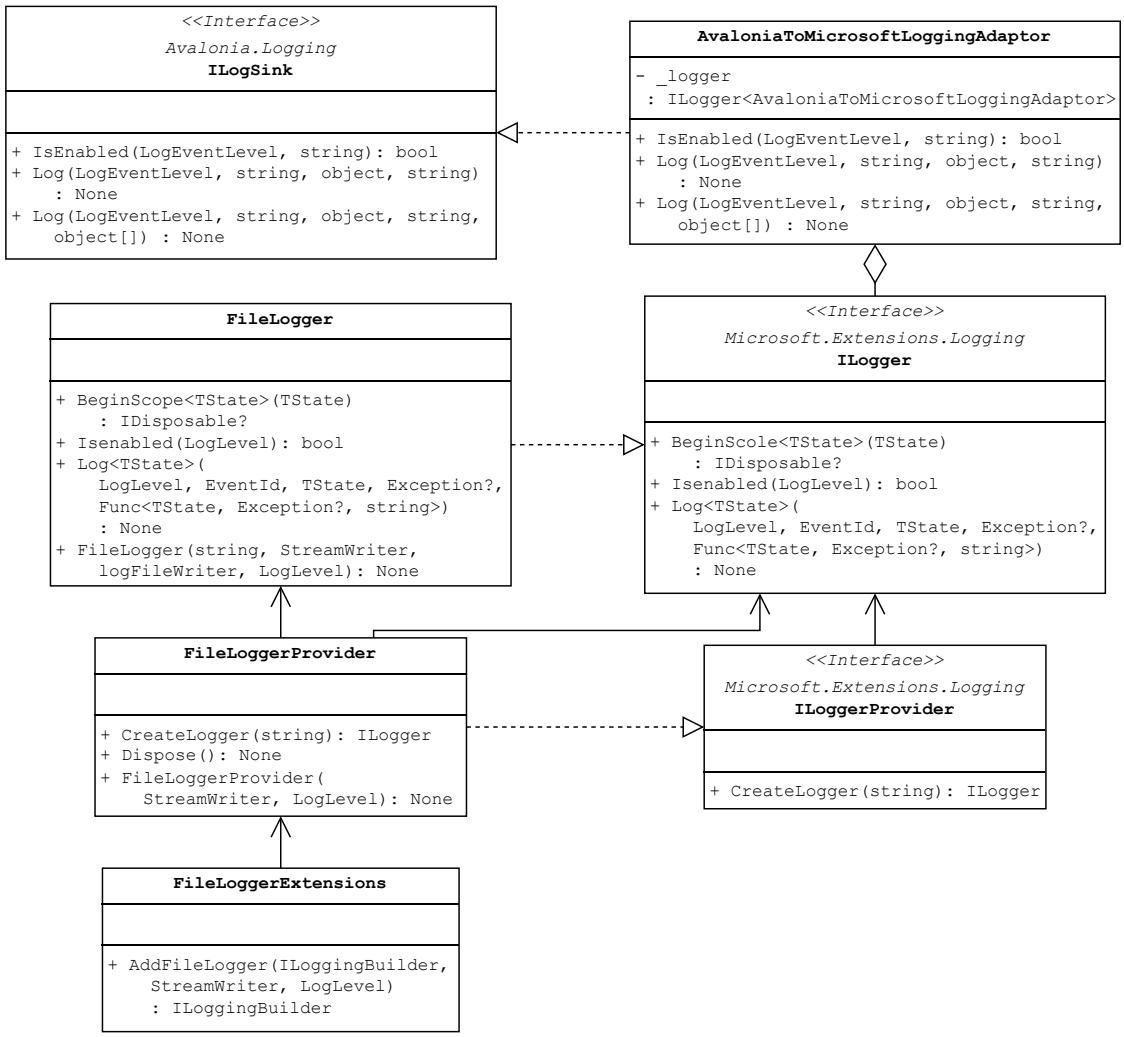


Figure 2.34: Class Diagram for Logging Services

Resources There are multiple map resources shared across different pages, and they are stored in the `MapResourcesProvider` class, which includes the shape files, the styles for the hypocentres, and the prefectures of Japan (which is further stored in the `PrefectureData` class). The class diagram for this is in Figure 2.35.

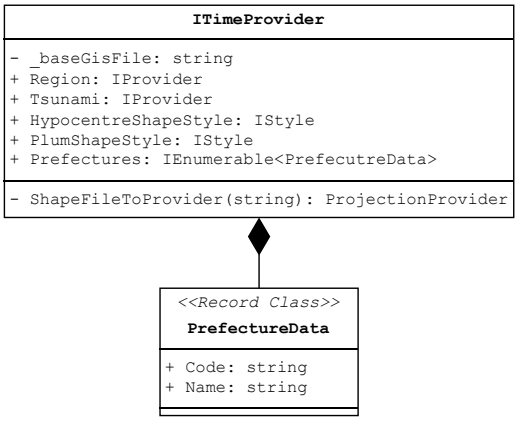


Figure 2.35: Class Diagram for Map Resources

Time Abstraction Although there is an abstraction for a time provider provided by Microsoft as well, the author preferred to implement his own in the application, since the Microsoft one is too heavy but also lacks some functionality at the same time. It only achieves two things: getting the current time, and convert the time to JST, Japanese Standard Time, for communication with the Kmoni APIs. The class diagram for this is in Figure 2.36.

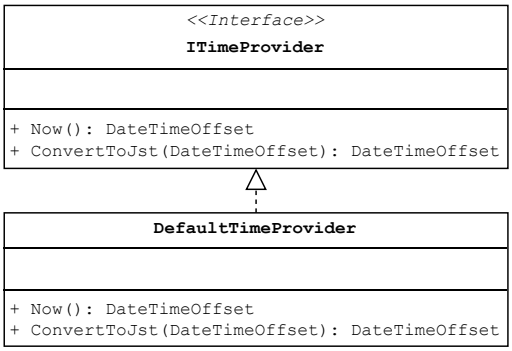


Figure 2.36: Class Diagram for Time Abstraction

This design might look trivial, but this first means that time could be mocked up for unit testing, and secondly past earthquakes are easier to be recreated, since a mock time provider could be provided to mock time in the past, and finally it improves code readability and code reusability, especially with the conversion of time to JST. There could also be a further functionality added, in the future where necessary, for the user to indicate whether they would like time to be displayed locally, or in JST, or in UTC.

Kmoni Settings Manager The reason why this (Kmoni) settings in particular is separated from other settings and have a full set of abstractions/options/DTO, is that it has its unique complexities: it is shared between the setting page and the real-time page, it involves the writing of the current option to a file, and it has default options if the file was not able to read/does not exist.

The **abstraction** involved here is the interface `IKmoniSettingsHelper`, which exposes the current measurement choice, and the current sensor choice. It also has an event to notify when the settings have

changed (which is then used to write the new settings to the file, and in the real-time page to update the scale of the GUI to correspond to the new choice). Therefore, in the **event arguments**, it also includes a copy of the new settings.

There is a **DTO** called `KmoniSettings`, which stores the sensor type and measurement type, and is serialisable/deserialisable to JSON format, which then could be written to a file/read from a file. Furthermore, this class has **options** in class `KmoniSettingsHelperOptions`, which contains the file path to store the settings, and the default settings, both to be read from the `appsettings.json` file.

The concrete implementation is `KmoniSettingsHelper`, which is responsible for exposing the property as defined by the interface, while at the same time raising the event if the new sensor choice has changed. It also has logging involved.

Furthermore, there is a **factory method** provided as extension to the dependency injection container, which reads the current options from the files, and adds the write-to-file method to the event handler. It also does the error handling of the file read and write, and the JSON serialisation/deserialisation – this shows a clear **separate of concerns** in the design.

The class diagram for this is in Figure 2.37.

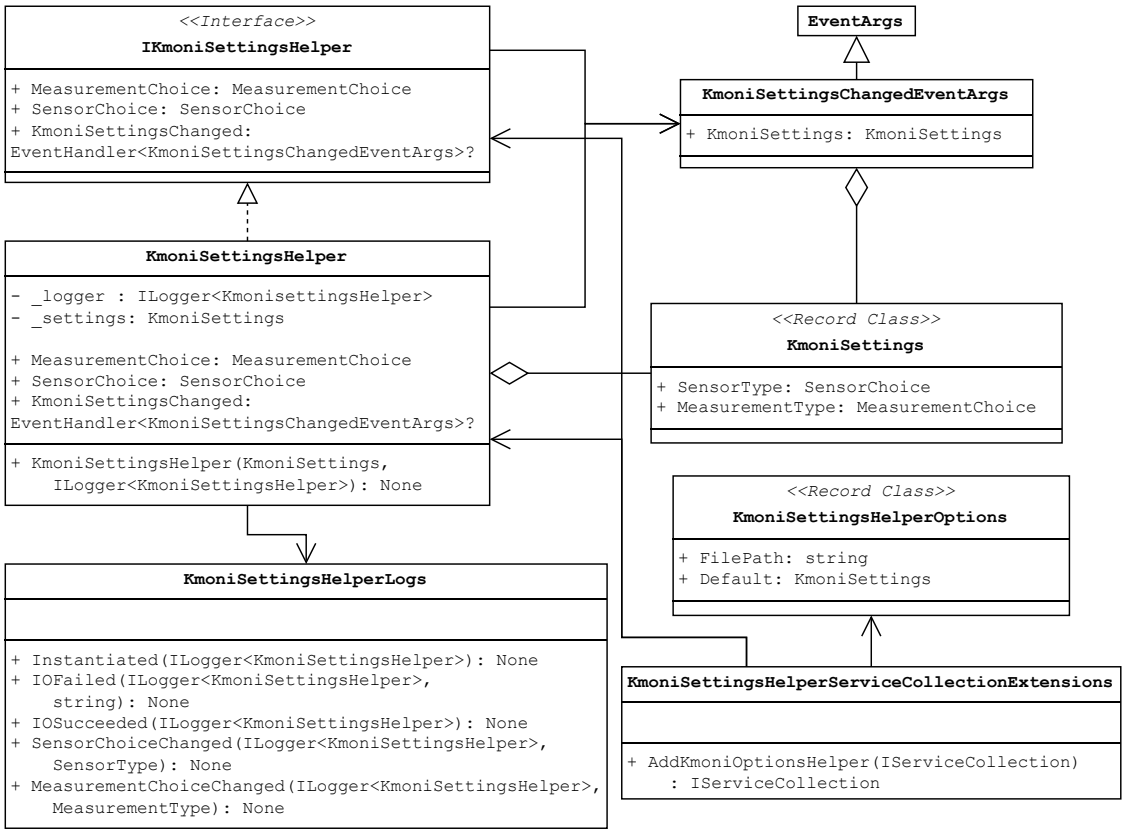


Figure 2.37: Class Diagram for Kmoni Settings Manager

2.5.3.2 Views, View Models, and Models

One-to-one correspondence between view and view model This principle is just as straightforward as the title suggests – there is one view model corresponding to each view, for the view to bind to.

There are four view models involved in the application: the main window (which contains functionality of the sidebar and switching the view-model), the setting page, the past earthquake page, and the real-time page. Figure 2.38 is a class diagram for the design of the view models.

The details of the view models of the pages will be detailed later.

Even though each view model might look bulky and heavy, the author believes this still does not break the principle of single responsibility – since each view model is only responsible for one page after

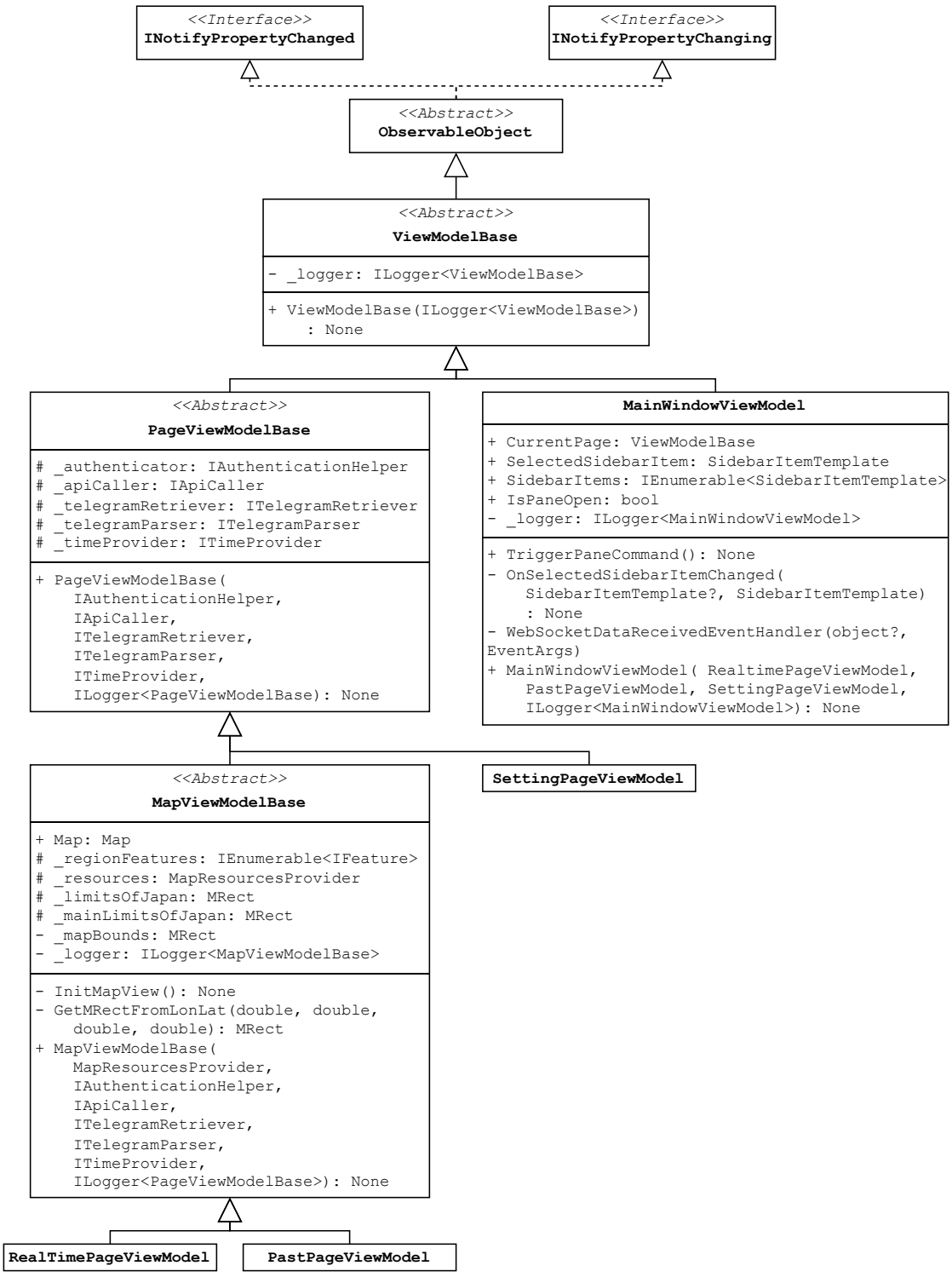


Figure 2.38: Class Diagram for View Models

all.

Abstract Base Classes **Abstract base classes (ABCs)** are used here which describes the shared parts for view models. All view models inherit from `ViewModelBase`. Then there is a `PageViewModleBase` which is the base view model for view models for a page (i.e. excluding the view model for the window itself). The `MapViewModelBase` inherits from that and provides additional functionalities for the map. **Protected fields and methods** are used to provide shared functionalities.

`ViewModelBase` This is a class which inherits from the `ObservableObject` from the MVVM toolkit, and only involves the `ILogger` object. This might look unnecessary, but it would be extremely useful in the future, in case we would like to change the MVVM toolkit we use, or if we would like to add more shared functionalities.

`PageViewModelBase` This is a class which inherits from the `ViewModelBase`. It has five protected fields: the authentication helper, the API caller, the telegram retriever, the telegram parser, and the time provider for the application. They are all **aggregated** but not composed in the classes, since they are not part of the life cycle of the view model, and are passes in as constructors for the view model (as in dependency injection).

`MapViewModelBase` This is a class which inherits from the `PageViewModelBase`. It has a private field marked as an observable property (which makes it public anyway) for the map, and four protected fields for:

- the features of the regions read from the shape file;
- the static resources for the map, such as the shape files themselves, and the two styles for the hypocentre to be reused;
- the limits of Japan; and
- the main limits of Japan (containing the tiny islands as well).

The Window The view model for the main window has quite simple purposes: providing the sidebar to switch between views (view models), and provide a button to expand/retract the side pane as well.

The property `IsPaneOpen` is a boolean indicating whether the sidebar pane is open, and it is bound to the property of the pane. The button to expand/retract side pane is bound to `TriggerPaneCommand`, which changes the boolean value (by XOR with true).

The only model associated with this, is the item displayed in the sidebar, `SidebarItemTemplate`. It is initiated with the instance of the view model, the icon key to display on the sidebar, and the text to display on the sidebar. The view will be able to bound the related properties by accessing the list of sidebar items. Since this instance is created in the constructor of the view model, it is considered as **aggregation**, since their life cycle is strongly correlated.

Upon the chosen item changing in the sidebar, the `CurrentPage` will change as well, allowing the view of the window to change the binding.

It will also handle the event raised by the real time view model requesting to change the view model to the real time view model, and this will be done within the constructor.

Figure 2.39 is the class diagram for the window view model and the related components.

Models for Setting Page There are **four** primary functionalities of the settings page: changing the language setting, the Kmoni setting, the WebSocket settings, and the authentication settings.

Figure 2.40 is the class diagram for the settings view model and related components.

Language Settings There is an **event**, `LanguageChanged` to be invoked when the language is changed. This is not used in particular for external code – this is used as a means to clean up what is to be done when the language has changed. In the current design, only two tasks are to be preformed: change the current language of the resources files, and to write it to the file which stores the language. The existence of this event allows the functionality to be extended, such as popping a window for the user to confirm, in the future.

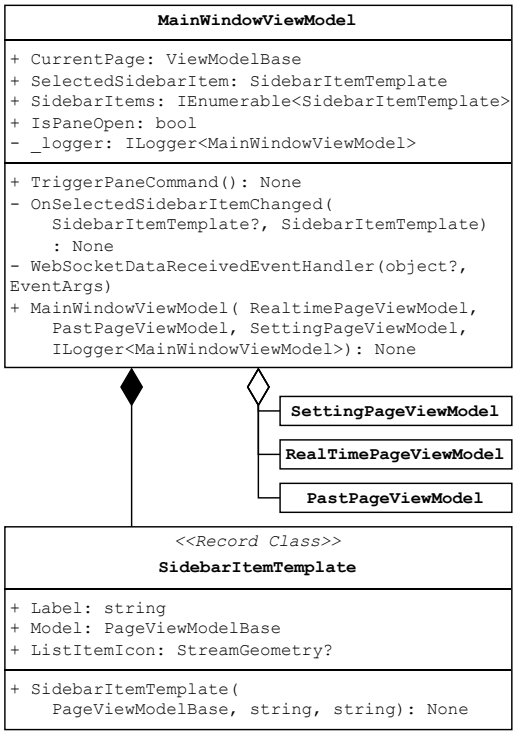


Figure 2.39: Class Diagram for Window

The property `LanguageChoice` is exposed to the view model to bind to the current language choice, and `LanguageChoices` is the list of languages available. To be consistent with the resources file, the type `CultureInfo` is used to represent the language.

Kmoni Settings For the Kmoni settings, there is a `IKmoniSettingsHelper` **aggregated** within the class, and two lists which contains the choice of the sensor type and the measurement type. The properties in the settings helper are directly bound to the view model, and all properties changed are handled within the settings helper.

WebSocket Settings This part aggregates a `IWebSocketClient`, which is the actual WebSocket connection. There is an event handler for the WebSocket connection status changing, since the view model has to notify the view that certain properties has changed (the current WebSocket status). There is a text to bind to the WebSocket connect/disconnect button (which will change when the WebSocket status is changed), and there is a command bound to it as well to connect/disconnect.

In the table there is a list of current `WebSocketConnections`, which is a `ObservableCollection` of the interface `IWebSocketConnectionTemplate`. The use of the observable collection is since it will be able to notify the view whenever its members has changed. The interface of the WebSocket Connection declares four properties: the WebSocket ID, the application name, the starting time of the connection, and whether the disconnect button is enabled. A method for disconnecting is also provided. This is an example of the **adaptor** pattern, since it adapts the objects returned by an API call to the contract we have with the view for binding.

The reason why there is a use of the interface here, is because of **polymorphism**: there are two concrete implementations of this, the first an empty WebSocket connection, which just stays put (and shares a common instance), and the other an actual WebSocket connection, as will be shown in the class diagram below. They behave differently (especially their disconnection command). The actual WebSocket connection also inherit from Observable Object to allow the notification to the view that its properties has changed.

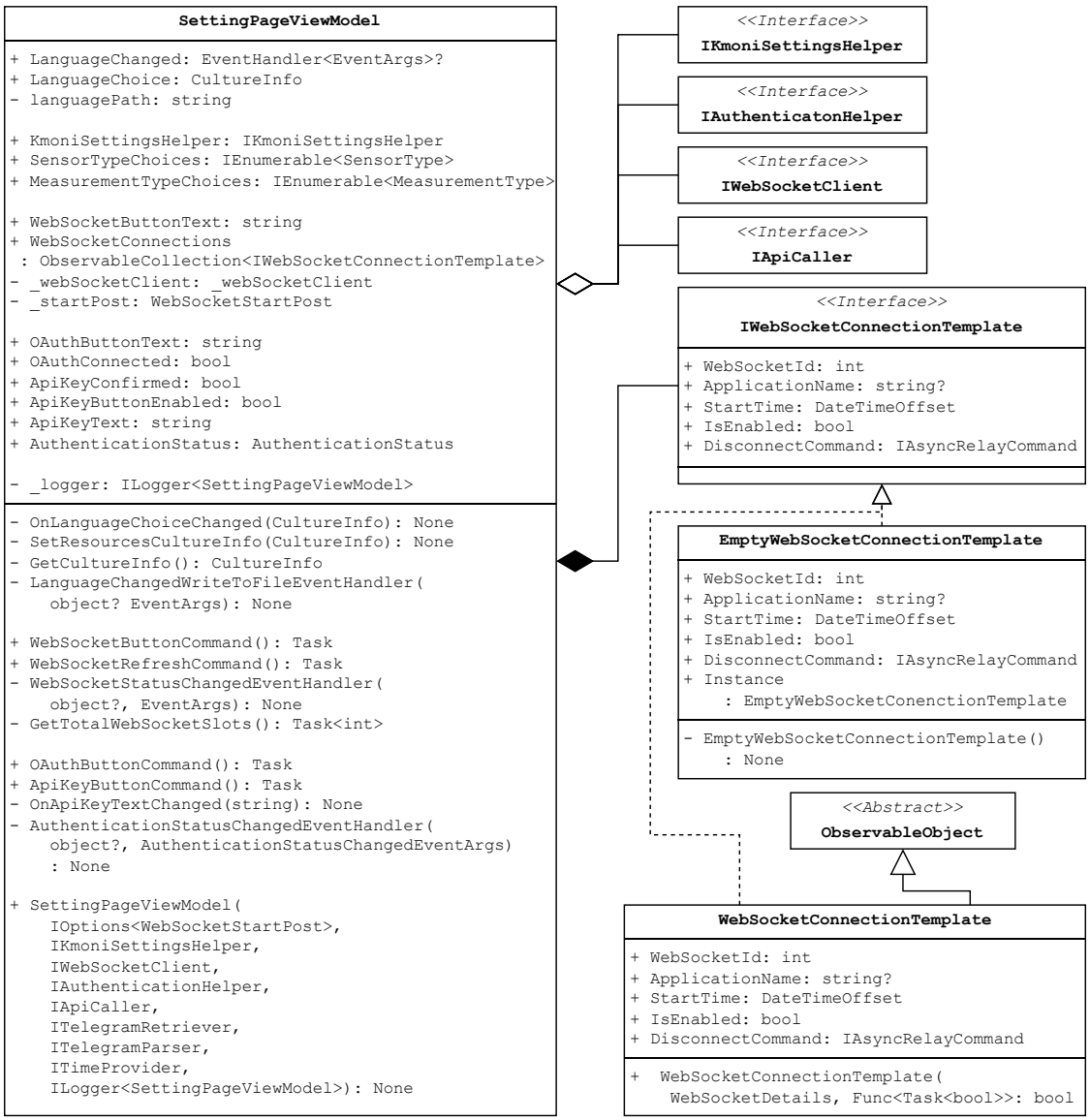


Figure 2.40: Class Diagram for Settings

To implement the refreshing the list of current WebSocket connections, there is a separate method to get the number of total WebSockets available to the user (which looks through the user's contracts), and a separate method to list all currently active WebSockets. The observable collection then could be altered to reflect the latest result.

Authentication Settings For the setting for authentication, there is first property exposing the current `AuthenticationStatus` to the view via the authenticator, to display the current authenticator. There is also an event handler to handle the authentication status changed event, which notifies the properties that are related with the current authentication status.

There are two commands exposing to the view model, the button to set the current API Key, and the button to connect/disconnect from OAuth 2.0. There is also a string for the current API Key Text, a string for the text to display on the OAuth button, a boolean for whether OAuth connection is connected, a boolean for whether API Key is confirmed, and button for whether the API Key button is enabled (i.e. the API Key fits the format, which will be checked every time if the API Key is currently valid).

Past Earthquake Page Splitting up the functionality for the past earthquake page, there are two main functionalities: the past earthquake list, and the past earthquake details display. Hidden behind this is a functionality to retrieve and cache the list of earthquake observation stations (which we will regard as the third functionality).

Figure 2.41 is the class diagram for the past earthquake view model and related components.

Earthquake List To achieve the list of earthquakes, the user would have to press the refresh button, which calls the command associated with it, and attempts to fetch a list of earthquakes, stored into an observable collection. **Note that the observable collection is annotated with observable property as well**, since we could just totally re-set the list to a new one, and the view will be able to reflect the change. In this method, the cursor token will also be stored to a private field, and used to load extra earthquakes. There is a boolean field reflecting whether the load extra button is enabled, by checking if the cursor token is null.

The model to store the earthquake list item takes one earthquake from the API call as in the constructor, and changing it to the contract to the view, which is another demonstration of the **adaptor pattern**. It only keeps the necessary properties for binding as well.

Observation Stations To load the list of earthquake observation stations, the user has to be authenticated first. If the constructor detects that the authenticator is in an authenticated stage, the constructor will attempt to fetch the station list at the start. When the authentication status is changed, there is also an event handler that attempts to fetch the list as well.

There is a method to load the stations manually as well, and allows for methods to check for if the list is initialised, before attempting to use it.

Earthquake Details This section has the most details and has the most complicated functionalities. First, three layers are repeatedly used over the life cycle of the application: the layer for the hypocentre, for the observation stations, and for the regional intensities. When a new earthquake is selected, it will not attempt to create a new layer – instead, it will change the data/style of a layer, and notify the layer that the data has changed.

Secondly there are constants defining the layer of the groups. It is designed such that the hypocentre always stays at the top of the map, stations the next, and the hypocentre in the bottom layer. There is also a **cancellation token source** – this is involved, since if the user has changed the mind of which earthquake they want to view the details of, before the details has been reflected on the map, we do not want it to be reflected any more – and we would like to cancel the asynchronous task. Therefore, in the rest details method, it will cancel and dispose of the current cancellation token source, and set up a new one for the next chosen earthquake.

There is a method to jump to the Yahoo earthquake page for the details of the earthquake as well, using a constant private field of string for the URL to be formatted.

Now comes the interesting part on how to deal with when a new earthquake item is selected.

- MVVM invokes the `OnSelectedEarthquakeChanged` method, which first resets the current earthquake details, and then attains a new cancellation token for the current earthquake.

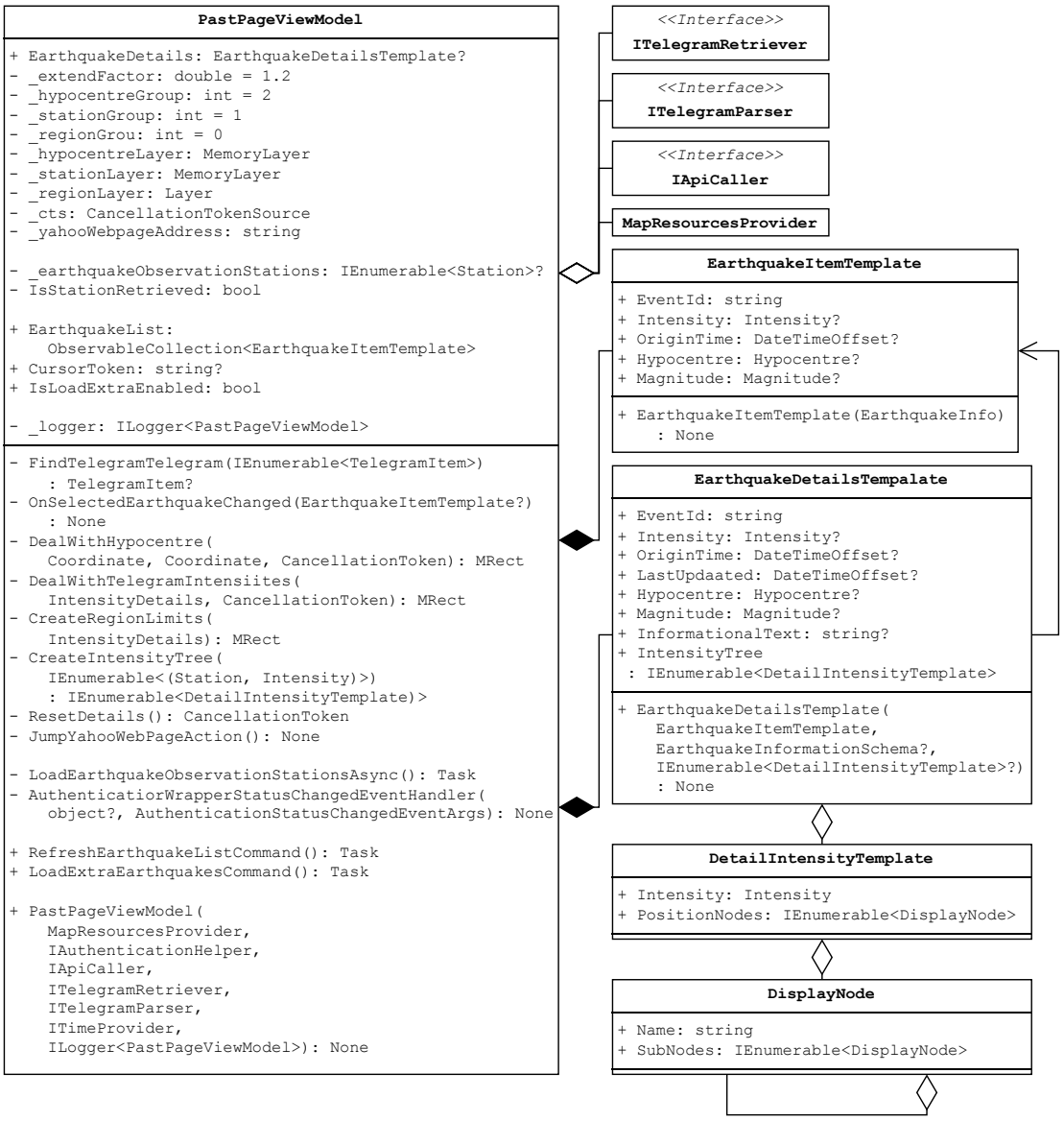


Figure 2.41: Class Diagram for Past Earthquake

- The method gets a list of telegrams for the current earthquake, and uses **filter, group by, select and order** methods as in `FindEarthquakeTelegram` to decide the most suitable telegram to get the information from, as described earlier in the design for the UI page.
- It creates the **intensity tree** and the **regional limits** (map bounds of regions which observed intensity) for the earthquake by calling the `DealWithTelegramIntensities` method.
- It marks the hypocentre on map and obtains limits for the hypocentre by `DealWithHypocentre` method.
- It zooms the map to the union of the regional limits and hypocentre, multiplied by a constant `_extendFactor`.
- It creates the details of the earthquake to display on the side panel, using the intensity tree created, the acquired telegram, and the earthquake list item.

In the `DealWithHypocentre` method, it will change the features of the hypocentre layer and notify the feature has changed, and return the limits for the hypocentre.

In the `DealWithTelegramIntensities` method, it will first **filter** where the intensity is not unknown, and **join** it with the list of observation stations.

Then, it will call method `CreateRegionLimits` which **aggregates** the region limits of each region involved using the union operation, and the `CreateIntensityTree` method which **joins** the station data further with a list of prefectures, and group by intensities, and group by prefecture, region, and finally by city, to create a **tree** structure.

There is a significant use of **list-manipulation methods** here, using existing **LINQ** methods including join, aggregate, select, where, group by, order by, and first.

As for the models used, `EarthquakeDetailsTemplate` is just `EarthquakeItemTemplate` with some more properties to display on the sidebar, and takes it and the telegram and the intensity tree within the constructor.

The particular interesting thing here is the **intensity tree**: it is a tree **with different data types** on different depths. Its root could be considered as the earthquake detail object, and on its first layer, the **intensity** is stored, but afterwards it stores a location, which could simply be it stores a **string**.

Therefore, there are two classes involved here: `DetailIntensityTemplate`, which is for the first layer, having an intensity and a list of `DisplayNode` (which is its children), which has a name, and a list of `DisplayNode` (which is its children). The whole tree is stored in a list of detail intensity template, which is a tree itself. Therefore, as shown in the class diagram, display node is aggregated within itself and detail intensity template, which is further aggregated into earthquake details template (it is not composition since it is provided at the constructor of earthquake details template).

Real-Time Page This is the view model with the most functionalities, but we can split it up into five parts: common parts of the view model, handling EEW, handling Tsunami, handling WebSocket, and Kmoni monitor functionality. Figure 2.42 is the overall class diagram for the real-time view model and related components.

Common Parts There are two **timers** shared by all the parts: a timer that invokes events every second (which includes refreshing the time display, removing expired EEW/Tsunami warnings, and refreshing the Kmoni layer), and a second one that invokes events every 2.5 seconds (which includes the switching of the EEW, if applicable). These event subscribers will be added to the timer within the constructor.

Furthermore, the management of group layers from to bottom is as follows, the layer number defined as a constant integer field (for code reusability and readability):

- Hypocentre,
- Wavefront,
- Tsunami,
- Kmoni,
- Intensity Regions.

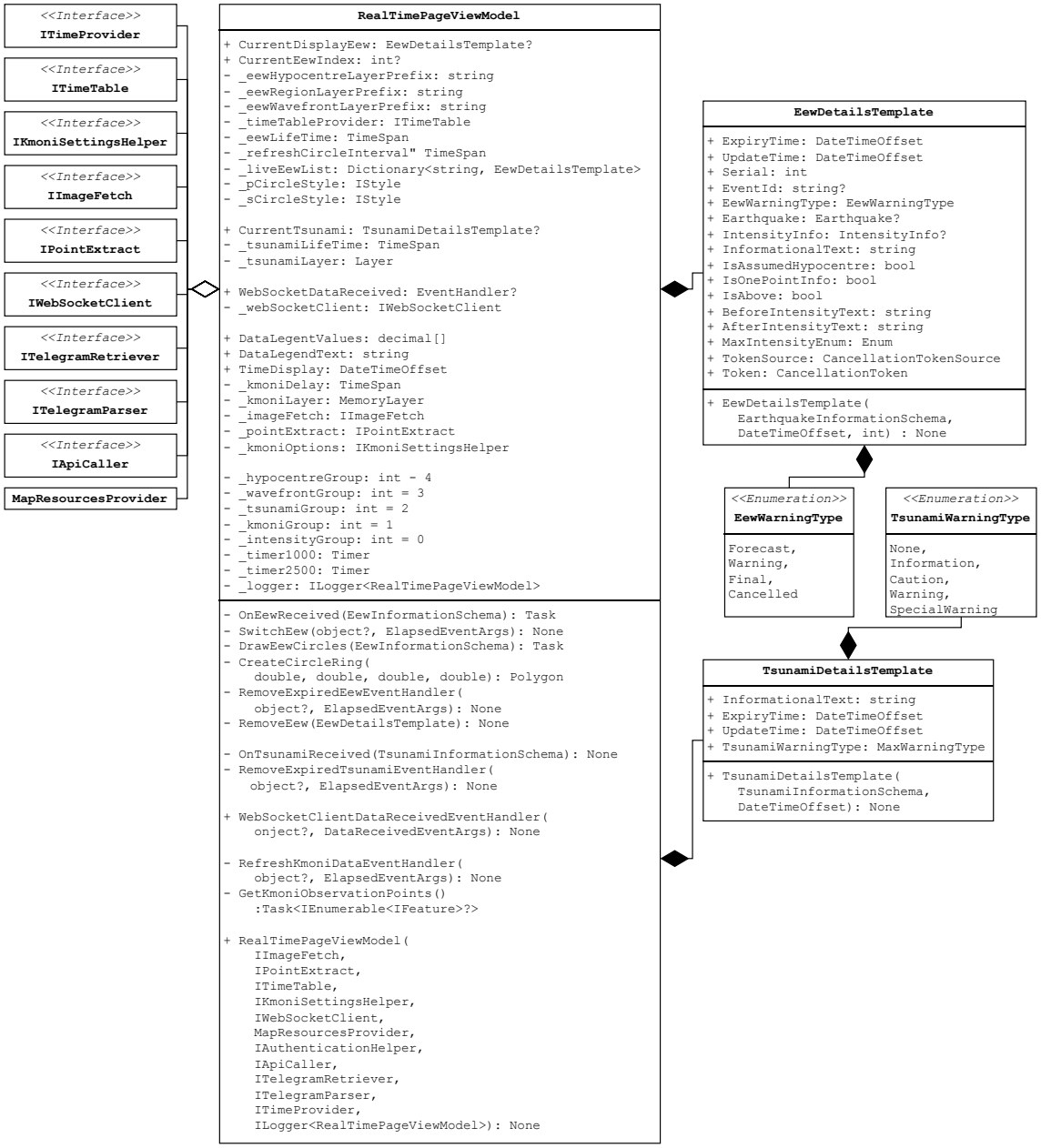


Figure 2.42: Class Diagram for Real-Time Page

EEW The key part of the EEW is that there could be multiple EEW in effect at the same time. However, there is one thing that is unique for each EEW: their event ID. Therefore, a suitable data structure to store this collection of EEWs is a **dictionary**, with the event ID in **string** as the key, and the EEW in `EewDetailsTemplate` as the value.

This naturally leads to switching the current display EEW. This is controlled by `CurrentEewIndex`, a nullable integer for the current EEW index, and `CurrentDisplayEew`, a nullable `EewDetailsTemplate` which has a get method using the index to display the earthquake. When the `SwitchEew` event handler is invoked, it will do the follows:

- If the number of current EEWs is zero, set the index to null;
- Else, if the current index is null, set the index to zero;
- Else, increment the current index by one, and modulo the length of the number of current EEWs.

The design of the `EewDetailsTemplate` is as follows: apart from the standard information to display, there are two properties of type string, `Before` and `AfterIntensityText`, which gives the text to display before/after the intensity (e.g. if the intensity provided in the EEW is 'above 6+', then the before text will be 'above'. In Japanese the after text will be '以上', since the sequence in the language is different.) There is also an enumeration determining which enumeration to display (if it is (6-, 6+), then it will display 6+; but if it is above 6- i.e. (6-, above), then it will display 6-). Finally, notice there is a cancellation token source and cancellation token here – this indicates whether this EEW has already been removed from the collection of EEWs, to allow for asynchronous operation to halt.

The type of the EEW is described by the enumeration `EewWarningType`, and the logic of deciding the type of the EEW was discussed before in the design of the UI.

The handler for receiving an EEW is `OnEewReceived`. First, it determines if the EEW is newer than the current EEW in the collection, with the same event ID (making the decision based on the serial number). If not, then it will do nothing; if yes, then it will remove the EEW in display. Then, it will plot the coordinates of the hypocentre on the map, using the corresponding shape style depending on whether the hypocentre is assumed, and also display the regional intensities colouring on the maps. Finally, it will start a long-running background task on a separate thread to draw the wavefronts for the EEW.

This is the asynchronous `DrawEewCircles`. First it checks if the EEW fits the condition to draw the wavefronts (e.g. is the earthquake, hypocentre, origin null, or is it an assumed hypocentre). If not, it will then check if the depth is within the acceptable range. If all the checks passed, it will then insert two layers onto the map, one for the P-Wave and another for the S-Wave, which will be re-used throughout the duration of the EEW, updating only the features.

Then there is a while loop, which will continue until the cancellation is requested. During the loop, it will find the time that the earthquake has elapsed, and find the distance the seismic waves has travelled through by checking with `ITimeTable`, and adding the feature onto the layer, and notifying the layer that its data has changed. The loop will sleep for a certain amount of time, and then repeat the process. After quitting the loop, the layers will be removed from the map.

`CreateCircleRing` is a method designed to draw a circle in `Polygon` at the given position and radius, to a good quality. Basic geometry is used here, but since the projection of the map changes the length, a magic constant is introduced to the radius to correctly plot the circle with approximately the correct radius.

The method to remove an EEW is as follows (assuming the EEW is removed from the collection): the cancellation token source will be cancelled and disposed of, and then the maps layer of hypocentre and regions, with the correct event ID attached to, will be removed. This will also be regularly called by a method which is invoked by the timer to regularly iterate through the dictionary to see if there are any expired EEWs.

Tsunami Since it is almost certain that there is only one tsunami warning in effect at one time, the model `CurrentTsunami` is simply a single object of type `TsunamiDetailsTemplate?`. The tsunami layer is also initialised once and used throughout the lifetime of the application, since the data source (shape file) stays the same, and only the colouring of the shape file (the style) changes.

The first method involved here is `OnTsunamiReceived`, which handles the tsunami when it is received. It is in charge of finding the valid date time of the tsunami (which is as indicated on the telegram, or if not then `_tsunamiLifeTime` which is 2 days in the program), and setting the `CurrentTsunami` object and changing the style of the map.

The second method involved is `RemoveExpiredTsunamiEventHandler` which is subscribed to the one-second timer. It checks if the current tsunami has expiry time before the current time (as indicated by the time provider), and if so, set the current tsunami to null and set the layer style to null as well.

In terms of the models involved, `TsunamiDetailsTemplate` is a record class that holds the details of the tsunami, including its informational text, expiry time, update time, and maximum tsunami warning type of the tsunami warnings of shorelines issued (using the **max** method in **LINQ**, which is an **aggregate**). The tsunami warning type is an enumeration, containing the tsunami warning type level. JMA uses a code to describe the tsunami warning type, so a conversion method is provided as well.

WebSocket This part consists of two functionalities: handling the WebSocket data received event (passing it to the appropriate handler for EEW or for Tsunami), and a separate event indicating data received from WebSocket (requesting main window to switch view model to the real-time page) named `WebSocketDataReceived`. The former is achieved by subscribing to the event in the WebSocket client (which is done in the constructor), and the latter will be raised in the event handler. Notice that subscription to the event raised by this will be done in the window view model.

Kmoni Monitor The first functionality this part achieves is displaying the **legend** for the colours of the Kmoni layer. This consists of two parts: the text for the legend in string `DataLegendtext`, displaying the type and units of the data measured, and the values on the scales of the legend in a decimal array `DataLegentValues`. The latter involves **list manipulation** using **LINQ method select** to **map** normalised height to the scale of the data that is concerned.

The second functionality is to display the Kmoni data on the map. To improve performance, the same layer `_kmoniLayer` is used, and only the data within the layer will be updated to be reflected on the map. To prevent 404 requests, a time delay is in `_kmoniDelay` which is subtracted from the current time when request is sent to fetch the image and extract the points. This is done by `RefreshKmoniDataEventHandler`, which fetches the Kmoni data, and updates the layer if the fetch is successful. The `GetKmoniObservationPoints` is the method called which actually fetches the data and converts them to the feature to display on the map.

2.5.4 Design of DTOs (Record Classes)

2.5.4.1 Common (Reusable) DTOs

In principle, the DTOs should be contained in the namespace for the abstraction requiring the use of the DTO, such as the API calls, the telegrams, or the WebSockets, and exposed to the user where necessary. However, there are some DTOs to be reused, primarily because they are shared across the different parts of the DM-D.S.S. API, WebSocket and telegram services.

JSON Components DM-D.S.S. designed **JSON components** for the application [16], which improves the reusability of typed records over the application. These components include:

- The description of the earthquake; which further incorporates:
  - The hypocentre (including the depth and the coordinate);
  - The magnitude;
- The coordinate position of some property.

This makes the records highly reusable, and these will be reused for the API calls (since it is included in the earthquake list and event calls), and the telegrams as well.

Enumerations are used wherever possible. In this case, it means that strings which can only take finitely-many enumerable values will be modelled as an enumeration. In this component, it means that:

- The condition for the depth (very shallow ' ごく浅い', very deep ' ７００ｋｍ以上', unclear ' 不明') are modelled with `DepthCondition`;
- The condition for the magnitude (unknown ' M不明', very big ' Ｍ８を超える巨大地震') are modelled with `MagnitudeCondition`;

- The unit for the magnitude (JMA magnitude 'Mj', normal magnitude 'M') are modelled with `MagnitudeUnit`;
- The source for the information of the earthquake (the earthquake advisory centres available) are modelled with `Source`;
- The geodetic coordinate used (world geodetic coordinate '世界測地系', Japanese geodetic coordinate '日本測地系') are modelled with `Geodetic`.

Telegrams Several components of the telegram are shared between different parts of the DM-D.S.S. functionality. Those include:

- `XmlReport` which composes of `XmlHead`, `XmlControl`, and describes the key components of the XML telegram. This is used within the API call for the earthquake event, and the data response from the WebSocket.
- `SchemaVersionInformation`, which includes the type and the version for the JSON schema used. It is shared with the API component (because in each telegram retrieved for earthquake event the schema will be included), and for parsing the telegram.
- `TelegramStatus` and `TelegramType`, which describes the status and the type of the telegram. They are composed within `XmlHead`, and for `Head` in the telegram component as well (described later).

Note that `TelegramStatus` and `TelegramType` are modelled as enumerations, since they only take a finite number of values, although provided by string from the original API.

Enumerations There are also a couple of other shared enumerations within the DM-D.S.S. services. They are all modelled from strings which can only take finitely many values.

- `Classification`, which could be used to describe the classification for a contract that user subscribed to, or the classification of a telegram.
- `Intensity` for earthquake intensity, `LgCategory` for LPGM motion category, and `LgIntensity` for LPGM intensity;
- `FormatType` and `TestStatus` for describing the format of a particular telegram (JSON, XML, Alpha Numeric or Binary) and the test status (whether it is a test or not), both used for setting up the WebSocket and in the WebSocket service.

2.5.4.2 API Calls

Since all API Calls must share the response ID, response time, and response status properties, this could be modelled as an **abstract base class** (record class) for the API calls. This is the `ApiBase` class, on the top of the diagram, composing of an enumeration for the status of the class.

Inheriting from it are two types of responses: a **successful** response (which is abstract as well), and an **error** response (which is concrete). The error response composes the details of the error within as a DTO.

There are three responses further abstracted, each inheriting from the previous one:

- a **list** response, which contains a `ItemList` containing all the items in the `item` property in the serialised JSON;
- a **token** response, which contains a `NextToken` property for the next API call; and
- a **pooling** response, which contains the next pooling and next pooling interval properties for the next API call.

As described in the previous behaviour of the API responses:

- `WebSocketStart` and `GdEarthquakeEvent` response are success responses and compose f only the details of the response;

- `ContractList` and `EarthquakeParameter` contains a list of contracts/stations within the response. In the parameter response, in addition to this, it also contains the last changed time for the data, and the version information;
- `WebSocketList` contains a token but not a pooling token, and contains a list of WebSocket details; and finally
- `GdEarthquakeList` contains a list of past earthquake information, and is a pooling response.

This design is accurately reflected in Figure 2.43 which shows a clear use of **inheritance, composition and abstract classes**.

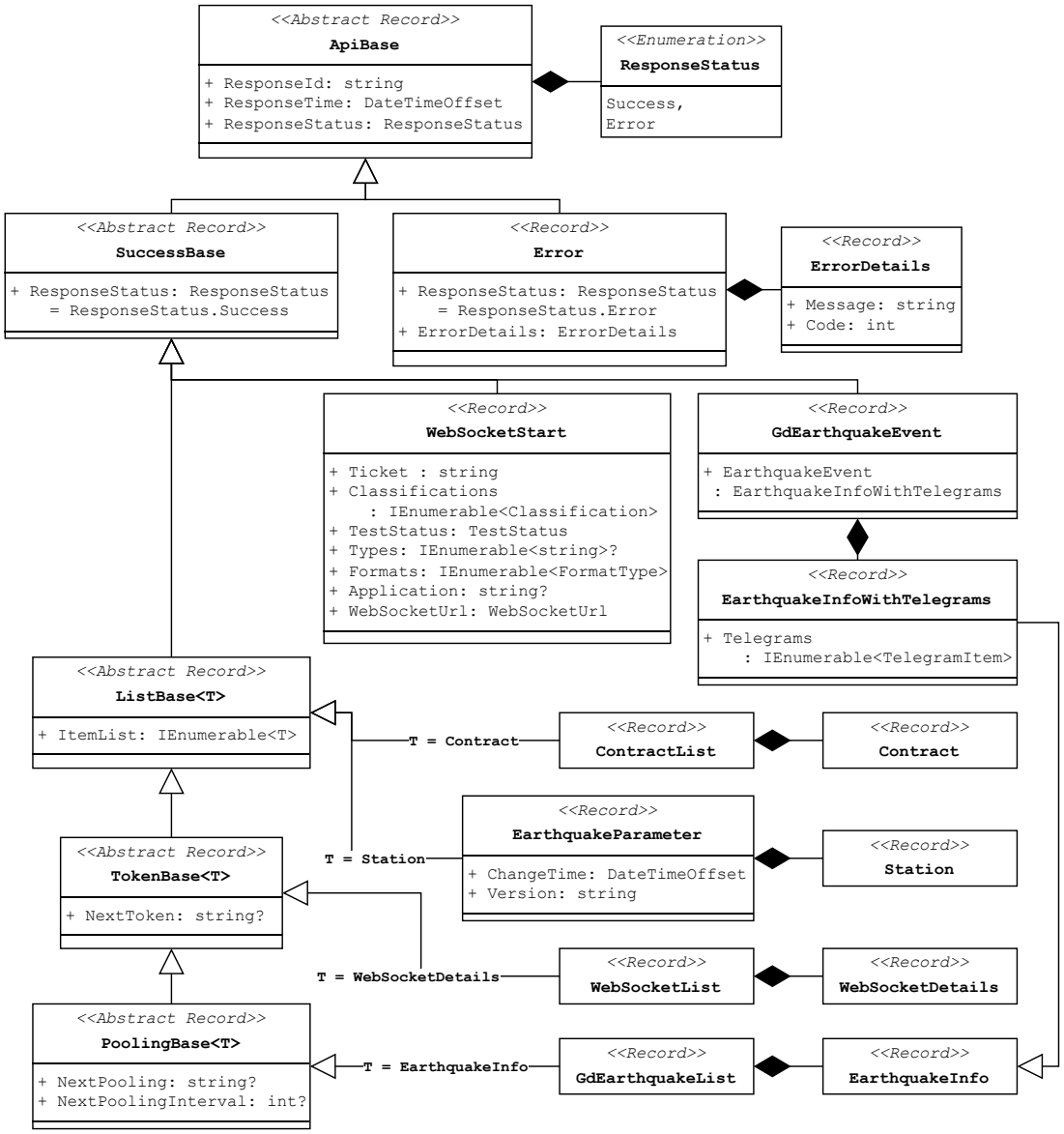


Figure 2.43: Class Diagram for DTOs for API Calls

Most other parts of the DTOs for API Calls is just simply the use of composition as modelled by the DM-D.S.S. API documentation, and the use of enumerations to model 'finitely many values'.

2.5.4.3 Telegrams

Similar to the API calls, a base class `Head` which contains the head information of the telegram is modelled. And then, classes are inherited from this class to provide the body part of the telegram. The reason why `Head` is not abstract, is because telegrams would have to be deserialised to it first and find the JSON schema, and then re-deserialised to the specific type of the telegram.

This is reflected in Figure 2.44 which shows a clear use of **inheritance and composition**.

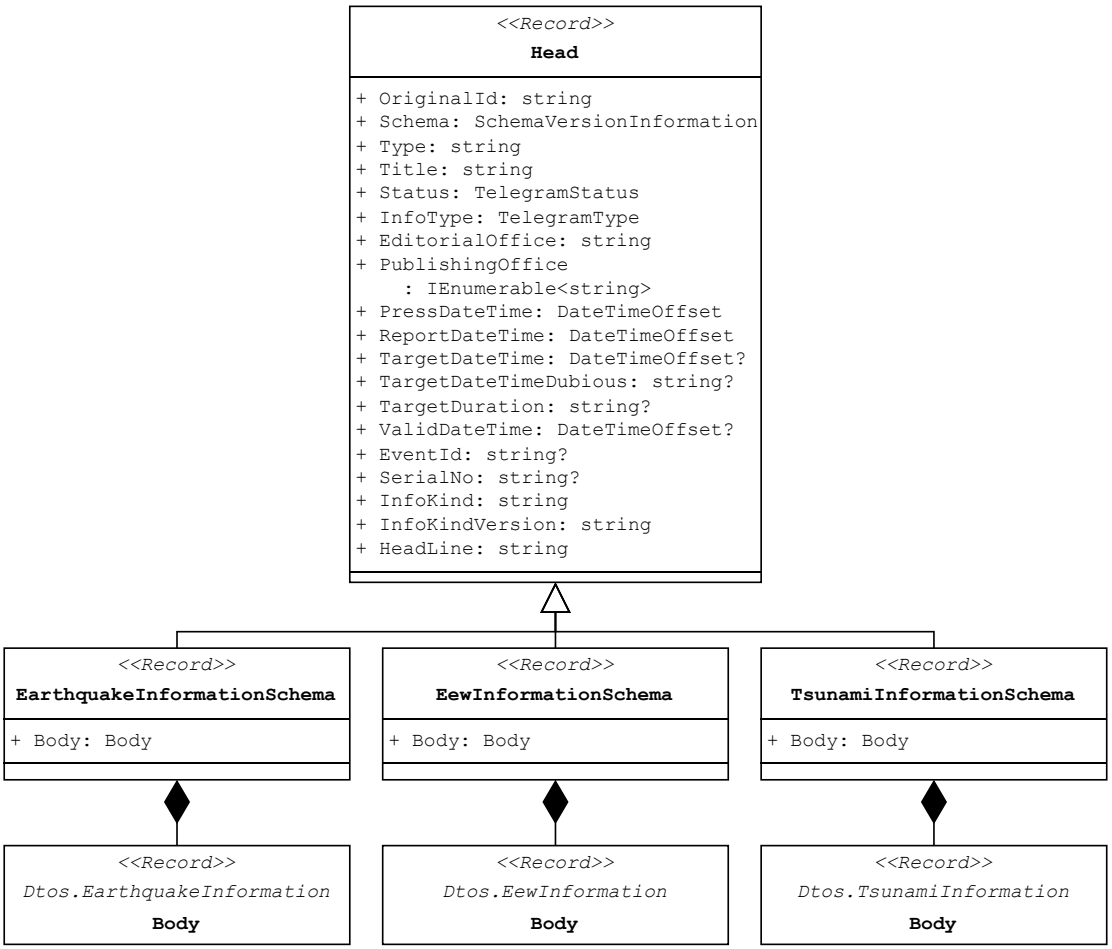


Figure 2.44: Class Diagram for DTOs for Telegrams

Otherwise, this it simply the use of composition as modelled by the DM-D.S.S. JSON Documentation [19].

2.6 Hardware and Software Requirements

Table 2.14 outlines the expected basic hardware requirements for the application.

Type	Minimum	Recommended
RAM	4 GB	8 GB
Storage Device	HDD	SSD
CPU	Intel Core 8th Gen (or equiv.)	Intel Core 10th Gen (or equiv.)
Display Resolution	720p	1080p
Display Size	9”	12”

Table 2.14: Hardware Specification for App

The amount of RAM is due to the amount of data that is processed (and considering other applications running as well). Storage space is not a substantial requirement of this application, while the CPU has to be of high standards to process all the data. To display the application properly (with appropriate size), a display of 1080p 12" is recommended.

It will be able to run on up-to-date Windows, macOS and Linux distributions (both x64 and ARM) due to the cross-platform nature of .NET, but out-of-date x86 platforms will not be explicitly supported.

The author uses a macOS 15 (beta) machine with 2.5K 13" display, Intel Core i5-1038NG7 (with Intel Iris Plus Graphics) and 16 GB of RAM (MacBook Pro 2020, 4 Thunderbolt Ports) and a Windows 11 (beta) machine with 2.5K 15" display, AMD Ryzen 7 5800H and 32 GB of RAM (with RTX 3070 for Laptop) (Legion R9000K 2021) to test the application, and with 2.5K 24" external display as well. An Ubuntu 22.04 virtual machine will be used to test the compatibility for Linux systems.

The application will be self-contained (i.e. comes with .NET runtime) to prevent the user from unnecessary technical complications.

The device needs to have stable connection to the internet using Wi-Fi/Ethernet/other means, to establish connection with the relevant APIs.

Chapter 3

Technical Implementation

3.1 Key Code Segments

3.1.1 Data structures

Implementation of ADTs and OOP Classes to be demonstrated.

3.1.2 Modularity

Code should be created and tested in separate modules that are integrated later. Use subheadings for each module, define the purpose of the module, and show unit testing of the module.

3.1.3 Defensive Programming/Robustness

Exception handling

Chapter 4

Testing

Consider how you will test your project. You should devise a test strategy that encompasses a range of methods.

4.1 Test Strategy

- Unit testing (of individual functions)
- Integration testing (e.g. different modules/class files)
- Robustness (demonstrating defensive programming skills/exception handling)
- Requirements testing (against your initial requirements - a table with test number, description, test data, expected result, evidence (screenshot/video time link) would be suitable)
- Independent end user beta testing (this will assist with your evaluation)

4.2 Testing Video

- You can include a video to assist (but you will need to reference the time point at which relevant evidence appears)
- If you include a video you will need to have it publicly available.
- It is suggested that you include a QR code in your testing to give a link to it the video (for the moderator) rather than just giving a long URL on its own.

4.3 System Tests (against original requirements' specification)

You need to give evidence in support of requirements that have been met e.g. reference to a relevant test/screenshot/relevant code.

Requirement №	Description	Success Criteria	Tests + Evidence

Table 4.1: Table of Tests.

Chapter 5

Evaluation

5.1 Requirements Specification Evaluation

Personal evaluation

- Copy and paste your original requirements from your project analysis
- You need to review each requirement and comment objectively on whether it was *fully met/partially met/not met*.

Requirement №	Description	Success Criteria	Fully/Partial/Not met (Reflective Comment)

Table 5.1: Table of Evaluation.

5.2 Independent End-User Feedback

End user/client evaluation

- there **must** be meaningful end user feedback
- You should hold a review meeting with your end user
- Write down any key feedback that they give you. E.g. Agreement that a particular requirement has been meet/comments as to aspects that they find suboptimal/comments as to additions they would like to see

Requirement №	Description	Acceptance Y/N	Additional Comments

Table 5.2: Table of Feedback.

5.3 Improvements

You need to give consideration to a number of potential future improvements that could be made. They may arise from either your experience or from feedback given to you by your end user. Ideally at least one should be in response to end user feedback.

- Write a paragraph for each potential improvement/change
- The improvements/changes could result from additional functionality that has been identified as being beneficial or could be as a result of required efficiencies if some processes are clunky or require faster run-times
- You should then comment on how the proposed change could be implemented moving forward. i.e. what would need to be changed/developed and how? You are not expected to actually make any changes; just comment on the possibilities.

Appendix A

Code Listing

```
# %% [markdown]
# # Polynomial fitting for $f : [0, 1] \to \mathcal{C}$

# %%
from PIL import Image, PyAccess # for image reading
import matplotlib.pyplot as plt # to plot graphs
from scipy import optimize # to do fitting

plt.rcParams['text.usetex'] = True # LaTeX in plots
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.family'] = 'STIXGeneral'

# %% [markdown]
# ## Plot $C = (H, S, V)$

# %%
orig_img_path: str = 'jma-scale.png'
orig_img: Image.Image = Image.open(orig_img_path) # read original image

orig_img_hsv: Image.Image = orig_img.convert('HSV') # convert to HSV colour format

img_width: int
img_height: int
img_width, img_height = orig_img.size # get size of image

img_width, img_height

# %%
# read the (H, S, V) in a fixed column

column: int = 5
hsv_list: list[tuple[int, int, int]] = [(orig_img_hsv.getpixel((column, y))) for y
↪  in range(img_height)] # type: ignore

hsv_list

# %% [markdown]
# ### Plot $f_H, f_S, f_V$ against $r$

# %%
r_list: list[int] = list(range(img_height))

# split up list into separate lists
```

```
hsv_h_list: list[int] = [c[0] for c in hsv_list]
hsv_s_list: list[int] = [c[1] for c in hsv_list]
hsv_v_list: list[int] = [c[2] for c in hsv_list]

# %%
# setup options for plotting graphs

marker_size: int = 15
marker_char: str = 'x'
line_width: float = 0.3

fig_size: tuple[int, int] = (10, 5)

h_colour: str = 'turquoise'
s_colour: str = 'gold'
v_colour: str = 'gray'

h_label: str = 'Hue $H$'
s_label: str = 'Saturation $S$'
v_label: str = 'Value $V$'

# %%
def init_plt(xlabel: str, ylabel: str, title: str) -> None:
    plt.figure(figsize = fig_size)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(True)
    plt.tight_layout()

# %%
# plot graph of (H, S, V) against r

init_plt('Row Number $r$', '$(H, S, V)$ Values', f'$(H, S, V)$ Values against Row
↪  Number $r$ along column {column}')

plt.scatter(r_list, hsv_h_list, label=h_label, color=h_colour, s=marker_size,
↪  marker=marker_char, linewidths=line_width)
plt.scatter(r_list, hsv_s_list, label=s_label, color=s_colour, s=marker_size,
↪  marker=marker_char, linewidths=line_width)
plt.scatter(r_list, hsv_v_list, label=v_label, color=v_colour, s=marker_size,
↪  marker=marker_char, linewidths=line_width)

plt.legend()

plt.savefig('hsv-against-row.png')

# %%
plt.close()

# %% [markdown]
# ### Plot $f_H, f_S, f_V$ against $h$

# %%
# setup the initial values for h and r

h_0 = 18
h_1 = 296 + 1
r_len = h_1 - h_0
```

```
def r_to_h(r: int) -> float:
    return 1 - ((r - h_0) / (r_len - 1))

h_list: list[float] = [r_to_h(r) for r in r_list] # convert list of r to list of h

h_r_list: list[float] = h_list[h_0:h_1] # extract the values of h within range

hsv_h_r_list: list[int] = hsv_h_list[h_0:h_1] # extract the values of (H, S, V)
↪  within range
hsv_s_r_list: list[int] = hsv_s_list[h_0:h_1]
hsv_v_r_list: list[int] = hsv_v_list[h_0:h_1]

h_r_list

# %%
# plot graph of (H, S, V) against h

init_plt('$h$', '$(H, S, V)$ Values', f'$(H, S, V)$ Values against $h$ along column
↪  {column}')

plt.scatter(h_r_list, hsv_h_r_list, label=h_label, color=h_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.scatter(h_r_list, hsv_s_r_list, label=s_label, color=s_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.scatter(h_r_list, hsv_v_r_list, label=v_label, color=v_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)

plt.legend()

plt.savefig('hsv-against-h.png')

# %%
plt.close()

# %% [markdown]
# ## Regression

# %% [markdown]
# ### Prepare Lists to do Regression

# %%
# normalise values of h, s, v to our desired values

hsv_h_n_list: list[float] = [hsv_h_r_list[i] / 255 * 360 for i in range(279)]
hsv_s_n_list: list[float] = [hsv_s_r_list[i] / 255 * 1 for i in range(279)]
hsv_v_n_list: list[float] = [hsv_v_r_list[i] / 255 * 1 for i in range(279)]

# %% [markdown]
# ### Regression on f_H

# %%
# define f_H with unkwnown parameters y_1, y_2 which operates on a list

def f_h_params(xl: list[float], y_1: float, y_2: float) -> list[float]:
    yl: list[float] = [0 for _ in xl]
    for i, x in enumerate(xl):
        if 0 <= x <= 0.1:
            yl[i] =  -150 * x + 237
```

```
        elif 0.1 <= x <= 0.6:
            yl[i] = (
                ((222  * (x - 0.3) * (x - 0.4) * (x - 0.6)) / ((0.1 - 0.3) * (0.1 -
                ↪  0.4) * (0.1 - 0.6))) +
                ((y_1 * (x - 0.1) * (x - 0.4) * (x - 0.6)) / ((0.3 - 0.1) * (0.3 -
                ↪  0.4) * (0.3 - 0.6))) +
                ((y_2 * (x - 0.1) * (x - 0.3) * (x - 0.6)) / ((0.4 - 0.1) * (0.4 -
                ↪  0.3) * (0.4 - 0.6))) +
                ((51   * (x - 0.1) * (x - 0.3) * (x - 0.4)) / ((0.6 - 0.1) * (0.6 -
                ↪  0.3) * (0.6 - 0.4)))
            )
        elif 0.6 <= x <= 0.9:
            yl[i] = -170 * x + 153
        elif 0.9 <= x <= 1:
            yl[i] = 0
        else:
            yl[i] = 0
    return yl

# %%
h_init: list[int] = [0, 0] # initial guess to indicate number of paameters

h_params: list[int]

h_params, _ = optimize.curve_fit(f_h_params, h_r_list, hsv_h_n_list, p0=h_init) #
↪  optimised parameters

h_params

# %%
def f_h(x) -> list[float]: # this is a partial function which gives in the fitted
↪  parameters
    return f_h_params(x, *h_params)

h_p_list: list[float] = [i / 1000 for i in range(0, 1001, 1)] # list to make sure
↪  the plotted graph looks quite smooth
hsv_h_fitted_list: list[float] = f_h(h_p_list) # fitted line

# %%
line_width = 1.25 # make the markers look a bit thicker

# %%
# plot result of fit

init_plt('$h$', h_label, '$H$ against $h$')

plt.scatter(h_r_list, hsv_h_n_list, label=h_label, color=h_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.plot(h_p_list, hsv_h_fitted_list, label='Fit')

plt.legend()
plt.savefig('h-against-h.png')

# %%
plt.close()

# %% [markdown]
# ### Regression on $f_S$
```

```
# %%
# define f_S

def f_s(xl: list[float]) -> list[float]:
    yl: list[float] = [0 for _ in xl]
    for i, x in enumerate(xl):
        if 0 <= x <= 0.2:
            yl[i] = 1
        elif 0.2 <= x <= 0.29:
            yl[i] = -2.611 * x + 1.522
        elif 0.29 <= x <= 0.4:
            yl[i] = 1.682 * x + 0.277
        elif 0.4 <= x <= 0.5:
            yl[i] = 0.5 * x + 0.75
        elif 0.5 <= x <= 1:
            yl[i] = 1
        else:
            yl[i] = 0
    return yl

hsv_s_fitted_list: list[float] = f_s(h_p_list) # fitted line

# %%
# plot result of fit

init_plt('$h$', s_label, '$S$ against $h$')

plt.scatter(h_r_list, hsv_s_n_list, label=s_label, color=s_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.plot(h_p_list, hsv_s_fitted_list, label='Fit')

plt.legend()
plt.savefig('s-against-h.png')

# %% [markdown]
# ### Regression on f_V

# %%
# define f_V

def f_v(xl: list[float]) -> list[float]:
    yl: list[float] = [0 for _ in xl]
    for i, x in enumerate(xl):
        if 0 <= x <= 0.1:
            yl[i] = 1.8 * x + 0.8
        elif 0.1 <= x <= 0.172:
            yl[i] = -4.444 * x + 1.424
        elif 0.172 <= x <= 0.2:
            yl[i] = 5.714 * x - 0.323
        elif 0.2 <= x <= 0.3:
            yl[i] = 1.6 * x + 0.5
        elif 0.3 <= x <= 0.4:
            yl[i] = 0.2 * x + 0.92
        elif 0.4 <= x <= 0.8:
            yl[i] = 1
        elif 0.8 <= x <= 0.9:
            yl[i] = -0.3 * x + 1.24
        elif 0.9 <= x <= 1:
            yl[i] = -2.9 * x + 3.58
```

```
        else:
            yl[i] = 0
    return yl

hsv_v_fitted_list: list[float] = f_v(h_p_list) # fitted line

# %%
# plot result of fit

init_plt('$h$', v_label, '$V$ against $h$')

plt.scatter(h_r_list[1:r_len - 1], hsv_v_n_list[1:r_len - 1], label=v_label,
↪  color=v_colour, s=marker_size, marker=marker_char, linewidth=line_width) # the
↪  first and last points are excluded from the plot
plt.plot(h_p_list, hsv_v_fitted_list, label='Fit')

plt.legend()
plt.savefig('v-against-h.png')

# %% [markdown]
# ## Back-Generate the image

# %%
# setup an HSV formatted image and get its pixels

image_generated: Image.Image = Image.new('HSV', (img_width, img_height))
generated_pixels: PyAccess.PyAccess = image_generated.load() # type: ignore

# returns a list of tuples of (H, S, V) for a column

def f(yl: list[int]) -> list[tuple[int, int, int]]:
    hl: list[float] = [r_to_h(y) for y in yl]

    hsv_h: list[float] = f_h(hl)
    hsv_s: list[float] = f_s(hl)
    hsv_v: list[float] = f_v(hl)

    hsv_h_i: list[int] = [int(h / 360 * 255) for h in hsv_h]
    hsv_s_i: list[int] = [int(s * 255) for s in hsv_s]
    hsv_v_i: list[int] = [int(v * 255) for v in hsv_v]

    return [(hsv_h_i[i], hsv_s_i[i], hsv_v_i[i]) for i in yl]

my_pixels: list[list[tuple[int, int, int]]] = [f(list(range(img_height))) for _ in
↪  range(img_width)]

# copy this into the generated_pixels reference

for x in range(img_width):
    for y in range(img_height):
        generated_pixels[x, y] = my_pixels[x][y]

# convert to RGB and output

image_generated_rgb: Image.Image = image_generated.convert('RGB')
image_generated_rgb.save('generated-colour.png')
```

Listing A.1: Code for Polynomial Fit of Colour

```
using System;
using System.Net.WebSockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        using (ClientWebSocket webSocket = new ClientWebSocket())
        {
            Uri serverUri = new Uri("wss://example.com/socket"); // Replace with
            ↪  your WebSocket server URI

            try
            {
                Console.WriteLine("Connecting to WebSocket...");
                await webSocket.ConnectAsync(serverUri, CancellationToken.None);
                Console.WriteLine("Connected!");

                // Sending a message
                string messageToSend = "Hello, WebSocket!";
                byte[] messageBytes = Encoding.UTF8.GetBytes(messageToSend);
                await webSocket.SendAsync(new ArraySegment<byte>(messageBytes),
                ↪  WebSocketMessageType.Text, true, CancellationToken.None);
                Console.WriteLine($"Sent: {messageToSend}");

                // Receiving a message
                byte[] buffer = new byte[1024];
                WebSocketReceiveResult result = await webSocket.ReceiveAsync(new
                ↪  ArraySegment<byte>(buffer), CancellationToken.None);
                string messageReceived = Encoding.UTF8.GetString(buffer, 0,
                ↪  result.Count);
                Console.WriteLine($"Received: {messageReceived}");

                // Closing the WebSocket
                Console.WriteLine("Closing WebSocket...");
                await webSocket.CloseAsync(WebSocketCloseStatus.NormalClosure,
                ↪  "Goodbye", CancellationToken.None);
                Console.WriteLine("WebSocket closed.");
            }
            catch (WebSocketException ex)
            {
                Console.WriteLine($"WebSocket error: {ex.Message}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Unexpected error: {ex.Message}");
            }
        }
    }
}
```

Listing A.2: Code generated by ChatGPT in [59]

Appendix B

Use of AI

The detailed prompt of the use of ChatGPT 4o [59] is as in Figure B.1, Figure B.2 and Figure B.3. For the ease of reference, the code is also included in A.2.

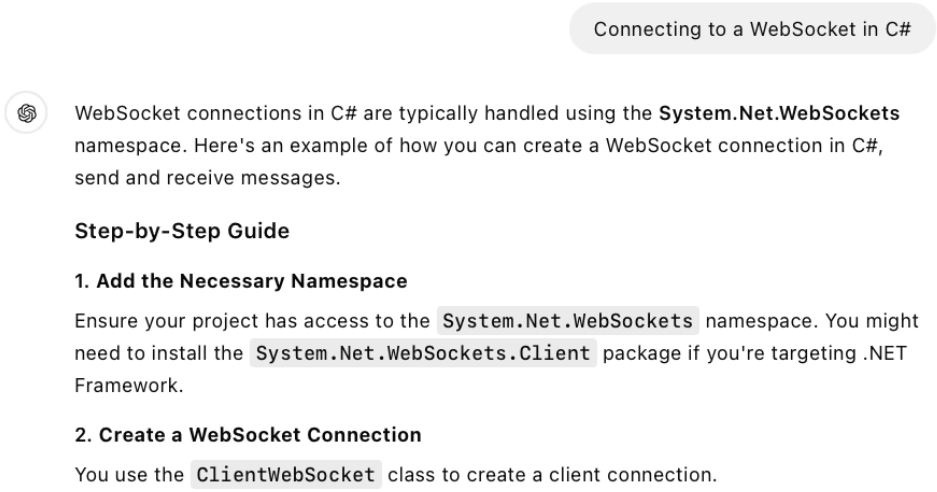
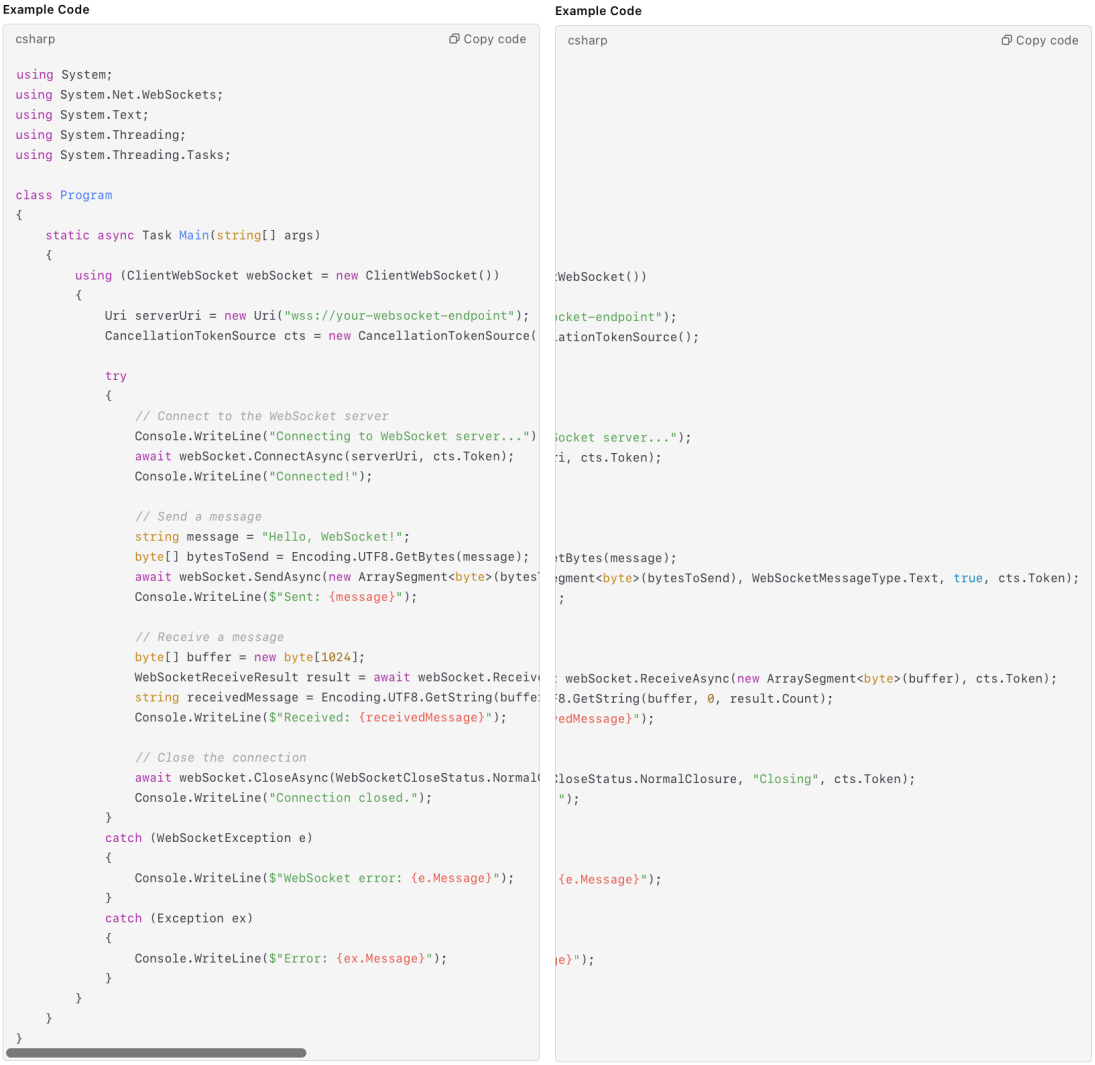


Figure B.1: Prompt to ChatGPT in [59], Part 1

This is used to outline a working example of `System.Net.WebSocket` class and the author made numerous edits and additions to eventually create commit `8f8f6e9` on the same day, and other references such as StackOverflow [20, 23] and the official Microsoft Documentation [49] has also been used to create the relevant code in WebSocket connections.

This code has eventually been refactored and removed in `60b660a`, still referencing the sources [20, 23].



Example Code

csharp Copy code

```csharp
using System;
using System.Net.WebSockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        using (ClientWebSocket webSocket = new ClientWebSocket())
        {
            Uri serverUri = new Uri("wss://your-websocket-endpoint");
            CancellationTokenSource cts = new CancellationTokenSource(

            try
            {
                // Connect to the WebSocket server
                Console.WriteLine("Connecting to WebSocket server...")
                await webSocket.ConnectAsync(serverUri, cts.Token);
                Console.WriteLine("Connected!");

                // Send a message
                string message = "Hello, WebSocket!";
                byte[] bytesToSend = Encoding.UTF8.GetBytes(message);
                await webSocket.SendAsync(new ArraySegment<byte>(bytes
                Console.WriteLine($"Sent: {message}");

                // Receive a message
                byte[] buffer = new byte[1024];
                WebSocketReceiveResult result = await webSocket.Receiv
                string receivedMessage = Encoding.UTF8.GetString(buffe
                Console.WriteLine($"Received: {receivedMessage}");

                // Close the connection
                await webSocket.CloseAsync(WebSocketCloseStatus.Normal(
                Console.WriteLine("Connection closed.");
            }
            catch (WebSocketException e)
            {
                Console.WriteLine($"WebSocket error: {e.Message}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
}
```

Example Code

csharp Copy code

```csharp
:WebSocket())

ocket-endpoint");
.ationTokenSource();

ocket server...");
ri, cts.Token);

tBytes(message);
gment<byte>(bytesToSend), WebSocketMessageType.Text, true, cts.Token);
;

webSocket.ReceiveAsync(new ArraySegment<byte>(buffer), cts.Token);
8.GetString(buffer, 0, result.Count);
edMessage}");

loseStatus.NormalClosure, "Closing", cts.Token);
");

{e.Message}");

e}");
```

Figure B.2: Prompt to ChatGPT in [59], Part 2

Explanation of Key Parts

1. `ClientWebSocket`: This is the main class for handling WebSocket connections.
2. `ConnectAsync`: Establishes the WebSocket connection to the specified URI.
3. `SendAsync`: Sends data to the WebSocket server.
4. `ReceiveAsync`: Receives data from the WebSocket server.
5. `CloseAsync`: Closes the WebSocket connection gracefully.
6. **Buffer Management**: The buffer is a byte array used to store incoming data temporarily.

Notes

- Replace `"wss://your-websocket-endpoint"` with your actual WebSocket server URI.
- Ensure you handle exceptions like `WebSocketException` for scenarios where the connection might fail.
- Use proper cancellation tokens to handle task cancellation in production code.

This code provides a basic framework for working with WebSocket connections in C#. You can expand it based on your application's needs.

Figure B.3: Prompt to ChatGPT in [59], Part 3

Appendix C

Bibliography

[1] E. Aparicio. *DemoMapsui*. 2024. URL: https://github.com/dev-elian/DemoMapsui.

[2] E. Aparicio. *MVVM with Maosui | Maps with .NET*. 3rd July 2024. URL: https://www.youtube.com/watch?v=5zcDIxUsV_Y (visited on 10/03/2025).

[3] Avalonia, ed. *Avalonia Docs*. URL: https://docs.avaloniaui.net/docs/welcome (visited on 10/03/2025).

[4] Avalonia, ed. *Built-in Controls*. URL: https://docs.avaloniaui.net/docs/reference/controls/ (visited on 10/03/2025).

[5] Avalonia. *Fluent Icons for Avalonia*. URL: https://avaloniaui.github.io/icons.html (visited on 10/03/2025).

[6] Avalonia, ed. *How to Create a Custom Data Binding Converter*. URL: https://docs.avaloniaui.net/docs/guides/data-binding/how-to-create-a-custom-data-binding-converter (visited on 10/03/2025).

[7] Avalonia, ed. *How To Log Errors and Warnings*. URL: https://docs.avaloniaui.net/docs/guides/implementation-guides/logging-errors-and-warnings (visited on 10/03/2025).

[8] Avalonia, ed. *How To Use Design-time Data*. URL: https://docs.avaloniaui.net/docs/guides/implementation-guides/how-to-use-design-time-data (visited on 10/03/2025).

[9] Avalonia, ed. *How to use INotifyPropertyChanged*. URL: https://docs.avaloniaui.net/docs/guides/data-binding/inotifypropertychanged (visited on 10/03/2025).

[10] Avalonia, ed. *Localizing using ResX*. URL: https://docs.avaloniaui.net/docs/guides/implementation-guides/localizing (visited on 10/03/2025).

[11] *Balsamiq*. Version 4.8.0. 2024. URL: https://balsamiq.com.

[12] BillWagner et al., eds. *Documentation comments*. 15th June 2023. URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/ (visited on 10/12/2024).

[13] M. Bloch. *mapshaper*. 2025. URL: https://mapshaper.org/.

[14] dmdata.jp, ed. *API v2*. 1st Dec. 2024. URL: https://dmdata.jp/docs/reference/api/v2/ (visited on 07/12/2024).

[15] dmdata.jp, ed. *EEW について*. 7th Feb. 2025. URL: https://dmdata.jp/docs/eew/ (visited on 10/03/2025).

[16] dmdata.jp, ed. *JSON 化データについて*. 7th Feb. 2025. URL: https://dmdata.jp/docs/reference/conversion/json/ (visited on 10/03/2025).

[17] dmdata.jp, ed. *OAuth 2 v1*. 1st Dec. 2024. URL: https://dmdata.jp/docs/reference/oauth2/v1 (visited on 10/12/2024).

[18] dmdata.jp, ed. *サンプル JavaScript*. 1st Dec. 2024. URL: https://dmdata.jp/docs/sample/javascript (visited on 10/12/2024).

[19] dmdata.jp, ed. *電文データ配信データの区分*. 17th Feb. 2025. URL: https://dmdata.jp/docs/telegrams/ (visited on 10/03/2025).

[20] FloriUni. *Create a client websocket based console app*. 17th Jan. 2021. URL: https://stackoverflow.com/a/65761228 (visited on 22/11/2024).

[21] gewarren and BillWagner, eds. *JSON serialization and deserialization (marshalling and unmarshalling) in .NET*. 25th Oct. 2023. URL: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview (visited on 01/12/2024).

[22] gewarren et al., eds. *Handling and throwing exceptions in .NET*. 15th Sept. 2021. URL: https://learn.microsoft.com/en-us/dotnet/standard/exceptions/ (visited on 10/03/2025).

[23] Harry. *Connecting to websocket using C# (I can connect using JavaScript, but C# gives Status code 200 error)*. Ed. by kevinarpe. 25th Aug. 2020. URL: https://stackoverflow.com/a/63574016 (visited on 22/11/2024).

[24] J. Hunter et al., eds. *Matplotlib 3.9.3 documentation*. 2002. URL: https://matplotlib.org/stable/users/index.html (visited on 07/10/2024).

[25] N. Iarocci. *How to implement a PKCE code challenge in C#*. 17th Jan. 2024. URL: https://nicolaiarocci.com/how-to-implement-pkce-code-challenge-in-csharp/ (visited on 01/01/2024).

[26] IEvangelist et al., eds. *.NET dependency injection*. 18th July 2024. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection (visited on 10/03/2025).

[27] IEvangelist et al., eds. *Configuration in .NET*. 9th Oct. 2024. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration (visited on 09/03/2025).

[28] IEvangelist et al., eds. *Handle and raise events*. 4th Oct. 2022. URL: https://learn.microsoft.com/en-us/dotnet/standard/events/ (visited on 10/03/2024).

[29] IEvangelist et al., eds. *Logging in C# and .NET*. 17th July 2024. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/logging (visited on 09/03/2025).

[30] IEvangelist et al., eds. *Make HTTP requests with the HttpClient class*. 8th Mar. 2025. URL: https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient (visited on 10/03/2025).

[31] ingen084. *【初心者向け】KyoshinMonitorLib チュートリアル*. 9th Dec. 2020. URL: https://qiita.com/ingen084/items/b9961576d1af1b140f10.

[32] ingen084. *KyoshinShindoPlaceEditor*. Version 0.0.6.0. 2018. URL: https://github.com/ingen084/KyoshinShindoPlaceEditor/.

[33] ingen084. *ShindoObsPoints.mpk.lz4*. 2024. URL: https://github.com/ingen084/KyoshinEewViewerIngen/blob/develop/src/KyoshinEewViewer/Assets/ShindoObsPoints.mpk.lz4 (visited on 08/03/2025).

[34] ingen084. *強震モニタの画像から揺れていることを検知する*. 14th Dec. 2022. URL: https://qiita.com/ingen084/items/82985e8d3227c97c608d (visited on 13/10/2024).

[35] ingen084. *強震モニタの画像から震度と地点を特定するまで*. 2nd Dec. 2020. URL: https://qiita.com/ingen084/items/7e91f8da2996972ac586#fnref2 (visited on 13/10/2024).

[36] JGraph. *diagrams.net, draw.io*. Version 15.5.2. 2021. URL: https://www.diagrams.net/.

[37] O. Kopp, C. C. Snethlage and C. Schwentker. 'JabRef: BibTeX-based literature management software'. In: *TUGboat* 44.3 (138 2023), pp. 441–447. ISSN: 0896-3207. DOI: 10.47397/tb/44-3/tb138kopp-jabref.

[38] F. Lundh, J. A. Clark and et al., eds. *Pillow (PIL Fork) 11.0.0 documentation*. 1995. URL: https://pillow.readthedocs.io/en/stable/ (visited on 07/10/2024).

[39] Mamma Mia Dev. *Avalonia UI - 03 - Sidebar Menu*. 3rd July 2024. URL: https://www.youtube.com/watch?v=UDbKVheMBY8 (visited on 10/03/2025).

[40] MammaMiaDev. *avalonia-the-series*. 2024. URL: https://github.com/MammaMiaDev/avaloniaui-the-series.

[41] Mapsui, ed. *API*. URL: https://mapsui.com/v5/api/Mapsui.html (visited on 10/03/2025).

[42] Mapsui, ed. *Logging*. URL: https://mapsui.com/documentation/logging.html (visited on 10/03/2025).

[43] Mapsui. *Mapsui*. Version 5.0.0 Beta 9. 10th Mar. 2025. URL: https://github.com/Mapsui/Mapsui.

[44] Mapsui, ed. *Mapsui Samples*. URL: https://mapsui.com/samples/ (visited on 10/03/2025).

[45] Mapsui, ed. *Mapsui Samples*. URL: https://mapsui.com/v5/samples/ (visited on 10/03/2025).

[46] Mapsui, ed. *Projections*. URL: https://mapsui.com/documentation/projections.html (visited on 10/03/2025).

[47] Matthias247. *Choosing a buffer size for a WebSocket response*. Ed. by Johan. 29th Jan. 2017. URL: https://stackoverflow.com/a/41926694 (visited on 22/11/2024).

[48] Microsoft, ed. *BindingFlags Enum*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.reflection.bindingflags?view=net-9.0 (visited on 10/03/2025).

[49] Microsoft, ed. *ClientWebSocket Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.websockets.clientwebsocket?view=net-9.0 (visited on 22/11/2024).

[50] Microsoft, ed. *HttpClient Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-9.0 (visited on 01/12/2024).

[51] Microsoft, ed. *HttpListener Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.httplistener?view=net-9.0 (visited on 01/12/2024).

[52] Microsoft, ed. *HttpRequest Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httprequestmessage?view=net-9.0 (visited on 01/12/2024).

[53] Microsoft, ed. *HttpRequestMessage Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httprequestmessage?view=net-9.0 (visited on 01/12/2024).

[54] Microsoft, ed. *MethodInfo Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.reflection.methodinfo?view=net-9.0 (visited on 10/03/2025).

[55] Microsoft, ed. *Type.GetMethods Method*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.type.getmethods?view=net-9.0 (visited on 10/03/2025).

[56] Microsoft, ed. *Uri Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.uri?view=net-9.0 (visited on 01/12/2024).

[57] Microsoft, ed. *UriBuilder Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.uribuilder?view=net-9.0 (visited on 01/12/2024).

[58] NoneType1 (フランソワ). *多項式補間を使用して強震モニタ画像から数値データを決定する*. 22nd Dec. 2020. URL: https://qiita.com/NoneType1/items/a4d2cf932e20b56ca444 (visited on 07/10/2024).

[59] OpenAI ChatGPT-4o. *Response on Connecting to a WebSocket in C#*. 22nd Nov. 2024. URL: https://openai.com/chatgpt (visited on 22/11/2024).

[60] Overleaf. *Bibliography management with biblatex*. URL: https://www.overleaf.com/learn/latex/Bibliography_management_with_biblatex (visited on 10/12/2024).

[61] Overleaf. *Code Highlighting with minted*. URL: https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted (visited on 14/10/2024).

[62] SciPy community, ed. *SciPy documentation*. 2008. URL: https://docs.scipy.org/doc/scipy/ (visited on 14/10/2024).

[63] Sergio0694, niels9001 and michael-hawker, eds. *Introduction to the MVVM Toolkit*. 7th Nov. 2024. URL: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/ (visited on 10/03/2025).

[64] TheSeeker. *Find the next TCP port in .NET*. 29th Sept. 2008. URL: https://stackoverflow.com/a/150974/ (visited on 03/12/2024).

[65] YoryeNathan. *Process.Start(url) fails*. 4th Apr. 2020. URL: https://stackoverflow.com/a/61035650/ (visited on 01/12/2024).

[66] こんぽ. *地震情報アプリ界隈 Advent Calendar 2021*. 2021. URL: https://adventar.org/calendars/6729 (visited on 15/10/2024).

[67] こんぽ. *地震界隈 Advent Calendar 2020*. 2020. URL: https://adventar.org/calendars/5903 (visited on 15/10/2024).

[68] こんぽ. *防災アプリ Advent Calendar 2022*. 2022. URL: https://adventar.org/calendars/7488 (visited on 15/10/2024).

[69] こんぽ. *防災アプリ開発 Advent Calendar 2023*. 2023. URL: https://adventar.org/calendars/9301 (visited on 15/10/2024).

[70] こんぽ. *防災アプリ開発 Advent Calendar 2024*. 2024. URL: https://adventar.org/calendars/9939 (visited on 10/12/2024).

[71] 気象庁 / Japan Meterological Agency. *JMA2001 走時表*. 2024. URL: https://www.data.jma.go.jp/svd/eqev/data/bulletin/catalog/appendix/trtime/trt_j.html (visited on 17/10/2024).

[72] 気象庁 / Japan Meterological Agency. *予報区等 GIS データ*. 2001. URL: https://www.data.jma.go.jp/developer/gis.html (visited on 14/03/2025).

[73] 気象庁 / Japan Meterological Agency, ed. *気象庁ホームページにおける気象情報の配色に関する設定指針*. July 2020. URL: https://www.jma.go.jp/jma/kishou/info/colorguide/HPColorGuide_202007.pdf (visited on 10/03/2025).

[74] 気象庁 / Japan Meterological Agency, ed. *気象庁防災情報 XML フォーマット　技術資料*. 9th Dec. 2022. URL: https://xml.kishou.go.jp/tec_material.html (visited on 10/12/2024).

[75] 気象庁 / Japan Meterological Agency, ed. *走時表フォーマット / Time Table Format*. URL: https://www.data.jma.go.jp/svd/eqev/data/bulletin/catalog/appendix/trtime/tttfmt_j.html (visited on 17/10/2024).

[76] 気象庁 / Japan Meterological Agency and dmdata.jp. *Sample Data*. URL: https://sample.dmdata.jp/ (visited on 10/03/2025).

[77] 防災科研 / National Research Institute for Earth Science and Disaster Resilience. *K-NET & KiK-net: 観測点一覧*. URL: https://www.kyoshin.bosai.go.jp/kyoshin/db/index.html (visited on 10/03/2025).

[78] 防災科研 / National Research Institute for Earth Science and Disaster Resilience. *NIED F-net*. 29th Mar. 2019. DOI: https://doi.org/10.17598/NIED.0005. URL: http://www.fnet.bosai.go.jp/auth/dataget/.

[79] 防災科研 / National Research Institute for Earth Science and Disaster Resilience. *NIED Hi-net*. 27th Feb. 2019. DOI: https://doi.org/10.17598/NIED.0003. URL: https://hinetwww11.bosai.go.jp/auth/download/cont/.

[80] 防災科研 / National Research Institute for Earth Science and Disaster Resilience. *NIED K-NET, KiK-net*. 28th Mar. 2019. DOI: https://doi.org/10.17598/nied.0004. URL: http://www.kyoshin.bosai.go.jp/kyoshin/.

[81] 防災科研 / National Research Institute for Earth Science and Disaster Resilience. *NIED MOWLAS (Monitoring of Waves on Land and Seafloor)*. 29th Mar. 2019. DOI: https://doi.org/10.17598/NIED.0009. URL: http://www.mowlas.bosai.go.jp/mowlas/.