COMP.CS.140

Programming 3: Interfaces and Techniques

Report: Weather App

Joonas Pinomäki
Jerri Tarpio
Jaakko Utriainen

Application structure

Classes *WeatherData, ForecastDataHourly, ForecastDataDaily, LocationData,* and *AirQuality* are used to contain the corresponding data. This data is gathered and accessed through *WeatherAPI*. *WeatherApp* calls a *WeatherAPI* object to display information on the application. The current state of *WeatherApp* is saved and loaded with *StorageSystem*, and *WeatherMap* is responsible for retrieving appropriate map weather images. OpenWeatherMap.org's weather API was used to retrieve weather and location data, and OpenStreetMap's tile API was used to get the map base images. More of the history and reasoning between the main application structure is detailed under section Division of labor.

UML class diagram

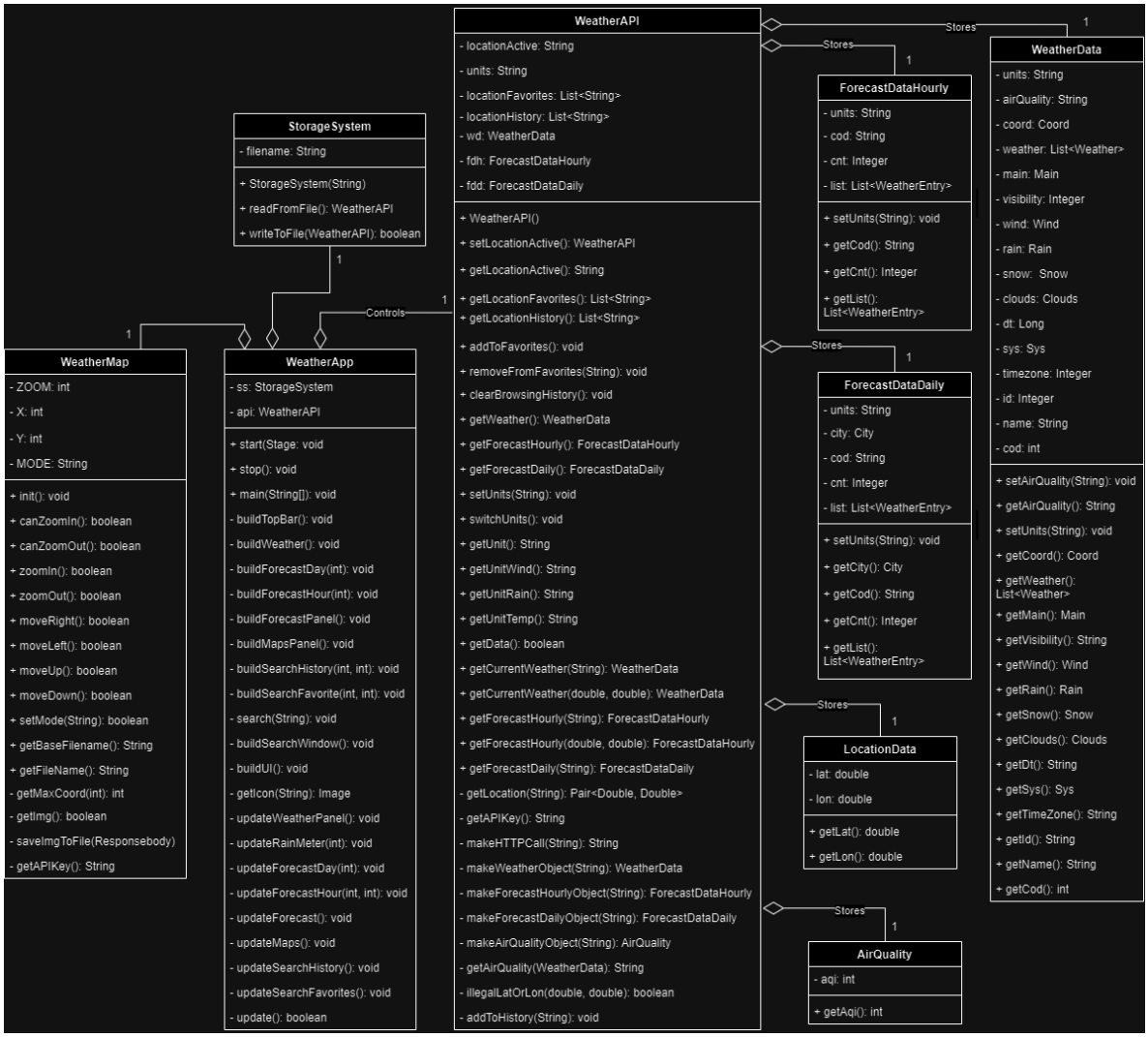
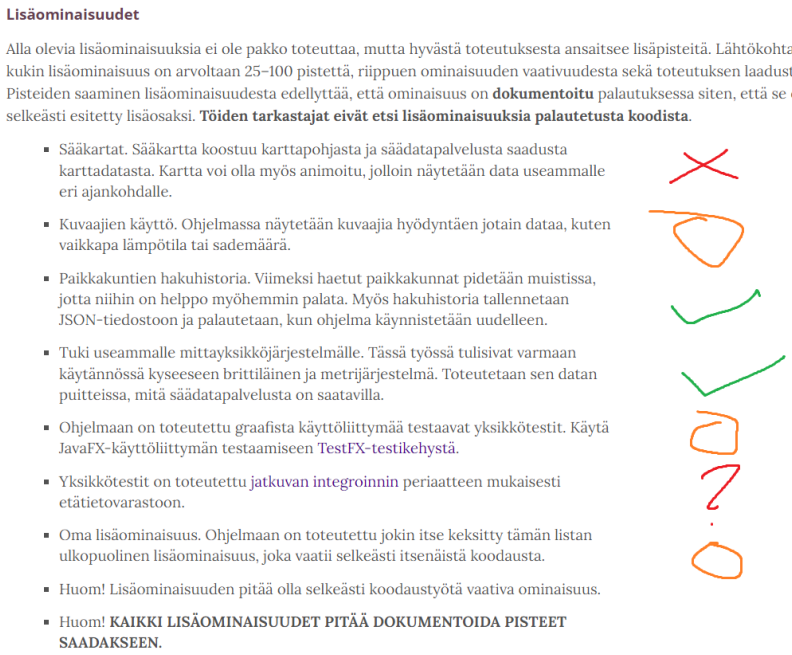


Figure 1: UML Class diagram

Implemented functions

The group unanimously agreed to target the maximum grade of 5. Due to unclear description of how the project would be scored, it was decided to finish as many additional features as possible within the time limit. This allowed us to develop the first draft of the software with some future additions already in mind. This ended up saving a lot of time that would have gone into refactoring especially the User Interface otherwise.



Lisäominaisuudet

Alla olevia lisäominaisuuksia ei ole pakko toteuttaa, mutta hyvästä toteutuksesta ansaitsee lisäpisteitä. Lähtökohtaisesti kukin lisäominaisuus on arvoltaan 25–100 pistettä, riippuen ominaisuuden vaativuudesta sekä toteutuksen laadusta. Pisteiden saaminen lisäominaisuudesta edellyttää, että ominaisuus on **dokumentoitu** palautuksessa siten, että se on selkeästi esitetty lisäosaksi. **Töiden tarkastajat eivät etsi lisäominaisuuksia palautetusta koodista**.

- Sääkartat. Sääkartta koostuu karttapohjasta ja säädatapalvelusta saadusta karttadatasta. Kartta voi olla myös animoitu, jolloin näytetään data useammalle eri ajankohdalle.
- Kuvaajien käyttö. Ohjelmassa näytetään kuvaajia hyödyntäen jotain dataa, kuten vaikkapa lämpötila tai sademäärä.
- Paikkakuntien hakuhistoria. Viimeksi haetut paikkakunnat pidetään muistissa, jotta niihin on helppo myöhemmin palata. Myös hakuhistoria tallennetaan JSON-tiedostoon ja palautetaan, kun ohjelma käynnistetään uudelleen.
- Tuki useammalle mittayksikköjärjestelmälle. Tässä työssä tulisivat varmaan käytännössä kyseeseen brittiläinen ja metrijärjestelmä. Toteutetaan sen datan puitteissa, mitä säädatapalvelusta on saatavilla.
- Ohjelmaan on toteutettu graafista käyttöliittymää testaavat yksikkötestit. Käytä JavaFX-käyttöliittymän testaamiseen TestFX-testikehystä.
- Yksikkötestit on toteutettu jatkuvan integroinnin periaatteen mukaisesti etätietovarastoon.
- Oma lisäominaisuus. Ohjelmaan on toteutettu jokin itse keksitty tämän listan ulkopuolinen lisäominaisuus, joka vaatii selkeästi itsenäistä koodausta.
- Huom! Lisäominaisuuden pitää olla selkeästi koodaustyötä vaativa ominaisuus.
- Huom! **KAIKKI LISÄOMINAISUUDET PITÄÄ DOKUMENTOIDA PISTEET SAADAKSEEN.**

Figure 2: Classification of additional features

The additional features were initially classified into categories of “easy”, “medium” and “hard” difficulty as shown in Figure 2. Whether these categories ended up being accurate was hit or miss, however they dictated the order of operations when we started developing additional features at the end. Some additional features, such as multiple unit systems and search history were so integral to the final product that they had to be developed from the very beginning.

Minimum features

- The program compiles. It can be compiled either in NetBeans IDE or using Maven via the command line.
- The program allows the user to search for different locations.
- The program displays the current weather and a simple forecast.
- The program uses both provided interfaces, although they have been heavily modified to suit the needs of the project.
- The complete version history of the project is available on Gitlab.
- You are reading the final document.

Intermediate features

- A completely custom graphical user interface (GUI) has been implemented.
- The application allows saving locations as favorites.
- The state of the application (current location, unit system, search history and favorite locations) is preserved between sessions.
- In addition to the current weather, a detailed forecast has been implemented.
- The application uses a custom set of icons. There are 22 icons in total, more than the 18 provided by OpenWeatherMap.
- The application resolves file handling errors in a controlled manner.
- Unit tests have been implemented for the application.

Additional features

- The application displays a map of the current weather.
- The forecast of rain intensity is displayed in a graph.
- The application tracks and maintains a history of search queries, including between sessions. The search history can be cleared by the user.
- Both Metric and Imperial units are supported, and the user can switch between them without restarting the program.
- Unit tests have been implemented with continuous integration.

Division of labor

The division of labor between group members was decided at the start of the project during our first meeting. The roles stayed constant during development, which posed some minor challenges but also forced us to work together more closely, which was good experience.

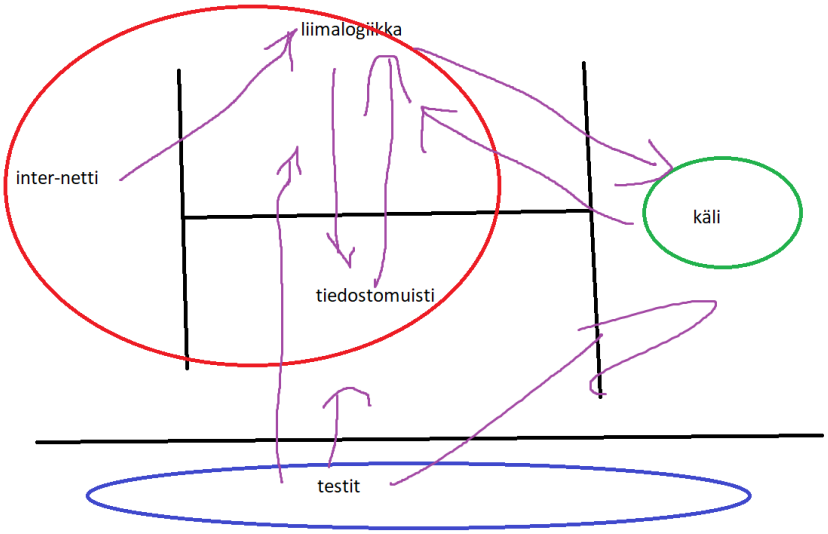


Figure 3: The initial division of labor

The initial division of labor was made as shown in Figure 3. The structure of the application was quickly divided into two parts: one section, named “Glue Logic” would fetch data from the API at OpenWeatherMap and handle static memory, while the other would contain the “User Interface”. This division was quite logical and helped to form the backbone of the project. As there were three (3) group members, a third section of the application was chosen as “Testing”. This third component would handle the unit testing and continuous integration of the application.

Highlighted by red, green, and blue ovals respectively, each section was then handed off to a separate group member: Mr. Pinomäki would implement the UI, Mr. Tarpio would implement the Glue Logic and Mr. Utriainen would implement the Testing. This is exactly how the roles ended up staying throughout the project.

In hindsight, the division of labor ended up being quite uneven. We agreed that that would be fine at the beginning of the project, though. More care could have been taken in ensuring that the implementation of the Testing section could have been started earlier, though, as the functions and methods of both other sections kept evolving throughout the project and would have required non-trivial refactoring of unit tests if implemented earlier.

User manual

This section contains a simplified user manual for the WeatherApp application.

Installation

1. Clone the source repository from https://course-gitlab.tuni.fi/compcs140-spring2024/group3681.git.
    - Use command ‘git clone https://course-gitlab.tuni.fi/compcs140-spring2024/group3681.git’
2. Make sure that you are on the correct branch, *main*.
    - Use command ‘git checkout main’ at the root of the repository.
3. Enter your OpenWeatherMap API key in file *WeatherApp/apikeys.json*.
    - Your 32-character API key goes in place of the default zeros.
    - Never share your private API key publicly.
    - You can use command ‘git update-index --skip-worktree WeatherApp/apikeys.json’ at the root of the repository to make sure changes to *apikeys.json* won’t be pushed to the repository.
4. Compile the application.
    - If using NetBeans IDE, click “run”.
    - If compiling from command line, use command ‘mvn package’ inside the *WeatherApp* directory.
5. Run the application.
    - If using NetBeans IDE, your application should be running.
    - If running from command line, use command ‘java -jar target/WeatherApp-1.0.one-jar.jar’.

UI overview

The Graphical User Interface (GUI) is closely modeled after the example solution given with the specification of the project.

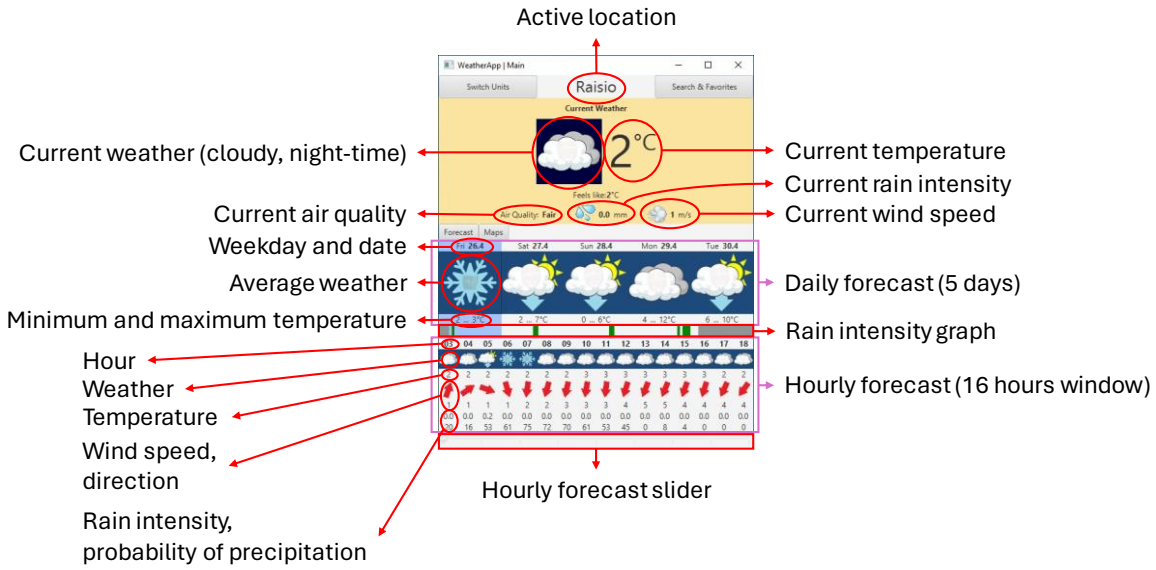


Figure 4: Main window of the GUI

The main window of the UI in its default state is displayed in Figure 4. The sections of the Forecast panel have been marked in purple while the main UI elements have been marked in red. The buttons have been purposefully left unlabeled, as their functionality is described in relation to their labels.

A day of the forecast can be selected by clicking anywhere within its bounds. The selected day controls which day the hourly forecast data is shown for. The hourly forecast slider can be used to select which 16 hours of 24 are shown on screen. If there are less than 16 hours of forecast data available for that day, the slider will be locked. Only valid data is shown on the UI. The missing data is also displayed as a dark gray color in the rain intensity graph.

The current unit system can be toggled with the “Switch Units”-button. The unit system is Metric by default, and this button toggles between Metric and Imperial units. The last selection is saved in nonvolatile memory when the application is closed properly.

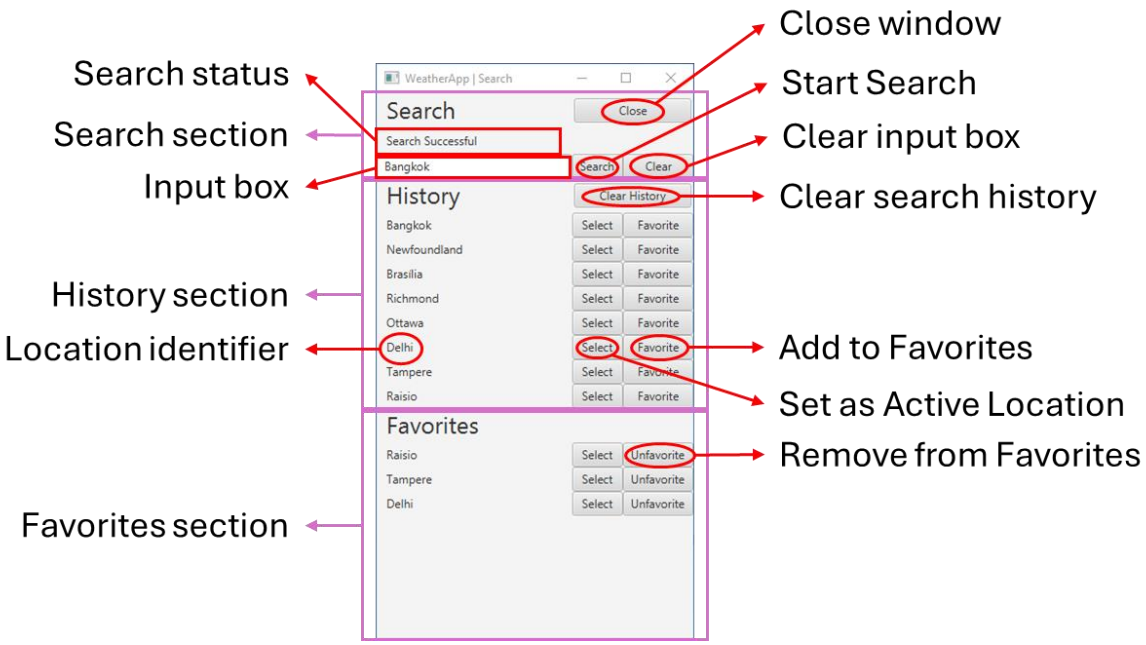


Figure 5: The Search window

The "Search & Favorites"-button opens the *Search*-window displayed in Figure 5. This is used to change the Active Location of the application. It also contains the search history and favorites. The last eight (8) history entries and favorites are displayed graphically, but they are all kept in non-volatile memory; removing a favorite will display more.

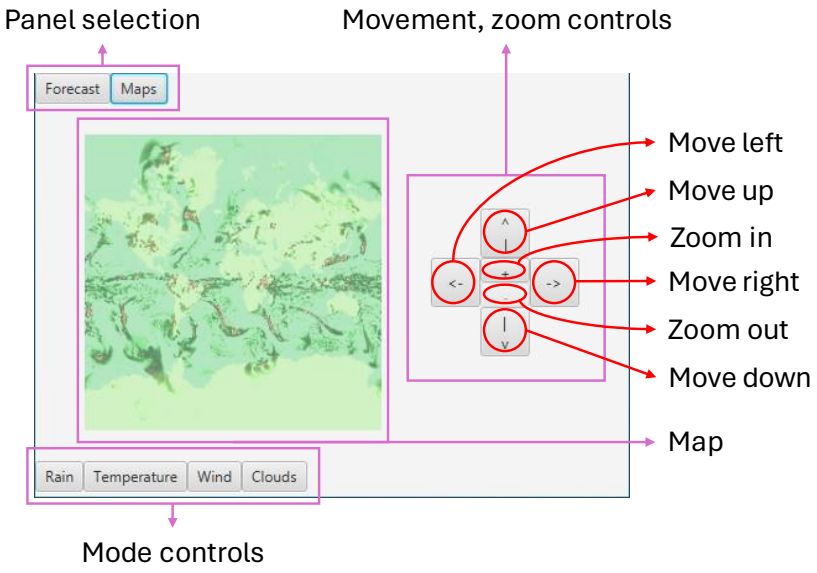


Figure 6: The weather map panel

The "Maps"-button opens the *Maps*-panel displayed in Figure 6. The move controls allow four-way movement and zooming in and out. The mode controls switch between rain, temperature, wind, and cloud maps. You can return to the *Forecast*-panel using the "Forecast"-button.

Known issues, etc.

- If the persistent data file (*WeatherApp/temp.json* by default) contains garbage data, the GUI might render without any data. This can be fixed by changing the Active Location, or by deleting *temp.json.*
- The timezone is always UTC and this cannot be changed by the user. It also doesn't automatically track the target location. A fix was attempted but it was deemed too hard to finish on time. However, the project requirements do not state that the displayed time must be in the local timezone.

On the use of artificial intelligence (AI) tools

No AI tools were used in the development of the user interface. It was deemed as a good learning experience for programming graphical user interfaces instead. If AI was used in place of manual work, it might have led to some of the interesting problems being solved without any thought, teaching us less. The structure of the UI class could also have become convoluted such that we would not know how to modify it or where to even start. On the other hand, using AI tools might have saved a lot of effort done to make the UI meet the requirements. The prototype for the GUI could have been generated based on the example photo, for example.

ChatGPT was used to generate the structure of the classes needed to contain the data from the OpenWeatherMap API Json responses. It was automating a task that would otherwise be repetitive and prone to errors. On the other hand, a lot of effort was still put into properly completing the classes as the AI only helped with their structure.

An attempt was made to use ChatGPT to help with testing and debugging the unit tests, but it was found to be not that helpful. Hence, ChatGPT did not affect the result in these test cases.