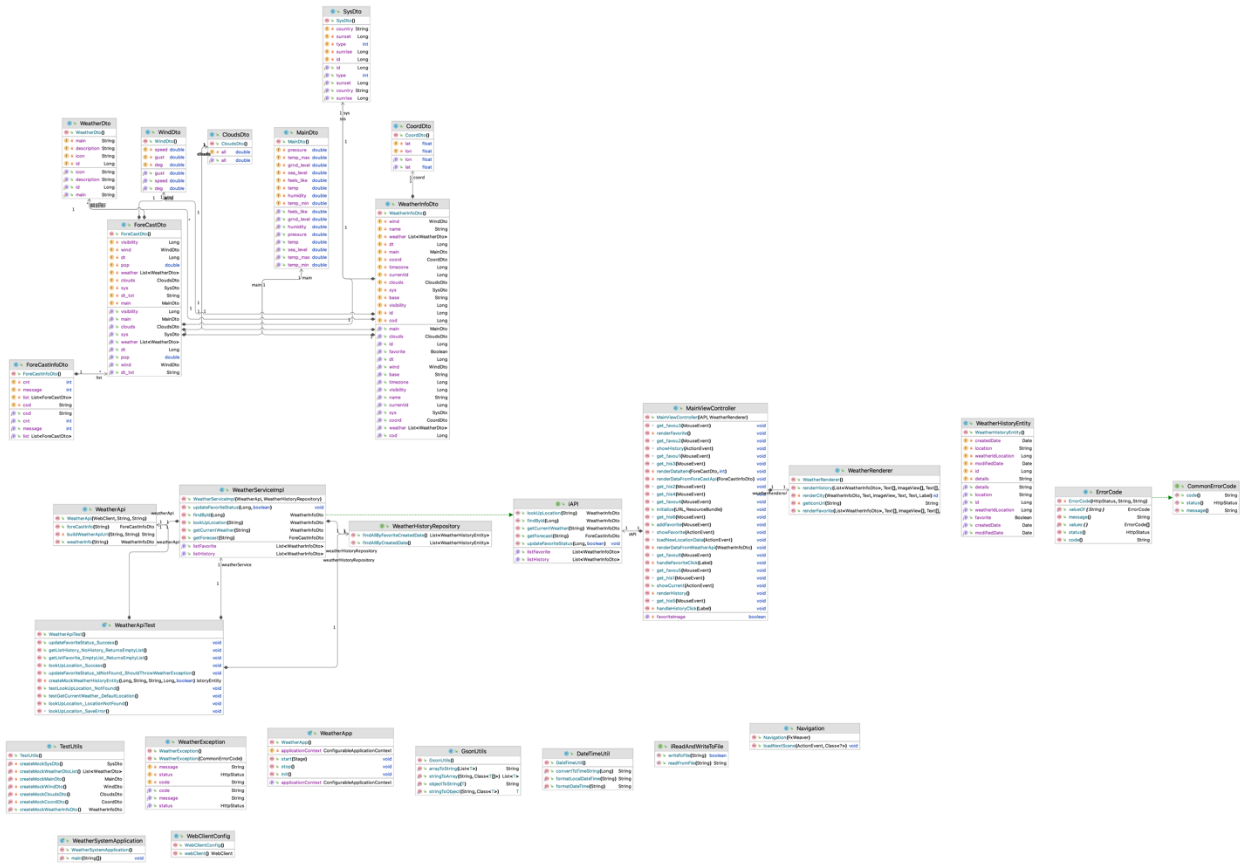
Project: Weather App

Minh Hoang, 152103143
Tuan Nguyen, 152060244

1. Structure of the Implemented Software

1.1 UML Class Diagram



1.2 Folder structure

config: is dedicated to handling configuration-related aspects of the application.
controller: houses classes responsible for handling the interaction between the user interface and the backend services.
dto: comprises Data Transfer Objects, which are used for transferring data between different layers of the application. Each DTO class represents a specific aspect of the weather data.
entity: contains classes related to data persistence and database representation.
exception: deals with exception handling in the application.
repository: includes the WeatherHistoryRepository class, which likely extends the Spring Data JPA CrudRepository or a similar interface. It provides methods for interacting with the WeatherHistoryEntity and performing database operations.
service: The WeatherServiceImpl class implements the iAPI interface and provides services related to weather information retrieval, storage, and management. It interacts with external weather APIs and a local database to offer functionality such as looking up locations, getting current weather, fetching forecasts, and managing user-specific data like search history and favorite locations.

util: The GsonUtils class provides utility methods for working with Gson library to convert between JSON strings and Java objects. It includes methods to convert JSON strings to arrays or objects, and vice versa. Additionally, it offers functionality to convert lists and individual objects to JSON strings. The class uses the Gson library for serialization and deserialization.
weatherapi: contain classes related to interfacing with external weather APIs.

2. Responsibilities of Key Classes

controller/MainViewController.java
+ The class is the central controller in the Weather application, managing interactions between the user interface and backend services. Annotated with @Component and @FxmlView, it seamlessly integrates with JavaFX and is controlled by the Spring context.
+ User Interface Control:
- Manages UI components like buttons, labels, and images.
- Handles user actions to display current weather, favorites, and historical data

+ API Interaction:
- Communicates with the iAPI service for real-time weather, forecasts, and historical data.
- Utilizes WeatherRenderer for efficient data rendering.

+ View Initialization:
- Implements Initializable for main view setup.
- Initializes weather and forecast data for default or user-specified locations.

+ User Actions:
- Responds to adding/removing locations from favorites.
- Enables switching between views for current weather, favorites, and history.

controller/ WeatherRenderer.java
+ The class, annotated with @Component, plays a pivotal role in rendering weather-related data within the Weather application. It specializes in dynamically displaying weather information for historical and favorite locations on the user interface.
+ Rendering Historical and Favorite Data:
- Utilizes methods renderHistory and renderFavorite to dynamically display weather details for a list of WeatherInfoDto objects.
- Renders information such as city names, weather icons, temperatures, and descriptions.

+ City Rendering:
- The private method renderCity centralizes the rendering process by populating JavaFX components (Text, ImageView, Label) with relevant weather details.
- Handles cases where weather information is unavailable, displaying a default message.

+ Icon URL Generation:
- The method getIconUrl constructs the URL for weather icons based on the provided icon code, facilitating the retrieval and display of weather images.

entity/ WeatherHistoryEntity.java
The WeatherHistoryEntity class is a JPA entity representing weather history data within the Weather application. Annotated with JPA and Spring Data annotations, it encapsulates

information about weather conditions, including location details, associated IDs, and timestamps.

repository/WeatherHistoryRepository.java

The WeatherHistoryRepository interface is a Spring Data JPA repository responsible for handling database operations related to weather history entities in the Weather application.

service/impl/WeatherServiceImpl.java

+ The WeatherServiceImpl class is the implementation of the iAPI service interface in the Weather application. It integrates with external weather APIs, manages weather data, and interacts with the database for storing and retrieving weather history. Key features include location lookup, current weather retrieval, forecast retrieval, history and favorite lists, updating favorite status, and finding weather information by ID.

+ Service Methods:

- ▫ lookUpLocation: Looks up a location, retrieves weather data, and stores it in the database.
- ▫ getCurrentWeather: Retrieves the current weather for a given location.
- ▫ getForecast: Retrieves the forecast for a given location.
- ▫ getListHistory: Retrieves a list of recent weather history entries.
- ▫ getListFavorite: Retrieves a list of favorite weather history entries.
- ▫ updateFavoriteStatus: Updates the favorite status of a weather entry.
- ▫ findById: Finds weather information by ID.

3. Project Functionality

The implemented software offers the following functionalities:

- Search Weather by Location: Users can search for weather information based on location.
- 5-Days Forecast: Provides a 5-day weather forecast.
- Recently Viewed Locations: Displays a list of recently viewed locations.
- Manage Favorite Locations: Users can add or remove locations from their favorite list.
- User-Friendly UI: The application boasts a well-designed and intuitive user interface.

4. Classes with Pre and Post Conditions

The team has implemented pre and post conditions for specific classes, ensuring the robustness of the code. Details can be found in the code documentation.

5. Agreed and Actual Division of Work

The agreed and actual division of work is detailed in the project management documentation. Each team member's responsibilities, tasks, and contributions are outlined to ensure a balanced workload.

Tuan Nguyen:

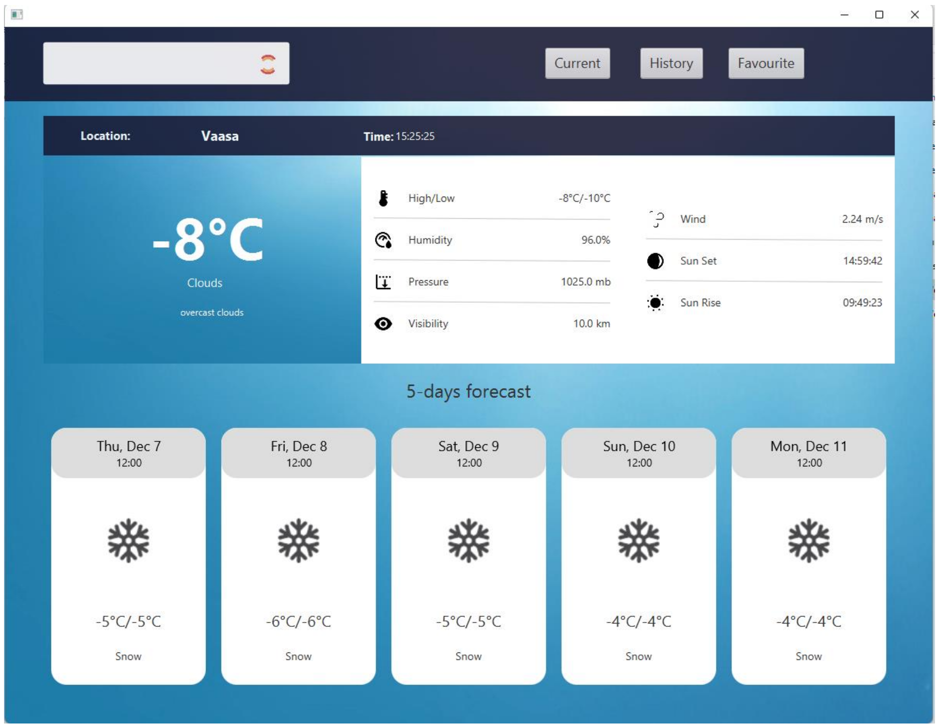
- Implement some methods in service class: update favourite, find by id and get history list.
- Write unit test cases.
- Render data for history and favourite view.

- Write util class for converting the object.
- Write the documentation.
- Clean code

Minh Hoang:

- Init project and implement the UI.
- Create and config application.
- Implement call endpoint from API.
- Implement interface service for all of methods.
- Render data from weather API for client site.
- Clean code.
- Write the documentation.

6. Short User Manual



Location: Vaasa Time: 15:25:25

-8°C
Clouds
overcast clouds

High/Low	-8°C/-10°C
Humidity	96.0%
Pressure	1025.0 mb
Visibility	10.0 km
Wind	2.24 m/s
Sun Set	14:59:42
Sun Rise	09:49:23

Thu, Dec 7 12:00	Fri, Dec 8 12:00	Sat, Dec 9 12:00	Sun, Dec 10 12:00	Mon, Dec 11 12:00
-5°C/-5°C	-6°C/-6°C	-5°C/-5°C	-4°C/-4°C	-4°C/-4°C
Snow	Snow	Snow	Snow	Snow

1. Open the application in **WeatherApp/target/WeatherApp-1.0.jar**
2. Type your location in the top left to check the weather.
3. Vaasa is set as the default. Click "current" to see Vaasa's weather.
4. Check your past searches in the history screen.
5. Like a place? Click the heart icon to add it to your favorites. Click again to remove it.

7. Known Bugs or Missing Features

Known Bugs

Bug 1: Null pointer exception ”Id not found”

Description: When user clicks to update the favorite, the status could not change and return an exception

Status: Resolved

Bug 2: Duplicate location in favorite views.
Description: When user clicks to favorite view, it shows duplication location.
Workaround: User can see only one location when click to favorite view.
Status: Resolved

Missing Features
Feature 1: C to F or F to C temperature
Description: User wants to see the temperature in F or C
Status: Under Development

Feature 2: Map
Description: User wants to see the map around the world
Status: Not Yet Scheduled