High level description:

Our application's root class is WeatherApp which is responsible for startup process keeping track of the state of the application and linking necessary components together.

We chose MVC for this project because it fits the best with the plan we had. For generating the application UI, we are using JavaFx library. JavaFx follows mostly MVC-architecture. Generated FXML-files are the views, each view has its own controller ja model class. Controller handles the interaction between the interface and the model. Model classes contains the information to linked each view.

We have four main MVC sets, each set represents single tab of the application. The tabs are weather, road and combined, settings. In addition to these sets, we also have root set, which function is to offer UI root component, for attaching all the other views. This makes class separation possible.

At startup all the views are anchored to the root view and rendered.

The preference class saves user's preferences by saving the tab the user is on and the buttons that are pressed at the moment on that tab.

We are using two classes to fetch the data from the APIs used. FmiApi fetches, parses and saves the data from the FMI api and DigiTrafficApi does the same to DigiTraffic api.

Classes and their dependencies are seen in this class diagram (Figure 1).

Components and class responsibilities:

WeatherController:

Controls the UI of the weather forecast tab. Reads user inputs and calls WeatherModel's functions. Also makes the function calls to work on action (pressing a button).

- Method setGraphContent: Reads user inputs from locationTextfield and sctDayDatePicker. Also checks which radiobutton is pressed and calls the drawGraph method with these three parameters.

RoadController:

Controls the UI of the road maintenances and tasks tab. Reads user inputs and calls RoadConditionModel's functions.

CombinedController:

Controls the UI of the combined tab. Reads user inputs and calls RoadConditionModel's and WeatherModel's functions.

SettingsController:

Controls the UI of the settings tab.

RootController:

Initializes the application UI.

- Method initialize: Initializes all the application tabs with fxml files.

App:

Root class of the application. Holds the data of the application.

- Method start: Starts the UI with root.fxml, so it gives the base to the application.

FmiApi:

Gets and parses the wanted data from FMI API.

- Method getWeatherData: calls the api call functions with given parameters for place and time
- Method formatDate: formats date to the wanted format
- Method formatDateStringToLocalDateTime: parses localDateTime
- Method parseForecastData: gets windspeed, temperature and cloudiness from the fetched data
- Method loadXMLFromString: gets xml file from string
- Method harmonieForecastSimple: the actual api call
- Method weatherObservationHourlySimple: the actual api call

DigiTrafficApi:

Gets and parses data from the digitraffi api into wanted roadData model.

Preference:

Saves the user's given options to a map by saving the current tab and selected options.

WeatherModel:

Draws weather graphs with user's preferences and fetches the data from weather API.

- Method drawGraph gets the selected radiobutton, location and date as parameters from the controller. Based on the radiobutton selection, the correct info for the chart is chosen
- Method fetchWeatherData calls the fmiApi's getWeatherData function to get the data based on the selected date and location.

RoadConditionModel:

Draws road condition graphs with user's preferences and fetches the data from road condition API.

CombinedModel:

Draws road condition and weather graphs with user's preferences and fetches the data from road condition and weather API.

- Method drawWeatherGraph draws the temperature chart based on the selected date and location which are passed as parameters
- Method fetchWeatherData calls the fmiApi's getWeatherData function to get the data based on the selected date and location.

Weather.fxml

Implements the UI for the Weather tab.

Root.fxml

Implements the UI base for the app.

RoadCondition.fxml

Implements the UI for the roadCondition tab.

Combined.fxml

Implements the UI for the Combined tab.

Settings.fxml

Implements the UI for the Settings tab.

Self-evaluation:

After the first TA meeting we had to change our plan to include which MVC variant we were going to use. We decided to use the basic MVC. In the updated class diagram (class_diagram_V0.2.drawio.png in gitlab and at the end of the document) you can see the added model classes that are made for all the controllers. This is the biggest change we have made. Also the methods of different classes have been updated. After starting the coding, we realized that the planned api manager classes and the ApiHandler class were unnecessary.

Otherwise, we feel that our design has supported the implementation quite well. We have found some variables to be unnecessary and figured that some important variables were missing from the design and class diagram. So far the quality of the app hasn't been perfected but it does what it's supposed to. For the remaining features we probably need to make some changes to the methods and variables we have planned.

Reasonings:

We decided to use a class WeatherApp to save and hold all the data. This way it's easy to use different UI controllers using the same data. The APIs have models for fetching the data from the DigiTrafficApi and FmiApi are managed through API managers.

For tracking project status and assigning tasks to developers we are using Trello and Kanban tables. We have decided to use JavaFx as an UI-library in the project and for common tools we have chosen SceneBuilder and NetBeans IDE.

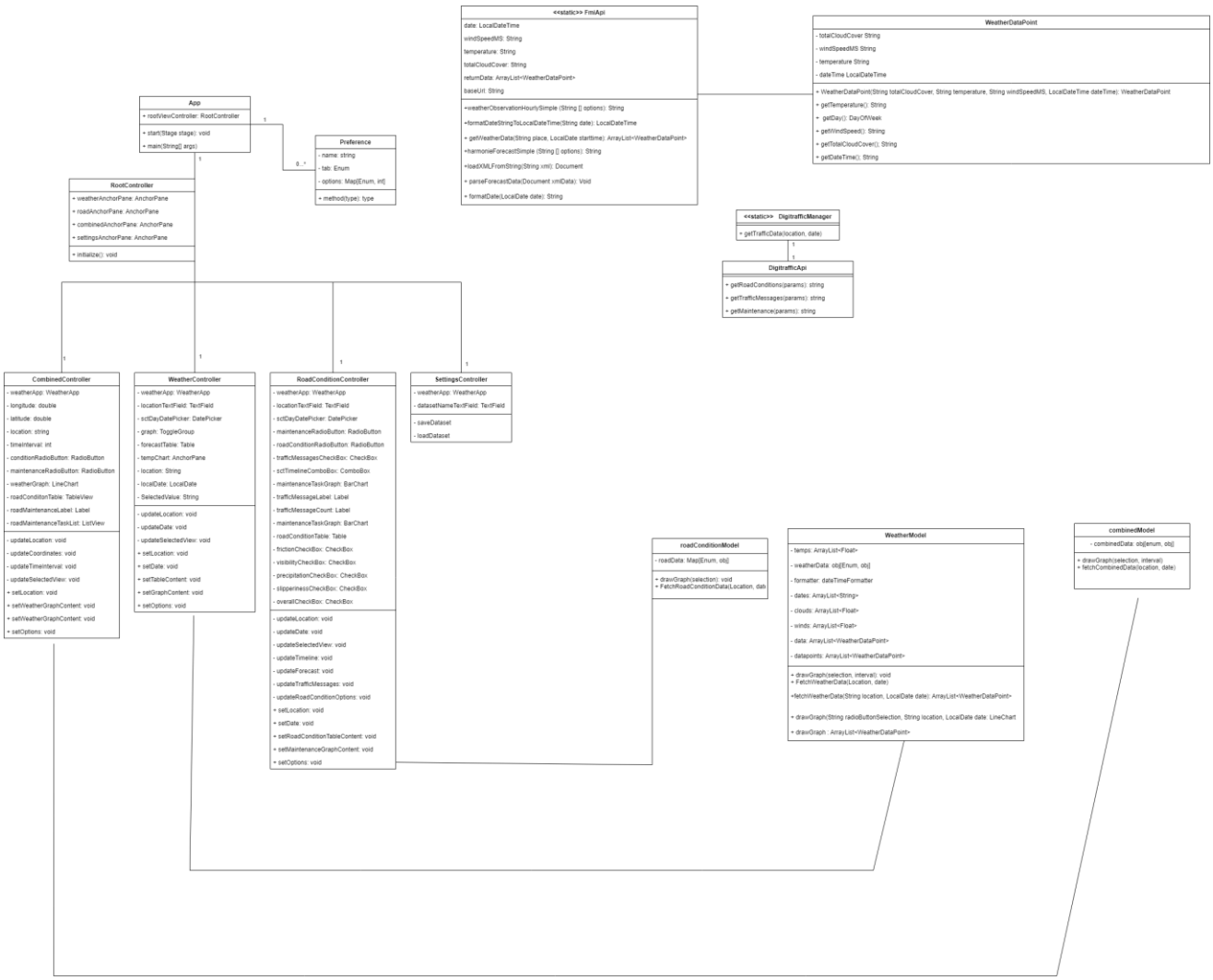

Figure 1 Application class diagram