Design document for prototype

Group Name: Here we go again

Alvin Wijaya

Juhana Kivelä

Yue Zhu

Chenshuo Dong

Table of contents

Contents

1. System architecture 3

1.1 Classes and interfaces 4

2. Used API’s 5

1. System architecture

Our application is designed to be run completely in a user’s device and only external call it makes are API calls to external databases. This means that it doesn’t have separate frontend running on client-side and backend running on server-side.

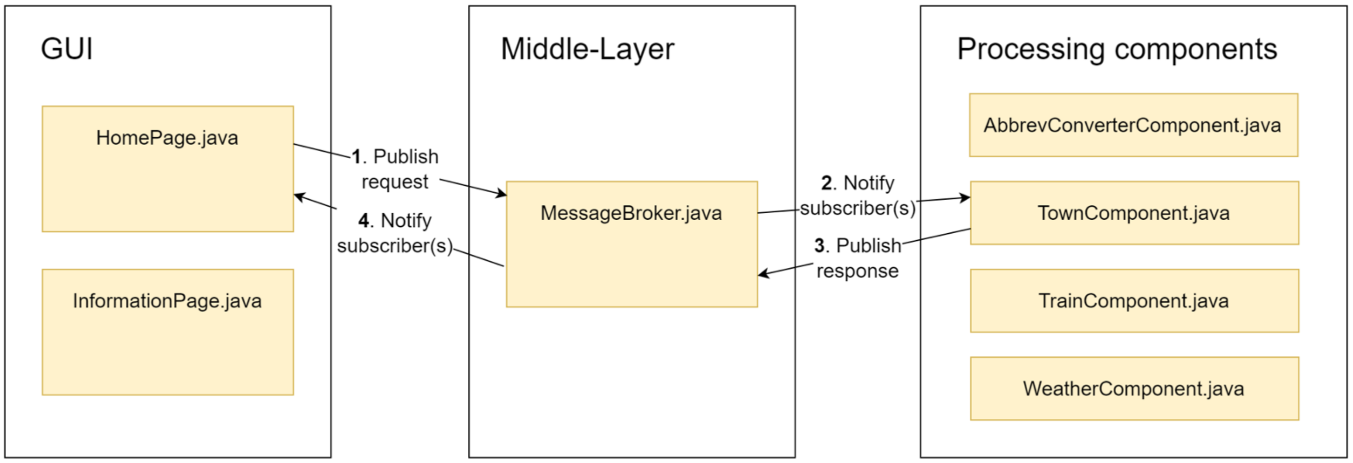
The most important attributes for our system are:

- Usability. The application is designed for end-user so it must be easy to use. The usage should feel intuitive, and the user should be able to use it without any help from the system.
- Reliability. The application should be stable and should not crash from any GUI usage. The application should also display correct error messages in case of internet issues or if API’s would be temporarily unavailable.
- Maintainability. The project should be designed in a way that it’d be possible to continue its development in the future. The project should be also divided into logical components so that the work can be divided for multiple developers, and each would have clear responsibility area.

After designing the systems structure, we ended up with a model that has centralized messaging system and clear division between GUI and processing components. Our system is divided into 3 parts:

1. GUI, which is responsible for managing UI and calling message broker for any information it needs
2. Middle-layer, which acts as a centralized messaging system and redirects topics to correct subscribers.
3. Processing components, which are responsible for business logic. They handle topics and return the responses as a new topics.

The systems design is visualized in Picture 1, which also shows one potential interaction between GUI and processing component.



Picture 1. Visualizing the system architecture.

The core idea for this design is that all the communication between GUI and processing components, as well as communication between processing components, would go through the middle-layer message broker. This adds clarity to the communication between components as all the traffic to them comes from one centralized message broker. This approach also allows the processing components to be independent of each other.

1.1 Classes and interfaces

This section explains the most important classes and interfaces of the system. They are described in Table 1.

Table 1. Classes and interfaces of the system.

Name of the component	Type	Description
HomePage.java	Application	Main entry point of the program GUI.
InformationPage.java	Application	Display the train arrival and departure stations, timetable, weather conditions, and information about the departure and arrival towns to the user.
MessageBroker.java	Class	Acts as a centralized communication component. Other classes can use MessageBroker.java to publish new Topics, subscribe to new topics and unsubscribe to topics. This class also handles the creation of processing components.

AbbrevConverterComponent.java	Class	Receive request with train station abbreviation, convert it to train station name using API, and send response with train station name back.
TownComponent.java	Class	Receive request with town name, get the information about the town (Population, Area, Density) using API, and send response with the town's information back.
TrainComponent.java	Class	Train information, including train arrival and departure times and running times.
WeatherComponent.java	Class	Get weather data from the API and convert it into a format that can be use by the program in GUI.
MessageCallback.java	Interface	Acts as a interface class that has one function onMessageReceived. All components that wish to receive Topics, must implement this interface.

Table 1, The most important classes and interfaces.

2. Used API's

This system integrates multiple APIs to provide detailed information about train travel, including weather conditions at departure and arrival stations, train schedules, and demographic data for the associated towns or cities.

1. **Weather Data (OpenWeatherMap API)**
   The OpenWeatherMap API is used to display the current weather conditions at both the departure and arrival stations. Weather information such as temperature and general conditions is fetched based on the location name of each station.

2. **Station Metadata (Digitraffic API - Station Shortcodes)**
   The station metadata is retrieved via the Digitraffic Station API:
   https://rata.digitraffic.fi/api/v1/metadata/stations
   This endpoint is used to convert the station shortcode (used internally by the train system) to its corresponding station name, enabling users to see the full name of the departure and arrival stations.

3. **Live Train Information (Digitraffic API - Live Trains)**
   Train departure and arrival information is fetched from the Digitraffic Live Trains API:
   https://rata.digitraffic.fi/api/v1/live-trains
   This provides real-time data on train schedules, including departure and arrival times, as well as the associated station shortcodes for further processing.

4. **Demographic Data (Statistics Finland API)**
   Demographic information about the departure and arrival towns/cities is sourced from

Statistics Finland:
https://pxdata.stat.fi/PxWeb/pxweb/en/StatFin/StatFin__vaerak/statfin_vaerak_pxt_11ra.px/table/tableViewLayout1/?loadedQueryId=75470143-d9dd-4179-ba1e-447b1e163243&timeType=top&timeValue=1
This provides key statistics, including population, population density, and the area of the relevant locations, offering users contextual information about the locations they are traveling to and from.

By combining data from these APIs, we provide a seamless and comprehensive overview of train journeys, including real-time train information, local weather conditions, and demographic insights.