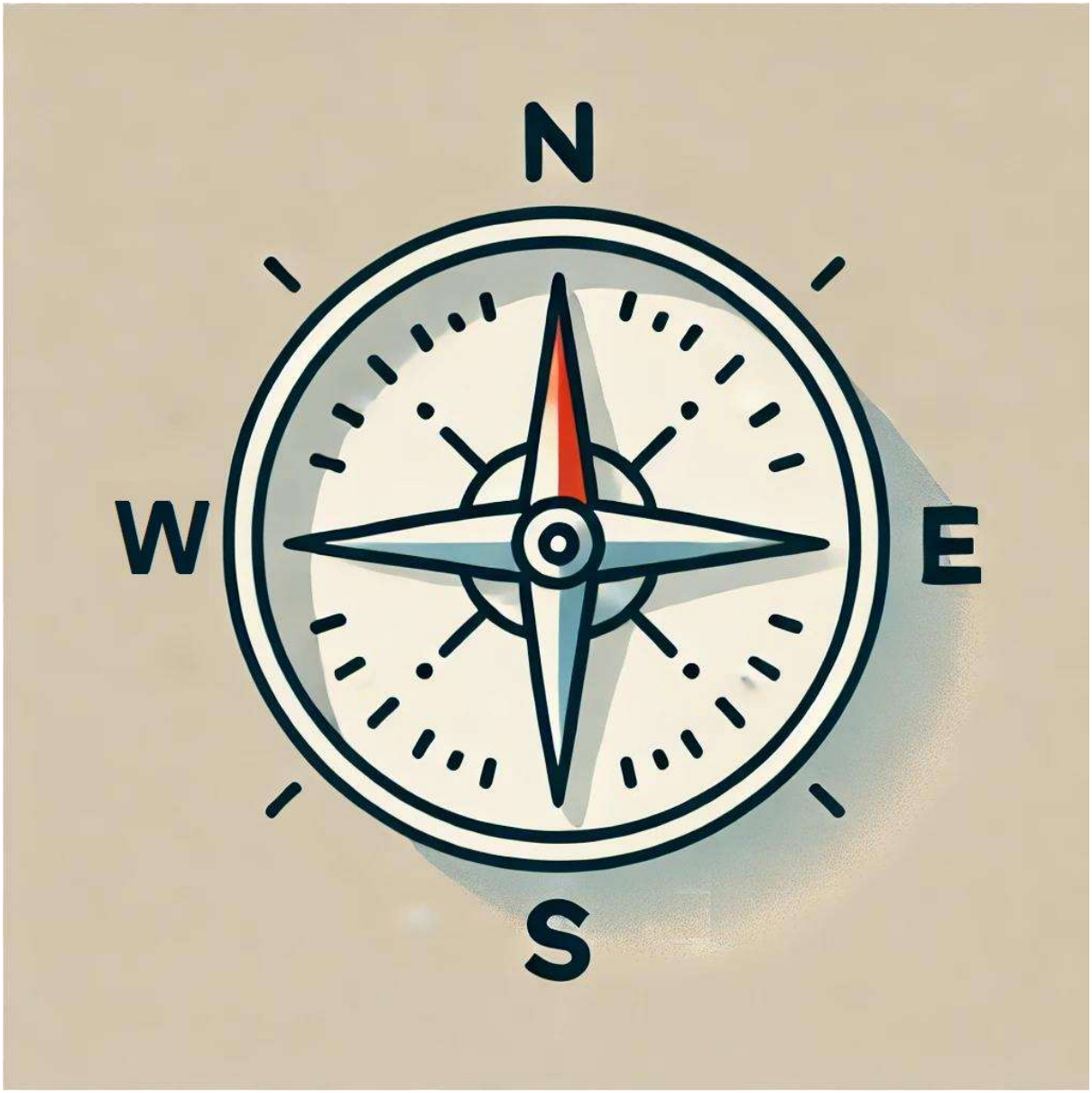
Koulu-Kompassi



Team Members: Niilo Hautamäki, Joni Hämäläinen, Max Jauhiainen, Tenho Laakkio

1. High-Level Description

1.1 Overview

The application allows the user to easily see the differences universities and universities of applied sciences as well as high schools and their appropriate programmes. There are also several filter options to visualize the data retrieved from public API Vipunen as well as web scraped from Government web sites.

The user can also choose the field they are interested in, and it filters out the universities that do not offer their wanted field. The user can also fill in their own grades from their matriculation examination and the application will show the fields the user would have gotten into.

1.2 Architecture

The program is a web application that utilizes Spring Boot with Java to create logic for the server side of things. The server uses Application.java as the entry point where our packages are configured using ComponentScan. The packages are logically divided to the following categories:

- Controllers
- Data
- ExcelParser
- Services
- Config

The controllers are the heart of our program. They allow us to logically divide the data pushing to different controllers. The mapping allows us to connect the JSON-data behind a certain localhost address by CrossOrigin approach defined by the mapping.

Data has our JSON-parsing related data classes as well as Cache classes for saving them to the disc.

ExcelParser is a critical component of our program to parse additional data from Vipunen that are not provided by the API. The data is from Excel files manually retrieved from Vipunen data Excel file set.

Services is our service layer that allows us to implement data handling functionality so that the single responsibility principle is true for our controllers. This layer can be used easily by other classes without violating good design practices.

1.3 Class Diagram

We used our IDE, IntelliJ IDEA, to create a class diagram of our software's Java classes for us.

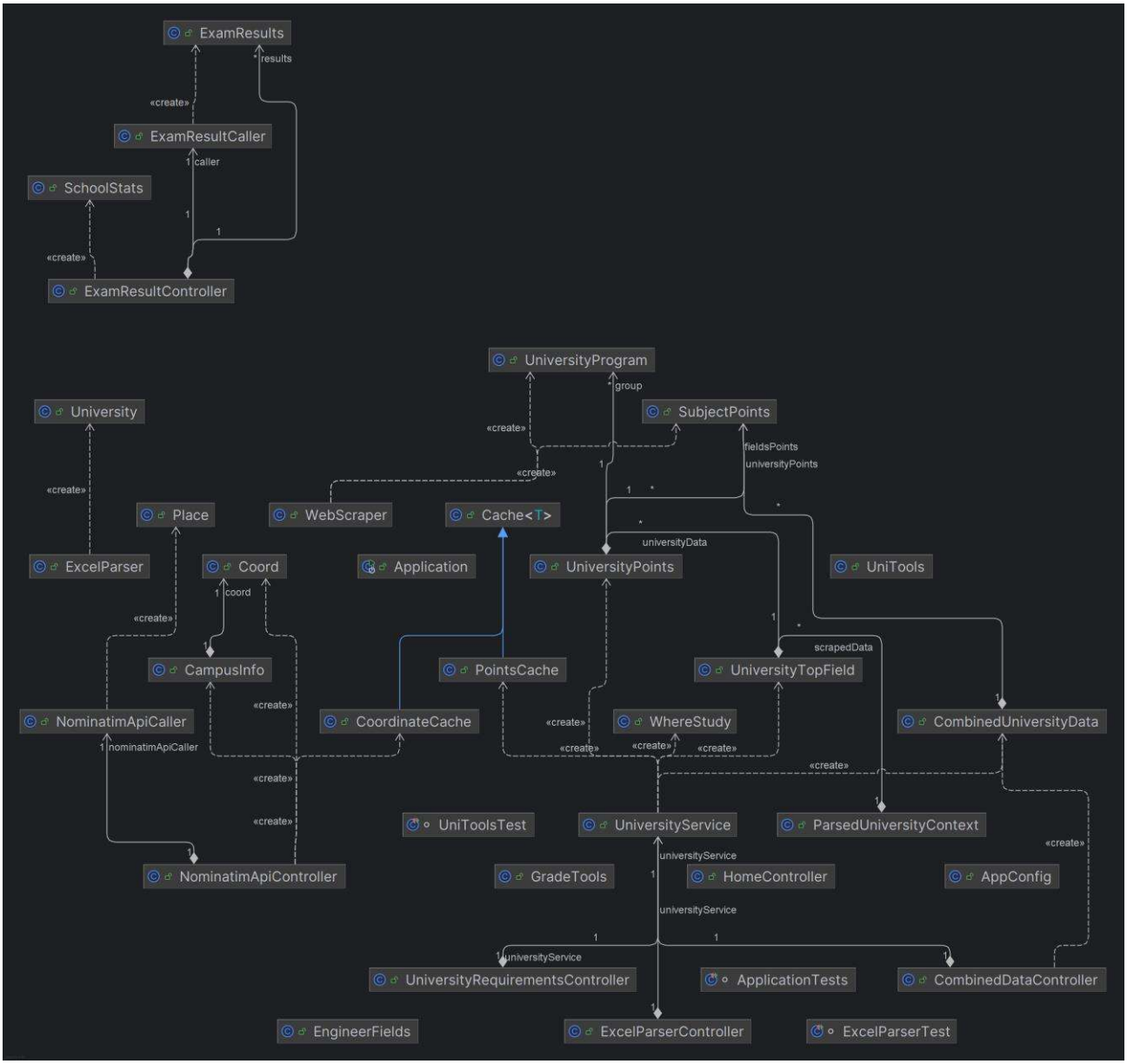


Diagram 1: Class diagram depicting the backend of the application (all the Java classes)

This diagram shows the relations between different classes. For example the blue line shows when the class is extended from another. Like the case with Cache and CoordinateCache. It also shows where some of the classes are Autowired like the case with UniversityService which shows how the Controllers depend on the UniversityService for their web scraping capabilities. The diagram also shows the division between high school and college classes. Like how the ExamResultController usages are not connected to the university classes.

There is also a large variety of different Singleton applications as static global classes. They can be seen how they are alone and not connected.

1.4 Technologies and Libraries

The tech stack for this project is the following:
Backend: Java with Spring Boot

Frontend: React with Vite

Primary libraries used in the backend of the application:

Apache POI – Used for parsing the excel files that contain the data for the universities, fields and required points for different years.

Jsoup – Used for Web scraping the government web sites

Primary libraries used in the frontend of the application:

Leaflet – Used for the map showing the universities and high schools as markers on the map

Recharts – Used for creating the charts for showing the required points for specific fields by year in the universities –view and showing the distribution of received grades from the matriculation examination in the high schools – view.

2. Boundaries and Interfaces (Internal)

2.1 Data Flow

Our application's dataflow is mainly one-sided, the frontend asks for the data and the backend provides it via REST API's. In the backend, there are API controllers which map a request into a URL and API callers, which will call the external API's (Nominatim and Vipunen) to get the asked data. In addition to these, we of course have base data classes, which are used to map the data received from the external API's and then extract the required information from them.

2.2 Interfaces

Nominatim API for managing the location data for universities and high schools as well as their campuses. Vipunen API for retrieving data for high schools, such as name, number of students who took matriculation examinations in a certain year and the average grade from all of the exams.

2.3 Interface Diagram

Below is a diagram that represents the interface between front-end and back-end.

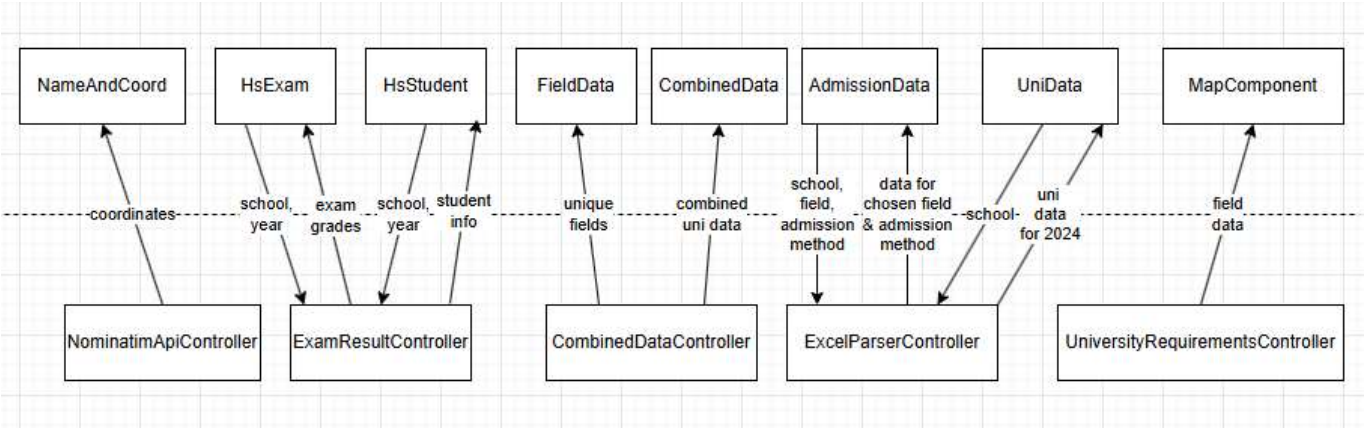


Diagram 2: This diagram illustrates the data flow between front-end and back-end.

This diagram shows only the components that are responsible for sending/receiving data through the interface between front- and backend. So, this diagram does not show in detail where the data comes from or how it is used further in front-end.

3. Component Descriptions and Responsibilities

3.1 Component Overview

In the application, there are three kinds of components:

User interface components, which handle the interaction with users, such as buttons and text input fields. These are all on the frontend side of the application and were made with react.

Service components, which are on the backend and provides the frontend with a service to retrieve some kind of data. These contain our business logic for handling and sending data.

Data components, which are on the backend, are used to hold the data that we then modify and send to the frontend via service components. These hold, for example, all the universities names and the fields you can study in said university.

3.2 Component Details

Two key components for the front-end are MapComponent.jsx and SidePanel.jsx. MapComponent is responsible for the map containing markers and it's also a parent component for every other component in map view (e.g. SidePanel, GradeFilter, SearchBar, TypeFilter). MapComponent is also responsible for the logic of adding and removing schools from favourites. MapComponent is quite extensive component, because it is responsible for many state variables and updating them. MapComponent passes down these state variables and different functions to its child components.

SidePanel might be the most important child component of MapComponent, because it holds some of the key features of this product, for example graphs for exam grades for high schools and for points for different admission methods for universities. SidePanel offers more features for universities, for example searching fields using a search bar.

The key components in the backend are the controller classes. CombinedDataController handles the requests related to the combined university data class. UniversityRequirementsController handles the requirements for all universities. ExcelParserController handles the data from the Excel files. ExamResultController and NominatimApiController hande the data gathered from Vipunen and Nominatim API's, respectively.

We have also unit tests for parsing and testing JSON data structure.

4. Design Decisions

4.1 Key Decisions

Our first key decision to making this project a reality was to select the core, key, component technologies that aid our development to a better pacing that allows us to concentrate more on the feature making instead of creating the tools needed for completing the given task.

React was chosen as it is the standard for web development and Java is the most commonly used to implement backend logic instead of doing native applications like JavaFX. React architecture like most Web Application styles are known for dividing two different codebases, one for frontend and the other one for backend. This is a generally a good practice since the code itself is very different from each other.

We do not have a database since our other means of saving data are used instead. Our Cache system is used for saving locally the data that has been downloaded. This allows very quick loading times if the unchanged data is used again. Also, our cookie system allows locally saved favourites to be saved to the computer which removes the need for database for saving preferences.

We also tried to focus on the SOLID principles. Single responsibility principle was mostly done by trying to divide functionality quite a lot so that overall, the number of lines of code per class was relatively small and the functionality of the classes was easy to decipher. That can be seen by the number of different packages and especially data classes we have on the backend code. Open-closed principle can also be seen in the backend, most notable in the Cache class where the generic programming design can be seen with the head Cache class from which all the other Cache classes are derived.

In the frontend open-closed principle can also be seen. The component-based design where UI-components can be reused is a good example of extendable code.

For Liskov substitution principle the Cache is a good example since using Cache as a superclass works for the variables it has without breaking the program.

4.2 Technology and Tools

Spring framework was chosen for the Java side because we were the most familiar with it out of all the Java frameworks. React was chosen because, as mentioned before, it is well suited for an application like this, and it is a standard for web development.

Most of the libraries we used were chosen because they were either the most common for the specific use case or because we were already familiar with them.

4.3 Quality Requirements and Constraints

Time was the biggest constraint. We also had no prior experience on React, which made the whole frontend development harder.

4.4 Rationale

One of our biggest time sinks was the web scraper. It was needed for providing some critical data for our grading system, since Vipunen API did not give us this data. This was scraped directly from the coalition of universities website https://yliopistovalinnat.fi/. The time investment was worth it, the scraper works really well and provides a lot of valuable data.

Cache system is other one that was created as a generic programming implemented class to provide a base class for Cache systems on our applications. This demanded time to get it right since generic programming can be more difficult.

One another non-programming related time sink was the acquisiton of Excel files from Vipunen's excel file system. We had to click through excel dropdown boxes for 2 hours per person to get all the data for the admission points for universities and universities of applied sciences.

5. Self-Evaluation

5.1 Design Evaluation

The original design was achieved, although it is not quite the same as what we had hoped for. The initial design was maybe a bit too complicated, and we had to scrap some of the optional plans for the application because of time constraints. That does not, however, negatively affect the application; all the key features have been implemented.

5.2 Changes Made

Our rating system had to be scrapped for both time and UI reasons. The simpler design might even prove to be more beneficial. The key features like graphs and filtering are there. The rating system in the original design was also quite superficial, the five-star

rating system is not flexible enough to tell a difference between for instance four-star and five-star universities on a fundamental level.

All the other promised features in the mid-term submission were done.

6. The Use of AI

6.1 Tools and Methods

ChatGPT was used in generating javadoc-comments. It has also been used in generating some code and unit tests. ChatGPT was also used to create a template for this design document. ChatGPT also created the compass icon for our software, although it had to be modified, since AI didn't get the points of compass right.

6.2 Prompts and Use Cases

For javadoc creating, ChatGPT was just given the class and asked to create javadoc-comments for the class. Unit tests were created with basically the same approach except that after giving it to the AI, the test was modified to be more useful to our usage of said class. Otherwise, the AI was used to think of ideas how to do something in our application. For example, when creating the charts for the high school and university data, we asked it how we could create this in react and it gave recharts as an option, so we ended up using it.

Using AI was both useful and problematic. Sometimes it wasted our time by providing code that did not fulfill the instructions. It could also with larger prompts lose the context. It is best used for small, self-contained, problems where the programmer knows the context and needs that self-contained code so that the vision can be fulfilled.