Chapter 1

Background

When designing the new Autograder front-end, it was necessary to develop other parts of the system for emulating dummy-data for the project. One of the most conventional ways of doing this is to create a database that stores the dummy-data, and have a server serve the frontend with that data. The system works in many ways as the fully functional Autograder, with predefined protocols for communications so that the new front-end can be implemented and deployed with a relatively small amount of effort. Making these protocols require us to both think about the current needs and future needs. Even though the system is currently up and running, many new ways of communicating with the new frontend is planned and will be implemented in the new system. It is however, no small task to create these sub-systems. We will go though some of the technologies we are using in this chapter.

1.1 Front-end

Making the new Autograder requires us to make a website to show and control all the systems that work in the back. Pages for showing grades, students, groups, teahers and admins have to be made. This includes pages for managing the pages and so on. Traditional webdesign takes a lot of time, where the process normally involves creating a design, coding the content and then styling the page. HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) is the most used way of showing content and styling it, however, this process takes a lot of time, and frankly, is in most cases the boring part of the job.

Making the new Autograder requires a solid framework for HTML and CSS that removed

the need to hard-code every DOM-element (Document object model), and eliminate the need to design both mobile and desktop-versions of the same website. It was decided early on to use *Twitter Bootstrap* [ref to about page]. The framework is in it's third iteration (soon 4th with Bootstrap 4.0 Alpha being shipped early 2016). Bootstrap removes the need to make all the elements of the page, by using predefined components placed on the page. The framework comes with build in styling, so the job of the designer (or rather lack of designers) is made easier. Compared to vanilla coding in HTML and CSS, the framework can remove countless of hours of coding, and the results look clean and functional.

As with the JavaScript and data binding, we use React. React is an open source JavaScript library designed and maintained by Facebook **(ref here)**. The framework provides a view for data rendered as HTML, and maintains a state that can change depending on the data that comes from the server. This removes a lot of overhead programming that the developers usually have to think about. If the data in the server changes, the change is reflected on the frontpage in realtime. Say for example that the users rights are changed from Student to Teacher. The frontend will have to reflect that change, and show data relevant for a teacher. With React, this change can be made without having to update the page manually. The document object model, DOM for short, is always updated. Updating of elements and data on the page is one of Reacts strongest sides, therefore, it is a very good library for Autograder frontend.

The choice between ReactJS and AngularJS (the other big JavaScript-library in the industry) is really based on preference. The component based pipeline that React offers is very appealing when the website will recycle a lot of the same elements. The new frontend can be compared to a single-page-application, where the different pages are loaded in without refreshing the page. Normally this requires a lot of loading and unloading of elements, and ReactJS already have this build in to the framework.

1.2 Back-end

To be able to make the front-end without using real datasets, it was necessary to create a functional server that can get data from the storage solution and serve it to the front-end. The original Autograder already use Go, short for Go language, as the programming language for the back-end. Go is very fast when it comes to web-servers. Making a webserver requires just a few lines of code, and handles multiple connections in go-routines, which is similar to threads. Therefore, it is not a hard choice when selecting the programming language of the back-end. The web-servers job is to be the link between

the data stored in the system and the front-end. As mentioned, predefined protocols for communication, such as data structures created with JSON (JavaScript Object Notation) is used to communicated with the front-end. Sending packages with content that the front-end can decode and handle in a predefined way. Consequently, the back-end can be written in many programming languages without effecting the front-end or storage.

Logic for roles, such as admin, teacher and student are also managed in the back-end. The logic behind the login page and authentication with Github is also managed back-end in the web-server. The Go language is a general-purpose language designed as a system programming language. It focuses heavily on concurrency and garbage-collection, and is therefore perfect for web-servers.

1.3 Storage solution

Even though the scope of the assignment is the front-end, a storage solution was necessary. The original system uses key-value-store, which is great for storing data which can be found using a key or index. The key-value-store is really fast when searching for keys, but lack the relational structure that Autograder needs. The new Autograder need a system that can store relations between roles and courses. Therefore, the solution that is used is a relational database. The database is filled with dummy-data from the beginning, so that the front-end and server can work with data. The dummy-data will represent real data, and look like the data that the real front-end will produce. The data can simply be replaced by real data later on if the new front-end is implemented in the existing Autograder-environment.

The data is stored in the relational database, and can be retrieved using MySQL. The data i retrieved using string-based commandos, such as "Get all from user_database". This is a very practical way of getting data, and combining this with Go can help make the back-end modular. A modular back-end can simplify the process of expanding the system in the future.