Autograder frontend

Thomas Darvik and Tomasz Gliniecki

February 15, 2016

Contents

Chapter 1

Introduction

[THERE ARE SOME PROBLEMS WITH FRONT-END VS FRONT END – FIX IT]

Compared to previous times when user interface was largely controlled by the software engineers that wrote them, many modern day systems are created by user experience and designers. Whether the user is a costumer or an employee, the path the user of the program has to take to get from point A to point B has become its own field. This requires the author of the program to be more aware of all the aspects of the program they are writing and always think of alternative ways to do the tasks at hand. Equally important is the usability of the program. Modern user interfaces on the web should also take into account that some users might have disabilities or other problems that prevent them from easily navigating the site.

Most programs go through many changes and iterations throughout the design-process and development phase. Therefore, it can be useful to revisit certain parts and rethink some aspects of the program. At the University of Stavanger, former master student Heine Furubotten (insert reference) created the Autograder-system for automatic code submission and a practical feedback for students. The system was designed to remove all the overhead that the students must think of when submitting the code. The former way of submitting code required the teacher and student to share the code, where the teacher would have to download the code and run it, before giving feedback to the student. This is a long time-consuming process, and the time it takes for the student to get feedback and approval is long. Therefore, the Autograder system solves many of the earlier problems of the old system. In addision, the Autograder system is connected to Github, a git version-control-system, where sharing and managing code is very easy. The system was created so that the teaching staff could do a faster and better job with feedback and learning. The idea of having the code automatically graded and corrected is not a new one, however, at the University of Stavanger, no such systems existed and thus the program was created.

The original Autograder system is created with a web-based front-end system and a server-side that handles all the assignments and other tasks. The program is divided in based on your rights as a user. There are different roles for students, teachers and admins, howerver, everything is managed on the web. The current design completes many of the goals of the system, however, there are some problems that needs to be addressed. One of the goals of the original author is to simplify the process of grading the students assignments. In the new Autograder system one of the main goals has been to improve the time it takes the student assistants and teaching staff to grade the students, by reducing the number of navigations and click the teacher has to use. This is one of the ways the new Autograder system has been redesigned and re imagined.

The system has already been tested on both master and bachelor level programming courses. These courses are advanced and require a lot of programming. Since the program has been up and running for a while, it is easier to find problems and obvious user interface issues. The goal of the approach is to see the system in action and testing new features, without having to make a fully functional system that is lock both in functionality and features. Talking to teaching staff and students after courses has gathered good feedback, and some improvements to the system has been planned out and changed. The scale of the system has also improved. Now that more courses have been put through the system, some aspects of the grading and approval will be redesign and rewritten. The natural transition from the original system to the new would not have been possible without users feedback. The main goal of the project is to improve the current design-faults and harden the system where it is possible. Teachers have given their feedback and questions about scalability has been raised.

Furthermore, there have been questions about mobile solutions and other feature of the page. Now that the system is maturing it will be a natural progression to think about mobile solutions. In the ever growing mobile-market, an increasing number of hand-held devices have been introduced to the market, and users will expect this to be integrated in the Autograder system. Nevertheless, the main focus of the new Autograder is not to implement mobile solution, rather rewrite and revisit the original design and front-end system.

The original backend-system is created using a combination of multiple technologies and programming languages. The main programming language is Go, a low level language created by Google. [insert reference]. The programming language it self is fairly new compared to other languages, however Google has done some incredible work when it comes to server-handling and server-oriented features. It is therefore very easy to make web-servers and a back-end systems that communicates with other systems, making Autograder a very fast application. The original web-server of the Autograder system has been rewritten, and discussed later on in this thesis. The storage solution of the original Autograder has also been rewritten from a key-value database to a more conventional relational database. This is also discussed later on in this thesis.

Chapter 2

Motivation

For maximum learning effect when learning a new language and working with multiple advanced courses at the same time, quick feedback is the key. Student should be able to get feedback in relatively short time, where it should be easy to see mistakes that were made and ways to improve it. Therefore, is should be easy for the teacher to make comments and give feedback to the students both on and off campus.

The original way of submitting and approving code is cumbersome in many ways and requires time. The original author of the Autograder system wanted this time and effort to be minimized, so that the quality of the feedback can be maximized. The advanced programming courses at the University of Stavanger normally contains a number of assignments with short deadlines, where the feedback students get can depend on a number of factors. The Autograder system is designed to remove some of these factors and effectively remove a lot of overhead work for the teaching staff and students. The program has been in use on a number of advanced programming courses over the last year and informal talk with students and teaching staff has generated a lot of ideas that should be implemented in the new system. This has been taken into consideration and we propose a new Autograder frontend with improved systems both for students, teachers and maintainers in the hope that even more overhead will be removed and the process of grading and giving feedback will be even simpler.

Chapter 3

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 dummy-data for the system, and have a server serve the front-end 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. Nevertheless, it is no small task to create these sub-systems and some of the technologies are introduced in this chapter.

3.1 Front-end

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 removed the need to make all the elements of the page, by using predefined components that can be placed on the page. The framework comes build in with styling, so that the job of the designers (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 will normally look clean and function in a good way.

As with the JavaScript and data handling we used a React, an open source JavaScript library designed and maintained by Facebook, [ref here] providing a view for for data rendered as HTML. The same way Bootstrap helps with DOM-elements, React removes a lot of overhead programming that the developer normally has to think about. Updating of elements and data on the page is one of Reacts strongest sides, therefore, it is a very good library for Autograder front-end.

3.2 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 server it to the front-end. The original Autograder already use Go as the programming language. The Golang is very fast when it comes to web-servers. 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. The logic behind the login page and authentication with Github is also managed back-end in the web-server.

3.3 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 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 be filled with dummy-data from the beginning, so that the front-end and server can work with data. This data will look like the real data that will be served to the system later on if the new front-end is implemented in the already working Autograder system.

Chapter 4

Design and Technology

Planning user interface involves a lot of time spent at the drawing board, without a lot of coding. This is necessary to avoid rewriting big shank's of code over and over again. Things like wireframes, use cases and user stories are used to build fundaments for a new user interface.

4.1 Wireframes

Creating wireframes is a crucial part of user interface design. The point of wireframes is to have something that is easier to change than the code you are going to write. Before coming to the programming phase, one can iterate over a lot more design propositions, and is able to change things up easily during this initial or planning phase. One could call wireframes a foundation of the application you are developing, wireframes are not supposed to show the creative design, but rather reflect on the technological and use related aspects of the software.

During this project a software development methodology called scrum was used, this helps manage product development process significantly. In scrum, planning phase never ends, there is a endless loop of planning and implementation as the project develops. This gives a bit more freedom while drawing wireframes, the focus is on one or two specific features of the software and iterating until the feature is finished.

4.2 User stories and use cases

To achieve the goad of creating good user interface, a lot of work needs to be put in going through how the applications is going to be used in theory. We create a set of use cases, these describe individual functions that the user would want to execute. This can be shown in a form of a graph, that starts usually at the default state of the software and traverses and lists actions that are needed to achieve the goal.

User stories are supposed to help develop good use cases, they are sentences that describe The type of a user, the goal and reasoning behind a given action. An example would be:
As a administrator, I want to get to the users section, so that I can remove a user.
This gives a great starting point for creating good use case, user stories are created by comping up with the actions relevant to the purpose of the software. By looking at user stories, decision is made on which of the user stories are a priority. Next an appropriate use case is made.

4.3 ReactJS

In this thesis a couple of new technologies and frameworks were used. ReactJS is a JavaScript library for building user interfaces, it abstracts away the DOM from a programmer, gives a programming model similar to OOP[1] to generate appropriate html document. An additional extension of React, called react-bootstrap was used to make the creative design of the application easier. React-bootstrap is just a normal twitter bootstrap[2] wrapped in syntax that reassembles that of React elements.

There are a lot of problems with the solution involving ReactJS. First of all this library is created with so called single-page applications in mind, alternatively React could be used to implement smaller portions of a website, like a real time chat window, while the rest of a page is generated in a different way.

First complication is the amount of computation that is needed for a large complicated application to be generated with React. While small application like messaging app or chat window works great with React, a bit more thought needs to go into design of much more complicated applications like Autograder.

Application that was written with React is being compiled to one JavaScript file, this is the source file of the whole front-end of the application. This means that server is serving out one file and the rest is being compiled on the client side, the necessary data is being fetched from the server as it's needed, see section 4.4 on websockets.

The problem arises when deciding on what should be rendered and when. It's obvious that we do not show or render every application element at once, we need a way to split the rendering into *views* in other words have multiple states of the application that are associated with given url. We can achieve that by having switch statements in out code that looks at the given url, and determines what part of the code should be evaluated and rendered. This might get complicated as the application grows, to help us manage both the url states and overall improve the code readability, another library that was build on top of React was used.

React-router is a routing library that takes care of managing urls, and helps organize routes by simply looking at the nesting patter of the UI components that you want to render, furthermore it is possible to determine the parents and child elements by looking on either url or nesting patter in code, this helps make the code more readable and is helpful when debugging.

4.4 WebSocets

WebSocets is a protocol that enables two-way communication between a client that is running untrusted code, and a host that is configured to communicate from that code. With this one can send and receive event-driven responses without having to poll the server for a reply. The decision to use WebSocets was somehow indirect, while working with React, it became obvious that the old pattern for receiving data from server would not be optimal for the the way React works and the way the development of the Autograder goes.

React works by listening to the changes on separate UI components, and updating their state in real time. When fetching data asynchronously, by using techniques like AJAX, the only way to update UI component is to poll the server for change and ask React to re-render on that basis. Although this is not a bad way to do things, and Autograder does not really need a lot of polling to work correctly, there are no obvious disadvantages of using WebSocets, on top of that, the way it works is more similar to the way React does things client side.

The advantages of having constant connection to the server are that the contents on the page can be updated in real time when the server updates its state. One of the use cases that came up while discussing Autograder, was to have some sort of notification system built in the system. The problem was

[1]Object oriented programming

[2]twitter url

that in the prevois version of Autograder, it was almost impossible to tell whenever someone enrolled to a course or some other important event happend in the system that needed Admin og Teacher attention, without looking for that specific event. Despite the fact that notification systems are fully possible to implement by doing it AJAX way, it would still require an event on the client side to trigger the update and notify the user, with connection based communication the update would happend almost instantly.

As mentioned before, WebSocets integrates pretty well with React, the way to do it, just simply listen to the incoming data through the WebSocet connection, and set the components underlying state - while react listens to the componetns state, it will render when changes are detected.

4.5 Bootstrap

```
<Button>
    <div>
        <a></a>
    </div>
</Butoon>
```