Sprintdoc 1608
Reimplementation einer Webapplikation für Schulen

Pascal Ernst
70302367
Ostfalia Hochschule für angewandte Wissenschaften

März 2016

1 Implementation des CRUD der Klassen

Auch bei den Klassen verlief die Implementation des grundlegenden CRUDs ohne Probleme. Hier will Ich eine weitere Funktionalität einbauen: Man soll die Schüler, die in den jeweiligen Klassen pro Schuljahr (inzwischen Semester genannt) sind, sehen können und bearbeiten können. Dazu habe Ich erstmal ein Block designed, mit dem dies relativ einfach möglich ist. Allerdings wurde auch klar, dass Ich dafür endlich einen größeren Anteil an Javascript-Code brauche.

2 React

Um erweiterte Funktionen und bessere Benutzerbedienbarkeit in das Programm einbauen zu können, wollte Ich Client-seitigen Code mithilfe einer Javascript-Bibliothek implementieren. Dies wollte Ich bereits vorher schon mit ReactJS, die den Html-Code in Komponenten strukturiert, machen. Die normale, unstrukturiertere Variante mit JQuery oder MooTools habe Ich aus Erfahrung nicht mehr in Betracht gezogen, die Übersicht geht bei größeren Javascript-Dateien verloren. JQuery werde Ich aber weiterhin als Ergänzung zu ReactJS benutzen, da es auch einige grundsätzliche Javascript-Funktionen (Z.b. für Arrays) implementiert.

Zuerst ging es darum, diese Bibliothek und möglicherweise weitere in das Projekt zu integrieren. Hier boten sich mir zwei Gems (Ruby-Bibliotheken) an, beide Open Source und kostenlos verwendbar:

- `react-rails`, eine relativ kleine Ruby-Implementation für Ruby on Rails Projekte, die es erlaubt einzelne ReactJS-Komponenten zu kompilieren und in die `assets` (die kompilierten Html-, CSS-, Bild- und JS-Dateien für den Client) packt.
- `react_on_rails` ist ein komplett neues Tool-Stack, der in das Ruby on Rails-Projekt mit integriert wird. Es benötigt NodeJS und NPM, ein Package Manager für Javscript-Bibliotheken, um weitere Abhängigkeiten für diesen Stack zu integrieren. Die größte davon ist Webpack, welcher sich anstatt Ruby on Rails um die assets kümmert und für den Client verpackt.

Der grundlegende Unterschied zwischen den beiden Bibliotheken ist die Tiefe, wie weit sie in das Ruby on Rails-Projekt eingreifen und eine Umstellung von der normalen Programmier-Umgebung erfordern.

`react-rails` erlaubt eine leichte Integration in bereits existierende Programme. Einzelne Komponenten lassen sich schnell verstreut in die Webseite integrieren, da es nichts an den restlichen assets verändert und keine weiteren Abhängigkeiten besitzt.

`react_on_rails` dagegen lässt sich nur sehr umständlich in eine existierende Applikation integrieren. Da alle assets mit Webpack gebaut werden sollen, müssen die Pfade

und teilweise die assets selber verändert werden. Die assets werden dann von Webpack in eigene Dateien gepackt, die wiederrum an Ruby on Rails zur Verarbeitung überreicht werden. Die Ruby on Rails-Engine muss so umkonfiguriert werden, dass sie diese externen assets mit beim Aufbau der Webseite einbaut. Dieser Aufwand macht sich allerdings auch bezahlt. Durch Webpack gewinnt der Entwickler die Hot-Reload-Funktion, mit der eine Webseite bei einer Code-Veränderung aktualisiert wird, ohne sie neuzuladen. Außerdem lassen sich damit große Clients, die den Html-Code selber rendern und den Server nur noch für Daten belasten, realisieren.

Während die erste Variante also einfach zu integrieren ist, vermisse Ich hier die Möglichkeit einen größeren Client, den Ich für das Frontend vorgesehen habe, zu realisieren. Ich versuche also, die `react_on_rails`-Bibliothek in das Programm zu integrieren.

2.1 Integration von `react_on_rails`

Das Projekt bietet ein Skript an, dass für eine relativ schmerzfreie Integration der Bibliothek in ein bereits vorhandenes Programm sorgen soll. Allerdings funktionierte das bei mir nicht, denn wie sich herausstellte ist dieser Skript veraltet und wird gerade überarbeitet.

Diese Erfahrung habe Ich bereits öfters mit Open Source Ruby-Bibliotheken gemacht. Sie verändern sich so schnell, dass Dokumentation und Tools kaum hinterherkommen.

Da dies nicht funktioniert hat, habe Ich die in der Bibliothek enthaltenen Beispielapplikation vorgenommen und manuell meine Applikation daran angepasst. Ein Vorteil an Ruby ist, dass der Code sehr lesbar ist - Ich fand mich bei Fehlern auch ohne ausführliche Dokumentation in der Bibliothek und Ihren Tools zurecht und konnte sie beheben.

Statt einen zwei Clients

Foreman changes