Sprintdoc 1610
Reimplementation einer Webapplikation für Schulen

Pascal Ernst
70302367
Ostfalia Hochschule für angewandte Wissenschaften

März 2016

1 Implementation des grundlegenden Frontends

1.1 Kompatibilität

Auf das Frontend sollen alle Schüler und Lehrer der Schule zugreifen können. Somit muss es weit kompatibler mit den Webbrowsern sein als das Backend. Unter den zu beachtenden Bereichen fallen:

- Die Webbrowser-Kompatibilität. Der Internet Explorer 9 unterstützt nicht die gleichen Features wie der Google Chrome 49. Hier kann es zu Problemen hinsichtlich der Darstellung (CSS) und der Funktionen (JS) kommen. Ältere Webbrowser beherrschen bestimmte HTTP-Request-Arten wie `PATCH` und `DELETE` nicht.
- Die Download- und Upload-Datenmengen einzelner Benutzer ist stark limitiert, sei es durch die Internetverbindung selbst oder Limitierungen im Vertrag des Anbieters.
- Ob Javascript bei dem Webbrowser eingeschaltet ist. Einige Benutzer schalten mit Absicht Javascript ganz aus, um Bandbreite zu sparen oder um sicherer im Web browsen zu können. Viele Seiten funktionieren nicht mehr ohne Javascript, weswegen diese Benutzer nur Teile des Webs nutzen können.

Der erste Punkt, die Webbrowser-Kompatibilität, hat sich in den letzten beiden Jahren durch Microsofts offizielle Einstellung des Supports für Internet Explorer 7 und 8 erheblich verbessert. Mithilfe verschiedener Polyfills und Server-seitigem vorkompilieren können eigentlich nicht unterstützte Bibliotheken auch in älteren Webbrowsern zuverlässig benutzt werden. Bei der Darstellung gibt es allerdings immer noch kleinere Unterschiede zwischen den einzelnen Browsern, hier muss man leider einfach ausprobieren wie man die Webseite auf allen Browsern ähnlich / gleich aussehen lassen kann.

Beim zweiten Punkt sollte man Browser-seitig Caching-Strategien korrekt anwenden, damit die Benutzer nicht unnötig die gleichen Dateien immer wieder herunterladen. Um die Erst-Datenmenge beim Laden der Seite weiter zu verringern, werde Ich die Daten der Funktionalität und der Darstellung in mehrere Dateien aufteilen, zum Beispiel in Frontend und Backend. Dann muss der Benutzer nur die Funktionen laden, die er auch wirklich braucht.

Ausgeschaltete / nicht funktionierende Javascript-Ausführung ist mittlerweile nur noch wenig verbreitet. Nur wenige Webseiten unterstützen eine gute Bedienbarkeit auch ohne Javascript, weil dies für moderne Interaktionen benötigt wird. Will man auch Webbrowser ohne Javascript unterstützen, so muss man Teile der Applikation neuschreiben oder im schlimmsten Fall sogar zwei Ansichten einbauen, eine, die mit und eine die ohne Javascript funktioniert. Das hoffe Ich mithilfe von ReactJS zu lösen. Die Idee hierhinter ist dass der clientseitige Javascript-Code, der die einzelnen Komponenten der Webseite rendered, auch serverseitig benutzt werden kann um die Seite

einmal vollständig vorzurendern und dann komplett an den Client zu schicken. Hat dieser Javascript, werden dort vom gleichen Code die modernen Interaktionen aktiviert, ansonsten hat er trotzdem eine funktionierende Webseite.

1.2 Implementierung

Ich benutze den gleichen Aufbau von Tools wie im Backend, mit dem Unterschied dass Ich für das Frontend die Webseiten serverseitig vorrendere. Dafür schreibe Ich meinen Programm-Code auf dem Server (Ruby on Rails) so, dass dieser ein JSON-Objekt erzeugt welches an den `ruby_on_rails` - Helper übergeben wird. Dieser speist diese Daten in eine virtuelle DOM (Der geparste Aufbau einer HTML-Webseite) ein. Diese DOM wird dann an den Webpack-Prozess gegeben, der den Javascript-Code zum rendern des Clients darauf ausführt und diese DOM wieder als HTML-Code an den Ruby on Rails Server zurückgibt. Dieser Code wird dann als Antwort an den Client gesendet.

Um auch nicht-Javascript-Nutzer die Webseite anzeigen zu können und Navigation für diese Nutzer zu erlauben, benutze ich die Bibliotheken `react-router` und `Redux`.

Die erstere erlaubt mir, Klicks auf Links zu anderen Bereichen der Webseite abzufangen und anders zu rendern. Benutzer ohne Javascript werden bei Klick einfach auf die neue URl geleitet, sodass der Server wieder die HTML-Seite generiert und an den Client sendet. Bei Benutzern mit Javascript wird im Hintergrund nur die neuen Daten vom Server abgefragt und der Client kümmert sich selber darum dass die neuen Komponenten generiert und angezeigt werden.

`Redux` ist ein Tool dass das Management des States des Clients, also der im Client gespeicherten Daten, vereinfacht. Dort kann Ich nur mithilfe wohldefinierter Prozeduren (`Reducers`) den State ändern, wodurch verhindert wird dass das Programm mit der Zeit unübersichtlich die Daten verändert und schlimmstenfalls die angezeigten Daten nicht mehr mit denen des Servers übereinstimmen. Redux vereinheitlicht auch die Behandlung der Daten beim Server-side Rendering und Client-side Rendering.

Für das grundlegende Layout erstelle Ich mir die drei Komponenten `TopNavbar`, `MainContent` und `Footer`. In den MainContent werden je nach URL die entsprechenden Daten reingerendert.

Für das Design lasse Ich mich von meiner bisherigen Arbeit inspirieren. Für Farben generiere Ich mir ein `HCL`-Farbschema mit den Grundfarben Gelb und Dunkelblau.

1.3 Menü Einstellungen

Als erste Komponente für das Frontend habe Ich Optionen für den Benutzer hinzugefügt, seine Email und sein Passwort verändern zu können. Außerdem kann man dort

für den Fall dass man seine Karte verloren hat seinen Account sperren.

2 Sprint Aufgabenfertigstellung

Die Gesamtzeit für die Erstellung des Frontends habe Ich auf etwa 20 bis 30 Stunden geschätzt. Am Ende brauchte Ich etwa 40 Stunden für die Erstellung und die erste Komponente des Frontends, was nur ein wenig über den geschätzten Aufwand lag.