Quicknotes AB-02 und AB-03

Anfänge + Benutzerauthentifizierung mit ‘devise’

Zu Beginn erstellte ich mit dem gängigen Befehl ‘rails new’ eine neue Rails Applikation und begann mit der Installation des Gem ‘devise’. Im Ruby on Rails Framework werden Gems genutzt, um die Funktionalität einer Rails Applikation zu erweitern und zu modifizieren. Damit wir das Rad nicht neu erfinden müssen und die Ganze Benutzerauthentifizierung selbst implementieren müssen, wird ‘devise’ verwendet (erstellt Anmeldeformular und Registrierungsformular). Dazu wird innerhalb des Gemfiles folgendes eingetragen: ‘gem devise’. Anschliessend wird mit ‘rails generate devise:install’ das Gem aufgesetzt und weitere Konfigurationen folgten. Nun musste ich nur noch mit ‘devise’ mit folgenden Befehl: ‘rails g devise User’ ein neues Model erstellen, die Tabellenstruktur anpassen und mit ‘rails db:migrate’ alles mit der sqlite3 Datenbank migrieren. Somit startete ich mit ‘rails s’ den Server und konnte schon mit einem vordefinierten Formular Benutzer registrieren und anmelden.

Bootstrap4 + jQuery + popper

Nun kam die Installation von Bootstrap4, jQuery und popper um die ganze Arbeit mit HTML, CSS und JavaScript zu vereinfachen (um beispielsweise vordefinierte Klassen zu benutzen, weniger JavaScript zu schreiben oder Positionierungen zu optimieren). Ein weiterer Vorteil ist Bootstraps Grid System, welches bis zu 12 Spalten beinhalten kann, ausserdem können mit den vier vordefinierten Klassen xs, sm, md, lg verschiedene Bildschirmgrössen angesprochen werden.

Navbar (Bootstrap)

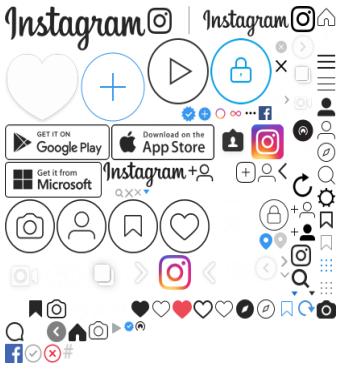
Ein wichtige Funktion von Bootstrap war das Erstellen einer Navbar mit der Klasse ‘navbar’ und ihren ganzen built-in Klassen wie ‘navbar-brand’ für Projektnamen, und ‘navbar-nav’ für die Navigation (meistens mittels -tag). Auch mit der ‘navbar’ Klasse ist die Benutzung von Displaygrössen-Klassen (sm|-md|-lg|-xl) möglich.

Für den Anfang benutzte ich das Beispiel auf: http://getbootstrap.com/docs/4.0/components/navbar/. Im Verlauf der Arbeite wurde sie deutlich angepasst und gestylt.

CSS + SCCS + SASS

Für die stylesheets wurde SCCS verwendet, man hätte auch SASS verwenden können, denn der einzige relevante Unterschied liegt im IU, darum lassen sich beide ineinander importieren. Zwei der Syntaxparser von SASS sind CSS und SCSS. SCCS und SASS enden schlussendlich sowieso als CSS Output. Was aber SCCS praktischer als das klassische CSS macht, ist beispielsweise die Verschachtelung von Klassen, mehr Features, einfachere Variablenschreibweise und die Möglichkeit benutzerdefinierte SCSS Dateien zu importieren.

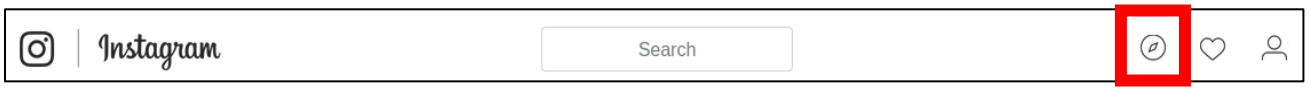
Navbar (CSS Image Sprites + Partial View)



Ein Image Sprite ist eine Sammlung von Bildern, welche in einer einzigen Bilddatei angezeigt wird (siehe Bild rechts). Eine Webpage mit sehr vielen Bildern dauert in der Regel länger zum Laden und generiert viele Serverabfragen. Mit der Benutzung von Image Sprite wird die Anzahl Serverabfragen reduziert und spart somit viel Bandbreite.

Um das Navbar-Menu zu verschönern, wurden jedem Navbar-Button die Klasse ‘core-sprite’ mit der Bilddatei des Sprites gegeben und je nach Button eine Klasse mit dem jeweiligen Symbol und dessen genauen Koordinaten auf dem Sprite zugeteilt. Um die exakten Koordinaten zu erhalten, nutzte ich die Seite: www.spritecow.com. Und um allen Text hinter den Navbar-Symbolen zu verstecken, wurde die Klasse ‘hide-text’ erstellt.

So sah das Ergebnis aus (inklusiv Code für den Entdecken-Button):




```
.explore-icon {
  background-position: -274px -356px;
  height: 24px;
  width: 24px;
}
```

Bis hierhin wurde das ganze HTML in derselben Datei geschrieben (app/views/layouts/application.html.erb). Um das Menu auszulagern, arbeitet man mit Partial Views, welche einzelne Webseitenelemente beinhaltet und in separaten Ordnern gespeichert werden. In meinem Fall wurde der Code für die Navbar in ‘app/views/shared/’ gespeichert und konnte im application.html.erb mittels Rubys render-Funktion angezeigt werden.

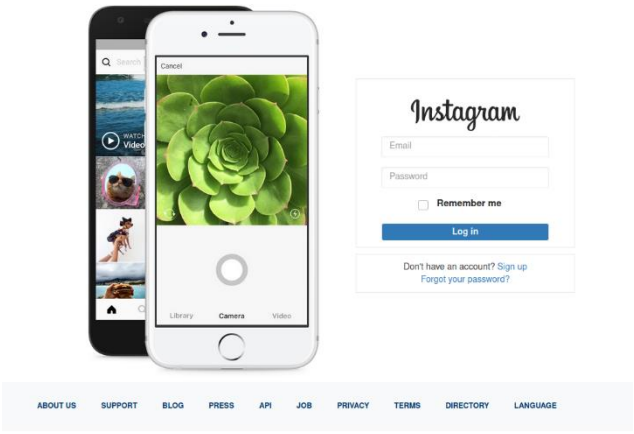
Jedoch sollte man diese Partial View nur angezeigt bekommen, wenn ein Benutzer sich angemeldet hat. Für solche Fälle bietet ‘devise’ eine sehr gute Hilfe an, da man bei erfolgreicher Anmeldung die mitgelieferte Variable ‘current_user’ anspricht. Somit konnte die Navbar angezeigt werden, unter der Bedingung, dass sich ein angemeldeter Benutzer in der Session befindet:

```
<%=render 'shared/navbar' if current_user %>
```

Login-View (Dummy Phone mit Carousel-Funktion)

Als nächstes machte ich mir die Carousel-Funktion von Bootstrap zu nutze um wechslende Bilder auf meiner Login-View zu erhalten. Dafür verschachtelte ich meine verschiedenen Bilder, welche in einem definierten Intervall nacheinander angezeigt werden sollen, in verschiedene . Die Klasse ‘active’ wird für ein einziges Bild verwendet, damit das Carousel überhaupt sichrbar wird. ‘item’ wird zum weiteren stylen verwendet. Aussen werden mit der Klasse ‘carousel-inner’ die jeweiligen spezifiziert. Es wird noch ein weiterer Wrapper verwendet mit ‘id: myCarouel’, damit die Steuerung des Carousels richtig funktioniert. Ausserdem braucht es noch eine Klasse welche angibt, wie sich das Carousel verhalten soll, dies wurde mit der Klasse ‘.carousel-fade’ und jeweiligen CSS-Adaptationen gelöst. Schlussendlich wurde noch alles in die Klasse ‘.carousel’ gepackt, diese Klasse erstellt das Carousel und mit dem Attribut data-ride=’carousel’ aktiviert.

Dieses Dummy-Phone-Element wurde schliesslich auch als Partial View erstellt und gerendert.



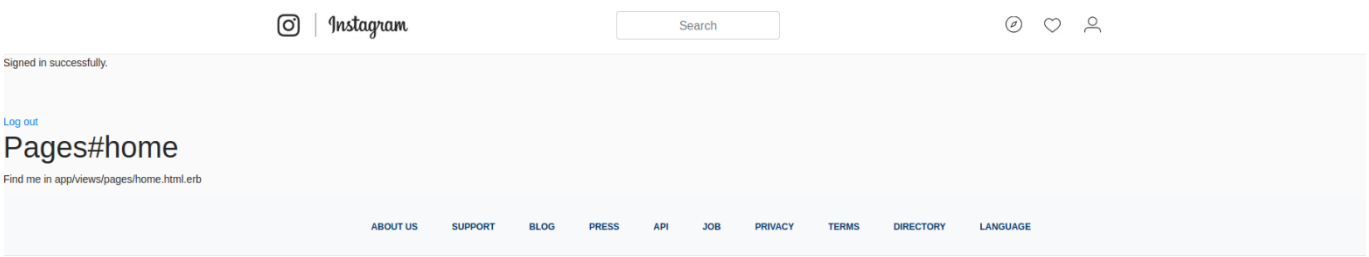
Login-View (neue Migration + Footer)

Was nun noch für Benutzer Authentifikation in der Login-View fehlte, war der Benutzername, weil man als Benutzer bis dahin nur eine E-Mail-Adresse und ein Passwort hatte. Somit erstellte ich mit ‘rails g migration AddNameToUser’ eine Migrationsdatei mit der ich eine weitere Spalte in die User-Tabelle erstellte. Mit dem Befehl ‘rails db:migrate’ wurde dann die Datenbankstruktur aktualisiert. Durch einige Änderungen im Formular der Login-View konnte man nun bei der Registrierung auch einen Benutzernamen angeben.

Nun folgten noch kleinere Anpassungen im Layout um die Seite wie das originale Instagram aussehen zu lassen und zu guter Letzt erstellte ich eine Partial View mit einem Footer, welcher immer angezeigt werden sollte.

So sieht die Login-View nun aus (Registrierung):

Und so die erste Seite nach erfolgreichem Login:



Selbstreflexion / Abschliessende Reflexion

› Was habe ich gelernt? Was wusste ich bereits?

› Wie bin ich vorgegangen beim Lernen bzw. Ausführen des Auftrages?

› Was waren die Schwierigkeiten, wie konnte ich diese lösen?

› Was habe ich nicht verstanden bzw. was konnte ich nicht lösen?

Da wir uns schon letztes Quartal mit Ruby on Rails befasst haben, war mir das Grundgerüst einer Ruby on Rails Applikation bereits bekannt. Vor diesem Instagram-Projekt war mir aber das Gem ‘devise‘ nicht bekannt, somit informierte ich mich vor der Installation darüber um sein Nutzen besser zu verstehen. Im Bereich Bootstrap hatte ich schon einige Vorkenntnisse und verstehe auch die Funktionsweise inklusive einiger Standardklassen. Was aber jQuery und popper betrifft, ist meine Erfahrung noch sehr gering, dies liegt auch dran, dass in den Arbeitsblättern kaum auf die beiden Libraries eingegangen wird. Wiederum fühle ich mich sehr sicher im Bereich CSS, und kann bei Bedarf Elemente verschönern und auch neue Properties hinzufügen, da in den Arbeitsblättern nicht immer alles 100% korrekt ist. Was die Sprite-Thematik angeht, fühle ich mich auch sicher, da ich eine ähnliche Vorgehensweise schon mal benutzt habe in einem anderen Projekt. Im Bereich Partial-Views erstellen, fühle ich mich nun viel sicherer, da wir drei Neue erstellten und es für mich zur Routine wurde.

Ein grösseres Problem für mich war das Carousel. Wenn man nur nach Anleitung der Arbeitsblätter ging, änderten sich die Bilder im Dummy-Phone nicht, somit musste ich lang nach einer Lösung suchen und Carousels von Tutorials anschauen und überlegen wo der Fehler war. Nach langen Recherchen passte ich im Stylesheet zwei Klassen an und importierte innerhalb der Partialview für das Dummy-Phone noch weitere Bootstrap links. Schlussendlich funktionierte alles wie es sollte!

Neue Migrationen zu erstellen um der Datenbank neue Strukturen zu geben, war für mich kein grosses Hindernis. Die neue Spalte „name“ war schnell erstellt. Was aber ein schwierigere Aufgabe war, war das Erstellen der Registrierungsseite mit dem neuen Input „name“. Im Code musste man noch viele Anpassungen tätigen, damit dem Formular die nötigen Daten übergeben werden. Dazu kam noch, dass es ein wenig mühsam war, alles genau so aussehen zu lassen wie auf den Arbeitsblättern gezeigt wird.

Schlussendlich muss ich sagen, dass ich nun im ersten Teil dieses Projektes viel Neues dazu gelernt habe und in den Bereich die ich schon kannte, eine zusätzliche Sicherheit erhielt.