QUICKNOTES AB151-04

Fabiyan Beyeler INF2017.3e

fbe106360@iet-gibb.ch

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Zusammenfassung & Anwendungszweck

In diesem Arbeitspaket haben wir unsere Instagram-Applikation um einiges Benutzerfreundlicher gemacht. Dies geschah als Erstes mit dem Einsetzen des Flash. Im Flash werden temporäre Nachrichten bzw. Meldungen gespeichert. Diese wurden aber noch nicht dargestellt. Diese darzustellen war kein Problem, leider waren diese nicht gestylt. Diese Meldungen hätte man mit Bootstrap sehr einfach stylen können. Im unseren Fall verwendeten wir ein gem namens «toastr». Dieses gem erstellt Toats mit den jeweiligen Mitteilungen. Diese mussten wir nicht selbst stylen und je nach Typ der Mitteilung wurde der Style automatisch angepasst. Ausserdem verwendenden wir eine for-Schleife, um durch alle Meldungen im Flash zu iterieren, damit diese nacheinander angezeigt werden. Diese Toasts mit den Meldungen werden dann oben rechts angezeigt und verschwinden nach einer bestimmten Zeit. Dies hat zum Vorteil, dass der User die Meldungen kaum übersehen kann, diese den User aber nicht beim surfen einschränken bzw. stören. Ein Nachteil können diese Toasts für ältere User sein, für die es zu schnell geht und sie die Meldung nicht lesen können. In diesem Stand werden alle Meldungen der Action übergeben. Validierungen von Formularen werden in unserem Fall beim Model erstellt. Diese Meldungen müssen auch angezeigt werden um eine gut UX zu haben. Diese Meldungen vom Model werden in einer Datei abgelegt. Auch hier wurde eine for-Schleife verwendet um durch diese Errors zu iterieren.

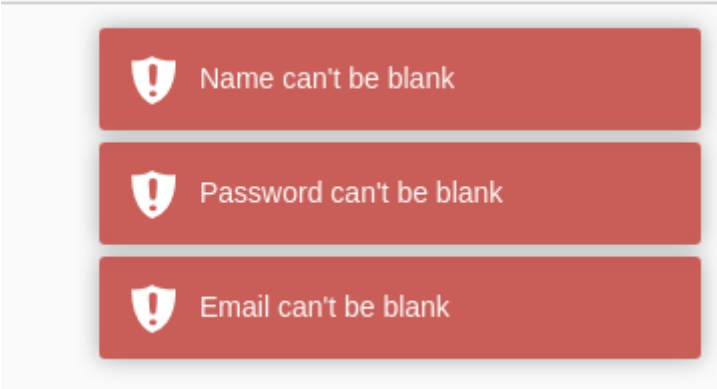

Abbildung 1 Toast Meldungen mit toastr

Um unsere Instagram Applikation so genau wie möglich nachzubauen, mussten wir unsere user-Tabelle erweitern. Bei Instagram ist es möglich eine Website , ein Profilbild und eine Kurzbeschreibung an ein Profil zu hängen. Dies geschah mit einer Migration. In dieser erstellten wir die nötigen Spalten mit dem Befehl «add_column :tabellenname, :spaltenname, :datentyp». Anschliessend muss der Befehl «rails db:migrate» durchgeführt werden, damit die Änderungen in der Datenbank übernommen werden. Ausserdem wurde im routes.rb folgender Code hinzugefügt: «resources :users, only: [:index, :show]». Dies dienst dazu, dass Rails automatisch die Routen zu den Funktionen im Controller erstellt. Wir haben die Routes eingeschränkt. Normalerweise werden sieben Routen erstellt. Man kann sagen, dass diese sieben Routen das CRUD der Tabelle abdeckt. Das heisst für unseren Fall es könnte ein User erstellt, bearbeitet, angezeigt und gelöscht werden. Wir wollen im Moment nur die Accounts bzw. die User anzeigen, deshalb werden bei uns nur 2 Routen erstellt. In der show-Methode wird der User anhand seiner Id angezeigt. Dies geschah im Controller. Der Code lautet «@user = User.find(params[:id])». Mit diesem Code wird das user-Objekt unter der globalen Variable «user» gespeichert und kann so von der View verwendet werden.

Das Profilbild des Users wird in der Datenbank nicht gespeichert. Hier wird mit einem Drittanbieter gearbeitet. Gravatar ist ein Dient, bei welchem man einer E-Mail ein Bild angeben kann. Dieses Bild kann dann über einen Link aufgerufen werden. Dieser Link wird im ApplicationHelper anhand der user-E-Mail erstellt.

Ausserdem wurde auf der Profil-Ansicht ein Modal von Bootstrap verwendet. Ein Modal ist eine Komponente von Bootstrap. Einfach gesagt ist es ein Dialog. Dieser wird ausgelöst, wenn der User auf das Settings-Icon klickt. Dann wird ein Dialog geöffnet und die Seite wird abgedunkelt. Im Dialog kann man ganz normalen HTML,CSS und JS Code schreiben. Der Dialog kann jederzeit mit einem Klick auf die Website geschlossen werden.

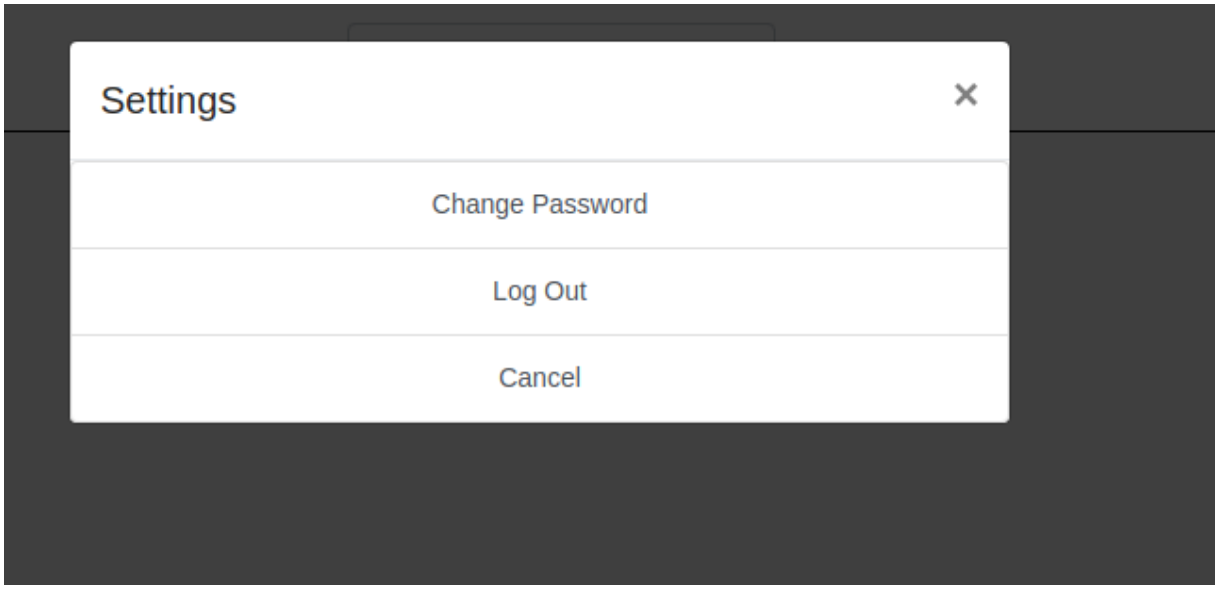

Abbildung 2 Dialog mit Modal

Als letztes wurde die Profil Seite mit dem Icon in der Navbar verlinkt und die Profil-edit Ansicht erstellt. Diese wurde vom gem schon erstellt aber von uns noch einmal überarbeitet.

2. Selbstreflexion

Was habe ich gelernt?

Ich habe durch dieses Arbeitspaket viel Neues gelernt. Mit Toasts habe ich zuvor zwar schon gearbeitet aber nicht in Ruby. Das Übergeben der Nachrichten vom Model zur View war neu für mich. Mit Bootstrap hatte ich auch schon einige Erfahrungen gesammelt, aber Modals habe ich zuvor noch nie verwendet.

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

Ich habe mich so gut wie nur möglich an das Arbeitsblatt gehalten. Bei den Problemen, welche dennoch auftauchen, suchte ich im Internet nach einer Lösung.

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

In diesem Arbeitsblatt fast Aufgaben lösen. Ich hatte zu Beginn ein paar Probleme mit den Toasts, da diese nur als weisse Balken angezeigt wurden. Dies konnte ich dank Tobias

aber beheben. Im späteren Verlauf hatte ich noch einen Bug mit dem Einbinden des Profilbildes. Aber auch dieser konnte ich schnell lösen.

Was kann ich nächstes Mal besser machen?

Diese Woche lief fast alles gut. Ich habe versucht, das was ich letzte Woche hätte verbessern können in dieser Woche umzusetzen. Dies gelang mir eigentlich sehr gut. Leider ist mir aufgefallen, dass ich zu ungenau arbeite. Mir sind während der Arbeit einige Flüchtigkeitsfehler aufgefallen, diese hätte man durch genaueres Arbeiten sicherlich vermeiden können. Ich nehme mir für nächste Woche vor, die Aufträge genauer zu lesen und die Arbeiten sorgfältig zu erledigen.

3. Reflexion über das Gelernte

Abschliessend kann ich sagen, dass ich diese Woche wieder Neues gelernt habe. Ich habe die neuen gems kennengelernt und kann diese im Code verwenden. Die Modals von Bootstrap verstehe ich jetzt und kann diese in mein Projekt einbetten. Ich denke, dass ich auf einem guten Level bin. Ich kann definitiv noch mehr über Ruby lernen. Denn besonders in den Bereichen Konsolenbefehlen fühle ich mich noch unsicher.

4. Instagram Applikation

Registrierungs-Seite:

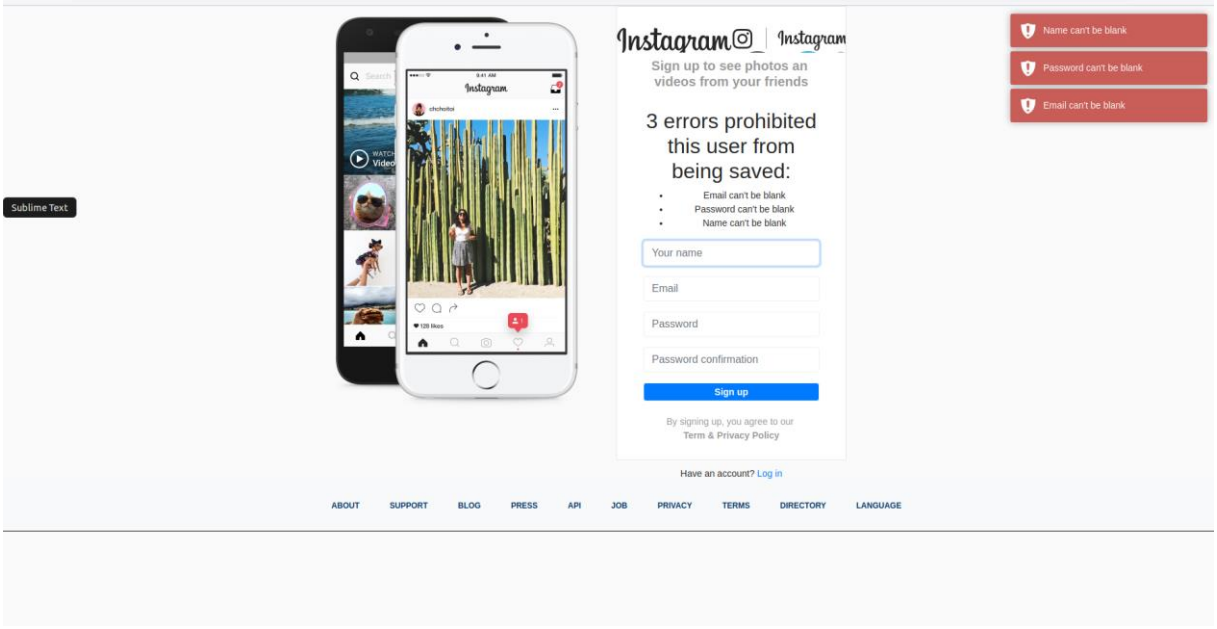

Abbildung 3 Registrierungs-View

Profil Übersicht:

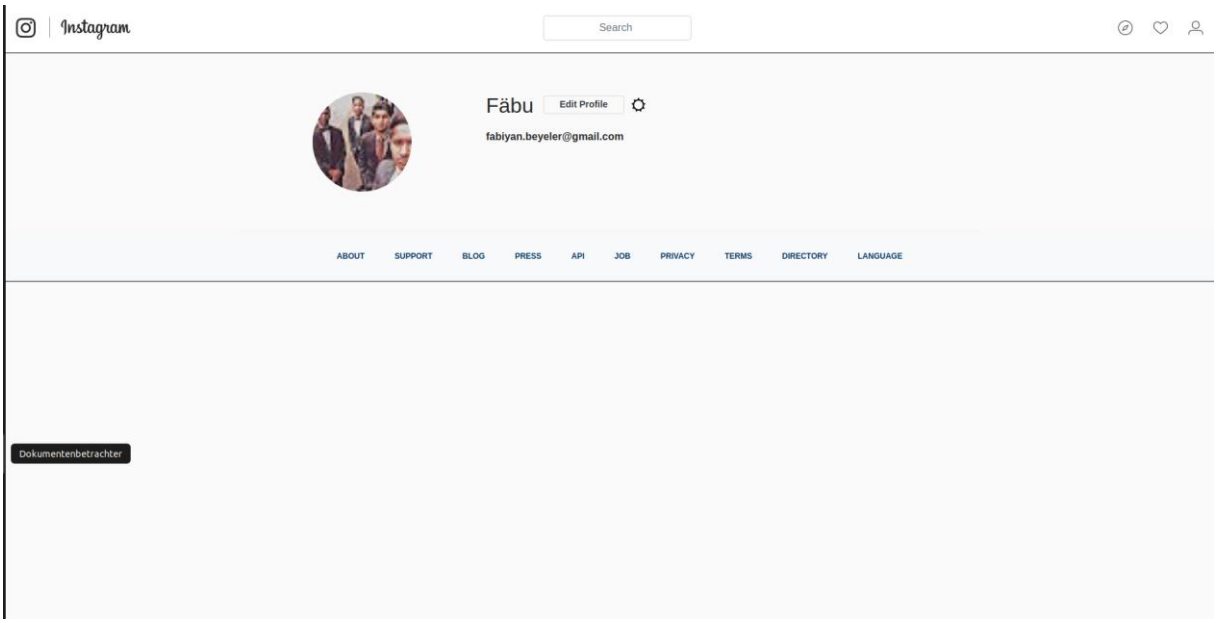

Abbildung 4 User-Show View

Profile-Edit Ansicht:

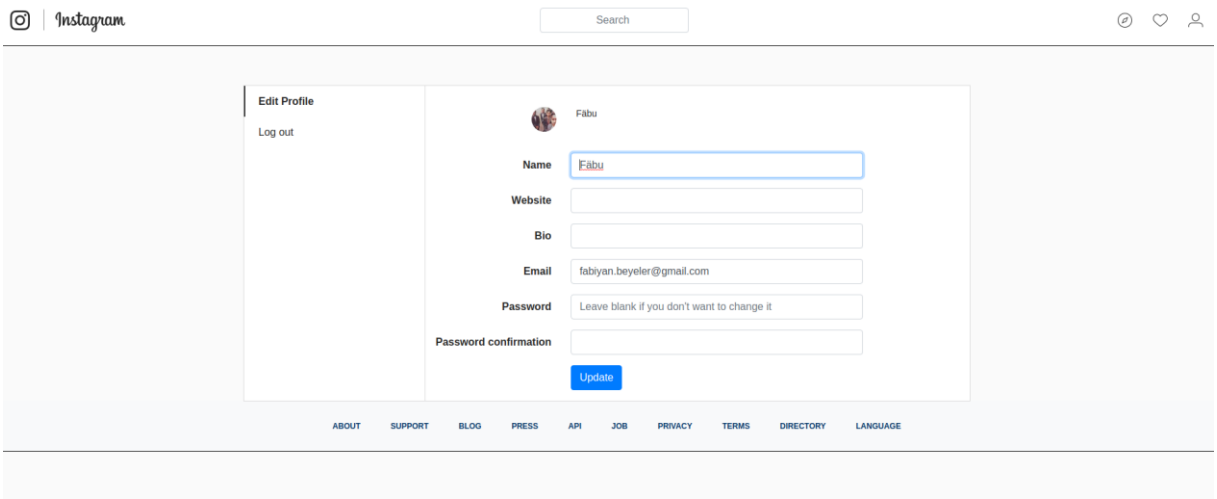

Abbildung 5 User-edit View

5. Lösungen

Partial View erstelllen (Seite 5):

1. File «_devisemes.html.erb» wurde im Folder app/views/shared erstellt.
2. Code wurde eingefügt

```
<% if resource.errors.any? %>
  <script type="text/javascript" charset="utf-8">
      <% resource.errors.full_messages.each do |value| %>
      toastr.error('<%= value %>');
      <% end %>
  </script>
<% end %>
```

Abbildung 6 Codes für das Iterieren durch die Errors

3. Partial View in «application.html.erb» mit der render Methode eingebunden.

Anpassungen um User zu bearbeiten:

1. «registrations_controller.rb» erstellt
2. Code eingefügt:

```
class RegistrationsController < Devise::RegistrationsController
  protected

  def update_resource(resource, params)
    resource.update_without_password(params)
  end
end
```

Abbildung 7 Code um bearbeiten der User zu ermöglichen

3. Routes.rb File mit «controllers: {registrations: 'registrations'}» ergänzt