Instagram

Dokumentation Arbeitsblatt 4

Inhalt

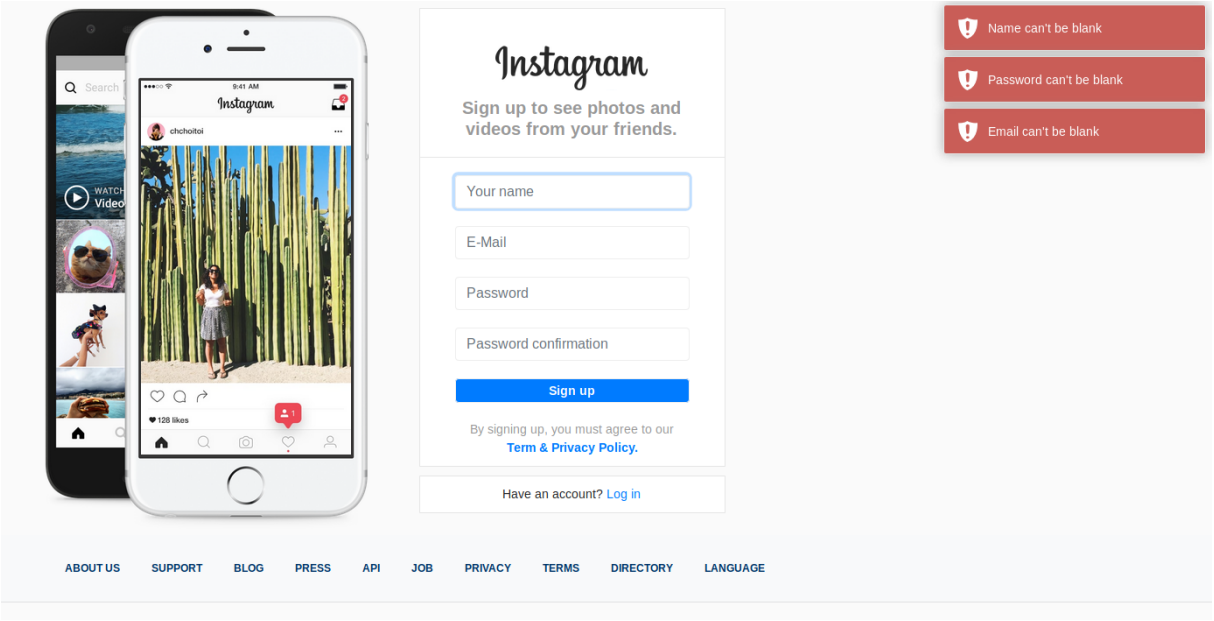
Zusammenfassung und Anwendungszweck

Flash in Rails

In Rails gibt es einen Flash Speicher, in welchem temporäre Variablen gespeichert werden, welche man auf der nächsten View wieder abrufen kann. Devise macht davon Gebrauch und speichert die aktuellen Fehler- oder Erfolgsmeldungen im Flash Speicher ab. Diese kann man dann auf der nächsten Seite auslesen und anzeigen. Beim Wechseln in die nächste View, werden die alten Flash Variablen gelöscht. In unserer App wird der Flash Speicher benutzt, um zum Beispiel den User zu informieren, wenn ein Feld bei der Registration nicht den Vorgaben entspricht.

JavaScript Toast-Meldungen mit gem toastr

Es gibt eine Library, welche Toasts in Rails anbietet. Diese können den Style an die Art von Nachricht anpassen. Bei einem Fehler sind die Toasts rot und bei einem Erfolg grün. Den Text und die Art der Meldung können aus dem Flash Speicher ausgelesen werden und in den Toast integriert werden. Somit braucht man nicht für jede Fehlermeldung selber Code zu schreiben und der Code bleibt übersichtlich. Die Fehlermeldungen aus dem Flash Speicher, zum Beispiel auf der Login Seite, bei einem falschen Passwort, werden damit angezeigt. Anbei ein Beispiel bei der Registration, wenn kein Feld ausgefüllt wurde und auf «Sign up» geklickt wurde:



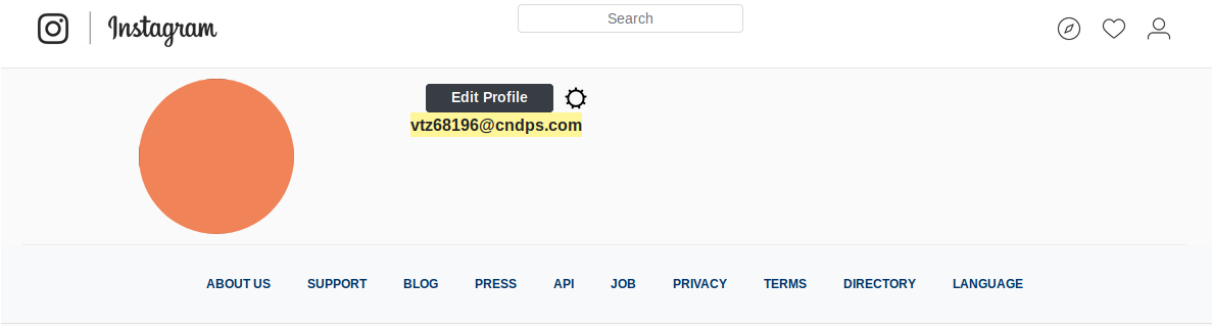
Flash-Meldungen vs. Meldungen des Models

Die Fehler, welche in der Datenbank generiert werden, sind in der Klasse «resource.errors» zu finden. Das ist nicht dasselbe Prinzip, wie beim Flash Speicher, da keine Daten aus der Vorherigen Ansicht gespeichert werden, sondern vom letzten Datenbankzugriff. Die Fehler werden also dann gespeichert, wenn die neue View bereits geladen wurde und sind somit direkt aufrufbar. Wenn ein User in unserer Instagram App versucht, mehr als die erlaubte Anzahl Zeichen als Namen einzugeben, wird der daraus resultierende Fehler aus der Datenbank im Browser angezeigt.

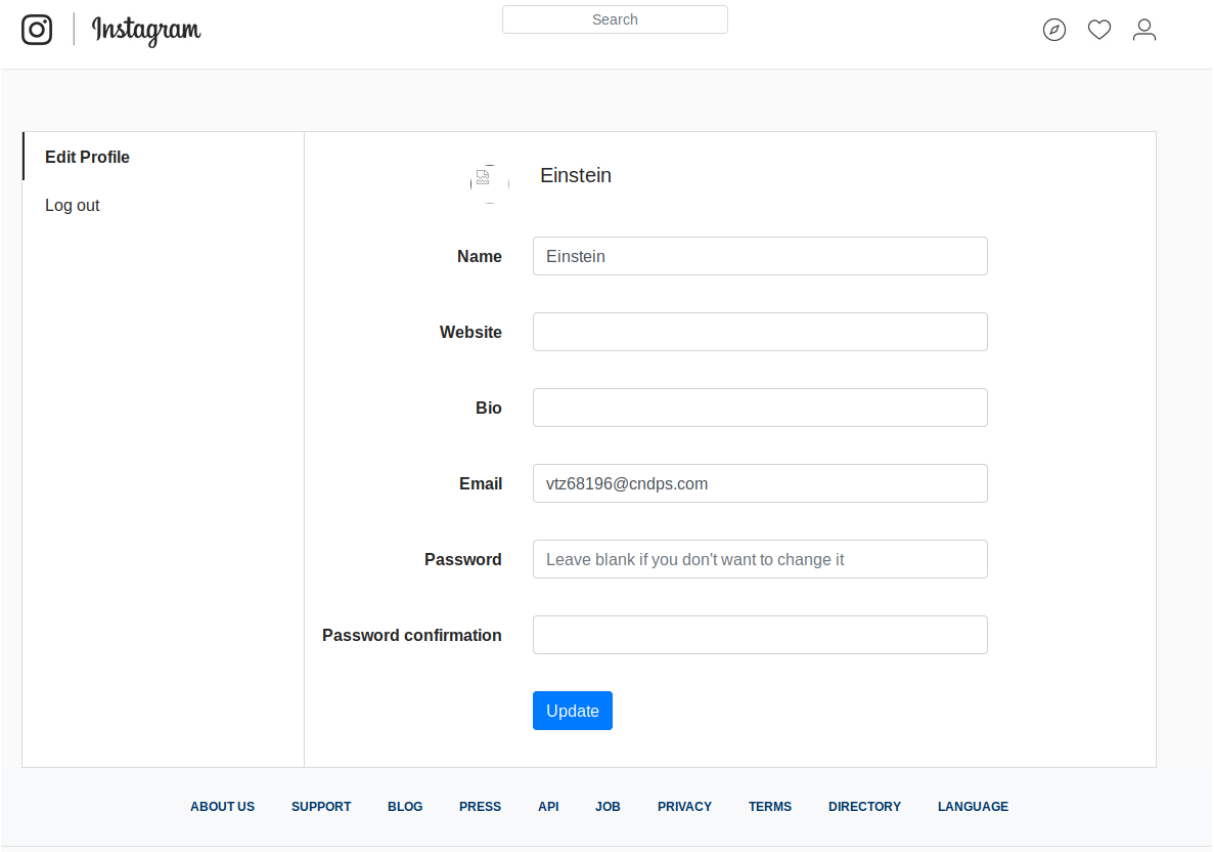
Routing mit resources :users

Mit dieser Erweiterung werden automatisch sieben vordefinierte Routen erstellt. Diese beinhalten alle Grundfunktionen, die ein User haben sollte, wie zum Beispiel das Ansehen des Accounts und dem Ändern und Löschen des Users. Die Erweiterung wird im Routing von Rails eingetragen und kann spezialisiert werden. Das heisst es werde nicht alle sieben Funktionen erstellt, sondern nur die, welche man benötigt. Die benötigten Routen kann man als Parameter mitgeben. Die beiden Routings «index» und «show» werde in Instagram verwendet. Anbei die Ansicht eines Users (Das Bild kann nicht angezeigt werden, da ich ein

Problem mit dem Internet auf meiner WM hatte. Ich habe es stattdessen markiert, um den Umriss darzustellen.):



Und hier die Ansicht des Edit Routings (Auch hier das selbe Problem mit dem Bild):



Bootstrap Modal

Bootstrap Modal bietet bereits fertig erstellte Notifications und Popups zur Verfügung. Diese können ganz einfach nach Belieben gestaltet und erweitert werden. In unser Instagram Applikation verwenden wir ein Popup, um die möglichen Einstellungen eines Users anzuzeigen.

Gravatar für Benutzerbilder

Gravatar ist eine Webseite, mit welcher man ein Bild mit einer E-Mail verknüpfen kann, welches dann auf jeder Webseite, welche Gravatar unterstützt und in welcher die E-Mail-

Adresse verwendet wird, angezeigt wird. Bei unserer Applikation Beispielsweise wird das zugewiesene Bild auf der Account-Ansicht angezeigt. Somit erspart man sich einen Bildupload in die Webseite zu integrieren.

Thema/Funktion	**Vorteile**	**Nachteile**
Flash in Rails	-Einfaches Übergeben von Daten in die nächste Ansicht -Rails stellt den Dienst bereits zur Verfügung, weshalb man nicht mehr tun muss, als ihn zu brauchen -Devise macht davon Gebrauch	-Nicht alle Librarys unterstützen den Flash Speicher
Toastr	-Einfaches einbetten der Meldungen -Design passt sich an die Meldung an	-Erschwertes Bearbeiten des Designs der Toastmeldungen
Routing mit resources: users	-Die Ansichten und Funktionen werden automatisch erstellt -Die Ansichten können gewählt werden	
Bootstrap Modal	-Ansprechendes Design -Wenig Aufwand	-Etwas kompliziertes Einbetten in den Code
Gravatar	-Kein Bildupload nötig auf der Webseite -Einfaches einbetten in die Webseite	-Alle User brauchen einen Gravatar Account

Lösung der Aufgaben

Aufgabe 1, Seite 4

Es wird eine Erfolgsmeldung in Form eines Toasts mit dem gem «toastr» erzeugt mit dem Inhalt «Wow, a success».

Aufgabe 2, Seite 6

Befehle:

rails g migration AddFieldsToUser

rails db:migrate (Nach dem Bearbeiten der Datei)

Inhalt:

```
def change
  add_column :users, :website, :string
  add_column :users, :bio, :text
  add_column :users, :provider, :string
  add_column :users, :uid, :string
  add_column :users, :image, :string
 end
```

Aufgabe 3, Seite 7

Es werden zusätzlich zu den Routen die vier Helpers «users_path», «new_user_path», «edit_user_path» und «user_path» erstellt. Diese ermöglichen das einfache Aufrufen der Routen im HTML.

Aufgabe 4, Seite 8

```
def show
  @user = User.find(params[:id])
 end
```

Aufgabe 5, Seite 10

Mit dieser Funktion wird das Bild ausgelesen, welches auf Gravatar mit der aktuellen E-Mail des Users verbunden wurde.

Aufgabe 6, Seite 11

Der erste Abschnitt lädt das Gravatar Bild und rundet es ab. Der zweite Abschnitt wird nur angezeigt, wenn ein angemeldeter User seine eigenen Daten ansieht. Anderen Nutzern bleibt die Einstellungsmöglichkeit verborgen. Der dritte Abschnitt definiert den Logout Button. Wenn dieser geklickt wird, wird die Usersession zerstört und der User wird ausgeloggt.

Das Erstellen der Partial-View

Es müssen sowohl in der Login-, wie auch in der Registrations-View Fehlermeldungen angezeigt werden können. Die einfachste und performanteste Lösung ist somit eine Partial-View zu erstellen, welche den Code beinhaltet, welcher die Meldungen aus der Klasse «resource.errors» ausliest und in Toasts lädt.

Anpassungen, um ein Benutzerprofil ändern zu können

Damit es einem User möglich ist, eine Änderung an seinem Account vorzunehmen, ohne ein neues Passwort setzen zu müssen, muss man für die Ansicht einen eigenen Controller erstellen und die bereits vorhandene Methode zum Updaten der Daten überschreiben. Im Routing von Rails muss dann noch definiert werden, dass Devise diesen Controller brauchen soll.

Selbstreflexion

Ich habe Modal von Bootstrap, das resources: users Routing und den Flash Speicher in Rails neu kennen gelernt. Ich wusste bereits, wie man Migrations und Controller erstellt.

Ich konnte den Auftrag nicht termingerecht fertigstellen, da ich über das ganze Wochenende, von Freitag beginnend, bis am Dienstag für das Geschäft im Ausland arbeiten musste. Somit bin ich nun eine Woche verspätet. Auch diesmal habe ich den Auftrag zum Ende des Wochenendes verschoben, bin aber nicht in Stress gekommen.

Ich hatte Schwierigkeiten, nachdem ich den Code für die Bearbeiten-Ansicht eines Users kopiert und eingefügt hatte, da eine unerwartete Fehlermeldung generiert wurde. Nachdem ich den Code etwas durchforstet hatte, musste ich feststellen, dass ich die Spalte «website» beim Migrieren falsch geschrieben hatte. Mit einer weiteren Migration konnte ich dieses Problem lösen.

Ich konnte alle Aufgaben vom Arbeitsblatt lösen und habe das Gefühl, soweit alles grob verstanden zu haben.

Nächstes Mal sollte ich noch etwas früher mit der Arbeit beginnen, um den Rest der freien Zeit geniessen zu können.

Fazit

Das meiste vom Arbeitsblatt 4 war mir unbekannt und ich konnte somit eine Menge lernen. Ich konnte das geforderte fertigstellen und habe das Gefühl, soweit alles verstanden zu haben.

Ich hatte das Gefühl, dass ich viel mehr gelernt habe, als in den letzten Arbeitsblättern und somit ein grosses Stück weiter mit meinem Wissen über Rails gekommen bin.