Quicknote 2 AB4 & AB5 Luca Frei

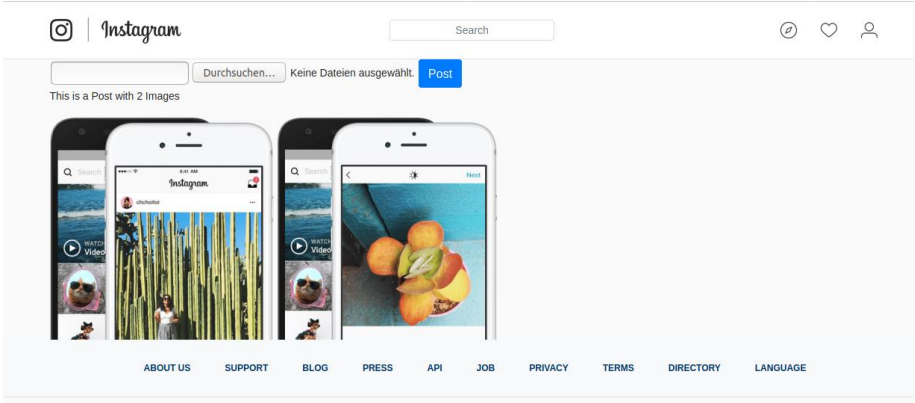
Zusammenfassung Arbeitsblätter

AB4

In diesem Arbeitsblatt haben wir Profilbilder mittels Gravarar und Flashmessages hinzugefügt. Die Falshmessages haben wir mit Toastr gestylt, damit sie etwas schöner aussehen. Ausserdem haben wir die Profil-Seite gemacht, in welcher man seine Benutzerdaten ändern und anschauen kann.

AB5

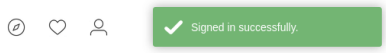
Im Arbeitsblatt 5 war das Uploaden/Posten von Bildern das Thema. Realisiert haben wird diese Funktion mit Hilfe von Cloudinary, wo die Bilder gespeichert werden, und einem Postcontroller, welche die Posts in unserer Datenback abspeichert. Ausserdem haben wir neue Models erstellt nämlich Photo und Post. Photo hat eine 1:m Beziehung mit Post und Post eine 1:m Beziehung mit User. Schlussendlich lag eine Postseite vor die wie folgt aussieht:



Verwendete Technologien

Flash & Toastr

Flash beziehungsweise Flashmessages werden bei und im Instagram verwendet, um dem Benutzer Rückmeldungen zu seinen Aktionen zu geben. Wie zum Beispiel im Bild unten, wenn sich ein Benutzer erfolgreich angemeldet hat. Normalerweise sehen Flashmessages nicht automatisch wie im Bild aus, sondern sind einfach ein oben eingeblendeter Text ohne jegliches Styling. Bei uns in der Applikation werden die Flashmessages durch Toastr gestylt damit sie so Fancy wie auf dem Bild aussehen.
Vorteile: Benutzer erhält Feedback, welches er ansonsten nicht bekommen würde.
Nachteile: -



Gravatar

Mit den Gem Gravatar werden Profilbilder mit einer bestimmten E-Mail verbunden. Wenn sich nun jemand mit dieser E-Mail bei unserem Instagram anmeldet wird das verbundene Profilbild geladen und angezeigt.
Vorteile: Profilbilder müssen nicht in der Datenbank abgespeichert werden.
Nachteile: Bei zu vielen Anfragen von einem Netz aus blockt Gravatar weitere Anfragen ab, Verbinden von vielen E-Mail-Adressen ist umständlich.

Cloudinary

Ist ein Cloudservice, welcher und Speicherplatz für unsere Bilder zur Verfügung stellt. Beim Registrieren wird dem Benutzer ein Server zugewiesen, sowie der API-Key und das API-Secret geliefert. Die Einrichtung ist simpel. Es muss lediglich ein neues Gem im Gemfile eingetragen werden und noch zusätzlich ein Kommando im Terminal ausgeführt werden.
Vorteile: Einfache Installation/Einbindung, Daten müssen nicht Lokal gespeichert werden
Nachteile: Genauer Standort der Daten ist nicht bekannt, Daten könnten vom Betreiber weiterverkauft werden, während der Entwicklungsphase muss zum Testen eine Internetverbindung vorhanden sein

Figaro

Wie die oben bereits genannten Technologien ist Figaro ebenfalls ein Gem, welches im Gemfile eingefügt werden muss. Figaro wird zum Verstecken von sensitiven Informationen verwendet. Im Instagram verwenden wir Figaro, um die Verbindungsdetails von Cloudinary zu verstecken, da mit diesen Infos direkter Zugriff auf unsere Daten möglich ist.
Vorteile: Daten können auf simple weise versteckt werden um vor unerwünschten Zugriffen zu schützen
Nachteile: Die Verbindungsinformationen werden in einer Datei gespeichert, welche nicht auf Git gepusht wird. Sprich müssen die Informationen nach dem Clonen des Projekts wieder eingetragen werden.

Bootstrap Modal

Bootstrap Modal gehört wie es der Name schon verrät zu Bootstrap. Die Modalfunktion wird verwendet, um Popups zu machen. Wenn ein solches Modalelement aktiv ist wird die Seite im Hintergrund gesperrt. Auf der Seite kann erst wieder etwas gemacht werden, wenn das Popup geschlossen wird.
Vorteile: Informationen können hervorgehoben werden, gibt viele neue Möglichkeiten.
Nachteile: -

Flash-Meldung vs. Meldungen des Models

Der Unterschied von Flash und Model Meldungen liegt darin, dass Flashmessages eigentlich zu jedem Zeitpunkt ausgelöst werden können. Hingegen werden Model Meldungen nur angezeigt, wenn sich etwas am Model verändert. In unserem Fall haben wir beim Instagram Flash-Meldungen verwendet, da wir nicht bei jeder Benutzung auch das Model ändern. Wirkliche Vorteile und Nachteile lassen sich nicht herauskristallisieren, da beide für ihren Nutzen eigentlich sehr gut sind. Ein Vorteil von Flash ist jedoch, dass man unabhängig Meldungen an den Benutzer senden kann.

Selbstreflexion

Was habe ich gelernt?

In den Arbeitsblättern 4 und 5 habe ich gelernt wie wichtig das es ist dem Benutzer genau mitzuteilen was gerade im Hintergrund passiert ist oder passiert. Einen einfachen Weg einem Benutzer ein Profilbild zuzuweisen, ohne die eigene Datenbank voll zu stopfen.

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

Während dem Bearbeiten der Arbeitsblätter habe ich mir Notizen gemacht, damit ich beim Schreiben der Quicknote nicht nochmal das ganze Arbeitsblatt durchlesen muss, um mich zurecht zu finden. Ich habe jeden Auftrag genau durchgelesen und mir überlegt wie dieser erledigt werden muss.

Was waren die Schwierigkeiten, wie konnte ich diese lösen?

Schwierigkeiten hatte ich lediglich mit meiner Arbeitsgeschwindigkeit. Ich habe lange für Arbeitsblatt 4 gebraucht, da ich etwas langsam gearbeitet habe.

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

Ich konnte alles Lösen und habe alles verstanden.

Was kann ich nächstes Mal besser machen?

Am liebsten würde ich mein Arbeitstempo erhöhen, um weniger Stress für die Quicknote zu haben.

Fazit

Abschliessend habe ich in den beiden Arbeitsblättern viel dazugelernt und Neues kennengelernt. Insbesondere fand ich den Teil mit den Flashmessages interessant, da ich es wichtig finde das der Benutzer so viele Informationen von den Vorgängen wie nur möglich erhält und das mit Flashmessages ziemlich einfach ist.

Lösungen Aufgabenblätter

AB4

Notieren Sie die notwendigen Kommandos, um die Migration durchzuführen und den Inhalt der Migrationsdatei:

```
rails g migration AddFieldsToUser
rails db:migrate
Migrations datei:
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
```

Erklären Sie in groben Zügen, was das bedeutet:

Die GravatarId des Benutzers wird mittels der Email herausgesucht und dann bei Gravatar gesucht um das Profilbild zu finden.

Erklären Sie kurz folgende Codeausschnitte:

```
…
<%= image_tag avatar_url(@user), width: '152', height: '152',
class: "round-img" %>
…
<% if @user == current_user %>
  <%= link_to "Edit Profile", edit_user_registration_path, class:
"btn btn-outline-dark common-btn edit-profile-btn" %>
  <button type="button" class="core-sprite setting" data-
toggle="modal" data-target="#exampleModal"></button>
<% end %>
…
<%= link_to "Log out", destroy_user_session_path, method: :delete,
class: "list-group-item list-group-item-action" %>
…
```

1. Codeteil: definiert ein Bild welches mit der avatar_url Methode eine Bildquelle zugewiesen bekommt.
2.Codeteil: Zeigt einen Button an, welcher zum bearbeiten des profils verlinkt. Dieser wird nur angezeigt wenn der Benutzer der aktuelle Benutzer ist.
3. Codeteil: ist ein button, welcher den Benutzer ausloggt und die Session beendet.

AB5

Was bewirkt das Schlüsselwort `references` auf den Attributen `Post.user` und `Photo.post` in der Migration (in den entsprechenden Dateien)?

References erstellt einen Foreignkey, welcher auf das angegebene Objekt verweisst.

Wie ist die Namensgebung der Fremdschlüssel-Attribute im Tabellenschema von `Post` und `Photo`?

Post = user_id
Photo = post_id

Ergänzen Sie die Beziehungen zwischen `User – Post` (ein Benutzer kann mehrere Posts haben) und zwischen `Post – Photo` (ein Post kann mehrere Photos haben) und notieren Sie die Einträge in den entsprechenden Dateien:

Anpassungen user.rb: has_many :posts
Anpassungen post.rb: has_many :photos

Welche Anpassung müssen Sie dazu in `app/models/user.rb` machen:

has_many :posts, dependent: :destroy

Welche Anpassung müssen Sie dazu in `app/models/post.rb` machen:

has_many :photos, dependent: :destroy

Erklären Sie die `create`-Methode im Post Controller.

Beschreiben Sie als Erstes, was in der 1. Zeile gemacht wird:
`@post = current_user.posts.build(post_params)`

Um es besser zu verstehen, können Sie die Rails Console zur Hilfe nehmen und das Kommando `User.first.posts.build` ausführen.

Die Grundstruktur eines Userposts wird mittels User.posts.build aufgerufen. in der oben angegebenen Zeile wird die Grundstruktur des aktuellen users aufgerufen um einen neuen Post zu erfassen, wobei die post_params mitgegeben werden.

Erklären Sie die `index`-Methode im Post Controller:

```
5     def index
6       @posts = Post.all.limit(10).includes(:photos)
7       @post = Post.new
8     end
```

In der index-Methode werden von allen Posts maximal 10 zusammen mit den dazugehörigen Fotos in der Variable @posts abgespeichert.
der Variable @post wird ein neuer Post zugewiesen

Erklären Sie die folgende Zeile:

```
3     before_action :find_post, only: [:show, :destroy]
```

Durch diese Zeile sind solange die :find_post methode nicht ausgeführt wurde nur die methoden :show und :destroy ausführbar.

Erklären Sie den Rest der `create`-Methode:

```
10    def create
11      @post = current_user.posts.build(post_params)
12      if @post.save
13        if params[:images]
14          params[:images].each do |img|
15            @post.photos.create(image: img)
16          end
17        end
18
19        redirect_to posts_path
20        flash[:notice] = "Saved ..."
21      else
22        flash[:alert] = "Something went wrong ..."
23        redirect_to posts_path
24      end
25    end
```

In der Methode wird überprüft ob der Post abgespeichert werden kann. Wenn dies nicht der Fall ist, erhält der Benutzer mittels Flashmessage eine Nachricht, dass etwas falsch gelaufen wird. Ausserdem wird er auf auf die Posts seite weitergeleitet. Wenn der Post abgespeichert werden kann wird überprüft ob es images gibt. Wenn ja werden die Fotos dem Post zugewiesen ud abgespeichert. Wenn alle Bilder zugewiesen sind oder der Post keine Bilder hat wird der Benutzer auf die Posts seite weitergeleitet und erhält das Feedback, dass der Post gespeichert wurde