Quicknote AB-151-05 Joel Liechti

Zusammenfassung:

In diesem AB ging es um das Hochladen und Anzeigen von Bilder. Ebenfalls wurden Themen wie Verschlüsselung und Sicherheit angesprochen. Schlussendlich ging es noch um Before Actions.
Ich kannte das meiste in diesem AB noch nicht und lernte einiges. Viele der besprochenen Dinge sind sehr praktisch und einfach anzuwenden.

Wie und wo können die vorgestellten Techniken, Methoden und Konzepte in einer Rails-App angewandt werden?

Cloudinary & Carrierwave:

- **Anwendung**: Mithilfe dieser Komponenten kann man Bilder hochladen und per API wieder abfragen.
- **Vorteile**: Gratis und sehr einfach zu benutzen.
- **Nachteile**: Man ist natürlich auf diesen Service angewiesen ähnlich wie bei Gravatar.

Figaro/Verschlüsselung:

- **Anwendung**: Wenn man sensible Daten hat, welche nicht aufs git übertragen werden sollen und innerhalb des Projekts nicht als normalen Text herumgegeben werden soll, kann man Figaro verwenden um mehr Sicherheit zu erreichen.
- **Vorteile**: Leicht zu benutzen, zusätzliche Sicherheit ohne viel Aufwand.
- **Nachteile**: -

Datenbankbeziehung:

- **Anwendung**: Wenn man einfache oder auch komplexe Datenbankbeziehungen hat kann man diese sehr einfach mithilfe des Models definieren. Ebenfalls kann man ohne viel Aufwand Constraints setzen.
- **Vorteile**: Wenig Aufwand für komplexe Beziehungen. Verhält und ist simpel.
- **Nachteile**: -

Before Actions:

- **Anwendung**: Wenn man Code hat, welcher immer vor einer bestimmten Aktion ausgeführt werden soll, kann man das per Before Actions machen und somit Redundanzen verhindern.
- **Vorteile**: Man verhindert redundanten Code.
- **Nachteile**: -

Aufgaben:

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

Das Schlüsselwort references erstellt eine Spalte für einen Foreign-Key auf die angegebene Tabelle.

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

Der Name des Models wird klein geschrieben und ein _id angehängt. Führt zu: user_id und 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:

In user.rb (Model)
has_many :posts

und in post.rb:
has_many :photos

in post.rb steht bereits belongs_to :user

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

Bezogen auf eine casecade delete Beziehung.

In beiden Models (user.rb und post.rb) muss man hinter das «has_many :blabla» noch «, dependent: :destroy» anhängen.

Führt zu: «has_many :posts, dependent: :destroy» und «has_many :photos, dependent: :destroy»

Beschreiben Sie als Erstes, was in der 1. Zeile gemacht wird:

```
@post = current_user.posts.build(post_params)
```

Bei dem aktuellen User wird auf posts zugegriffen. Dort wird ein neuer Post erstellt mithilfe der Post-Parameter. Diese sind im Controller definiert.

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
```

Der Post wird gespeichert. Wenn das fehlschlägt gibt es eine Errormeldung und man wird weitergeleitet.
Falls der Post jedoch gespeichert wird, werden alle Bilder, die mitgegeben wurden (über das Formular) genommen und als neues Photo-Objekt erstellt. Diese Photos werden direkt auf dem Post-Objekt erstellt und gehören somit zu diesem Post.
Danach kommt eine Meldung, dass der Post gespeichert wurde und man wird weitergeleitet.

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
```

Als erstes werden die ersten 10 Posts genommen. Dabei wird die Beziehung direkt aufgelöst und alle Photos, die zu einem Post gehören, werden dem entsprechenden Post-Objekt zugewiesen und sind so abrufbar.
Ebenfalls wird mit Post.new ein neuer Post bereitgestellt.

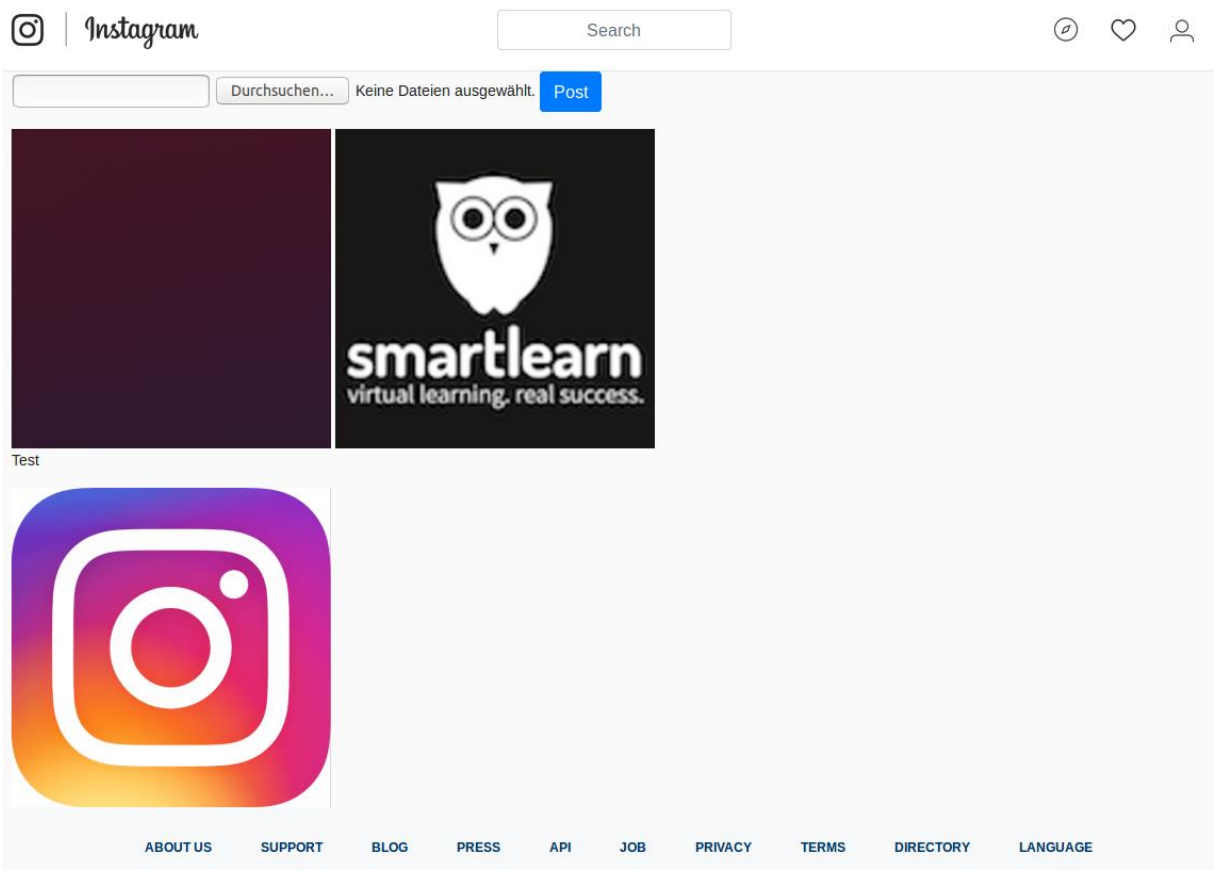
Erklären Sie die folgende Zeile:

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

Vor jeder Methode (Action), die entweder show oder destroy ist, wird die Methode find_post ausgeführt. Ohne das only würde die Methode find_post vor jeder anderen Methode ausgeführt werden.

Screenshots:

Posts (einer ohne und einer mit Text):



Selbstreflexion

Was habe ich gelernt?

In diesem Arbeitsblatt war fast alles neu für mich. Ich kannte weder Cloudinary noch Carrirewave und verwendete beide zum ersten Mal. Figaro kannte ich noch nicht auch wenn ich das Prinzip und die Vorteile der Verschlüsselung natürlich kenne. Before Actions habe ich noch nie verwendet und finde ich etwas verwirrend aber sie scheinen praktisch zu sein.

Was hat mich behindert?

Viel mühsames Abschreiben ohne wirkliche Erklärung.

Was habe ich nicht verstanden?

Ich habe nicht verstanden wie die Verschlüsselung effektiv funktioniert. Allerdings denke ich, dass das auch nicht nötig ist für mich.

Was kann ich beim Studium besser machen?

-

Schlussfolgerung

Ich lernte in diesem AB viel Neues und benutzte einige sehr praktische Tools. Allerdings kann ich wie immer das meiste nicht verwenden da ich weder Privat noch in der Firma Ruby/Rails verwende. Diese Clouddienste per API zu verwenden ist aber sicherlich etwas das ich später noch einmal verwenden werde. Verschlüsselung etc. ist immer wichtig und wird natürlich immer verwendet werden von mir.