



MODUL 151

TEIL 6: DESIGN VON POST VIEW

Ralph Maurer

Inhaltsverzeichnis AB151-06

Modul 151: Instagram mit Rails bauen

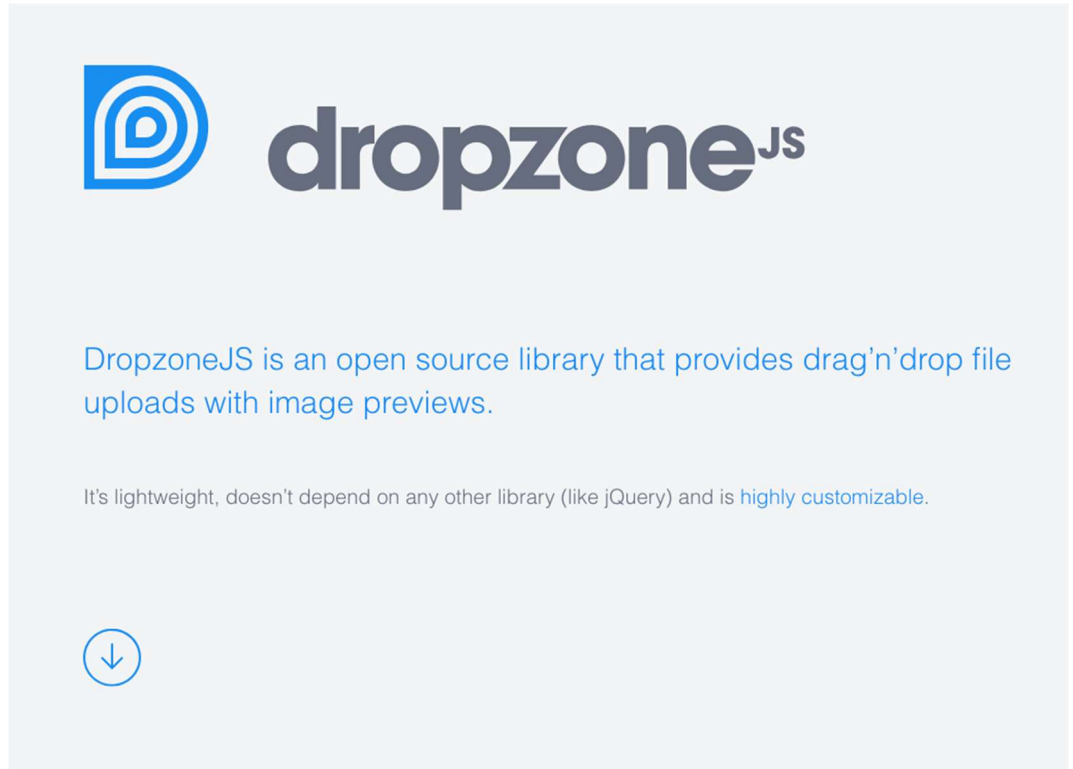
Teil 6: Instagram Design von Post View

Inhaltsverzeichnis AB151-06	1
Modul 151: Instagram mit Rails bauen	1
Post View.....	2
DropzoneJS	2
Icons von Font Awesome.....	3
View posts#index	4
SCSS-Klassen für die View posts#index	6
+ Button mit DropzoneJS gestalten	7
Posts ohne Photos unterbinden	8
Posts mit Photos testen	9
Bildspeichergrößen ändern	9
Design für die Liste mit den 10 Posts	10
Partial-View für die Anzeige der Posts	12
Styling der Posts inkl. Icons	16
Bildgröße verändern	18
Posts als Startseite	19
Auftrag: Quicknote AB151-06	20

Post View

DropzoneJS

Zur Gestaltung des Uploads nutzen wir DropzoneJS. Ein Gem, das uns eine super Anzeige für den Bilderupload ermöglicht. Alle Informationen zu DropzoneJS finden Sie unter <http://www.dropzonejs.com>



Erfassen Sie im Gemfile den Eintrag:
`gem 'dropzonejs-rails'`

Installieren Sie das Gem mit `bundle install`

Starten Sie anschliessend den Server neu.

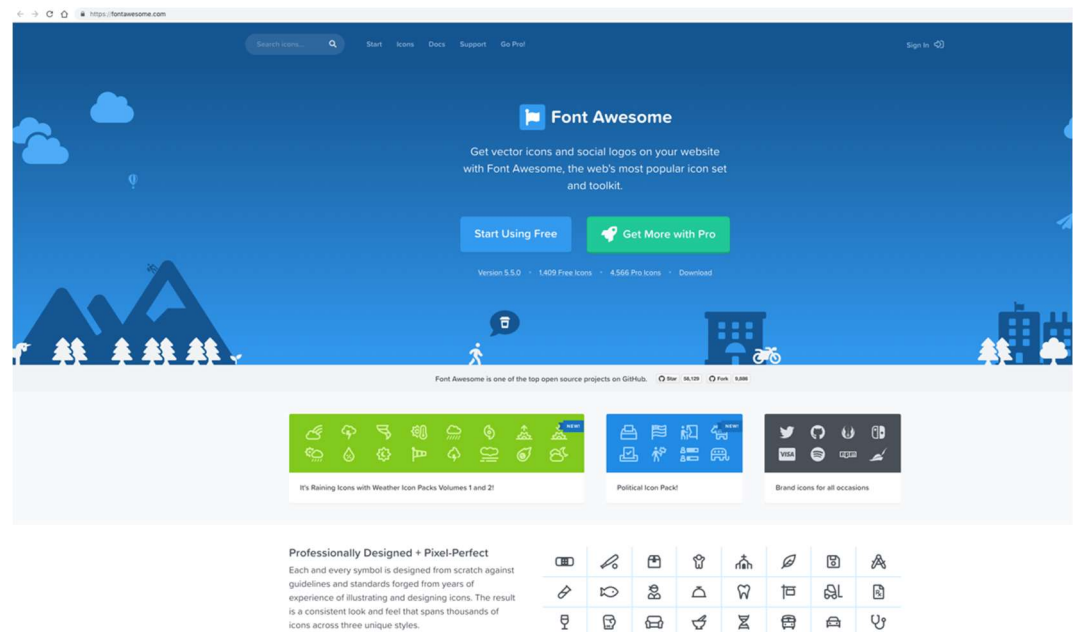
In `application.js` muss `dropzone` referenziert werden (Achtung: die Referenz muss nach `bootstrap-sprockets` und vor `tree .` formuliert werden!):

```
application.scss | application.js | upload_post_images.js | index.html.erb
1 // This is a manifest file that'll be compiled into application.js, which will include all the files
2 // listed below.
3 //
4 // Any JavaScript/Coffee file within this directory, lib/assets/javascripts, or any plugin's
5 // vendor/assets/javascripts directory can be referenced here using a relative path.
6 //
7 // It's not advisable to add code directly here, but if you do, it'll appear at the bottom of the
8 // compiled file. JavaScript code in this file should be added after the last require_* statement.
9 //
10 // Read Sprockets README (https://github.com/rails/sprockets#sprockets-directives) for details
11 // about supported directives.
12 //
13 //= require rails-ujs
14 //= require activestorage
15 //= require turbolinks
16 //= require jquery3
17 //= require popper
18 //= require bootstrap-sprockets
19 //= require dropzone
20 //= require_tree .
21 //= require toastr
22
```

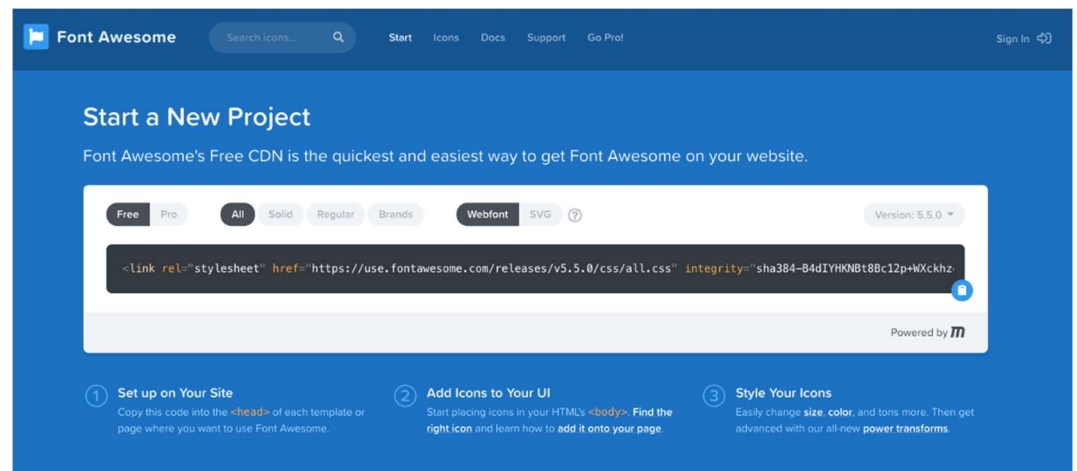
Und in `application.scss` wird `dropzone` importiert:
`@import "dropzone/dropzone";`

Icons von Font Awesome

Icons für unsere Applikation übernehmen wir von <http://fontawesome.com>



Gehen Sie auf «Start Using Free», kopieren Sie den Link:



Und fügen Sie diesen in `app/views/layouts/application.html.erb` ein:

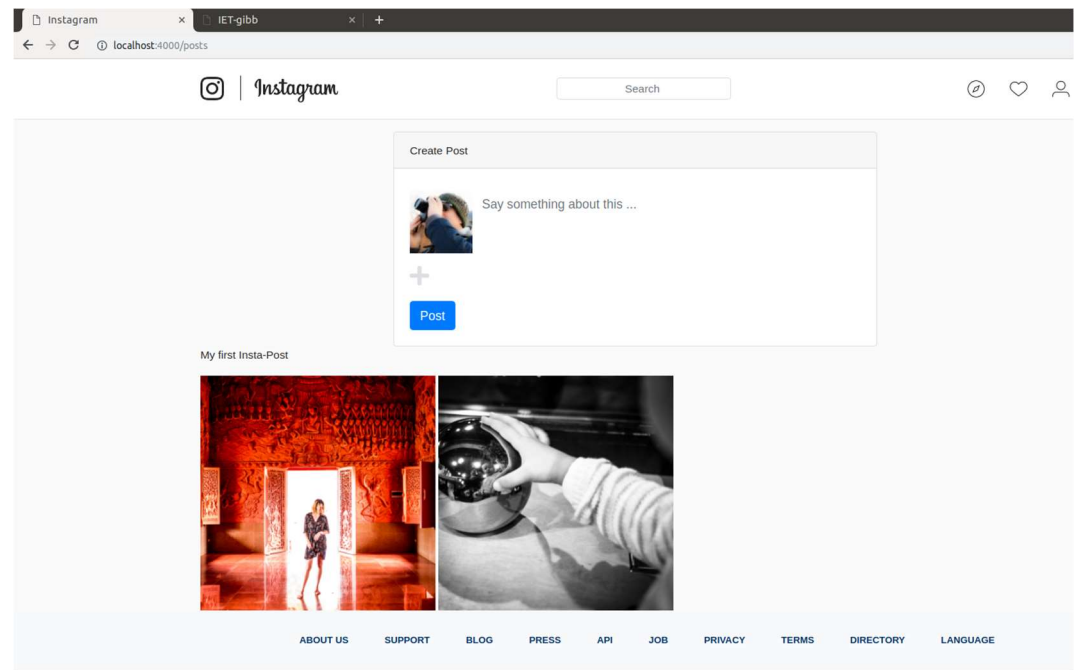
```
application_controller.rb  registrations_controller.rb  Gemfile  application.js  application.html.erb  application.scss  routes.rb
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Instagram</title>
5     <%= csrf_meta_tags %>
6     <%= csp_meta_tag %>
7
8     <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>
9     <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
10    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.5.0/css/all.css" integrity="sha384-B4dIYHKNBt8Bc12p+WXckhzcICo0wtJ
11  </head>
12  <body>
13    <%= render 'shared/navbar' if current_user %>
14    <div class="container">
15      <%= yield %>
16    </div>
17    <%= render 'shared/footer' %>
18    <% if flash.any? %>
19      <script type="text/javascript">
20        <% flash.each do |key, value| %>
21          <% type = key.to_s.gsub('alert', 'error').gsub('notice', 'success') %>
22          toastr.<% type %>('<%= value %>')
23        <% end %>
24      </script>
25    <% end %>
26  </body>
27 </html>
28
```

View posts#index

Jetzt müssen wir die View `app/views/posts/index.html.erb` anpassen. Dies verlangt ein genaues Arbeiten:

```
application_controller.rb  registrations_controller.rb  Gemfile  application.js  application.html.erb  index.html.erb  application.scss
1 <div class="d-flex flex-column align-items-center mt-3">
2   <div class="col-xl-7 col-lg-8 col-md-10 col-sm-11 post-card">
3     <div class="card">
4       <div class="card-header">
5         Create Post
6       </div>
7       <div class="card-body">
8         <%= form_for @post, :html => {:multipart => true, :class => "dropzone upload-images p-0 border-0"} do |f| %>
9           <div class="form-group row mt-2">
10            <div class="col-auto pr-0">
11              <%= image_tag avatar_url(current_user), class: "post-author-icon" %>
12            </div>
13            <div class="col pl-0">
14              <%= f.text_field :content, class: "form-control border-0",
15                placeholder: "Say something about this ..." %>
16            </div>
17          </div>
18          <div class="fallback">
19            <%= file_field_tag "images[]", type: :file, multiple: true %>
20          </div>
21          <div class="dz-message m-0"></div>
22          <div class="dropzone-previews mb-3">
23            <div class="upload-photos-icon">
24              <i class="fa fa-plus fa-2x" aria-hidden="true" style="color:#dddfe2"></i>
25            </div>
26          </div>
27          <%= f.submit "Post", class: "btn btn-primary" %>
28        <% end %>
29      </div>
30    </div>
31  </div>
32 </div>
33
34 <%= @posts.each do |post| %>
35   <p><%= post.content %></p>
36   <%= post.photos.each do |photo| %>
37     <%= image_tag photo.image.url(:standard) %>
38   <% end %>
39 <% end %>
40
```

Ihre Ansicht sollte nun wie folgt aussehen:



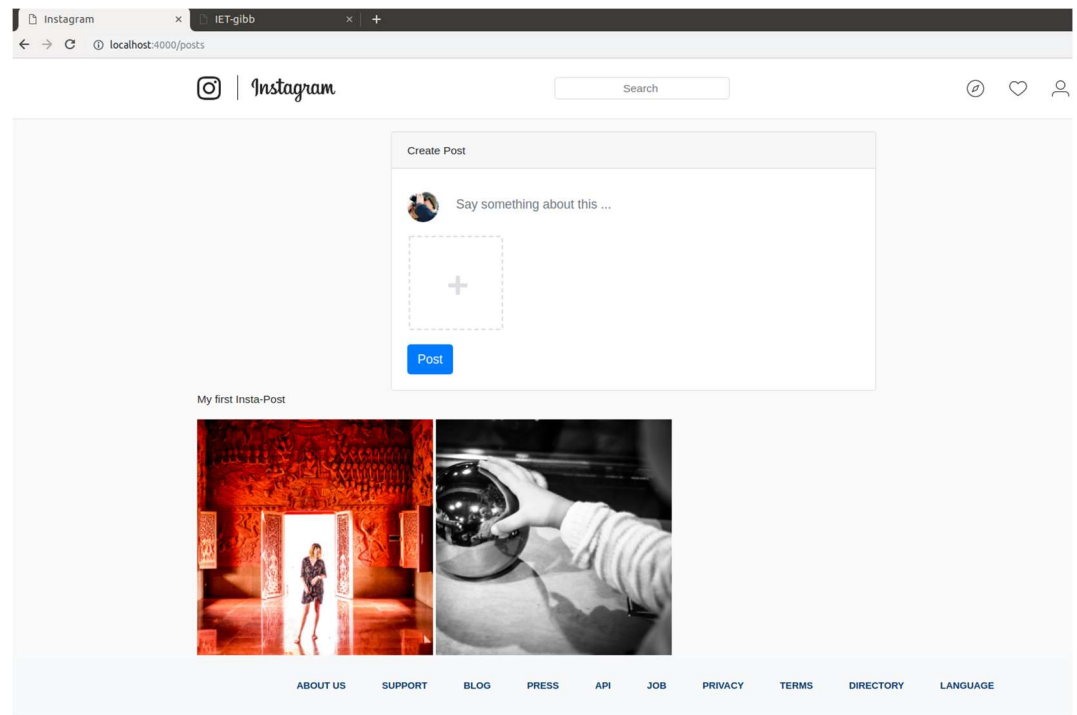
SCSS-Klassen für die View posts#index

Den Abschnitt auf der Seite zum Erfassen eines Posts («Create Post») bauen wir nun Instagram-Like aus. Insbesondere der + Button soll schöner aussehen. Hierzu schreiben wir reichlich Code und nutzen dabei SCSS-Klassen von DropzoneJS.

Ergänzen Sie `app/assets/stylesheets/application.scss`:

```
application.scss      index.html.erb
244     font-size: 16px;
245   }
246 }
247 }
248 // Posts
249 .post-author-icon {
250   height: 40px;
251   width: 40px;
252   border-radius: 50%;
253   margin-right: 10px;
254 }
255
256 // Customize upload photos form
257 .upload-photos-icon:hover {
258   cursor: pointer;
259 }
260
261 .dz-preview.dz-image-preview {
262   margin: 2px;
263 }
264
265 .dz-image {
266   border-radius: 0px !important;
267
268   img {
269     width: 100%;
270   }
271 }
272
273 .dz-remove {
274   @extend .core-sprite;
275   @extend .hide-text;
276   position: absolute;
277   top: 9px;
278   right: 5px;
279   z-index: 20;
280   width: 18px;
281   height: 18px;
282   background-position: -386px -25px;
283   opacity: 0;
284   -webkit-transition: opacity 0.2s linear;
285   transition: opacity 0.2s linear;
286 }
287
288 .dropzone .dz-preview:hover .dz-remove {
289   opacity: 1;
290 }
291
292 .upload-photos-icon {
293   height: 120px;
294   width: 120px;
295   border: 2px dashed #dddfe2;
296   border-radius: 2px;
297   padding: 48px;
298   display: inline-block;
299   margin: 2px;
300 }
```

Wenn wir das Ganze nun betrachten, stellen wir fest, dass der + Button noch gar nichts macht.

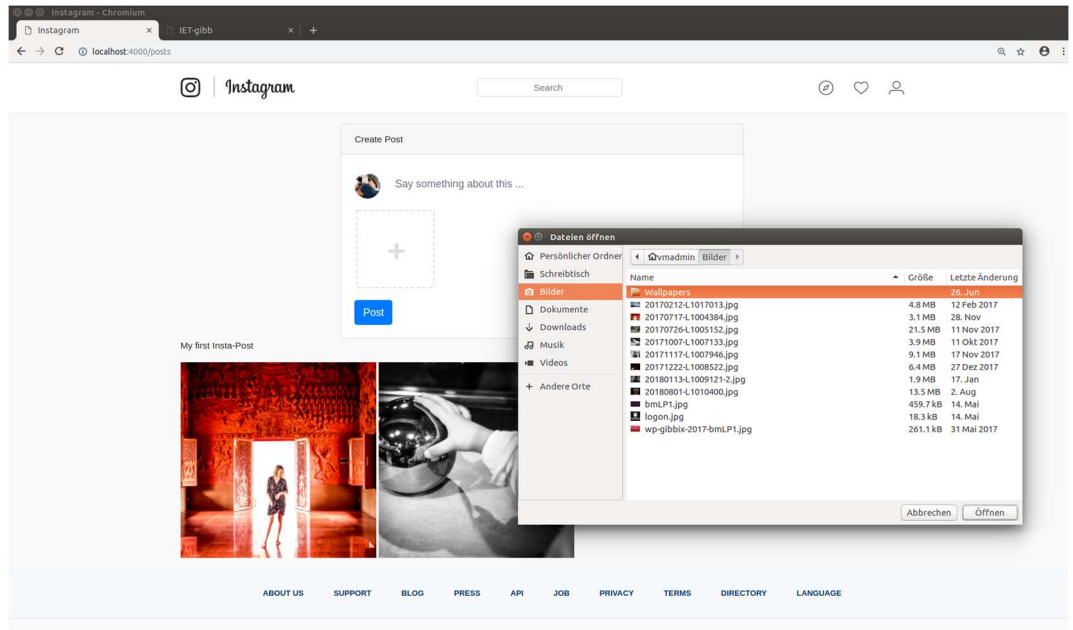


+ Button mit DropzoneJS gestalten

Der + Button soll nun seine Funktionalität erhalten mit DropzoneJS in Javascript. Erstellen Sie die neue Datei `app/assets/javascripts/upload_post_images.js` und erfassen Sie folgenden Code:

```
application.scss | upload_post_images.js | index.html.erb
1 Dropzone.autoDiscover = false;
2
3 $(document).ready(function(){
4   $(".upload-images").dropzone({
5     addRemoveLinks: true,
6     maxFileSize: 1,
7     autoProcessQueue: false,
8     uploadMultiple: true,
9     parallelUploads: 100,
10    maxFiles: 100,
11    paramName: "images",
12    previewsContainer: ".dropzone-previews",
13    clickable: ".upload-photos-icon",
14    thumbnailWidth: 100,
15    thumbnailHeight: 100,
16
17    init: function(){
18      var myDropzone = this;
19
20      this.element.querySelector("input[type=submit]").addEventListener("click", function(e){
21        e.preventDefault();
22        e.stopPropagation();
23        myDropzone.processQueue();
24      });
25
26      this.on("successmultiple", function(files, response){
27        window.location.reload();
28      });
29
30      this.on("errormultiple", function(files, response){
31        toastr.error(response);
32      });
33    }
34  })
35 });
36
```


Nun sollte der + Button funktionieren:



Posts ohne Photos unterbinden

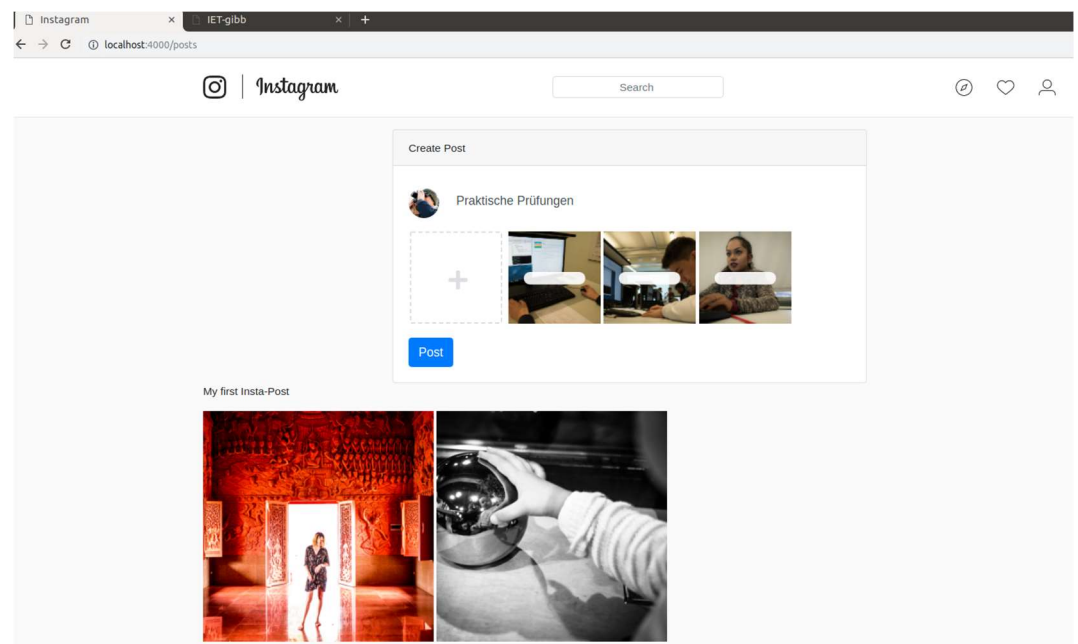
Wie können Sie sicherstellen, dass ein Post mind. ein Photo hat? Überlegen Sie sich, wo eine Überprüfung möglich ist und wo sie am meisten Sinn macht. Testen Sie Ihre Lösung!

In `app/controllers/posts_controller.rb` müssen wir noch die Bildübergabe als Parameter erfassen:

```
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[1])
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
26
```


Posts mit Photos testen

Erstellen Sie nun einen Post mit drei Bildern und testen Sie das Ganze:



Bildspeichergrößen ändern

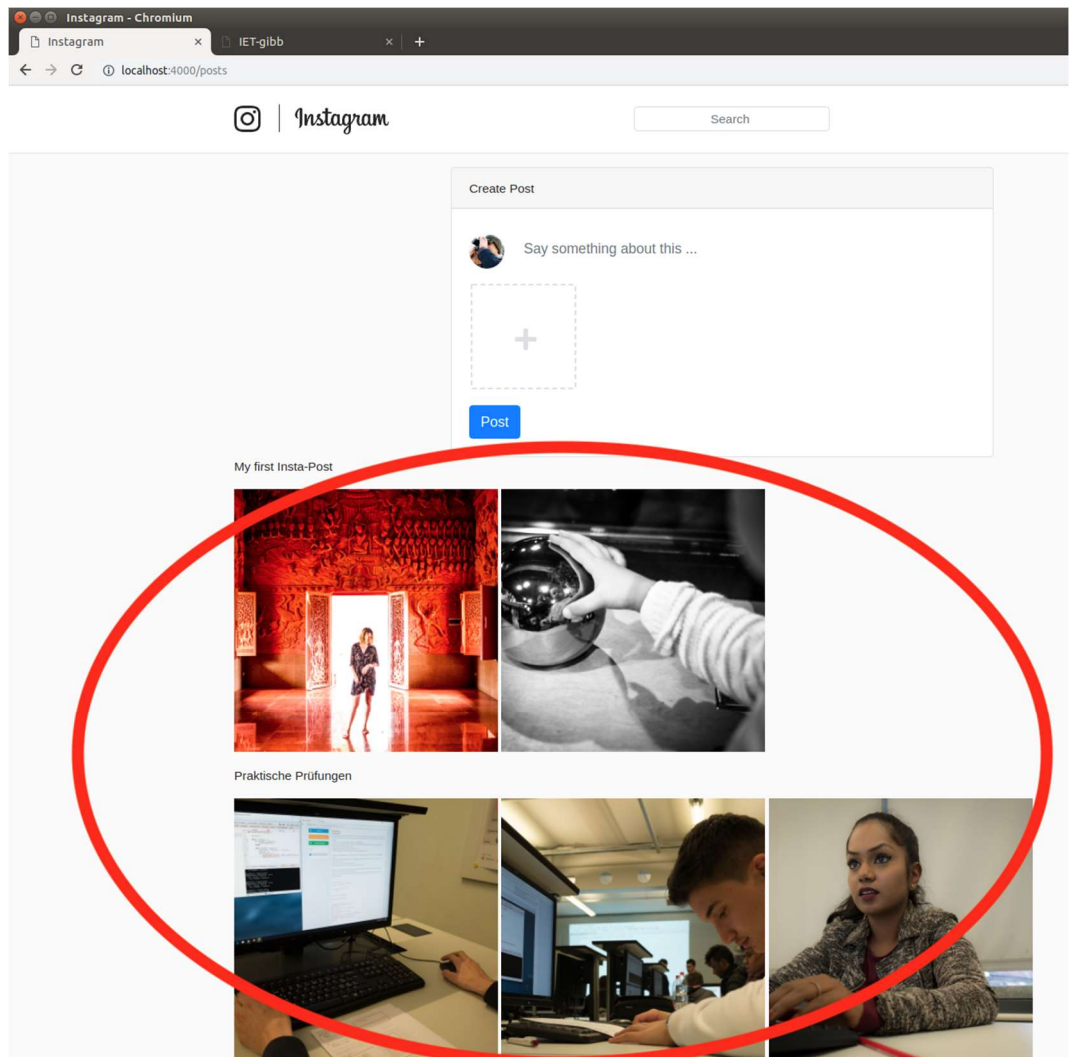
Sie können nur Bilder mit einer Grösse bis zu 1 MB hochladen. Finden Sie den entsprechenden Parameter und verändern Sie diesen, so dass 4 MB grosse Bilder verwendet werden können. Welche Datei und welcher Eintrag sind betroffen?



Design für die Liste mit den 10 Posts

Erinnern wir uns: Es werden max. die 10 letzten Posts angezeigt. Diese Anzeige wollen wir nun gestalten. Folgende Anforderungen sind zu erfüllen:

1. Username und Avatar sollen angezeigt werden.
2. Sortierung: Von neu nach alt (Neuster zuoberst).



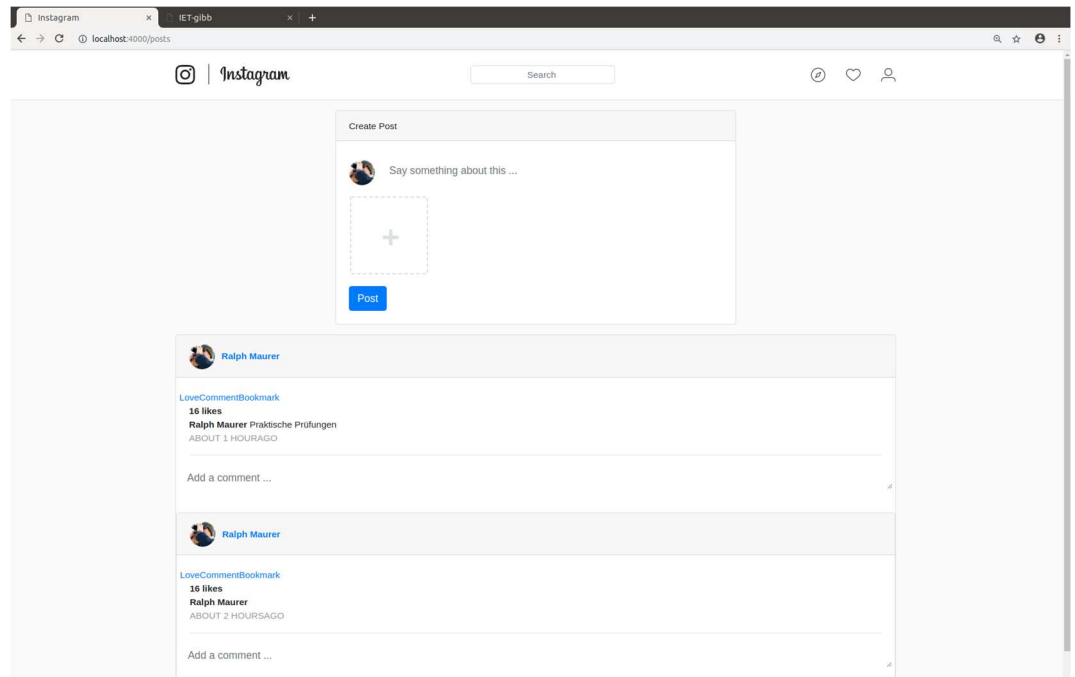
Diese Anforderungen für die Sortierung können wir ohne grossen Aufwand in `app/controllers/posts_controller.rb` umsetzen:

```
5   def index
6     @posts = Post.all.limit(10).includes(:photos, :user).order('created_at desc')
7     @post = Post.new
8   end
9
```

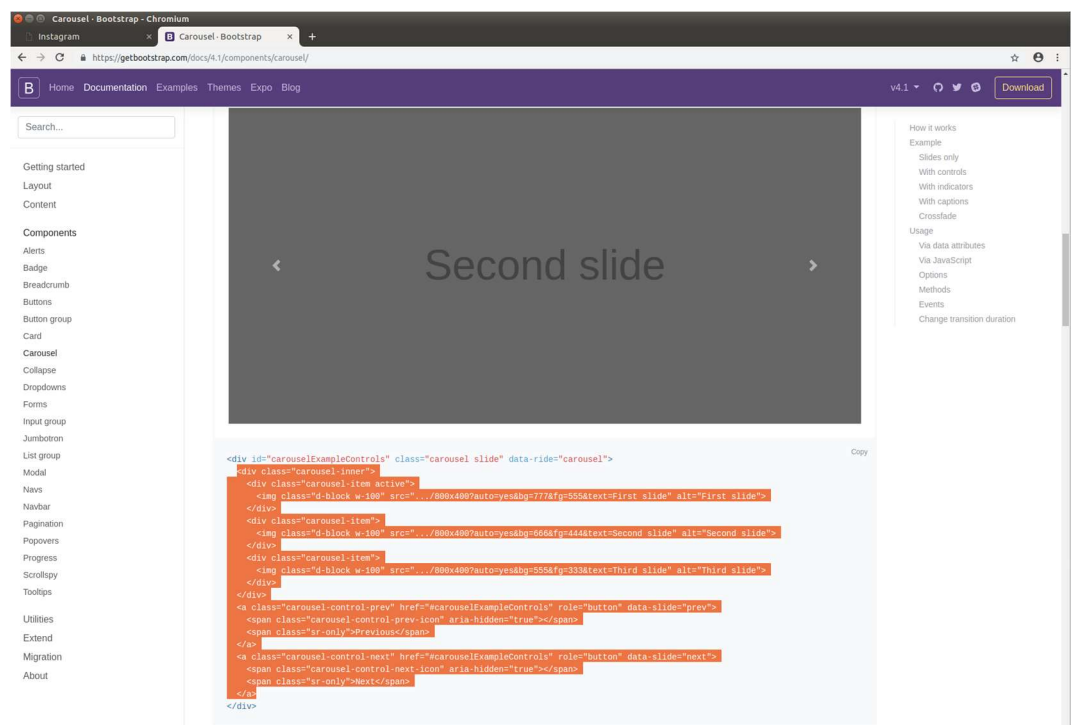
Der Aufwand für die Darstellung der Posts (app/views/posts/index.html.erb) ist wesentlich grösser. Viel Spass beim Schreiben 😊:

```
index.html.erb
1 <div class="d-flex flex-column align-items-center mt-3">
2   <div class="col-xl-7 col-lg-8 col-md-10 col-sm-11 post-card">
3     <div class="card">
4       <div class="card-header"> Create Post </div>
5       <div class="card-body">
6         <%= form_for @post, :html => {:multipart => true, :class => "dropzone upload-images p-0 border-0"} do |f| %>
7           <div class="form-group row mt-2">
8             <div class="col-auto pr-0">
9               <%= image_tag avatar_url(current_user), class: "post-author-icon" %>
10            </div>
11            <div class="col pl-0">
12              <%= f.text_field :content, class: "form-control border-0",
13                placeholder: "Say something about this ..." %>
14            </div>
15          </div>
16          <div class="fallback"> <%= file_field_tag "images[]", type: :file, multiple: true %> </div>
17          <div class="dz-message m-0"></div>
18          <div class="dropzone-previews mb-3">
19            <div class="upload-photos-icon">
20              <i class="fa fa-plus fa-2x" aria-hidden="true" style="color:#dddfe2"></i>
21            </div>
22          </div>
23          <%= f.submit "Post", class: "btn btn-primary" %>
24        <% end %>
25      </div>
26    </div>
27  </div>
28 </div>
29 <%= @posts.each do |post| %>
30   <div class="card mt-3 post">
31     <div class="card-header d-flex align-items-center">
32       <%= link_to user_path(post.user), class: "no-text-decoration" do %>
33         <%= image_tag avatar_url(post.user), class: "post-author-icon" %>
34       <% end %>
35       <%= link_to user_path(post.user), class: "normal-color no-text-decoration",
36         title: post.user.name do %>
37         <strong><%= post.user.name %></strong>
38       <% end %>
39     </div>
40     <div class="card-body">
41       <div class="row actions">
42         <a href="#" class="core-sprite love hide-text"> Love </a>
43         <a href="#" class="core-sprite comment hide-text"> Comment </a>
44         <a href="#" class="core-sprite bookmark hide-text"> Bookmark </a>
45       </div>
46       <div><strong><%= pluralize(16, "like") %></strong></div>
47       <div>
48         <span><strong><%= post.user.name %></strong></span>
49         <span><%= post.content %></span>
50       </div>
51       <div class="light-color post-time"><%= time_ago_in_words(post.created_at).upcase %>AG0</div>
52       <hr>
53       <div class="row actions">
54         <form action="#" class="w-100">
55           <div>
56             <textarea class="form-control comment-input border-0" placeholder="Add a comment ..." rows="1"></textarea>
57           </div>
58         </form>
59       </div>
60     </div>
61   <% end %>
```

Die View sollte nun - ohne Bilderanzeige - wie folgt aussehen:



Für die Darstellung der Bilder wollen wir wieder Bootstrap Carousel nutzen. Gehen Sie auf <http://www.bootstrap.com> und suchen Sie in der Dokumentation nach Carousel. Wir wollen den Code für einen Slider mit Controls verwenden (with controls):



Partial-View für die Anzeige der Posts

Mit dem Code für den Slider wächst der Code für die Anzeige der Posts langsam immer mehr an, und es ist an der Zeit, diesen Teil in eine Partial-View auszulagern:

Erstellen Sie die Datei `app/views/posts/_posts_list.html.erb` und verschieben Sie die Hälfte des Codes von `app/views/posts/index.html.erb` in diese Datei:
Sämtlicher Code ab `<% @posts.each do |post| %>`.

Kopieren Sie nun den Slider-Code von der Bootstrap-Site und fügen diesen in die Datei
app/views/posts/index.html.erb an der richtigen Stelle ein:

```
<% @posts.each do |post| %>
  <div class="card mt-3 post">
    <div class="card-header d-flex align-items-center">
      <%= link_to user_path(post.user), class: "no-text-decoration" do %>
        <%= image_tag avatar_url(post.user), class: "post-author-icon" %>
      <end %>
      <%= link_to user_path(post.user), class: "normal-color no-text-decoration",
        title: post.user.name do %>
        <strong><%= post.user.name %></strong>
      <end %>
    </div>

    <div class="carousel-inner">
      <div class="carousel-item active">
        
      </div>
      <div class="carousel-item">
        
      </div>
      <div class="carousel-item">
        
      </div>
    </div>
    <a class="carousel-control-prev" href="#carouselExampleControls" role="button" data-slide="prev">
      <span class="carousel-control-prev-icon" aria-hidden="true"></span>
      <span class="sr-only">Previous</span>
    </a>
    <a class="carousel-control-next" href="#carouselExampleControls" role="button" data-slide="next">
      <span class="carousel-control-next-icon" aria-hidden="true"></span>
      <span class="sr-only">Next</span>
    </a>

    <div class="card-body">
      <div class="row actions">
        <a href="#" class="love hide-text"> Love </a>
        <a href="#" class="comment hide-text"> Comment </a>
        <a href="#" class="bookmark hide-text"> Bookmark </a>
      </div>
      <div><strong><%= pluralize(16, "like") %></strong></div>
      <div>
        <span><strong><%= post.user.name %></strong></span>
        <span><%= post.content %></span>
      </div>
      <div class="light-color post-time"><%= time_ago_in_words(post.created_at).upcase %>AGO</div>
      <hr>
      <div class="row actions">
        <form action="#" class="w-100">
          <div>
            <textarea class="form-control comment-input border-0" placeholder="Add a comment ..." rows="1"></textarea>
          </div>
        </form>
      </div>
    </div>
  </div>
<%end%>
```


Wir müssen den Slider erheblich umbauen, damit er unseren Anforderungen entspricht:

```
1 <%= @posts.each do |post| %>
2   <div class="card mt-3 post">
3     <div class="card-header d-flex align-items-center">
4       <%= link_to user_path(post.user), class: "no-text-decoration" do %>
5         <%= image_tag avatar_url(post.user), class: "post-author-icon" %>
6       <end %>
7       <%= link_to user_path(post.user), class: "normal-color no-text-decoration",
8         title: post.user.name do %>
9         <strong><%= post.user.name %></strong>
10      <end %>
11    </div>
12
13    <%= if post.photos.size == 1 %>
14      <%= image_tag post.photos.first.image.url(:standard), class: "card_img_top" %>
15    <else %>
16      <div class="carousel slide" data-ride="carousel" id="carousel-post-<%= post.id %>">
17        <div class="carousel-inner">
18          <%= post.photos.each do |photo| %>
19            <%= if photo == post.photos.first %>
20              <div class="carousel-item active">
21                <%= else %>
22              <div class="carousel-item">
23                <end %>
24              <%= image_tag photo.image.url(:standard), class: "card_img_top" %>
25            </div>
26          <end %>
27        </div>
28        <a class="carousel-control-prev" href="#carousel-post-<%= post.id %>" role="button" data-slide="prev">
29          <span class="carousel-control-prev-icon" aria-hidden="true"></span>
30          <span class="sr-only">Previous</span>
31        </a>
32        <a class="carousel-control-next" href="#carousel-post-<%= post.id %>" role="button" data-slide="next">
33          <span class="carousel-control-next-icon" aria-hidden="true"></span>
34          <span class="sr-only">Next</span>
35        </a>
36      </div>
37    <end %>
38    <div class="card-body">
39      <div class="row actions">
40        <a href="#" class="core-sprite love hide-text"> Love </a>
41        <a href="#" class="core-sprite comment hide-text"> Comment </a>
42        <a href="#" class="core-sprite bookmark hide-text ml-auto"> Bookmark </a>
43      </div>
44      <div><strong><%= pluralize(16, "like") %></strong></div>
45      <div>
46        <span><strong><%= post.user.name %></strong></span>
47        <span><%= post.content %></span>
48      </div>
49      <div class="light-color post-time"><%= time_ago_in_words(post.created_at).upcase %>AGO</div>
50      <hr />
51      <div class="row actions">
52        <form action="#" class="w-100">
53          <div>
54            <textarea class="form-control comment-input border-0" placeholder="Add a comment ..." rows="1"></textarea>
55          </div>
56        </form>
57      </div>
58    </div>
59  </div>
60 <end %>
```

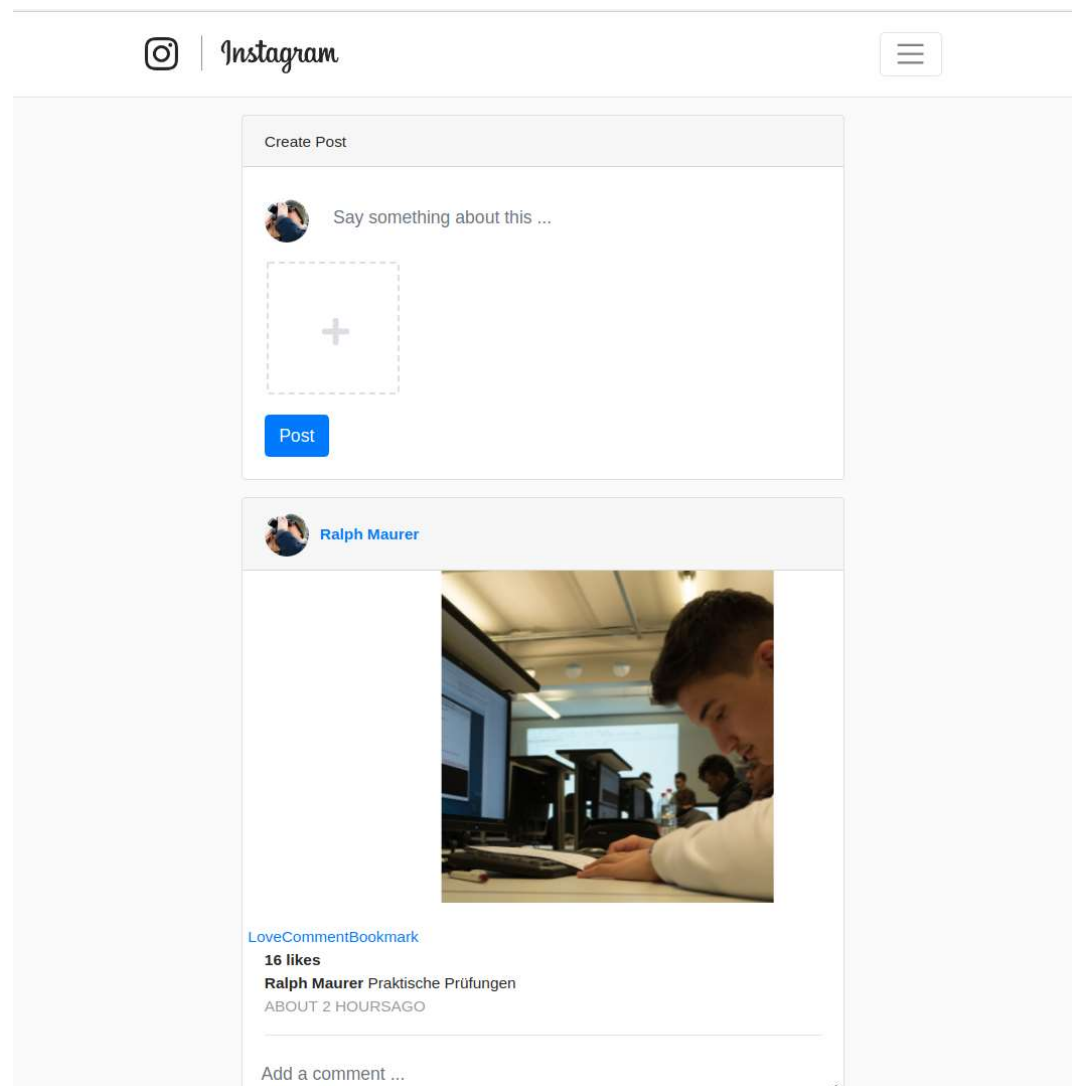


Es ist recht schwierig, den Überblick zu behalten: Es gibt viele <div>-Elemente und einige if-Statements. Die <div>-Elemente werden teilweise konditional erstellt, d.h. die Unterstützung der Verschachtelung durch den Editor (hier Atom) funktioniert nicht mehr.

In `app/views/posts/index.html.erb` packen wir das Rendering in den Container `<div id="post">`:

```
index.html.erb      application.scss      _posts_list.html.erb      upload_post_images.js
1 <div class="d-flex flex-column align-items-center mt-3">
2   <div class="col-xl-7 col-lg-8 col-md-10 col-sm-11 post-card">
3     <div class="card">
4       <div class="card-header"> Create Post </div>
5       <div class="card-body">
6         <%= form_for @post, :html => {:multipart => true, :class => "dropzone upload-images p-0 border-0"} do |f| %>
7           <div class="form-group row mt-2">
8             <div class="col-auto pr-0">
9               <%= image_tag avatar_url(current_user), class: "post-author-icon" %>
10            </div>
11            <div class="col pl-0">
12              <%= f.text_field :content, class: "form-control border-0",
13                placeholder: "Say something about this ..." %>
14            </div>
15          </div>
16          <div class="fallback"> <%= file_field_tag "images[]", type: :file, multiple: true %> </div>
17          <div class="dz-message m-0"></div>
18          <div class="dropzone-previews mb-3">
19            <div class="upload-photos-icon">
20              <i class="fa fa-plus fa-2x" aria-hidden="true" style="color:#dddfe2"></i>
21            </div>
22          </div>
23          <%= f.submit "Post", class: "btn btn-primary" %>
24        <%= end %>
25      </div>
26    </div>
27    <div id="post">
28      <%= render 'posts_list' %>
29    </div>
30  </div>
31</div>
32
```

Betrachten wir das Resultat, das schon sehr ansprechend aussieht:



Styling der Posts inkl. Icons

Wir müssen noch ein paar SCSS-Klassen definieren für eine verbesserte Anzeige von Posts und für die Anzeige der Icons *Love*, *Comment* und *Bookmark* an. Dazu erstellen wir folgende SCSS-Klassen in `app/assets/stylesheets/application.scss`:

```
301
302 // Customize post views
303 .post {
304   .card-header {
305     background-color: white;
306   }
307   .card-body {
308     padding: 5px 16px 5px;
309   }
310   .actions {
311     margin: 12px 0;
312   }
313   .post-time {
314     margin-top: 5px;
315     font-size: 10px;
316   }
317   .comment-input {
318     font-size: 14px;
319     resize: none;
320   }
321   .previous-btn {
322     background-position: -281px -280px;
323     height: 30px;
324     opacity: 1;
325     width: 30px;
326   }
327   .next-btn {
328     background-position: -353px 0px;
329     height: 30px;
330     opacity: 1;
331     width: 30px;
332   }
333 }
334
335 .love {
336   background-position: -325px -329px;
337   height: 24px;
338   width: 24px;
339 }
340 .loved {
341   @extend .love;
342   background-position: -200px -330px;
343 }
344 .comment {
345   @extend .love;
346   background-position: -258px -175px;
347   margin-left: 8px;
348 }
349 .bookmark {
350   @extend .love;
351   background-position: -150px -355px;
352 }
353 .bookmarked {
354   @extend .love;
355   background-position: -353px -273px;
356 }
357
```

Damit das SCSS greift, müssen in der Partial-View `app/views/posts/_posts_list.html.erb` noch folgende Links für *Love*, *Comment* und *Bookmark* angepasst werden:

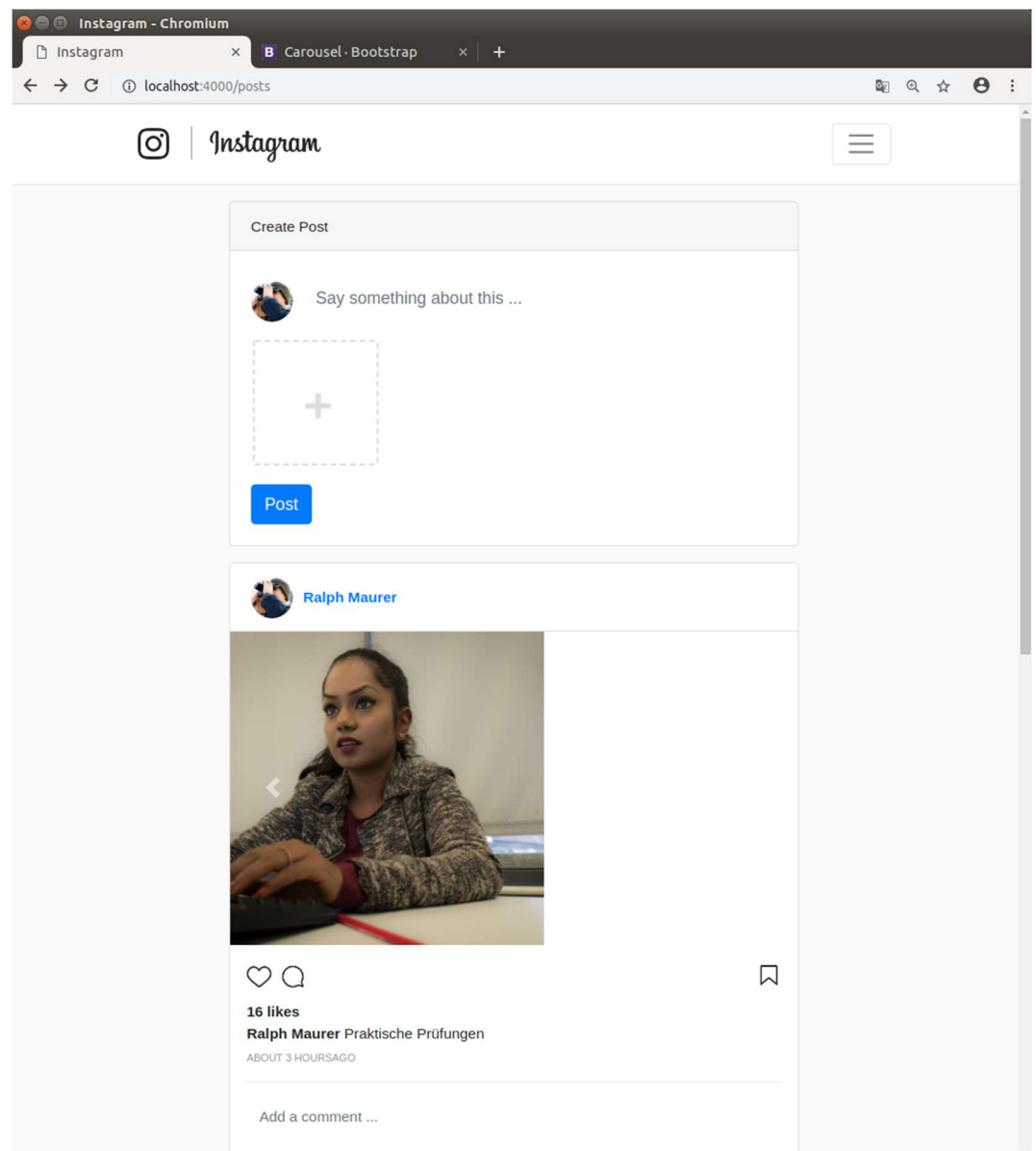
```
<div class="card-body">
  <div class="row actions">
    <a href="#" class="core-sprite love hide-text"> Love </a>
    <a href="#" class="core-sprite comment hide-text"> Comment </a>
    <a href="#" class="core-sprite bookmark hide-text ml-auto"> Bookmark </a>
  </div>
</div>
```

Alternative Icons

Da wir das Stylesheet von *Font Awesome* einbinden (Kapitel *Icons von Font Awesome*), können wir – anstatt die Icons vom Sprite-Bild zu verwenden – aus der riesigen Auswahl von Icons unsere gewünschten auswählen. Das könnte beispielsweise so aussehen:

```
<a href="#" class="far fa-heart fa-2x" style="color:#888888"></a>
<a href="#" class="far fa-comment fa-2x" style="color:#888888"></a>
...<a href="#" class="far fa-bookmark fa-2x" style="color:#888888"></a>
```

Nun sind wir schon fast zufrieden mit dem Resultat:



Störend wirkt die blaue Schrift für den Benutzernamen-Link neben dem Avatar:



Ralph Maurer



Erfassen Sie die zwei SCSS-Klassen `.no-text-decoration: hover` und `.normal-color` in `app/assets/stylesheets/application.scss`:

```
index.html.erb  application.scss  _posts_list.html.erb  upload_post_images.js

302 .no-text-decoration: hover {
303   text-decoration: none;
304   color: #262626;
305 }
306
307 .normal-color {
308   color: #262626;
309 }
310
311 // Customize post views
312 .post {
313   .card-header {
314     background-color: white;
315   }
}
```

Bildgrösse verändern

Die Bildgrösse für den Upload können wir in der Datei `app/uploaders/photo_uploader.rb` anpassen.

```
1 class PhotoUploader < CarrierWave::Uploader::Base
2   # Include RMagick or MiniMagick support:
3   # include CarrierWave::RMagick
4   # include CarrierWave::MiniMagick
5
6   # Choose what kind of storage to use for this uploader:
7   # storage :file
8   # storage :fog
9
10  # Override the directory where uploaded files will be stored.
11  # This is a sensible default for uploaders that are meant to be mounted:
12  def store_dir
13    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
14  end
15
16  include Cloudinary::CarrierWave
17
18  process :convert => 'jpg'
19  process :tags => ['post_picture']
20
21  version :standard do
22    process :resize_to_fill: [600, 480, :center]
23  end
24
25  version :thumbnail do
26    resize_to_fit(100, 100)
27  end
end
```



Probieren Sie verschiedene Einstellungen mit verschiedenen Fotos aus (Fotos sowohl im Hoch- als auch im Querformat). Die Dokumentation dazu finden Sie unter https://cloudinary.com/documentation/rails_integration

Für den Co-Autor ist es unverständlich, dass eine Anpassung der Werte `resize_to_fill` auch auf bereits veröffentlichte Posts eine Auswirkung hat. Die Werte von `600 x 480` und `center` haben sich bewährt. `center` bedeutet, dass die Bilder zentriert werden und nicht links oben positioniert sind.

Posts als Startseite

Die Post View soll zuletzt noch als Root-Page definiert werden:

```
root 'posts#index'
```

Auftrag: Quicknote AB151-06

Alle Aufträge des Typen Quicknote sind Bestandteil der Bewertung. Erstellen Sie Quicknotes mit 2-4 A4-Seiten Text. Dazu kommen Screenshots und Bilder, so dass schlussendlich mehr als 4 Seiten resultieren können.

Die Dokumentation muss zusammen mit der Rails-Applikation via Git abgegeben werden (z.B. GitLab oder GitHub). Das Format ist entweder *md* (Markdown) oder *pdf*. Eine Anleitung zu GitLab der Abteilung IET finden Sie auf dem Modulshare. Vergessen Sie nicht, das Projekt für die Lehrperson freizugeben!

Zusammenfassung und Anwendungszweck

Die Quicknotes sollen eine kurze, prägnante Zusammenfassung des Dokuments AB151-06 und einen Überblick über **alle** vorgestellten Techniken, Methoden und Konzepte beinhalten. Fassen Sie nicht die Arbeitsschritte des Auftrages zusammen, sondern beschreiben Sie nur kurz, was sie gemacht haben. Z.B. «Installation des Gem devise mit einem Eintrag im Gemfile und dem Ausführen des Befehls 'bundle install'».

Führen Sie ausführlicher auf, was die Techniken, Methoden und Konzepte bedeuten. Z.B. «Mit Gem devise wird eine Authentifizierung in Rails erstellt. Dazu gehören das Model (Tabelle 'users') und die Seiten/Views Sign-up (users/sign_up) und Sign-in (users/sign_in). Nach erfolgreichem Sign-in wird eine Session erstellt...usw...».

Achten Sie darauf, dass alle wesentliche Themen inkl. Anwendungszweck beschrieben werden und wenn möglich in Zusammenhang gebracht sind.

Die Fragen zum Anwendungszweck werden direkt mit der Zusammenfassung beantwortet: Wie und wo können die vorgestellten Techniken, Methoden und Konzepte in einer Rails-Applikation (z.B. Instagram) angewendet werden?

Zu folgenden Themen/Kapiteln im AB151-06 sollen die Vor- und Nachteile aufgeführt werden:

- gem 'dropzonejs-rails'
- Icons von Font Awesome
- Design der Seite mit den Posts

Vergessen Sie nicht, auch die übrigen Themen/Techniken in der Zusammenfassung zu beschreiben!

Selbstreflexion

Folgende Fragen müssen bei der Selbstreflexion beantwortet werden:

- 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?
- Was kann ich nächstes Mal besser machen?

Es sollen alle wesentlichen Themen des Arbeitsblattes bei der Reflexion berücksichtigt werden. Wenn Sie etwas bereits kannten/wussten und Sie deshalb bei diesem Aspekt nichts lernten, dann muss das bei der Reflexion erwähnt werden! Mit anderen Worten: Reflektieren Sie über alle Themen und Arbeitsschritte und überlegen Sie sich dabei, was Sie bereits gekannt haben, was Sie gelernt haben und wie Sie dabei vorgegangen sind.

Formulieren Sie vollständige Sätze und seien Sie bei der Selbstreflexion ausführlich! Die Selbstreflexion sollte mind. eine halbe Seite lang sein.

Lösungen der Aufgaben

Das Dokument muss die Lösungen der Aufgaben beinhalten. Die Lösungen können - anstatt in einem separaten Kapitel - auch in der Zusammenfassung enthalten sein. In diesem Fall sollten sie speziell markiert werden, damit die Lehrperson diese schnell findet!

Dieses Arbeitsblatt hat nur 2 Aufgaben: S. 8 und 9.

Abschliessende Reflexion über das Gelernte:

Schreiben Sie eine abschliessende, zusammenfassende Reflexion:

- Eine Zusammenfassung über das Gelernte (das ist eine Wiederholung der Selbstreflexion in komprimierter Form).
- Der Lernfortschritt, den Sie mit diesem Arbeitsblatt erzielt haben, in Bezug auf das Rails-Framework allgemein und/oder in Bezug auf die Instagram-Applikation im Speziellen.

Instagram Applikation

Ziel der Arbeitsblätter ist in erster Linie das Entwickeln der Instagram Applikation, nicht das Erstellen der Dokumentation. In den Quicknotes sollen deshalb Screenshots den Projektfortschritt dokumentieren.

Folgende Screenshots müssen in dieser Dokumentation enthalten sein:

- Screenshot von Liste mit Posts, am Ende des Arbeitsblattes:
 - Post mit Kommentar
 - Post mit einem Bild
 - Post mit mehreren Bildern

Sie bauen die Screenshots am besten in die Zusammenfassung ein.

In diesem Arbeitsblatt gab es viel zu codieren und nur wenige Aufgaben. Das Endresultat ist umso wichtiger und fliesst in diesem Fall mit mehr Punkten in die Bewertung ein!

Abgabe der Quicknotes und der Applikation

Termin für die Abgabe: Gemäss Angaben der Lehrperson.

Checken Sie Ihr Projekt am Ende des Arbeitsblattes auf dem Git Server ein. Vergessen Sie nicht, dass auch die Dokumentation enthalten sein muss. Versehen Sie das Projekt anschliessend mit einem Release-Tag und pushen auch dieses auf den Server.

Eine Anleitung zu Git finden Sie auf dem Modulshare (Dokument "AB151_GitAnleitung.pdf").