



# MODUL 151

## TEIL 3: INSTAGRAM AUTHENTICATION VIEW

Ralph Maurer

# Inhaltsverzeichnis AB151-03

## Modul 151: Instagram mit Rails bauen

### Teil 3: Instagram Authentication View

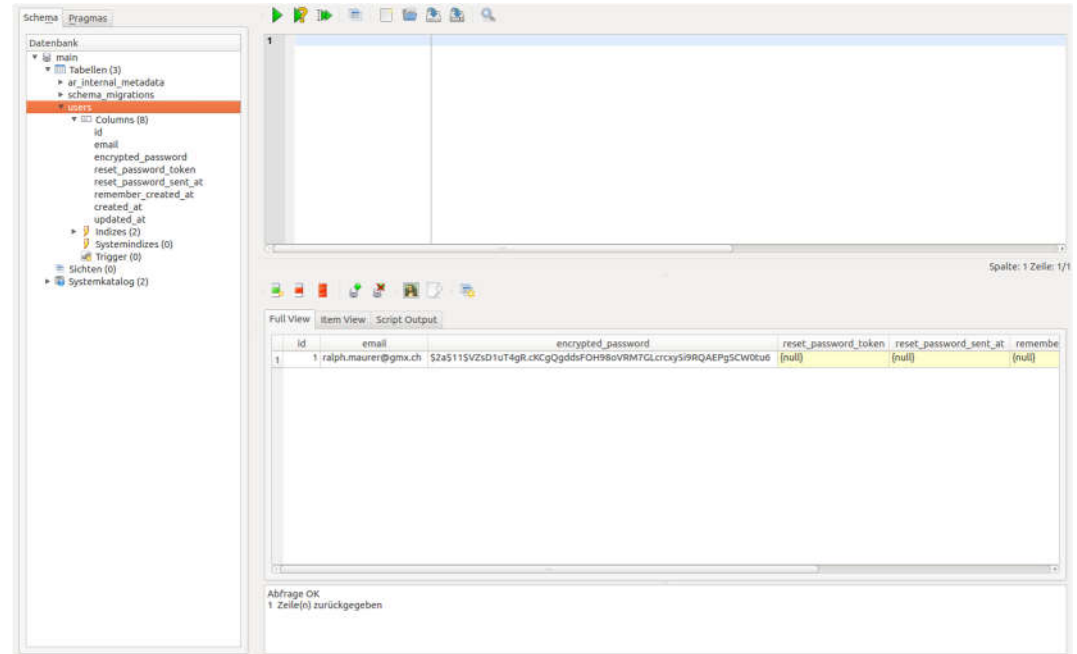
|  |    |
|--|----|
| Inhaltsverzeichnis AB151-03 .....                              | 1  |
| Modul 151: Instagram mit Rails bauen .....                     | 1  |
| Attribut Benutzernamen erfassen.....                           | 2  |
| Login-View anpassen.....                                       | 4  |
| Menu ausblenden im Logout-Bereich.....                         | 4  |
| Dummy-Phone mit Carousel-Funktion in Login-View einbauen ..... | 5  |
| Carousel in Login-View einbauen.....                           | 8  |
| Footer einbauen .....  | 12 |
| Registration-View anpassen.....                                | 13 |
| Partial-View mit Dummy-Phone erstellen .....                   | 13 |
| Code von Login-View übernehmen.....                            | 14 |
| Auftrag: Quicknote AB151-03 .....                              | 17 |

# Attribut Benutzernamen erfassen

Betrachten wir die Datenbank, die «devise gem» generiert hat.

Sqliteman

```
$ cd workspace/instagram/db  
$ sqliteman development.sqlite3
```



Rails Konsole

```
2.4.1 :001 > User.connection  
...  
2.4.1 :002 > User  
=> User(id: integer, email: string, encrypted_password: string,  
reset_password_token: string, reset_password_sent_at: datetime,  
remember_created_at: datetime, created_at: datetime, updated_at:  
datetime)
```

## Das Attribut Benutzername fehlt!

Fügen Sie dieses Attribut mit einer Migration mit Namen `AddNameToUser` hinzu. Rails nutzt Namenskonventionen und wenn Sie bei der Migrationserstellung `Add«Attributnamen»To«Tabellennamen»` zusammen mit dem Attributnamen und Datentyp verwenden, wird gleich die fixfertige Migrationsdatei erstellt.

```
1 class AddNameToUser < ActiveRecord::Migration[5.2]  
2   def change  
3     add_column :users, :name, :string  
4   end  
5 end  
6
```

Wie lautet der Migrationsbefehl, damit die Migrationsklasse wie abgebildet erstellt wird?

Erstellen Sie die Migration und führen Sie diese aus. Notieren Sie die notwendigen Kommandos:

```
rails g migration AddNameToUser name:string  
rails db:migrate
```

Wir wollen nun den Benutzernamen nutzen und validieren. Erstellen Sie in der Datei `instagram/models/user.rb` eine Validierung, die sicherstellt:

- › dass der Benutzername angegeben worden ist
- › dass die max. Länge 50 Zeichen nicht überschreitet

Notieren Sie die Validierung:

```
application_controller.rb      user.rb  
1  class User < ApplicationRecord  
2    # Include default devise modules. Others available are:  
3    # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable  
4    devise :database_authenticatable, :registerable,  
5           :recoverable, :rememberable, :validatable  
6  
7    validates :name, presence: true, length: { maximum: 50 }  
8  end
```

Die «devise gem» lässt aktuell nur die E-Mail Adresse als Benutzernamen zu. Um dies zu ändern müssen wir `instagram/controllers/application_controller.rb` anpassen.



Beachten Sie, dass der folgende Code «devise gem» spezifisch ist und einfach übernommen werden kann bzw. sollte.

Wollen Sie mehr wissen, finden Sie hier die «devise gem»-Dokumentation:

<https://www.rubydoc.info/gems/devise/Devise>

```
application_controller.rb  
1  class ApplicationController < ActionController::Base  
2    protect_from_forgery with: :exception  
3  
4    before_action :configure_permitted_parameters, if: :devise_controller?  
5  
6    protected  
7  
8    def configure_permitted_parameters  
9      devise_parameter_sanitizer.permit(:sign_up, keys: [:name])  
10     devise_parameter_sanitizer.permit(:account_update, keys: [:name])  
11   end  
12 end
```

Achten Sie darauf, den Code korrekt zu übernehmen. Andernfalls wird die Rails-App nicht mehr funktionieren.

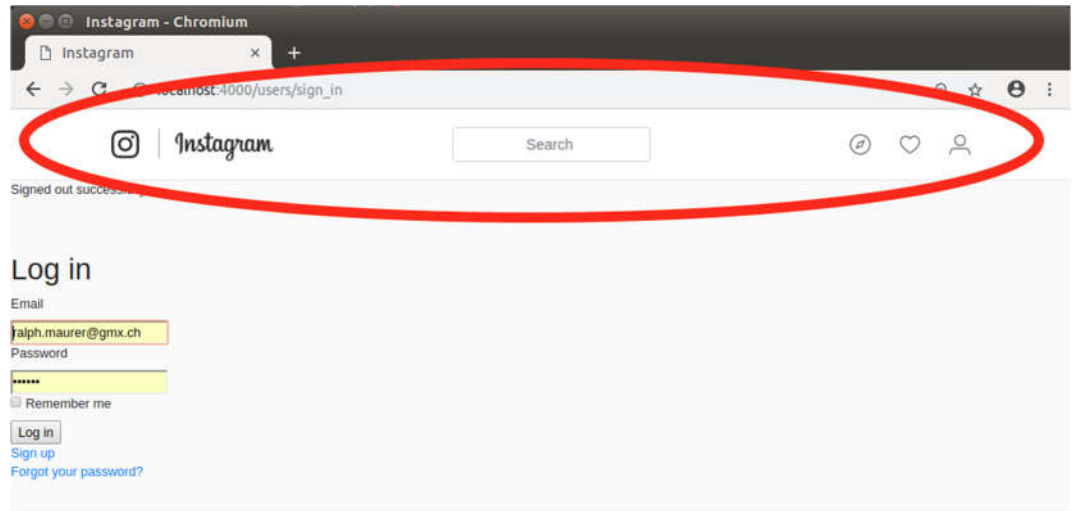
Sie können es testen indem Sie den Rails Server neu starten. Mögliche Syntaxfehler werden genau beschrieben und können korrigiert werden.

# Login-View anpassen

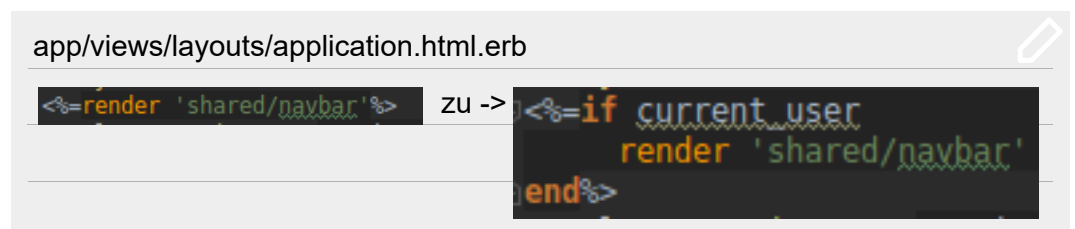
Die Login-View «sign\_in» ist unter [http://localhost:4000/users/sign\\_in](http://localhost:4000/users/sign_in) aufrufbar.

## Menu ausblenden im Logout-Bereich

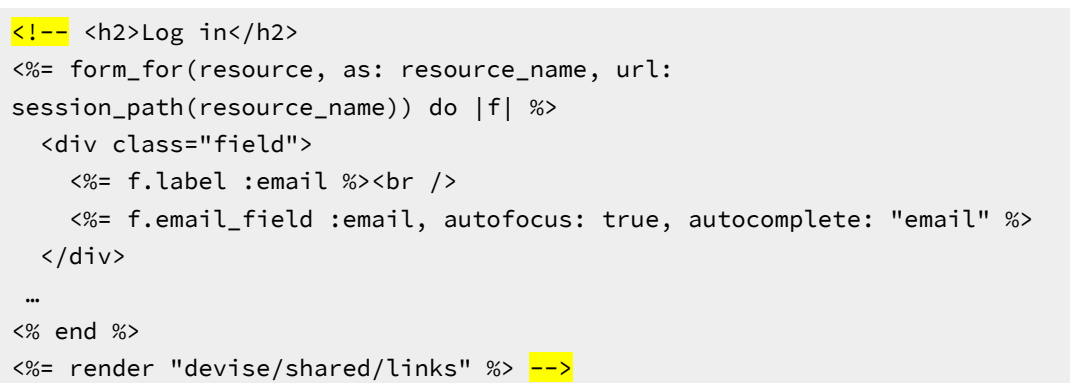
Bevor wir nicht eingeloggt sind, soll das obere Menu verschwinden. Dazu bietet uns «devise gem» die Variable `current_user`. Mit der Abfrage `if current_user` wird geprüft, ob eine valide Authentifikation durch einen Benutzer stattgefunden hat.



Bauen Sie die Abfrage in der richtigen View am richtigen Ort ein. Notieren Sie Pfad und Namen der View und die geänderte Codezeile:



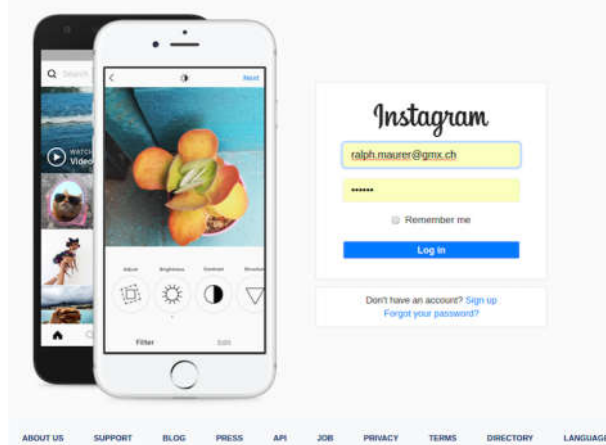
Wir passen nun die Login-View an, was einiges an Schreibaufwand verursacht. Öffnen Sie `/app/view/devise/sessions/new.html.erb` und kommentieren Sie alles aus. Einige Teile können später wiederverwendet werden (`<ctrl> + /`).



# Dummy-Phone mit Carousel-Funktion in Login-View einbauen

Bootstrap 4 bietet Carousel-Klassen, mit denen automatisch wechselnde Bilder realisiert werden können. Wir wollen von dieser Möglichkeit Gebrauch machen.

Die Login-View soll am Ende des Arbeitsblattes in etwa so aussehen:



Auf der linken Seite platzieren wir ein Dummy-Phone, dessen Display mit einem Image-Slider gestaltet wird. Auf die rechte Seite kommt das Registrierungsformular. Passen Sie die Datei `/app/view/devise/sessions/new.html.erb` wie folgt an:


```
1 <div class="container">
2   <div class="row">
3     <div class="col-lg-6 landing-left">
4       <div class="dummy-phone">
5         <div class="screen-shot">
6           <div class="carousel carousel-fade" data-ride="carousel">
7             <div class="carousel-inner">
8               <div class="carousel-item active">
9                 <%= image_tag "screenshot1.jpg", height: '427', width: '240' %>
10              </div>
11              <div class="carousel-item">
12                <%= image_tag "screenshot2.jpg", height: '427', width: '240' %>
13              </div>
14              <div class="carousel-item">
15                <%= image_tag "screenshot3.jpg", height: '427', width: '240' %>
16              </div>
17              <div class="carousel-item">
18                <%= image_tag "screenshot4.jpg", height: '427', width: '240' %>
19              </div>
20              <div class="carousel-item">
21                <%= image_tag "screenshot5.jpg", height: '427', width: '240' %>
22              </div>
23            </div>
24          </div>
25        </div>
26      </div>
27    </div>
28    <div class="col-lg-6 landing-right">
29
30
31  </div>
32 </div>
33 </div>
```

Ergänzen Sie die SCSS-Datei `app/assets/stylesheets/applications.scss` wie folgt:

```
74 // Landing
75 .landing-left {
76   .dummy-phone {
77     background-image: image-url("dummy-phone-2x.png");
78     background-repeat: no-repeat;
79     background-position: right 0;
80     background-size: 454px 618px;
81     position: relative;
82     height: 618px;
83     margin-left: -35px;
84     margin-right: -15px;
85   }
86   .screen-shot {
87     position: absolute;
88     top: 99px;
89     right: 63px;
90
91     .item {
92       position: absolute;
93     }
94   }
95 }
96
```

Was fällt Ihnen auf? Stichwort SCSS, CSS und «nesting»?

```
.landing-left {
  .dummy-phone {
    ...
  }
  .screen-shot {
    ...
    .item {
      ...
    }
  }
}
```

Die Verschachtelung in SCSS ist sehr übersichtlich und praktisch. 

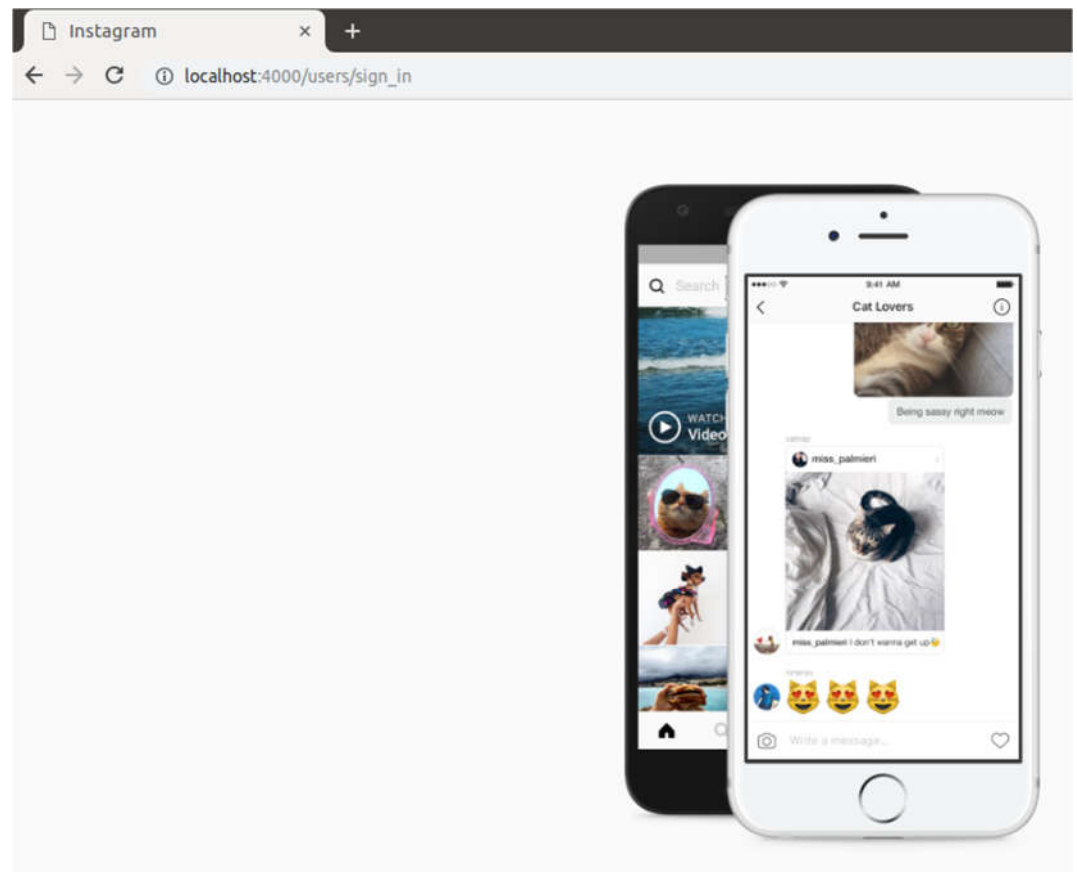
In CSS müsste man die "item"-Klasse so selektieren:

```
.landing-left .dummy-phone .item{
```

Man kann in SCSS ausserdem CSS-properties verschachteln:

```
.demo {
  background: {
    image: url("image.jpg");
    position: fixed;
  }
  margin : {
    top: 1px;
    right: 1px;
  }
}
```

Schauen wir uns das Resultat im Browser an:



Welches Attribut wechselt die Bilder im Dummy-Phone?

data-ride

Welche Klasse sorgt dafür, dass die Bilder im Wechsel angezeigt werden?

carousel-item definiert ein Bild des Karusells. Diese werden automatisch rotiert.

Mit welchen Bootstrap- und SCSS-Klassen werden die Bilder des Sliders positioniert?

```
.screen-shot{  
  position: absolute;  
  top: 99px;  
  right: 63px;  
}
```

Welche SCSS-Klassen unterteilen die Seite in links (Dummy-Phone) und in rechts (Registrierung)?

col-lg-6 bedeutet, dass das Element 6/12 der Breite bekommt. (Desktop größe)  
Somit sind landing-left und -right je 1/2 der Gesamtbreite breit.

Wo wird der Hintergrund des Dummy-Phone definiert (das Mobiltelefon Gehäuse)?

```
.dummy-phone{  
  background-image: image-url("dummy-phone-2x.png");  
}
```



## Carousel in Login-View einbauen

Jetzt können Sie den auskommentierten Bereich in  
`app/view/devise/sessions/new.html.erb` wiederverwenden und anpassen.

Fügen Sie den gesamten Code in das `<div>`-Tag ein:

```
<div class="col-lg-6 landing-right">
...
</div>
```

Machen Sie die Anpassung mit SCSS-Klassen von Bootstrap. Die verwendeten Klassen sind unter <https://getbootstrap.com/docs/4.0/components/forms/> genauer erklärt.

```
<div class="form-login box">
  <h3 class="core-sprite brand-name-img"></h3>
  <%= form_for(resource, as: resource_name, url: session_path(resource_name)) do |f| %>

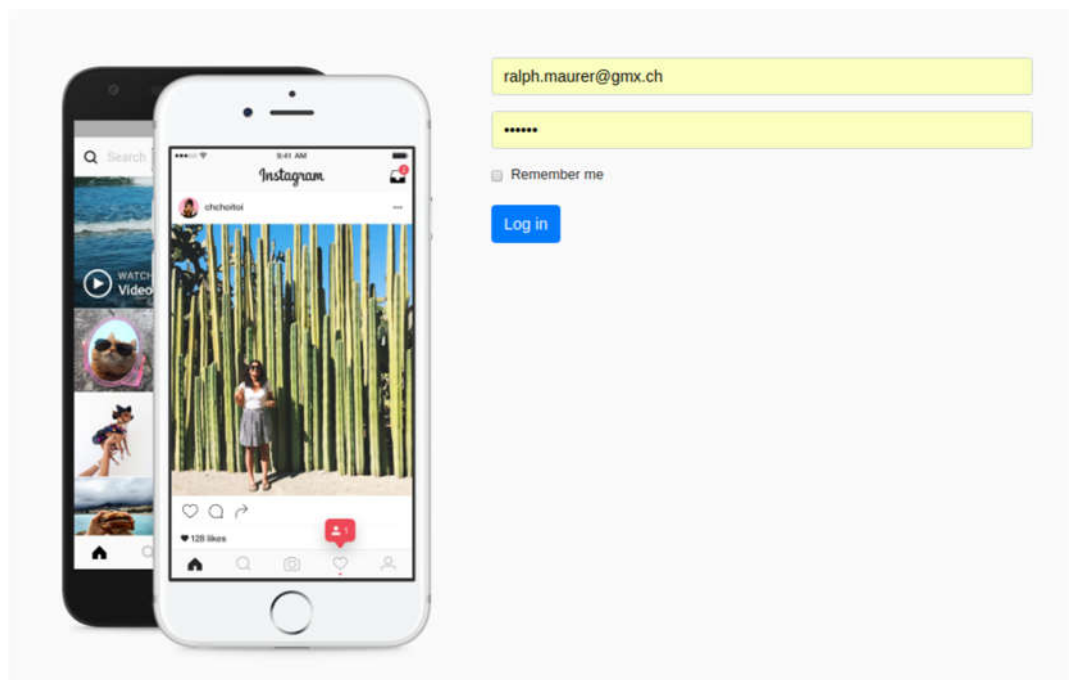
  <div class="form-group">
    <%= f.email_field :email, autofocus: true, placeholder: "Email", class: "form-control" %>
  </div>

  <div class="form-group">
    <%= f.password_field :password, autocomplete: "off", placeholder: "Password", class: "form-control" %>
  </div>

  <% if devise_mapping.rememberable? %>
    <div class="form-check">
      <%= f.check_box :remember_me, class: "form-check-input" %>
      <%= f.label :remember_me, class: "form-check-label" %>
    </div>
    <br>
  <% end %>

  <div class="actions">
    <%= f.submit "Log in", class: "btn btn-primary" %>
  </div>
  <br>
  <% end %>
</div>
```

Sie sollten nun folgende Ansicht im Browser erhalten:



Wir beginnen unsere Authentifikation zu gestalten:

```
<div class="form-login box">
```

```
<h3 class="core-sprite brand-name-img"></h3>
```

Hierzu müssen wir die Klassen `.box` und `.brand-name-img` im SCSS definieren.

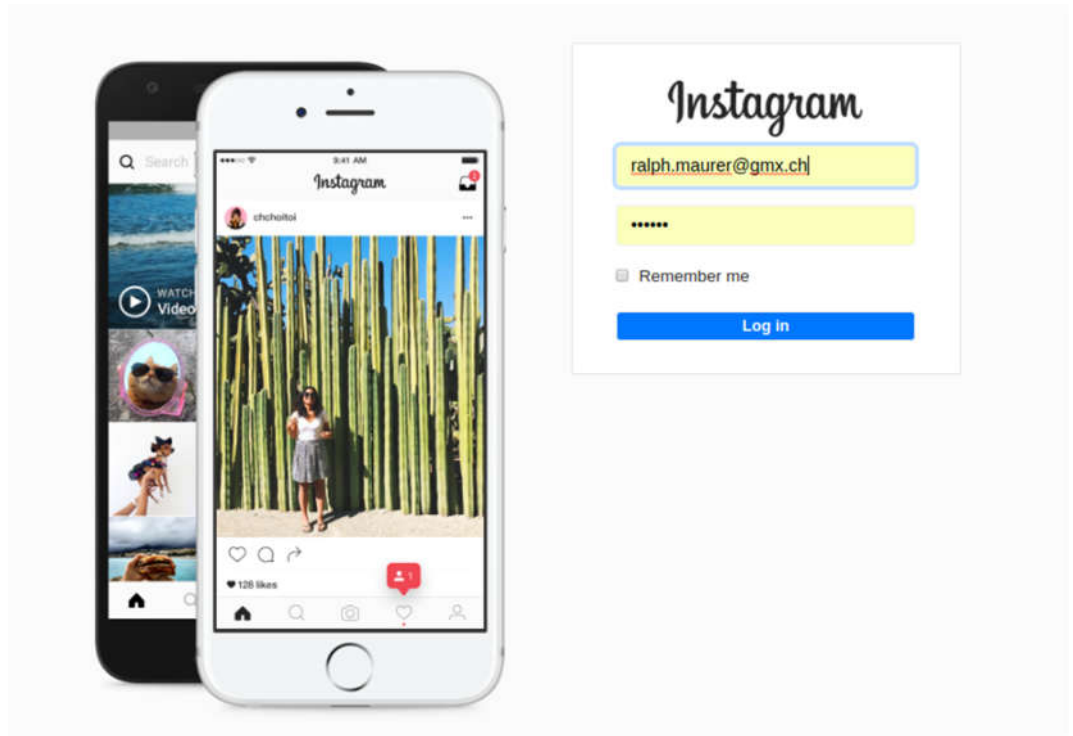
Ergänzen Sie `app/assets/stylesheets/applications.scss` um die zwei Klassen:

```
.box {  
  background-color: #fff;  
  border: 1px solid #e6e6e6;  
  border-radius: 1px;  
  padding: 10px 0;  
  margin: 0 0 10px;  
}  
  
.brand-name-img {  
  background-position: 0 0;  
  height: 51px;  
  width: 175px;  
}
```

Die ganze rechte Seite muss formatiert werden. Erstellen Sie dazu die SCSS-Klasse `.landing-right` mit folgendem Inhalt:

```
.landing-right {  
  .form-login {  
    width: 350px;  
  
    form {  
      padding: 0 40px;  
  
      .btn-primary {  
        width: 100%;  
        line-height: 14px;  
        font-size: 14px;  
        font-weight: 600;  
      }  
  
      input {  
        border: 1px solid #efefef;  
      }  
  
      .form-check-label {  
        padding: 0;  
        font-size: 15px;  
        color: #262626;  
      }  
    }  
  
    .brand-name-img {  
      margin: 22px auto 8px;  
    }  
  }  
  
  .redirect-links {  
    width: 350px;  
  }  
}
```

Wenn Sie alles richtig gemacht haben, sollte Ihre Login-View nun so aussehen:

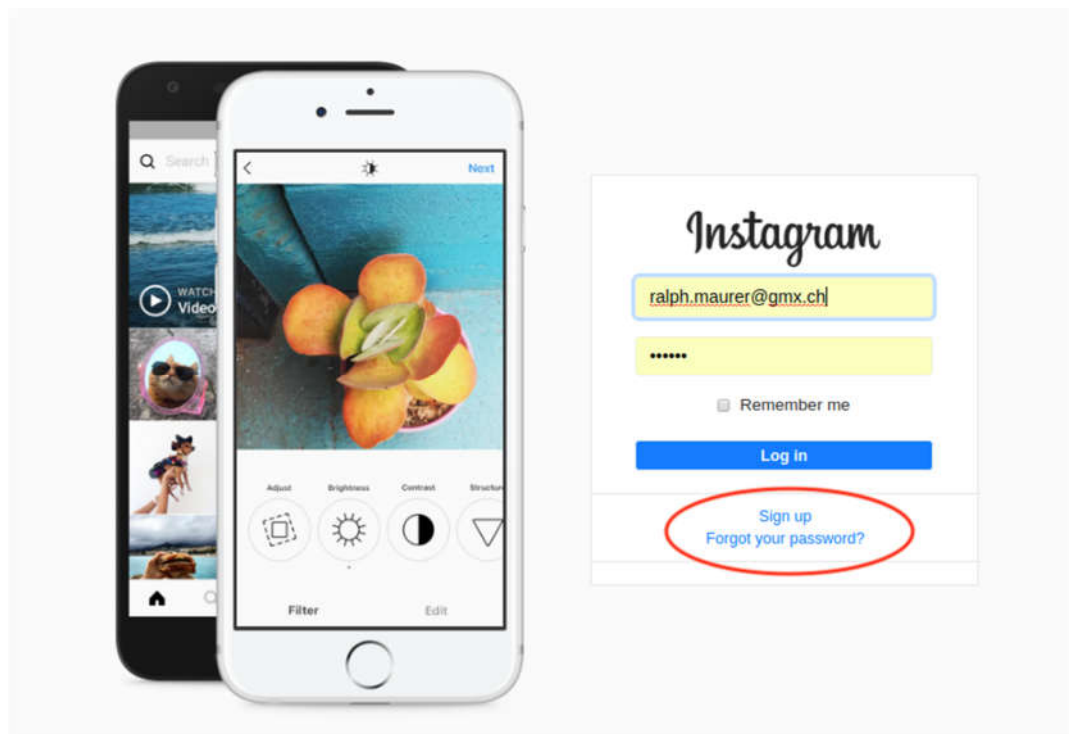


Das Resultat sieht schon ziemlich ansprechend aus!

Zentrieren wir noch die Elemente des `<div>`-Containers mit der Klasse `.landing-right` horizontal und vertikal:

```
<div class="col-lg-6 landing-right text-center d-flex align-items-center">
```

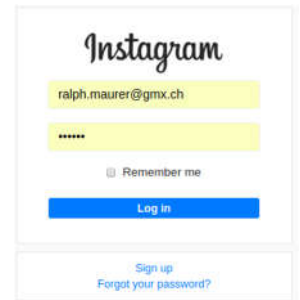
Die Links [Sign up](#) und [Forgot your password?](#) können wir als Partial-View mit Hilfe der «gem devise» einblenden:



Fügen Sie dazu den folgenden Code direkt nach `<% end %>` (Abschluss von `form_for`) ein:

```
<div class="redirect-links box">
  <%= render "devise/shared/links"%>
</div>
```

Zwischen dem Formular und den Links [Sign up](#) und [Forgot your password?](#) wollen wir eine Begrenzung mit einem `<div>`-Container einbauen:



Führen Sie die Anpassung gemäss folgender Abbildung durch:

```
28 <div class="col-lg-6 landing-right text-center d-flex align-items-center">
29   <div>
30     <div class="form-login box">
31       <h3 class="core-sprite brand-name-img"></h3>
32       <%= form_for(resource, as: resource_name, url: session_path(resource_name)) do |f| %>
33         <div class="form-group">
34           <%= f.email_field :email, autofocus: true, placeholder: "E-Mail", class: "form-control" %>
35         </div>
36         <div class="form-group">
37           <%= f.password_field :password, autocomplete: "off", placeholder: "Passwort", class: "form-control" %>
38         </div>
39         <%= if devise_mapping.rememberable? %>
40           <div class="form-check">
41             <%= f.check_box :remember_me, class: "form-check-input" %>
42             <%= f.label :remember_me, class: "form-check-label" %>
43           </div>
44         <br />
45         <%= end %>
46         <div class="actions">
47           <%= f.submit "Log in", class: "btn btn-primary" %>
48         </div>
49       <br />
50     <%= end %>
51   </div>
52   <div class="redirect-links box">
53     <%= render "devise/shared/links" %>
54   </div>
55 </div>
56 </div>
```

Der vorhin eingefügte Code `<div class="redirect-links box">`... muss noch eine `<div>`-Ebene nach aussen verschoben werden, siehe grossen roten Kreis.

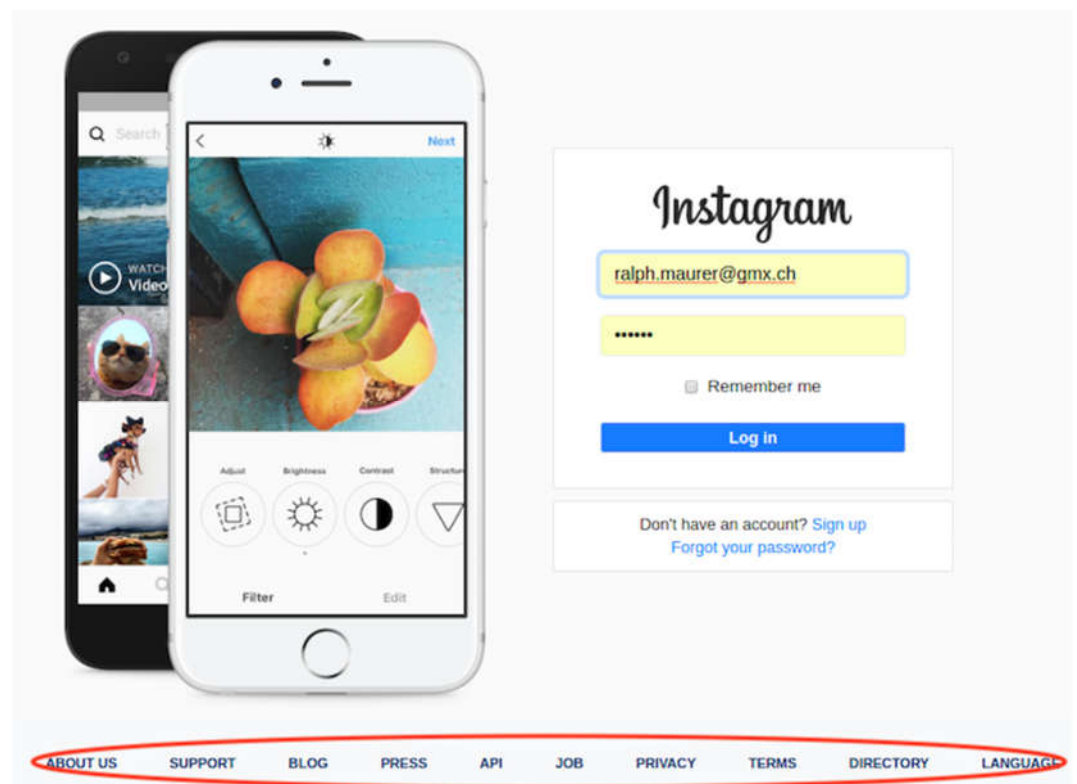
Damit die Links passender aussehen, ergänzen wir die «gem devise» Seite `app/views/devise/shared/_links.html.erb` wie folgt:

```
1 <%= if controller_name != 'sessions' %>
2   <%= "Have an account?" %>
3   <%= link_to "Log in", new_session_path(resource_name) %><br />
4 <%= end %>
5
6 <%= if devise_mapping.registerable? && controller_name != 'registrations' %>
7   <%= "Don't have an account?" %>
8   <%= link_to "Sign up", new_registration_path(resource_name) %><br />
9 <%= end %>
10
11 <%= if devise_mapping.recoverable? && controller_name != 'passwords' && controller_name != 'registrations' %>
12   <%= link_to "Forgot your password?", new_password_path(resource_name) %><br />
13 <%= end %>
14
15 <%= if devise_mapping.confirmable? && controller_name != 'confirmations' %>
16   <%= link_to "Didn't receive confirmation instructions?", new_confirmation_path(resource_name) %><br />
17 <%= end %>
18
19 <%= if devise_mapping.lockable? && resource_class.unlock_strategy_enabled?(:email) && controller_name != 'unlocks' %>
20   <%= link_to "Didn't receive unlock instructions?", new_unlock_path(resource_name) %><br />
21 <%= end %>
22
23 <%= if devise_mapping.omniauthable? %>
24   <%= resource_class.omniauth_providers.each do |provider| %>
25     <%= link_to "Sign in with #{OmniAuth::Utils.camelize(provider)}", omniauth_authorize_path(resource_name, provider) %><br />
26   <%= end %>
27 <%= end %>
```

## Footer einbauen

Der Footer soll auf jeder Seite vorhanden sein, deshalb erstellen wir eine Partial-View für den Footer. Der Footer soll nur einmalig gerendert werden und trotzdem in jeder View sichtbar sein.

Das Endresultat soll wie folgt aussehen:



HTML-Vorschlag für den Footer:

```
<footer>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbar">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbar">
      <div class="navbar-nav mx-auto">
        <a href="#" class="nav-item nav-link" href="#">ABOUT US</a>
        <a href="#" class="nav-item nav-link" href="#">SUPPORT</a>
        <a href="#" class="nav-item nav-link" href="#">BLOG</a>
        <a href="#" class="nav-item nav-link" href="#">PRESS</a>
        <a href="#" class="nav-item nav-link" href="#">API</a>
        <a href="#" class="nav-item nav-link" href="#">JOB</a>
        <a href="#" class="nav-item nav-link" href="#">PRIVACY</a>
        <a href="#" class="nav-item nav-link" href="#">TERMS</a>
        <a href="#" class="nav-item nav-link" href="#">DIRECTORY</a>
        <a href="#" class="nav-item nav-link" href="#">LANGUAGE</a>
      </div>
    </div>
  </nav>
</footer>
```

CSS-Vorschlag für den Footer:

```
footer .nav-item {  
  color: #003569 !important;  
  font-size: 12px;  
  font-weight: bold;  
}
```

1. In welcher Datei haben Sie den Code für den Footer als Partial-View erfasst?
2. Wo wird die Datei mit dem Footer gerendert?
3. Wie lautet der Befehl zum Rendern?

`/app/views/shared/_footer.html.erb`

`/app/views/layouts/application.html.erb` unter `<%= yield%>`

`<%= render 'shared/footer' %>`

## Registration-View anpassen

Die Registration-View «sign\_up» ist unter [http://localhost:4000/users/sign\\_up](http://localhost:4000/users/sign_up) aufrufbar.

Um nicht bei null zu beginnen, wollen wir so weit wie möglich den Code der Login-View für die Registration-View wiederverwenden. Wir unterscheiden zwei Fälle: Das Dummy-Phone mit der Carousel-Funktion ist in beiden Views identisch und kann 1:1 übernommen werden. Beim Formular auf der rechten Seite hingegen wird nur die Struktur der Login-View übernommen, der Inhalt wird von der (bereits existierenden) Registration-View verwendet.

## Partial-View mit Dummy-Phone erstellen

Erstellen Sie nun eine Partial-View mit dem Dummy-Phone Teil aus `/app/view/devise/sessions/new.html.erb`.

Die Partial-View soll `_dummy_phone.html.erb` heißen. Es wird sämtlicher Code innerhalb und inklusive des Tags `<div class="dummy-phone">` in diese Datei verschoben.

1. In welcher Datei haben Sie den Code für das Dummy-Phone erfasst?
2. Wo wird die Datei mit dem Dummy-Phone für die Views «sign\_in» und «sign\_up» gerendert?
3. Wie lautet der Befehl zum rendern dieser Datei?

`/app/views/shared/_dummy_phone.html.erb` wird in

`/app/views/devise/sessions/new.html.erb` und

`/app/views/devise/registrations/new.html.erb` eingebunden



## Code von Login-View übernehmen

Sie können den rot markierten Teil aus der Login-View kopieren und in `app/view/devise/registrations/new.html.erb` einfügen:

```
<div class="container">
  <div class="row">
    <div class="col-lg-6 landing-left">
      <%= render "devise/shared/dummy_phone"%>
    </div>
    <div class="col-lg-6 landing-right text-center d-flex align-items-center">
      <div>
        <h2>Sign up</h2>

        <%= form_for(resource, as: resource_name, url: registration_path(resource_name)) do |f| %>
          <%= devise_error_messages! %>

          <div class="field">
            <%= f.label :email %><br />
            <%= f.email_field :email, autofocus: true, autocomplete: "email" %>
          </div>

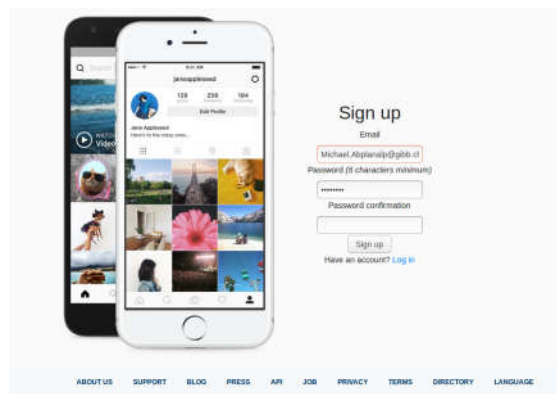
          <div class="field">
            <%= f.label :password %>
            <%= if @minimum_password_length %>
              <em>(<%= @minimum_password_length %> characters minimum)</em>
            <%= end %><br />
            <%= f.password_field :password, autocomplete: "new-password" %>
          </div>

          <div class="field">
            <%= f.label :password_confirmation %><br />
            <%= f.password_field :password_confirmation, autocomplete: "new-password" %>
          </div>

          <div class="actions">
            <%= f.submit "Sign up" %>
          </div>
        <%= end %>

        <%= render "devise/shared/links" %>
      </div>
    </div>
  </div>
</div>
```

Die Registration-View sollte nun so aussehen:



Fügen Sie folgende Klassen in `app/assets/stylesheets/application.scss` ein:

```
.carousel-fade {
  .carousel-item.active {
    opacity: 1;
    display: block;
  }

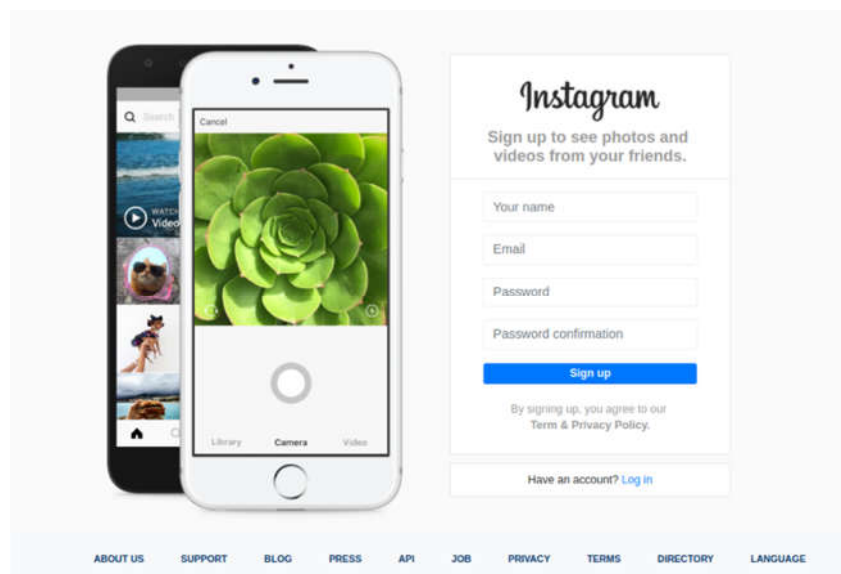
  .carousel-item {
    transition: opacity 0.6s ease-in-out;
    opacity: 0;
    position: relative;
    display: block;
  }

  .carousel-item:not(.active) {
    position: absolute;
    top: 0px;
  }
}
```

```
.light-color {
  color: #999;
}
```

.carousel-fade sorgt für einen Fade-Effekt im Slider innerhalb des Dummy-Phone. Und mit .light-color erhalten wir einen leicht eingefärbten Einführungstext (das wenden wir weiter unten an).

Passen Sie nun `app/view/devise/registrations/new.html.erb` so an, dass die View der folgenden Vorgabe entspricht:



Verwenden Sie dazu wie erwähnt die Struktur der Login-View und fügen den Inhalt der aktuellen Registration-View ein. Zudem müssen Sie fehlende Elemente ergänzen: Den Benutzernamen und die Texte *Sign up to see photos and videos from your friends.* und *By signing up, you agree to our Term & Privacy Policy.*

Um die Texte einzufügen, können Sie folgenden Code verwenden:

```
<h5 class="light-color"><strong>Sign up to see photos and <br />
                                videos from your friends.</strong></h5>
```

und

```
<p class="light-color">By signing up, you agree to our <br />
                                <strong>Term & Privacy Policy.</strong></p>
```

Beachten Sie die CSS-Klasse `.light-color`, die wir vorhin eingefügt haben.



## Lösung RMA:

```
<div class="container">
  <div class="row">
    <div class="col-lg-6 landing-left">
      <%= render "devise/shared/dummy_phone"%>
    </div>
    <div class="col-lg-6 landing-right text-center d-flex align-items-center">
      <div>
        <div class="form-login box">
          <h2 class="core-sprite brand-name-img"></h2>
          <h5 class="light-color"><strong>Sign up to see photos and <br> videos from your friends.</strong></h5>
          <hr>
          <%= form_for(resource, as: resource_name, url: registration_path(resource_name)) do |f| %>
            <%= devise_error_messages! %>
            <div class="form-group">
              <%= f.text_field :name, autofocus: true, placeholder: "Your name", class: "form-control" %>
            </div>
            <div class="form-group">
              <%= f.email_field :email, autofocus: true, placeholder: "Email", class: "form-control" %>
            </div>
            <div class="form-group">
              <%= f.password_field :password, autocomplete: "off", placeholder: "Password", class: "form-control" %>
            </div>
            <div class="form-group">
              <%= f.password_field :password_confirmation, autocomplete: "off", placeholder: "Password confirmation", class: "form-control" %>
            </div>
            <div class="actions">
              <%= f.submit "Sign up", class: "btn btn-primary"%>
            </div>
          <end %>
          <br>
          <p class="light-color">By signing up, you agree to our <br><strong>Term & Privacy Policy.</strong></p>
        </div>
        <div class="redirect-links box">
          <%= render "devise/shared/links"%>
        </div>
      </div>
    </div>
  </div>
</div>
```

# Auftrag: Quicknotes AB151-03

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 bekommen Sie im Unterricht. Vergessen Sie nicht, das Projekt für die Lehrperson freizugeben!

## Zusammenfassung und Anwendungszweck

Die Quicknotes sollen eine kurze, prägnante Zusammenfassung des Dokuments AB151-03 und einen Überblick über die vorgestellten Techniken, Methoden und Konzepte beinhalten. Achten Sie darauf, dass alle wesentliche Themen vollständig erwähnt 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?
- Was sind die Vor- und was die Nachteile?

Zu folgenden Themen/Kapiteln im AB151-03 müssen der Anwendungszweck und/oder die Vor- und Nachteile aufgeführt werden:

- Benutzernamen erfassen
- Dummy-Phone mit Carousel-Funktion
- Partial-Views 'Footer' und 'Dummy-Phone'
- Anpassungen des AB151-03 in Zusammenhang mit Gem 'devise' und Bootstrap bringen

## Selbstreflexion

Folgende Fragen müssen bei der Selbstreflexion beantwortet werden:

- Was habe ich gelernt?
- 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?

## Lösungen der Aufgaben

Das Dokument muss die Lösungen der wesentlichen Aufgaben beinhalten. Weglassen können Sie Fragen mit kurzer Antwort, z.B. "welche CSS-Klasse sorgt dafür, dass...".

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!

## Abschliessende Reflexion über das Gelernte:

Schreiben Sie eine abschliessende, zusammenfassende Reflexion über den Lerninhalt (Fazit). Beschreiben Sie ebenfalls den 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. 2 bis max. 4 Sätze.

## 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:

- Login-View am Ende des Arbeitsblattes
- Registration-View am Ende des Arbeitsblattes
- Beide Views mit Dummy-Phone und Footer

Sie bauen die Screenshots am besten in die Zusammenfassung ein.

Nach der Abgabe der jeweiligen Quicknotes kommt die Lehrperson bei jedem Lernenden vorbei und schaut sich den Projektfortschritt an. Dies fließt in die Bewertung mit 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").