smartlearn
virtual learning. real success.

MODUL 133

TEIL 2: STATISCHE SEITE MIT RAILS ERSTELLEN, EDITIEREN UND MIT TWITTER-BOOTSTRAP DESIGNEN

Ralph Maurer / Michael Abplanalp

Inhaltsverzeichnis AB133-02

Modul 133: Webapplikationen mit Session-Handling realisieren

Teil 2: Statische Seite mit Rails erstellen, editieren und mit Twitter-Bootstrap designen

Rails Server starten und stoppen

Der Rails-Server kann auf zwei Arten gestartet und gestoppt werden:

- Entweder mit einem Script, das automatisch beim Systemstart (oder auch von Hand) ausgeführt wird. Wenn Sie die praktische Arbeit im letzten Arbeitsblatt durchgeführt haben, wird der Server auf diese Weise gestartet.
- Stoppen können Sie den Server mit:

```
$ sudo kill $( cat
/home/vmadmin/workspace/myapp/tmp/pids/server.pid )
```

- Am besten verpacken Sie auch den Stopp-Befehl in ein Skript, das Sie bei Bedarf einfach ausführen können.
- Starten Sie den Rails-Server von der Konsole aus:

```
$ workspace/myapp$ rails server
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.11.2 (ruby 2.4.2-p198), codename: Love Song
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

  Wie Sie nach dem Starten des Rails-Server sehen können, wird dieser mit ctrl-C wieder gestoppt.

Vergewissern Sie sich, dass der Rails-Server auf der bmLP1 gestartet worden ist. Geben Sie darauf im Browser http://localhost:3000 ein, Sie sollten die Startseite von Rails erhalten.

Beenden Sie nun den Server wieder, mit kill oder mit ctrl-C.

Eine statische Seite mit Rails erstellen

HTML5 und CSS3 sind grundsätzlich Voraussetzung für dieses Modul und werden deshalb nicht näher behandelt. Es werden immer wieder Elemente aus HTML5 und CSS3 verwendet, die notwendigen Erklärungen finden Sie unter https://wiki.selfhtml.org/

Eine Rails-App hat immer genau die gleiche Verzeichnisstruktur, diese kann nicht verändert werden:

```
|-- app
|   |-- assets
|   |   |-- images
|   |   |-- javascripts
|   |   |   `-- application.js
|   |   `-- stylesheets
|   |       `-- application.css
|   |-- controllers
|   |   |-- application_controller.rb
|   |   `-- concerns
|   |-- helpers
|   |   `-- application_helper.rb
|   |-- mailers
|   |-- models
|   |   `-- concerns
|   `-- views
|       `-- layouts
|           `-- application.html.erb
.... →
```

```
|-- bin
|   |-- bundle
|   |-- rails
|   |-- rake
|   |-- setup
|   `-- spring
|-- config
|   |-- application.rb
|   |-- boot.rb
|   |-- database.yml
|   |-- environment.rb
|   |-- environments
|   |   |-- development.rb
|   |   |-- production.rb
|   |   `-- test.rb
|   |-- initializers
|   |   |-- assets.rb
|   |   |-- backtrace_silencers.rb
|   |   |-- cookies_serializer.rb
... →
```

```
|   |   |-- filter_parameter_logging.rb
|   |   |-- inflections.rb
|   |   |-- mime_types.rb
|   |   |-- session_store.rb
|   |   `-- wrap_parameters.rb
|   |-- locales
|   |   `-- en.yml
|   |-- routes.rb
|   `-- secrets.yml
|-- config.ru
|-- db
|   `-- seeds.rb
|-- Gemfile
|-- Gemfile.lock
|-- lib
|   |-- assets
|   `-- tasks
|-- log
… →
```

```
|-- public
|   |-- 404.html
|   |-- 422.html
|   |-- 500.html
|   |-- favicon.ico
|   `-- robots.txt
|-- Rakefile
|-- README.rdoc
|-- test
|   |-- controllers
|   |-- fixtures
|   |-- helpers
|   |-- integration
|   |-- mailers
|   |-- models
|   `-- test_helper.rb
|-- tmp
|   `-- cache
|       `-- assets
`-- vendor
```

Wir werden die Rails-Verzeichnisstruktur später genauer analysieren und begnügen uns jetzt lediglich mit deren Anwendung.

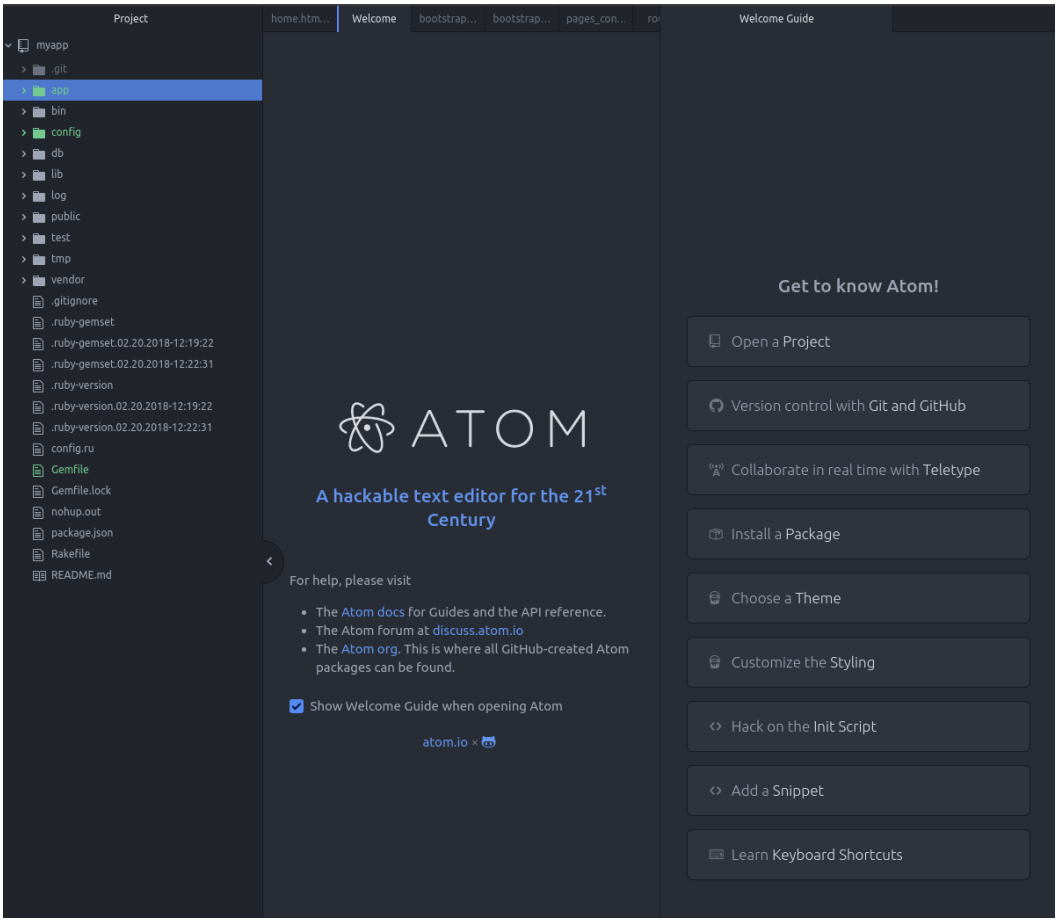
Wir wollen in unserer Applikation eine Seite, die wir *home* nennen, erstellen. Wechseln Sie zuerst ins Verzeichnis `myapp` und machen Sie folgende Kommandozeilen-eingabe:

```
~/workspace/myapp$ rails generate controller pages home
```

Nun starten wir die Projektverwaltung mit atom.io:

```
~/workspace/myapp$ atom .
```

Atom öffnet sich und sieht in etwa so aus:



Sie sehen ganz links die Rails-Verzeichnisstruktur, dann ein Welcome-Fenster und einen Welcome-Guide. Die Tabs der letzteren zwei können Sie schliessen.

pages_controller.rb

Die Datei `pages_controller.rb` wurde soeben im Verzeichnis `app/controllers` erstellt. Die Endung `.rb` steht für eine Ruby-Datei. Controller sind die logische Einheit oder das Herzstück einer Rails-App. Folgende Aufgaben werden von Rails-Controller erledigt (wir gehen später noch genauer darauf ein):

› Controller sind verantwortlich für das Weiterleiten externer Anfragen an interne Aktionen.
› Controller verwalten im Hintergrund das Caching (Zwischenspeichern) von Sessions, damit die Rails-App maximal performt.
› Controller verwalten Hilfsmodule (Gem), die die Funktionen der Webansicht (View) erweitern, ohne ihren Code zu vergrössern.
› Controller verwalten Sitzungen und erlauben eine fortlaufende Interaktion mit Anwendungen.

Betrachten wir nun den Inhalt von `pages_controller.rb`:

```
class PagesController < ApplicationController
  def home
  end
end
```

`PagesController` ist eine in Ruby definierte Klasse und beinhaltet u.a. die Funktionen des Application-Controllers (`PagesController` erbt vom `ApplicationController`).

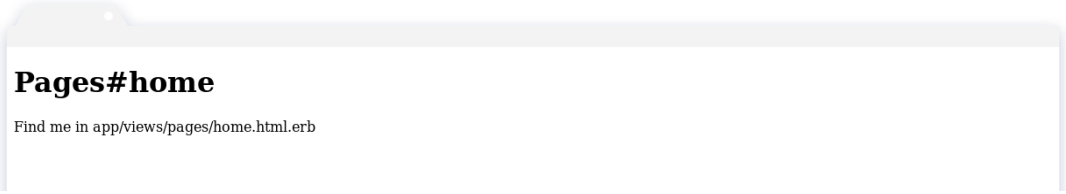
Wichtig ist, dass in der Klasse `PagesController` eine Methode definiert ist mit Namen `home`. Die Bezeichnung `home` stammt von unserem Kommando, mit dem wir eine neue Seite erstellt haben:

```
~/workspace/myapp$ rails generate controller pages home
```

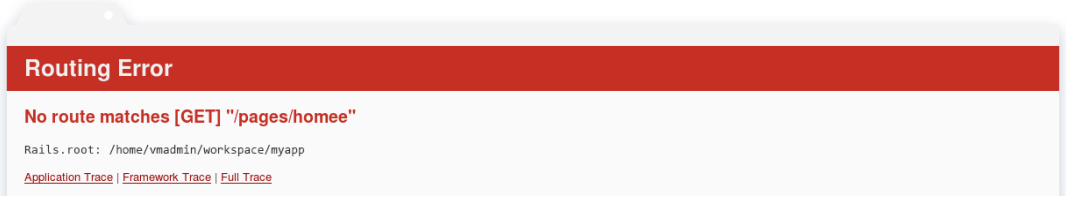
`def home` macht nichts anderes, als eine Anfrage durch den Benutzer weiterleiten. Starten Sie den Rails-Server wieder:

```
~/workspace/myapp$ rails server
```

Öffnen Sie http://localhost:3000/pages/home und Sie gelangen auf folgende View:



Geben Sie die URL falsch ein, z.B. http://localhost:3000/pages/homee, erhalten eine Fehlermeldung:



`PagesController < ApplicationController` beinhaltet alle Webaufrufe. Jede dort definierte Methode ist eine Aktion, die auf eine View leitet, genauer auf `view/pages/*`. In unserem Beispiel ist das auf `view/pages/home.html.erb`.

Wenn die Datei `home.html.erb` in Atom verändert und abgespeichert wird, muss der Rails-Server nicht neu gestartet werden.

Löschen Sie den Inhalt von `home.html.erb` und schreiben Sie in die Datei:

```
<h1>Welcome! I am a rails programmer!</h1>
```

routes.rb

Als nächstes wollen wir `home.html.erb` als Startseite von http://localhost:3000 definieren. Dazu müssen wir das Routing verändern, dies macht man in `config/routes.rb`. Ändern Sie den Quelltext wie folgt ab:

```
Rails.application.routes.draw do
  # get 'pages/home'
  root 'pages#home'
  # For details on the DSL available within this file, see …
End
```

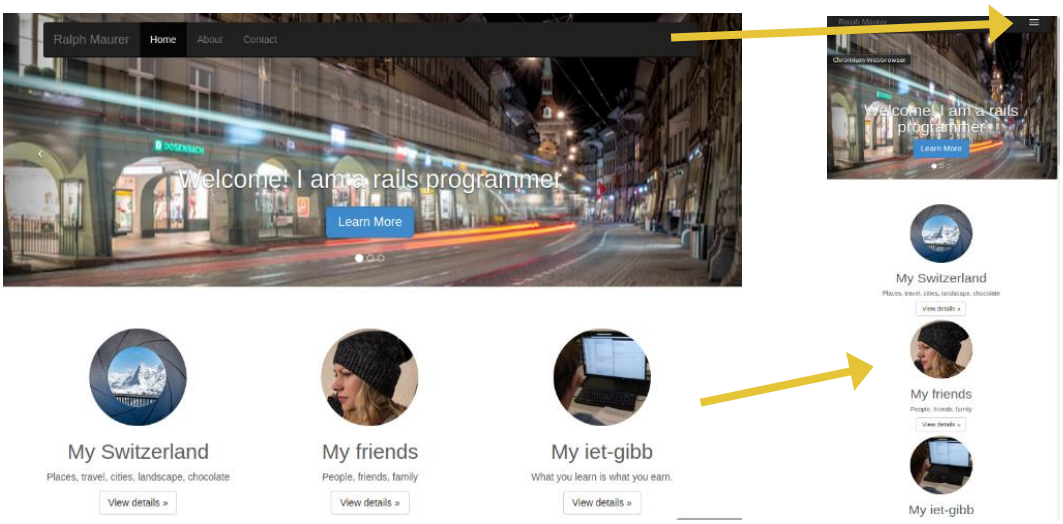
Die Root-Page (Startseite) ist nun auf `pages#home` gesetzt. Mit `# get 'pages/home'` ist das ursprüngliche Routing unserer Seite deaktiviert resp. auskommentiert (der Eintrag `get 'pages/home'` wurde beim Erstellen der Seite automatisch generiert).

Der Rails-Router ermittelt anhand der URL die Controller-Aktion. In unserem Fall wird die Controller-Aktion `def home` der Root-Page zugewiesen:

```
class PagesController < ApplicationController
  def home
  end
end
```

Bootstrap 4 und jQuery der Applikation myapp hinzufügen

Neben Airbnb, GitHub, Shopify ist auch Twitter in Rails geschrieben. Mit Bootstrap bietet Twitter ein sensationelles Frontend-CSS-Framework an, welches sämtliche Designkomponenten aus Twitter zur Verfügung stellt. Bootstrap ist responsive und lässt sich sehr gut in Rails integrieren. Responsive bedeutet, dass Bootstrap Medien (bzw. verschiedene Bildschirmauflösungen) zur Wiedergabe von Webseiten erkennt und sich diesen anpasst. So werden normale Menüs mit Menüeinträgen nebeneinander auf einem Handy als Dropdown-Menü dargestellt. Oder nebeneinander wird untereinander wie im folgenden Beispiel:



Weil Bootstrap 4 jQuery für die verschiedenen interaktiven Elemente (Tooltips, Popups, Modals) verwendet, müssen wir einen Weg finden, um jQuery zu installieren und mit Webpack zu kompilieren. In Rails 5 wurde dies mit Bootstrap- und jQuery-Gems gelöst. In Rails 6 wird Yarn benutzt, um die Javascript Node Module zu installieren und Webpack, um die Module auszuführen.

Führen Sie folgende Schritte aus:

1. Wechseln Sie ins Applikationsverzeichnis

```
$ cd ~/workspace/myapp
```

2. Installieren Sie Bootstrap, jQuery und Popper mit Yarn:

```
$ yarn add bootstrap jquery popper.js
```

3. Damit jQuery nicht in jedes Webpack integriert werden muss, führen wir eine Konfiguration durch, die diese Aufgabe übernimmt.

   Fügen Sie folgenden Code in `config/webpack/environment.js` ein (erste und letzte Zeile sind bereits vorhanden):

```
const { environment } = require('@rails/webpacker')

const webpack = require('webpack')
environment.plugins.append('Provide',
  new webpack.ProvidePlugin({
    $: 'jquery',
    jQuery: 'jquery',
    Popper: ['popper.js', 'default']
  })
)

module.exports = environment
```

4. Öffnen Sie die Datei `app/javascript/packs/application.js` und importieren unterhalb der `require`-Anweisungen Bootstrap und zudem addEventListener für Tooltips und Popovers in Bootstrap:

```
import "bootstrap";

document.addEventListener("turbolinks:load", () => {
  $('[data-toggle="tooltip"]').tooltip()
  $('[data-toggle="popover"]').popover()
})
```

5. Anstelle von CSS wollen wir das komfortablere SCSS verwenden. Benennen Sie in Atom die Datei `app/assets/stylesheets/application.css` zu `application.scss` um.
6. Fügen Sie in die Datei `app/assets/stylesheets/application.scss` die folgende Import-Anweisung ein (den Rest der Datei können Sie löschen):

```
@import "bootstrap/scss/bootstrap";
```

● ● ● Die Installationsanleitung oben wurde übernommen von https://www.timdisab.com/installing-bootstrap-4-on-rails-6/ und https://blog.capsens.eu/how-to-write-javascript-in-rails-6-webpacker-yarn-and-sprockets-cdf990387463

Dort finden Sie zusätzliche Informationen zu Javascript in Rails 6, Webpacker, Yarn und Sprockets.

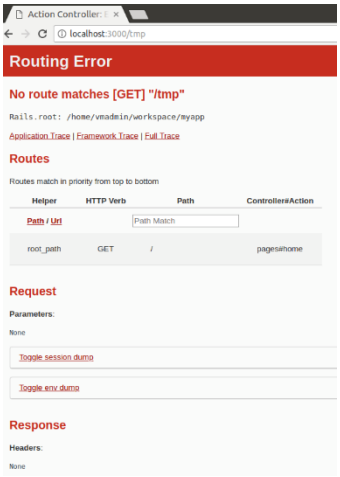
Berechtigung für temporäres Verzeichnis setzen

Bootstrap schreibt und löscht temporäre Dateien in den Ordner `myapp/tmp`, der Ordner ist eine Art Cache. Damit alles im Entwicklungsmodus funktioniert, müssen Sie dem Ordner Vollberechtigungen geben:

```
~/workspace/myapp$ sudo chmod 777 -R tmp
```

Nur so funktioniert Bootstrap wunschgemäss mit allen zur Verfügung gestellten Elementen.

Da keine explizite Route auf http://localhost:3000/tmp referenziert, hat diese Vollberechtigung auch keine sicherheitsrelevanten Konsequenzen. Beim Deployment auf einen öffentlichen Server müsste man den Zugriff von Bootstrap auf den `tmp`-Ordner sicher regeln.



Bootstrap-Template Carousel verwenden

Als nächstes beschaffen wir uns ein frei verfügbares Bootstrap-Template. Es gibt dafür viele entsprechende Anbieter. Das Template, das wir verwenden wollen, stammt von https://getbootstrap.com/docs/4.0/examples/ und heisst Carousel (https://getbootstrap.com/docs/4.0/examples/carousel/). Das Template wurde für das Modul 133 leicht angepasst, z.B. wollen wir die Bilder auf unserem Server speichern.

Zur Vereinfachung verwenden wir die bestehenden Dateien und fügen den Inhalt der Carousel-Dateien ein:

› Löschen Sie den Inhalt der Datei `app/views/pages/home.html.erb`. Fügen Sie darauf den Inhalt der Datei `AB133-02_carousel_layout.html` in diese Datei ein (copy&paste).
› Fügen Sie den Inhalt der Datei `AB133-02_carousel_layout.css` in die Datei `app/assets/stylesheets/application.scss`. ein (copy&paste).

Die Dateien `AB133-02_carousel_layout.html` und `AB133-02_carousel_layout.css` finden Sie bei den Arbeitsblättern im Ordner `AB133-02`.

Kopieren Sie die Dateien `400X400.png` und `placeholder.png` vom Ordner `AB133-02/images` (Modulshare) in den Ordner `app/assets/images` ihrer Applikation (bmLP1).

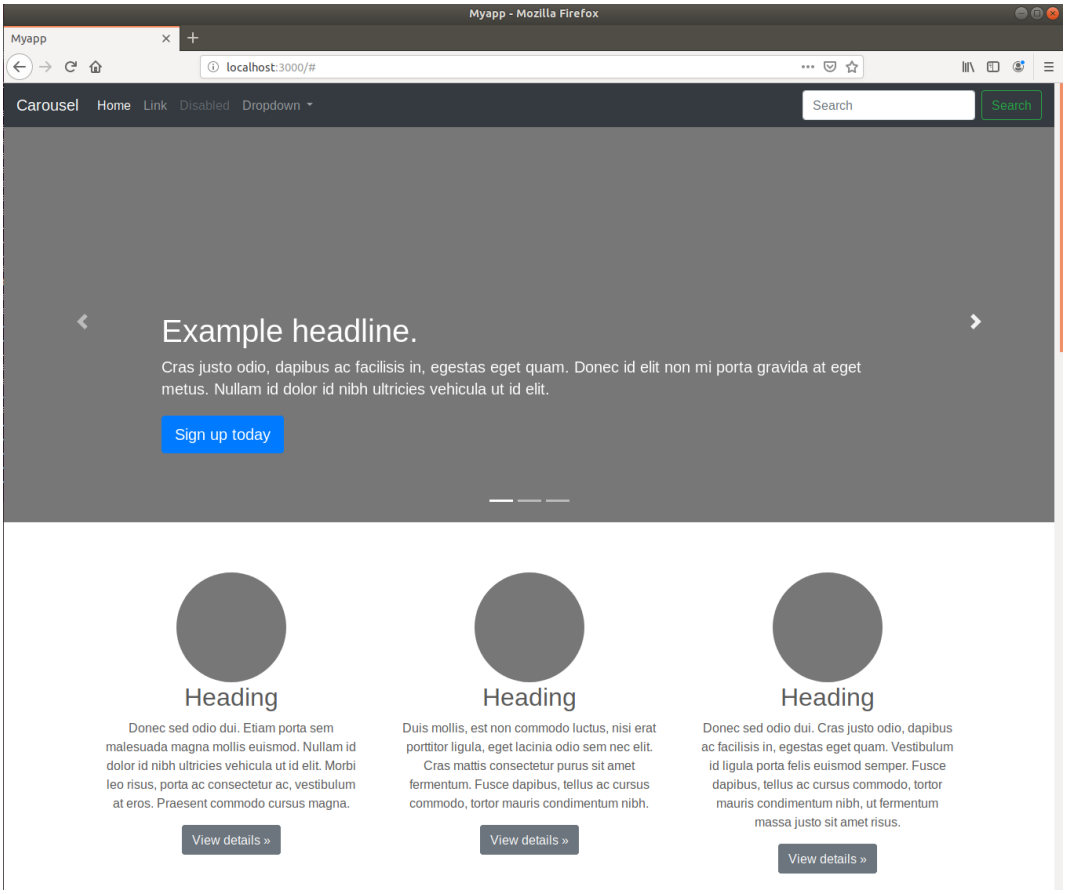
Da wir die Assets (css) unserer myapp verändert haben, müssen wir die Rails-App neu starten:

```
$ sudo kill $( cat /home/vmadmin/workspace/myapp/tmp/pids/server.pid )
```

bzw. ctrl-C

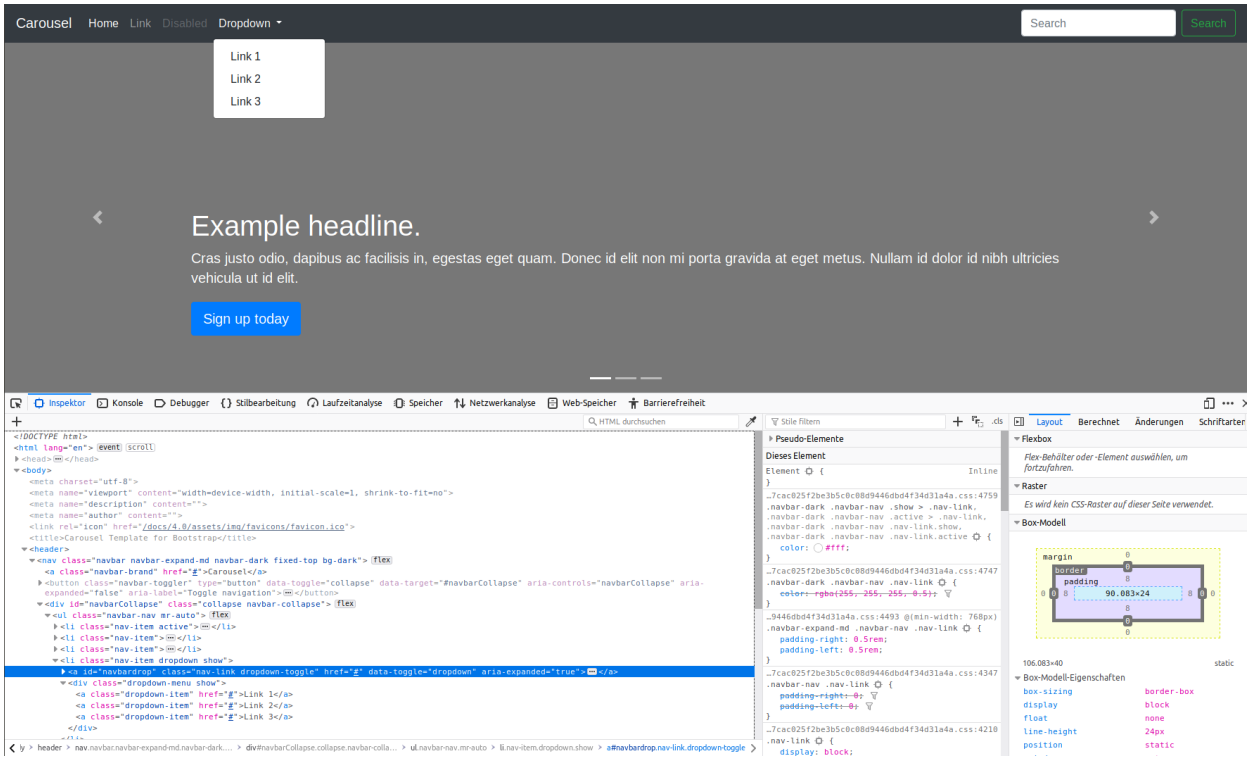
```
$ rails server
```

Öffnen Sie nun http://localhost:3000 und schauen Sie sich das leere Template an:



Bootstrap-Template Carousel moderat verändern

Um sich im HTML Code zurecht zu finden und die einzelnen Elemente zu lokalisieren, können Sie mit <F12> die Entwicklertools anzeigen lassen. Wechseln Sie in Firefox in den Tab *Inspector*. und in Chrome in den Tab *Elements*. Sie können beispielsweise mit dem Mauszeiger auf der Webseite über den Menüpunkt *Dropdown* fahren, die rechte Maustaste betätigen und im Popup-Menü den Eintrag *Element untersuchen* (Firefox) bzw. *Untersuchen* (Chrome) auswählen. Die Markierung springt an die Stelle im HTML-Code, wo der Code für den Menüpunkt *Dropdown* steht:



Wir wollen nun dieses Dropdown-Menü entfernen. Hierzu merken wir uns den markierten Code, wechseln in die Datei `home.html.erb` im Atom und suchen diesen Code. Wenn Sie ihn gefunden haben, löschen sie das Dropdown-Menü inklusive Untermenü:

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbardrop"
                                        data-toggle="dropdown">
    Dropdown
  </a>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="#">Link 1</a>
    <a class="dropdown-item" href="#">Link 2</a>
    <a class="dropdown-item" href="#">Link 3</a>
  </div>
</li>
```

Sie können den Code direkt in den Entwicklertools löschen, aber dies ist dann nicht dauerhaft. Es wird lediglich temporär die Browserview verändert.

Praktische Arbeit: Seite anpassen und neue Seiten erstellen

Verschiedene Bootstrap-Elemente ausprobieren

Ziel dieses Kapitels ist es, Bootstrap besser kennenzulernen und mit Bootstrap ein paar Webseiten zu gestalten. Bei allen weiteren Arbeitsblättern sind die Seiten mit Bootstrap gestaltet. **Obwohl der Einsatz von Bootstrap nicht obligatorisch ist, wird dieser doch sehr empfohlen!**

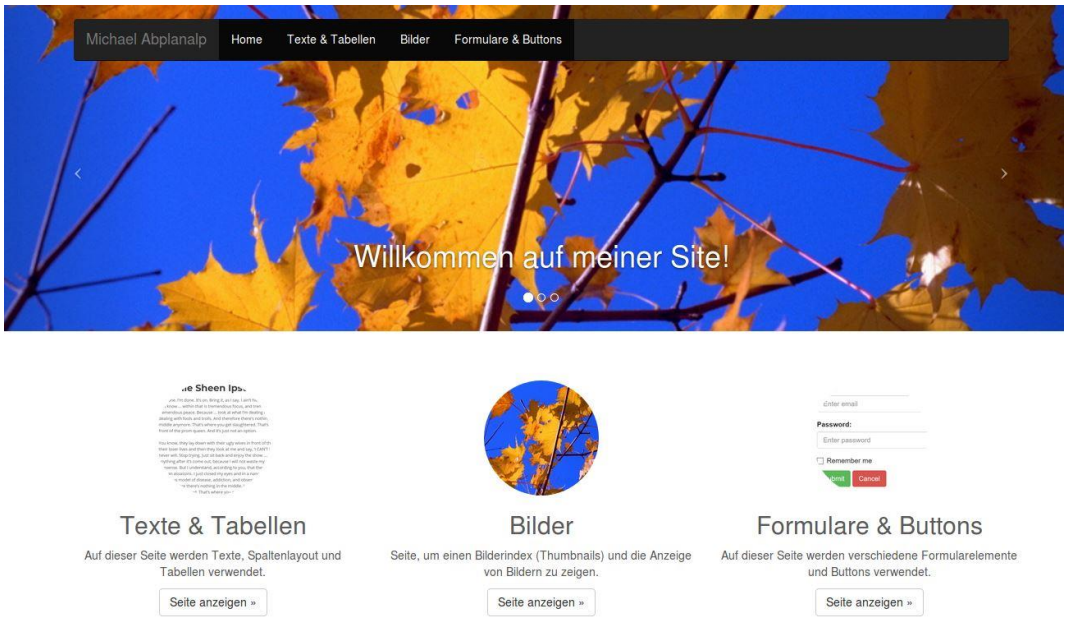
Es gibt viele Bootstrap-Tutorials. Eines der besten ist auf w3schools zu finden:

https://www.w3schools.com/bootstrap4/

Wir arbeiten mit Bootstrap Version 4. Arbeiten Sie die Kapitel *Bootstrap 4 Tutorial* und *Bootstrap 4 Grid* durch. Erstellen Sie parallel zum Tutorial Webseiten in Ihrer Rails-Applikation, gemäss den Vorgaben unten.

Hauptseite erstellen

Erstellen Sie eine Hauptseite, die drei Unterseiten erhalten soll. Sie können das Carousel-Template verwenden, das wir weiter vorhin erstellt haben. Führen Sie die notwendigen Anpassungen durch, damit die Startseite in etwa so (bzw. nach Ihrem Geschmack) aussieht:



Die Elemente auf der Seite:

- Menü mit vier Einträgen: *Home* (diese Seite), *Texte & Tabellen*, *Bilder* und *Formulare & Buttons*. Das Menü soll auf allen vier Seiten erscheinen.
- Hintergrundbilder im Kopfbereich, die mit der Carousel-Vorlage wechseln.
- Drei Bilder mit kurzem Text und Link auf die entsprechende Seite.

› Verwenden Sie eigene Bilder und Texte.

Die Startseite *Home* haben wir bereits generiert mit dem Befehl
`$ rails g controller pages home`

Auf den Seiten 8-10 haben wir die Seite bearbeitet, passen Sie diese nun Ihren Bedürfnissen an.

Seite «Texte & Tabellen» erstellen

Um eine neue Seite mit dem Namen *texte* zu erstellen, gehen Sie wie folgt vor:

1. Methode im Pages-Controller erstellen, Datei `app/controllers/pages_controller.rb`:

```
def texte
end
```

   Es handelt sich um eine leere Methode, d.h. die Methode führt keine Aktionen durch. Am Ende wird die View `texte.html.erb` gerendert (siehe nächsten Punkt).

2. Erstellen Sie mit Atom im Verzeichnis `app/views/pages` die Datei `texte.html.erb`. Die Datei (ohne Endung) muss exakt gleich heissen wie die Methode aus Punkt 1.

3. In der Datei `config/routes.rb` muss die Route definiert werden, damit die Seite via URL aufgerufen werden kann. Erstellen Sie in der Datei folgenden Eintrag:

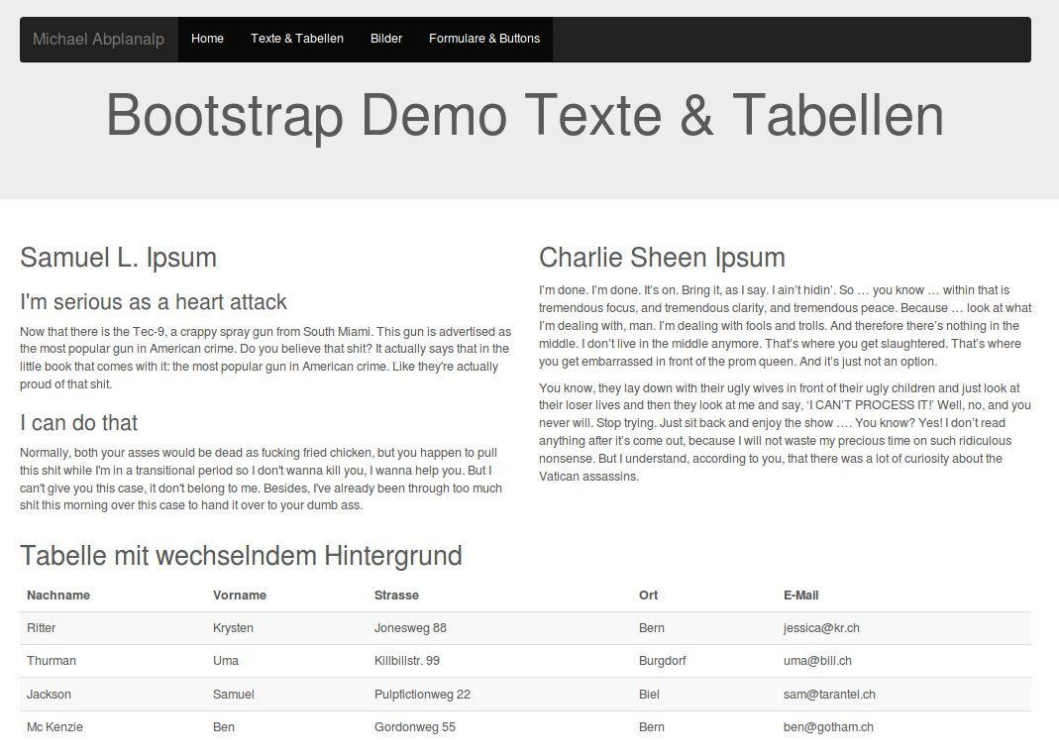
```
get '/pages/texte'
```

   `pages` ist der Name des Controllers, den wir auf Seite 5 erstellt haben. `texte` ist wiederum der Name der Methode aus Punkt 1.

4. Als Letztes muss auf der Startseite in der Bootstrap-Navbar der Link auf die Seite korrekt erfasst werden. Ändern Sie den entsprechenden Eintrag in `app/view/pages/home.html.erb` wie folgt:

```
<a class="nav-link" href="/pages/texte">Texte&Tabellen</a>
```

Wenn Sie nun http://localhost:3000/pages/form im Browser eingeben, sollte eine leere Seite angezeigt werden (und keine Fehlermeldung!). Erstellen Sie die Seite mit Bootstrap, sie sollte am Ende in etwa wie folgt aussehen:

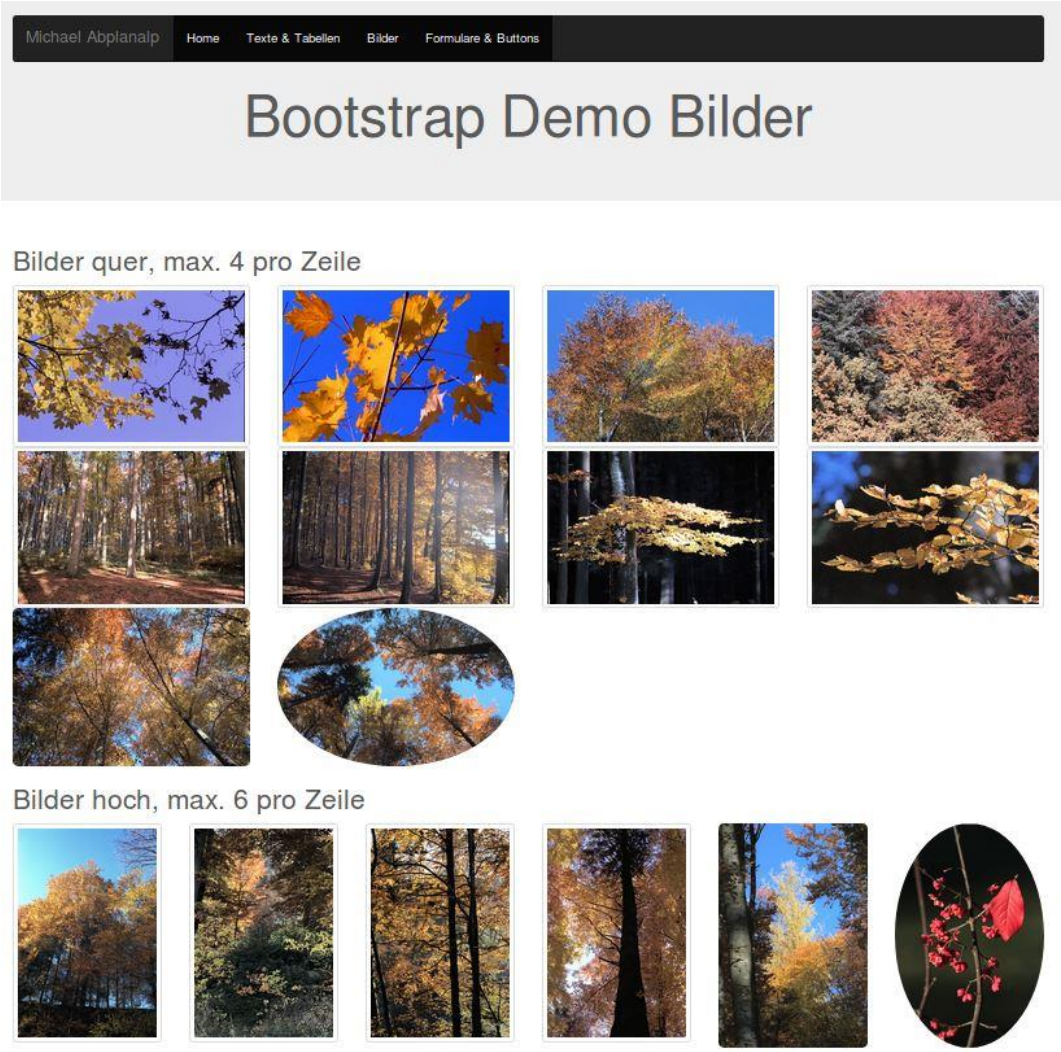


Die Elemente auf der Seite:

› Ein Kopfbereich mit dem bekannten Menü.
› Ein Textbereich mit mind. 2 Spalten, verwenden Sie dazu das Grid-System.
› Setzen Sie auch Titel ein.
› Wählen Sie eigene Texte aus.
› Eine Tabelle mit mind. fünf Spalten.
› Wählen Sie eine Tabelle, in der Sie persönliche Daten einfüllen können.

Seite «Bilder» erstellen

Führen Sie wiederum die vier Punkte aus dem vorherigen Kapitel aus, um die neue Seite *bilder* zu erstellen. Erstellen Sie die Seite mit Bootstrap, sie sollte am Ende in etwa wie folgt aussehen:



Die Elemente auf der Seite:

› Ein Kopfbereich mit dem bekannten Menü.
› Platzieren Sie mind. 10 verschiedene Bilder auf der Seite.
› Zeigen Sie die Bilder als Thumbnails an.
› Beim Klicken auf ein Bild wird dieses im Browser gross angezeigt.
› Probieren Sie das Grid-System aus: Verschiedene Anzahl Bilder pro Zeile.
`<div class=«col-xs-6 col-md-4 col-lg-2»>`
bedeutet z.B., dass bei kleinem Display ein Bild 6 Spalten beansprucht (= 2 Bilder pro Zeile), bei mittlerem Display 3 Spalten (= 4 Bilder pro Zeile) und bei grossem Display 2 Spalten (= 6 Bilder pro Zeile).
› Probieren Sie verschiedene Formen für die Bilder aus.

Verkleinern Sie das Browserfenster, um zu sehen, wie sich die geänderte Auflösung auf die Darstellung der Thumbnails auswirkt.

Nachfolgend sehen Sie zum Beispiel, dass sich die Anzahl Bilder bei schmalem Browserfenster pro Zeile um die Hälfte reduziert hat:

✈ Zusatzaufgabe für schnelle Lernende.

Zusatzaufgabe 1

Setzen Sie Lightbox für eine verbesserte Bildanzeige ein. Sie können Lightbox entweder selbst programmieren oder eine Open Source Lösung verwenden.

Seite «Formulare & Buttons» erstellen

Führen Sie wiederum die vier Punkte aus dem Kapitel *Seite «Texte & Tabellen» erstellen* aus, um die neue Seite *formulare* zu erstellen. Erstellen Sie die Seite mit Bootstrap, sie sollte am Ende in etwa wie folgt aussehen:

Die Elemente auf der Seite:

- Ein Kopfbereich mit dem bekannten Menü.
- Erstellen Sie ein Formular mit mind. folgenden Elementen:
  - Einzeiliges Textfeld
  - Mehrzeiliges Textfeld
  - Mind. 2 Checkboxes
  - Optionsgruppe mit mind. 2 Optionen
  - Drop-Down Liste
- Erstellen Sie folgende Buttons mit verschiedenen Bootstrap-Klassen (Farben):
  - Button 1 mit Link auf Homepage
  - Button 2 mit Link auf Seite «Texte & Tabellen»
  - Button 3 mit Link auf Seite «Bilder»
  - Button zum Senden des Formulars
  - Button zum Zurücksetzen des Formulars

Zusatzaufgabe 2

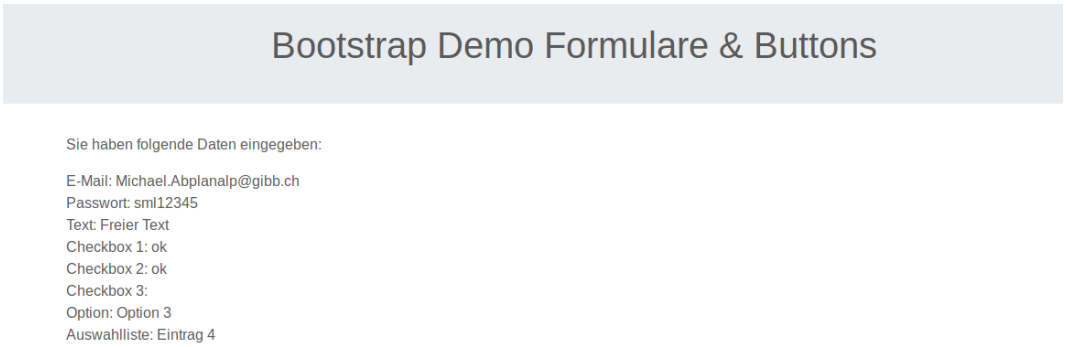
✈ Zusatzaufgabe für schnelle Lernende.

Erstellen Sie einen weiteren Button. Beim Klick darauf soll ein modales Fenster mit einem kurzen Text erscheinen.

Zusatzauftrag 3: Erfolgsmeldung nach dem Senden des Formulars

Mit einer Meldung wollen wir zeigen, dass das Formular tatsächlich gesendet worden ist:
Es sollen die Eingaben des Benutzers aufgelistet werden.

✈ Zusatzaufgabe für schnelle Lernende.



Beachten Sie dabei Folgendes:

- Führen Sie wiederum die vier Punkte aus dem Kapitel *Seite «Texte & Tabellen» erstellen* aus
- In `routes.rb` muss der Eintrag mit der Methode `post` anstatt `get` erstellt werden (z.B. `post 'pages/success'`).
- Aus Sicherheitsgründen muss in Rails bei der Übermittlung von Formularen ein CSRF-Token mitgegeben werden (Schutz vor Cross-Site-Request-Forgery). Damit wir das Token nicht einfügen müssen, können wir den CSRF-Schutz vorläufig ausschalten.
  Tragen Sie in der Datei `application_controller.rb` folgendes ein:
  `skip_before_action :verify_authenticity_token`
- Nun können Sie für das Formular die Methode *post* verwenden.
- Anstatt die Eingaben des Formulars können Sie als einfache Variante eine Erfolgsmeldung anzeigen ("Das Formular wurde erfolgreich übermittelt!" oder ähnlich).

Zügeln von Header und Menü in das Applikations-Template

Es gibt in Rails-Applikationen jeweils Dateien mit dem Namen `application.xy`, die für alle Seiten, Controller und Views zur Anwendung kommen. Nachfolgend sind die Wichtigsten aufgeführt.

app/javascript/application.js

Dieser JavaScript-Code kann auf allen Seiten verwendet werden.

app/assets/stylesheets/application.scss

Tag-Selektoren haben Auswirkung auf alle Seiten (wenn z.B. die Eigenschaften von `table` geändert werden). Und es können eigene Klassen definiert werden, die auf allen Seiten verwendet werden können.

app/controllers/application_controller.rb

Im Application-Controller können wir Funktionen ausführen und Instanzvariablen generieren, die für alle Seiten gültig sind.

app/views/layouts/application.html.erb

Das Template legt die Grundstruktur aller Seiten der Applikation fest und wird als Erstes gerendert. Zu Beginn sieht die Datei wie folgt aus:

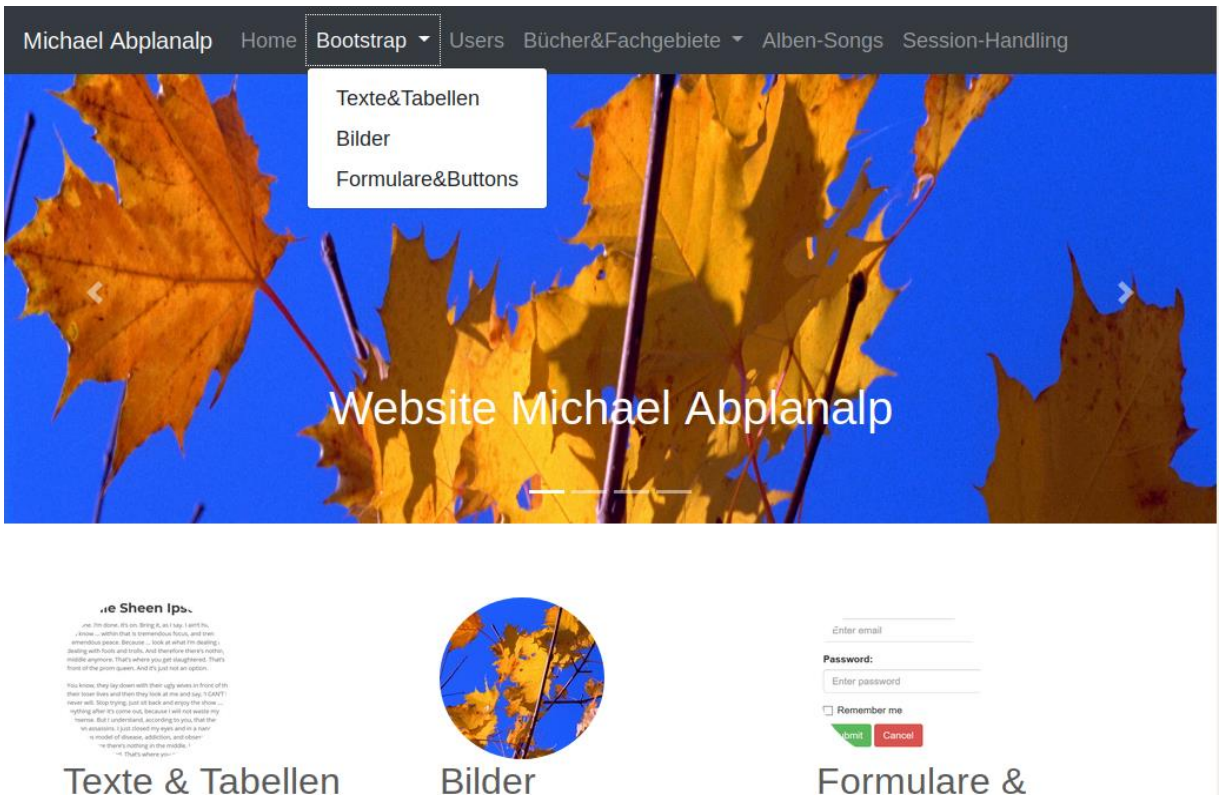
```
<!DOCTYPE html>
<html>
  <head>
    <title>myapp</title>
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>

    <%= stylesheet_link_tag    'application', media: 'all',
                               'data-turbolinks-track': 'reload' %>
    <%= javascript_pack_tag 'application',
                               'data-turbolinks-track': 'reload' %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

Die Elemente `<!DOCTYPE html>`, `<html>`, `<head>` und `<body>` werden also hier festgelegt und sollten deshalb aus allen anderen Templates entfernt werden!

Denn eine Seite wird wie folgt aufgebaut: Es wird der Code von `application.html.erb` bis zu `<%= yield %>` gerendert. Anstelle von `<%= yield %>` wird die entsprechende View eingesetzt, z.B. `pages#home`. Danach werden noch der Rest von `application.html.erb` abgearbeitet.

Im Verlauf des Moduls werden wir die Applikation stetig erweitern mit statischen und dynamischen Seiten. Wir wollen auf allen Seiten stets dasselbe Menü (Bootstrap-Navbar) anzeigen. Deshalb zügeln wir das Menü in `application.html.erb`, und bei späteren Änderungen müssen wir diese an nur einer Stellt durchführen.

Am Ende könnte unsere Startseite mit dem Menü wie folgt aussehen:



Gehen Sie für die Anpassung von `application.html.erb` wie folgt vor:

- Ergänzen Sie die Datei mit der Headerinformation der Datei `home.html.erb`, die Sie vorhin angepasst haben.
- Löschen Sie darauf den `DOCTYPE`, den gesamten Header und das `<body>`-Tag in der Datei `home.html.erb`.
- Zügeln Sie das gesamte Menü in die Datei `application.html.erb` (alles innerhalb und inklusive `<div class="navbar-wrapper">`-Tag), unterhalb von `<body>` und oberhalb von `<%= yield %>`.