

WEBTECHNOLOGIEN

03 – WEB APP BASICS

PROF. DR. MARKUS HECKNER

THE STORY SO FAR

- HTML und CSS für das Frontend der Website
- Replit als statischer Webserver, der die aufgerufenen Seiten an den Client zurückgibt
- JavaScript als Programmiersprache

Jetzt: Wie lassen sich diese Elemente kombinieren, um dynamische Webseiten zu erzeugen?

AB JETZT: ENTWICKLUNG EINER PLAYLIST APP ALS DEMOPROJEKT FÜR LECTURE UND LABS

Features am Ende:

- Registrierung
- Login, Logout
- Playlists anlegen und löschen
- Songs einer Playlist hinzufügen und löschen
- Ermitteln der Gesamtdauer einer Playlist

The image displays a mockup of a web application for managing playlists. The top section shows a playlist titled "Happy Mood" with a subtitle "Shortest song: Valerie". It contains a table with three songs: Valerie by Amy Winehouse (90s), 22 by Taylor Swift (180s), and Happy by Pharrell Williams (136s). Each song has edit and delete icons. Below the table is a form to add a new song with fields for Title, Artist, and Duration, and an "Add Song" button. The bottom section shows two forms: a "Log-in" form with fields for Email and Password, and a "Signup" form with fields for First name, Last name, Email, and Password. Both forms have a blue button to submit the data.

Playlist 5 Dashboard About

Happy Mood
Shortest song: Valerie

Song	Artist	Duration		
Valerie	Amy Winehouse	90		
22	Taylor Swift	180		
Happy	Pharrell Williams	136		

Title Artist Duration

Add Song

Playlist Signup Login About

Log-in

Email

Password

Login

Signup

First name

Last name

Email

Password

Register

AGENDA

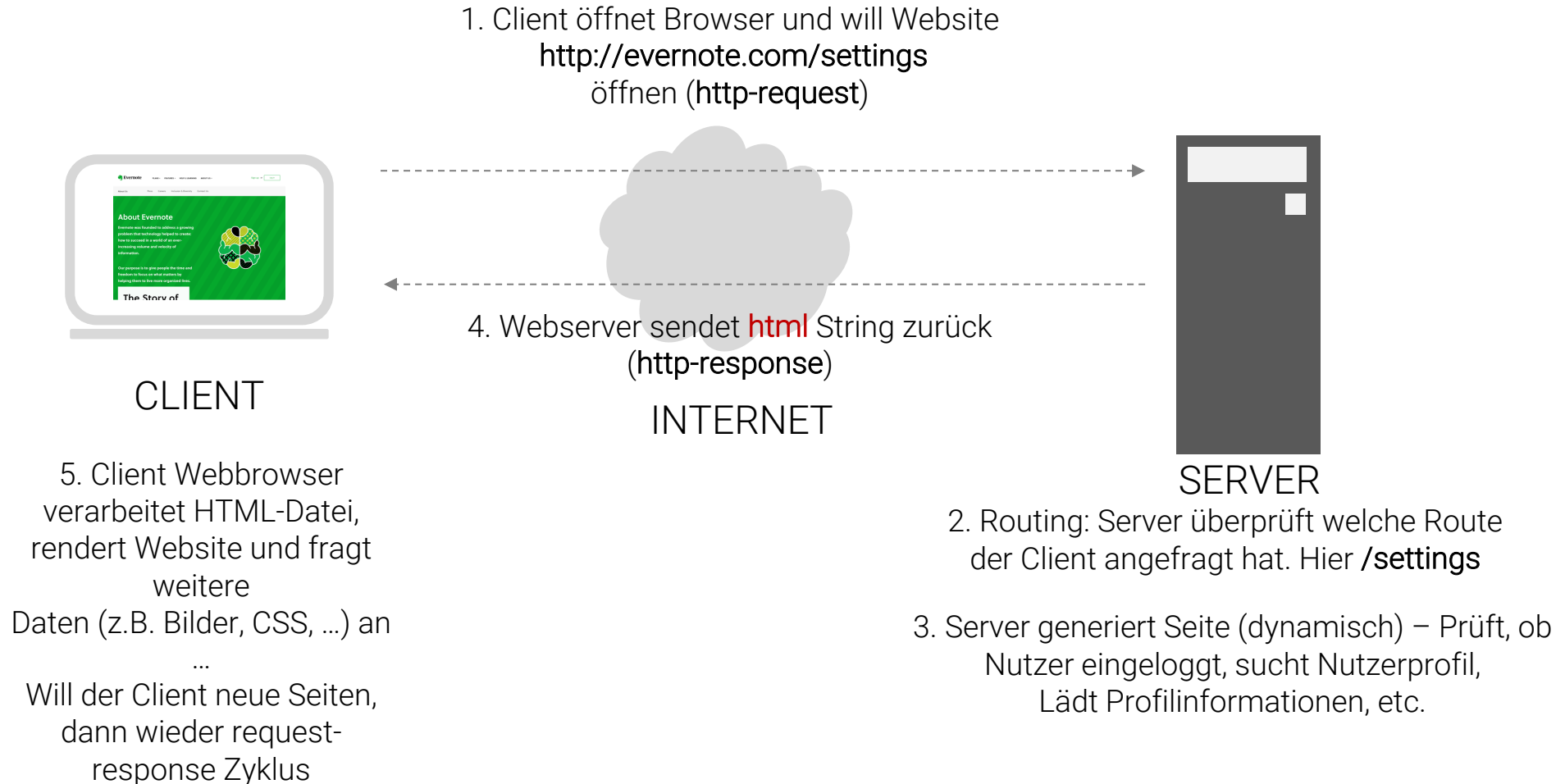
HTTP und serverseitig generierte Webseiten

Walkthrough Web-App Template – Struktur einer Web App

Request – Response Lebenszyklus – Zusammenspiel von View und Controller

Templates und Schleifen

HTTP IST EIN PROTOKOLL UND LEGT FEST WIE CLIENT UND SERVER KOMMUNIZIEREN



HTTP-METHODEN BESTIMMEN DIE ART DER ANFRAGE (= REQUEST) AN DEN SERVER

- GET – Daten vom Server holen
- POST – Erstellen einer Ressource (Daten auf dem Server ablegen) } Nicht heute
- PATCH – Daten auf dem Server aktualisieren
- DELETE – Löschen von Daten auf dem Server } Nicht in diesem Kurs
- ...

AGENDA

HTTP und serverseitig generierte Webseiten

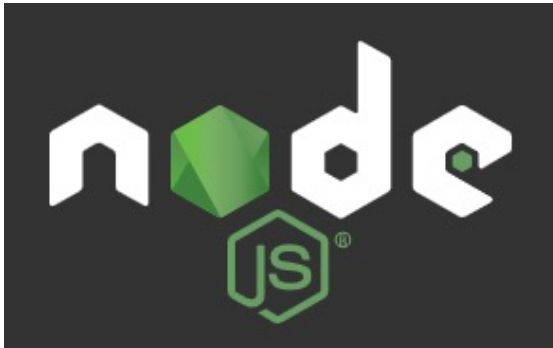
Walkthrough Web-App Template – Struktur einer Web App

Request – Response Lebenszyklus – Zusammenspiel von View und Controller

Templates und Schleifen

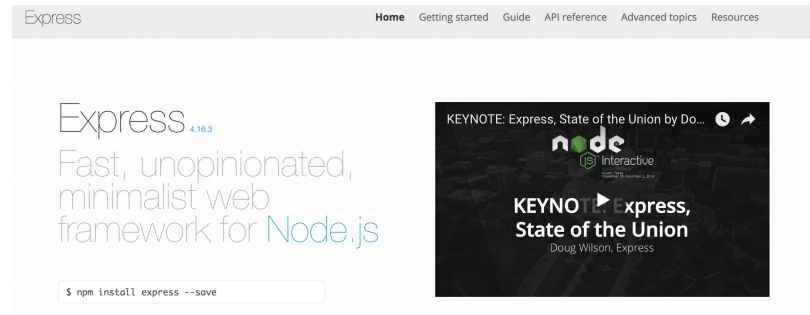
WEB APPS MIT NODE.JS, EXPRESS UND HANDLEBARS

Node.js



- Laufzeitumgebung, mit der man JS Code auf dem Server ausführen kann
- Rudimentäre Unterstützung von http-Kommunikation
- Erweiterbar durch packages (z.B. Express)

Express



- Package für Node.js
- "Fast, unopinionated minimalist web framework for Node.js"
- Baut auf Node.js auf
- Ermöglicht die Erstellung und das Ausliefern von Webapplikationen

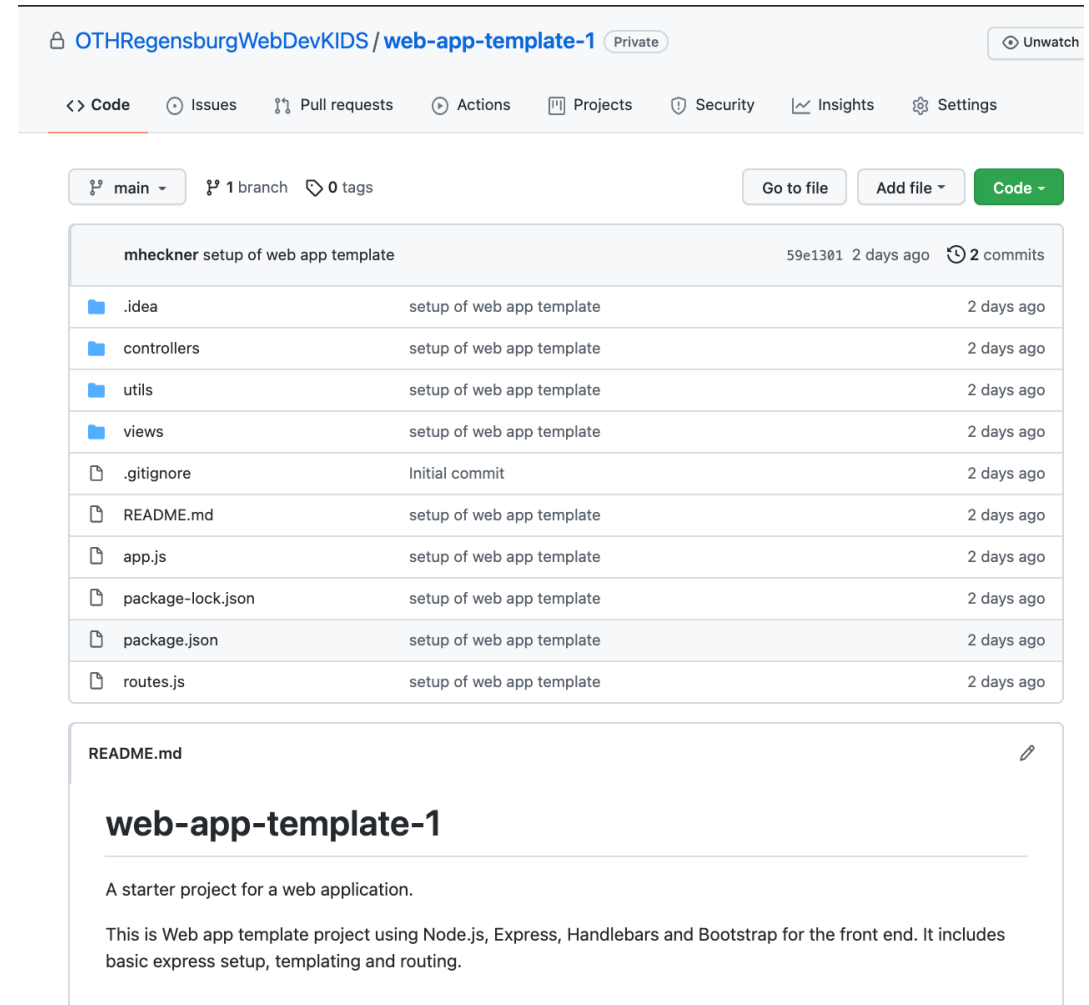
Handlebars



- Package für Node.js
- Dynamische Frontends mit einer Template Engine
- Webseite wird dynamisch aus Daten von Objekten und Arrays erstellt
- Komponenten können in mehreren Seiten verwendet werden

STARTERPROJEKT

- web_app_template_1 enthält enthält alle notwendigen Elemente für den Start der Entwicklung der eigenen Web-App (man muss nicht bei 0 loslegen)
- Auf dieser Basis lassen sich zahlreiche interessante (und komplexe) Projekt verwirklichen
- In replit lassen sich Projekte mit Startercode erstellen (siehe Lab)



OTHRegensburgWebDevKIDS / web-app-template-1 Private Unwatch

<> Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

mheckner setup of web app template 59e1301 2 days ago 2 commits

.idea	setup of web app template	2 days ago
controllers	setup of web app template	2 days ago
utils	setup of web app template	2 days ago
views	setup of web app template	2 days ago
.gitignore	Initial commit	2 days ago
README.md	setup of web app template	2 days ago
app.js	setup of web app template	2 days ago
package-lock.json	setup of web app template	2 days ago
package.json	setup of web app template	2 days ago
routes.js	setup of web app template	2 days ago

README.md

web-app-template-1

A starter project for a web application.

This is Web app template project using Node.js, Express, Handlebars and Bootstrap for the front end. It includes basic express setup, templating and routing.

Template: <https://github.com/OTHRegensburgWebDevKIDS/web-app-template-1>

DAS STARTERPROJEKT VERFÜGT ÜBER 2 SEITEN

/

Web App Template About

Welcome to the Web app template!

Web app template

A web app template.

/about

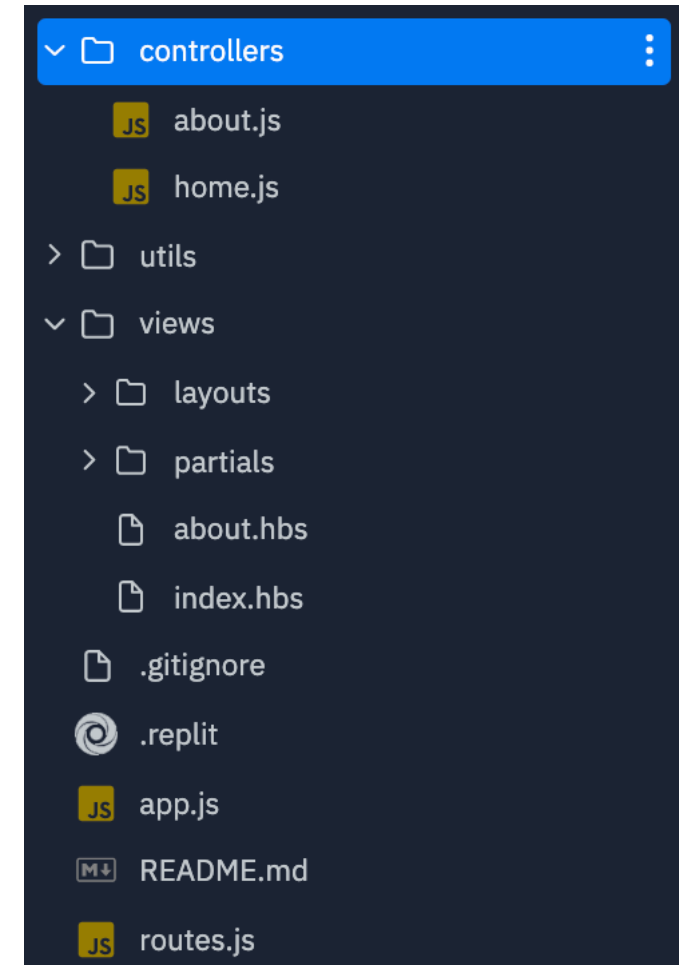
Web App Template About

Web app template

Content goes here...

STRUKTUR DER WEB-APP

- Besteht aus
 - Backend:
 - `app.js` – Startpunkt
 - `routes.js` – Welche URLs unterstützt werden
 - `controllers` – Objekte die requests an die Routes verarbeiten
 - `utils` – Hilfsobjekte (heute Logger, später mehr, z.B. Ermittlung der Dauer einer Playlist)
 - Frontend
 - `views` – Templates zum rendern der Seiten der Web-App



APP.JS IST DIE ZENTRALE KONFIGURATION DER WEB APP

- Importiert node-packages (= Bibliotheken wie express und handlebars)
- Konfiguriert Template-Engine (handlebars)
- Importiert den Router
- Wird über die Kommandozeile gestartet
- Rest erstmal nicht wichtig...

```
app.js x
1  const express = require("express");
2  const logger = require("./utils/logger");
3  const handlebars = require("express-handlebars");
4
5  const dotenv = require("dotenv");
6  dotenv.config();
7
8  const app = express();
9
10 app.engine('.hbs', handlebars.engine({extname: '.hbs'}));
11 app.set('view engine', '.hbs');
12 app.set('views', './views');
13
14 const routes = require("./routes");
15 app.use("/", routes);
16
17 app.listen(process.env.PORT, () => {
18   console.log(`Web App template listening on ${process.env.PORT}`);
19 });
20
21 module.exports = app;
```

AGENDA

HTTP und serverseitig generierte Webseiten

Walkthrough Web-App Template – Struktur einer Web App

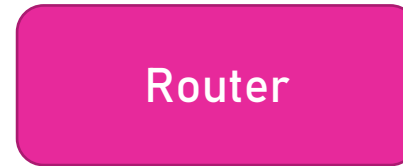
Request – Response Lebenszyklus – Zusammenspiel von View und Controller

Templates und Schleifen

REQUEST – RESPONSE LEBENSZYKLUS

ROUTER => CONTROLLER => VIEW

Request – User klickt
Link auf der Webseite



Überprüft, ob Route
verfügbar und findet
passendes
Controller-Objekt

REQUEST - USER KLICKT LINK AUF DER WEBSEITE



Requests erzeugt durch:

- Attribut href in Links (<a> Tags)
- Attribut href in Buttons
- Attribut action in Formularen (später)

ROUTER – FINDET PASSENDES CONTROLLER-OBJEKT

request landet beim Router



JS app.js
M+ README.md
JS routes.js

```
routes.js x
1  const express = require("express");
2  const router = express.Router();
3
4  const home = require("../controllers/home.js");
5  const about = require("../controllers/about.js");
6
7  router.get("/", home.index);
8  router.get("/about", about.index);
9
10 module.exports = router;
```

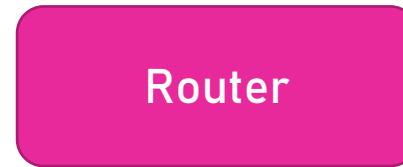
Router importiert zwei Controller, die sich um Verarbeitung der Anfrage kümmern

Router "match" die Anfrage auf die Controller-Objekte und leitet den request an die Controller-Methoden weiter (hier: die Methode index des about-Controllers)

REQUEST – RESPONSE LEBENSZYKLUS

ROUTER => CONTROLLER => VIEW

Request – User klickt
Link auf der Webseite



Überprüft, ob Router
verfügbar und findet
passendes
Controller-Objekt



Methode des
Controllers wird
aufgerufen, um
Request zu
verarbeiten

Controller schickt
Daten an View, um
Response zu
erstellen

DAS ABOUT CONTROLLER OBJEKT

Methode `index` mit zwei Parametern
(wird vom Router aus aufgerufen):

- `request` – Objekt mit Details zur Anfrage des Clients
- `response` – Objekt, das benutzt wird um Anfrage an den Client zurückzusenden

Erzeugt Objekt `viewData` mit einer Eigenschaft `title`

Export des `about`-Objekts, damit es vom Router verwendet werden kann

```
controllers/about.js x
1  const logger = require("../utils/logger.js");
2
3  const about = {
4    index(request, response) {
5      logger.info("about rendering");
6      const viewData = {
7        title: "About Web app template"
8      };
9      response.render("about", viewData);
10   }
11 };
12
13 module.exports = about;
14
```

Erzeugt Logausgabe auf der Konsole (in replit) – später

Controller schickt Daten an View (hier: `about.hbs`), um Response zu erstellen – Ergebnis wird als HTML-String an den Client zurückgesendet

REQUEST - RESPONSE LEBENSZYKLUS

ROUTER => CONTROLLER => VIEW

Request - User klickt
Link auf der Webseite



Router

Überprüft, ob Router
verfügbar und findet
passendes
Controller-Objekt



Controller

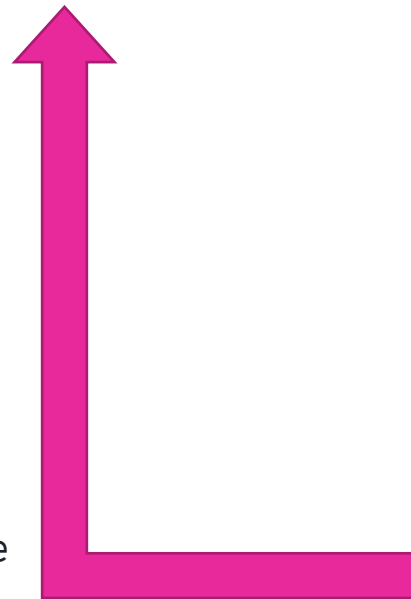
Methode des
Controllers wird
aufgerufen, um
Request zu
verarbeiten

Controller schickt
Daten an View, um
Response zu
erstellen



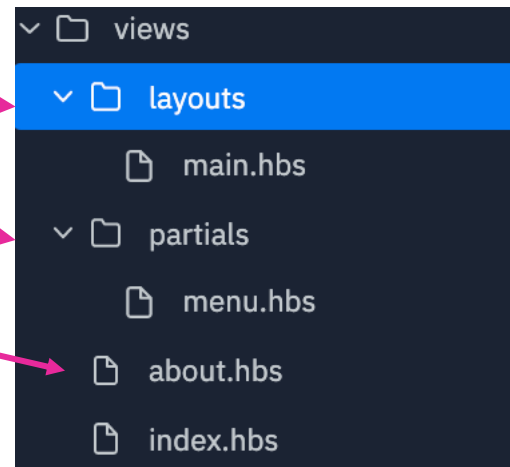
View

Response - Fertige
Seite wird an den
Browser
zurückgeschickt (der
diese dann anzeigt)



VIEWS SIND HTML TEMPLATES, DIE MIT DATEN DYNAMISCH BEFÜLLT WERDEN KÖNNEN

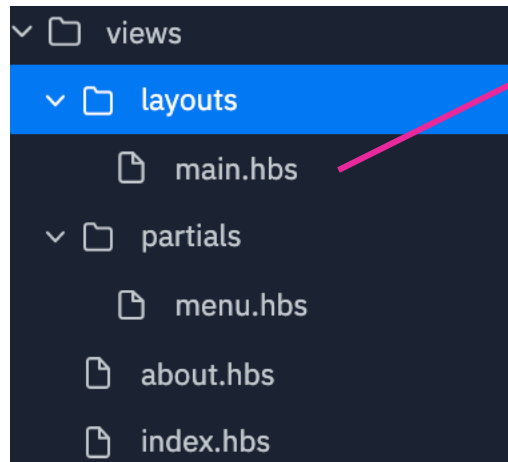
- Entwickelt in HTML und handlebars
- Handlebars: Template Engine – Befüllt views mit veränderlichen Daten.
 - Layouts
 - Partials
 - Views



LAYOUT

- main.hbs legt die Hauptstruktur für alle Seiten fest
- Bindet Bootstrap für Layout der Website ein

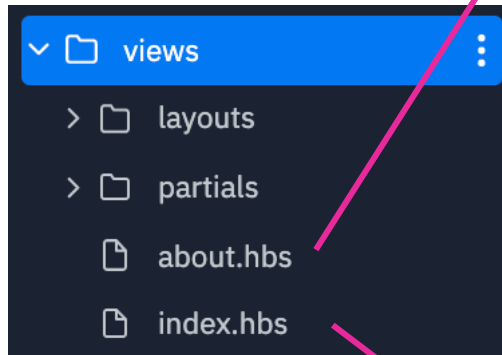
{{title}} ist ein Platzhalter und kann dynamisch ersetzt werden (mehr später)



```
views/layouts/main.hbs x
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>{{title}}</title>
6      <meta charset="UTF-8">
7      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.
8        integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jI
9        crossorigin="anonymous">
10     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bund
11       integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYs0g+0MhuP+IlRH9sENB00LRn5q+8nbTov4+
12       crossorigin="anonymous"></script>
13     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
14   </head>
15   <body>
16     <section class="container">
17       {{{body}}}
18     </section>
19   </body>
20 </html>
```

Inhalte der Views (hier: about.hbs und index.hbs) werden an die Stelle von {{{body}}} **automatisch** eingesetzt

ANZEIGE DER VIEWS IM FRONTEND



Web App Template About

Web app template

Content goes here...

```
views/about.hbs x
1  {{> menu id="about"}}
2
3  <div class="border p-2 my-2">
4    <h3>Web app template</h3>
5    <p>Content goes here...</p>
6  </div>
7
```

Web App Template About

Welcome to the Web app template!

Web app template

A web app template.

```
views/index.hbs x
1  {{> menu }}
2
3  <div class="card mt-3">
4    <div class="card-header">
5      {{title}}
6    </div>
7    <div class="card-body">
8      <h5 class="card-title">Web app template</h5>
9      <p class="card-text">A web app template.</p>
10   </div>
11 </div>
```

VIEWS BINDEN PARTIALS EIN (HIER MENÜ)

Partials sind Teile von Views die in unterschiedlichen Views verwendet werden können – Reduziert Redundanz!

```
views/partials/menu.hbs x
1 <nav class="navbar navbar-expand-lg navbar-light bg-light">
2   <div class="container-fluid">
3     <a class="navbar-brand" href="/">Web App Template</a>
4     <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
      data-bs-target="#navbarTogglerDemo02"
      aria-controls="navbarTogglerDemo02" aria-expanded="false" aria-label="Togg
5     <span class="navbar-toggler-icon"></span>
6   </button>
7   <div class="collapse navbar-collapse" id="navbarTogglerDemo02">
8     <ul class="navbar-nav me-auto mb-2 mb-lg-0">
9       <li class="nav-item">
10        <a class="nav-link" id="about" href="/about">About</a>
11      </li>
12    </ul>
13  </div>
14 </div>
15 </nav>
16
17 <script>
18   $("#{id}").addClass("active");
19 </script>
20
```

Web App Template About

Web app template

Content goes here...

```
views/about.hbs x
1 {{> menu id="about"}}
2
3 <div class="border p-2 my-2">
4   <h3>Web app template</h3>
5   <p>Content goes here...</p>
6 </div>
7
```

HIGHLIGHTEN DES AKTUELLEN MENÜPUNKTS (CLIENT-SIDE JS “MAGIC”)

Web App Template About

Web app template

Content goes here...

Variable `id` wird mit Wert “about”
an `menu.hbs` übergeben.

views/about.hbs x

```
1 {{> menu id="about"}}
```

```
2
```

```
3 <div class="border p-2 my-2">
```

```
4   <h3>Web app template</h3>
```

```
5   <p>Content goes here...</p>
```

```
6 </div>
```

```
7
```

```
<div class="collapse navbar-collapse" id="navbarTogglerDemo02">
  <ul class="navbar-nav me-auto mb-2 mb-lg-0">
    <li class="nav-item">
      <a class="nav-link" id="about" href="/about">About</a>
    </li>
  </ul>
</div>
</nav>

<script>
  $("#{{id}}").addClass("active");
</script>
```

Mithilfe der Bibliothek jQuery wird das Element mit der `id` `about` aus dem Menü selektiert und die Klasse “active” hinzugefügt – Menüeintrag `about` erscheint fett (clientseitiges (= im Browser ausgeführtes) JS ist *out of scope* für diesen Kurs)

WIEDERHOLUNG: DREI DATEIEN GENERIEREN EINEN VIEW

```
views/partials/menu.hbs x
1 <nav class="navbar navbar-expand-lg navbar-light bg-light">
2   <div class="container-fluid">
3     <a class="navbar-brand" href="/">Web App Template</a>
4     <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
5       data-bs-target="#navbarTogglerDemo02"
6       aria-controls="navbarTogglerDemo02" aria-expanded="false" aria-label="Toggle navigation">
7       <span class="navbar-toggler-icon"></span>
8     </button>
9     <div class="collapse navbar-collapse" id="navbarTogglerDemo02">
10      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
11        <li class="nav-item">
12          <a class="nav-link" id="about" href="/about">About</a>
13        </li>
14      </ul>
15    </div>
16  </div>
17  <script>
18    $("#{id}").addClass("active");
19  </script>
20 </nav>
```

```
views/layouts/main.hbs x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>{{title}}</title>
6     <meta charset="UTF-8">
7     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
8       rel="stylesheet"
9       integrity="sha384-1BmE4kWBq78iYhFdvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
10       crossorigin="anonymous">
11     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
12       integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYs0g+OMhuP+IlRH9sENB00LRn5q+8nbTov4+1p"
13       crossorigin="anonymous"></script>
14     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
15   </head>
16   <body>
17     <section class="container">
18       {{{body}}}
19     </section>
20   </body>
21 </html>
```

```
views/about.hbs x
1 <{{> menu id="about">}}
2
3 <div class="border p-2 my-2">
4   <h3>Web app template</h3>
5   <p>Content goes here...</p>
6 </div>
```

Gerenderter View

Web App Template About

Web app template

Content goes here...

EINE WEB-APP ALS ZUSAMMENSPIEL ZWISCHEN BACKEND UND FRONTEND

- Wie entwickelt man eine neue App
 - Start mit einem Template (z.B. web-app-template-1)
 - Schrittweises Erweitern, Anpassen und Ersetzen von Code aus dem Template
 - Im Router und in den Controllern
 - In den Handlebars Templates

AGENDA

HTTP und serverseitig generierte Webseiten

Walkthrough Web-App Template – Struktur einer Web App

Request – Response Lebenszyklus – Zusammenspiel von View und Controller

Templates und Schleifen

TEMPLATE ENGINES UND SCHLEIFEN

- Häufig werden in einer HTML-Seite Daten aus einer Liste beispielsweise in Tabellen, Dropdowns oder Aufzählungen verwendet. Hier: Darstellung mehrerer Playlists
- Handlebars bietet dafür eine Schleife

Web App Template About Dashboard

Happy mood

Iconic songs

DASHBOARD CONTROLLER

controllers/dashboard.hbs

```
const logger = require("../utils/logger.js");

const playlistCollection = [{title: "Happy mood"}, {title: "Iconic songs"}];

const dashboard = {
  index(request, response) {
    logger.info("dashboard rendering");
    const viewData = {
      title: "Dashboard",
      playlists: playlistCollection
    };
    logger.info('about to render', playlistCollection);
    response.render("dashboard", viewData);
  }
};

module.exports = dashboard;
```

Daten für den View (JS-Array aus Playlisten mit jeweils einem Titel)

Array wird als `playlists` im Objekt `viewData` gespeichert

View erhält das Array als Teil von `viewData`

RENDERN EINER LISTE VON PLAYLISTEN MIT EACH

views/dashboard.hbs

```
{{> menu id="dashboard"}}  
  
<section class="ui segment">  
  {{#each playlists}}  
    <div class="border p-2 my-2">  
      <h3>  
        {{this.title}}  
      </h3>  
    </div>  
  {{/each}}  
</section>
```

Für jedes Element in `playlists`...

... erzeuge jeweils ein eigenes `div`-Element mit Rahmen mit jeweils einer Überschrift...

... und füge den Titel der jeweiligen Playlist ein

```
const playlistCollection = [{title: "Happy mood"}, {title: "Iconic songs"}];
```

Web App Template About Dashboard

Happy mood

Iconic songs

FAZIT

- Request landet beim Router, der die Anfrage an die entsprechende Methode eines Controller-Objekts weiterleitet
- Der Controller holt sich Daten (hier bereits im Controller vorhanden) und befüllt den View mit Daten
- Abschließend schickt der Controller diesen View als response an den Client zurück
- Ein Controller-Objekt kann über mehrere Methoden verfügen die alle thematisch zusammengehörige requests verarbeiten (z.B. alle requests an ein Dashboard, oder alle requests für die Verwaltung von Nutzerdaten (login, logout, register, ...))
- Backend (Controller) und View (Templates) arbeiten zusammen um eine response zu erzeugen
- Was noch fehlt: Controller holt sich Daten aus Models (Anfang im Lab, mehr in der nächsten Unit...)