WEBTECHNOLOGIEN

03 – HTTP – CLIENT UND SERVER

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?

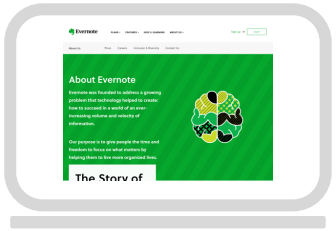
AGENDA

- **HTTP und serverseitig generierte Webseiten**
- Walkthrough Web-App Template – Struktur einer Web App
- Request – Response Lebenszyklus - Zusammenspiel von View und Controller
- Templates und Schleifen

WIE KOMMEN WEBSEITEN ZUM BROWSER?

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

1. Client öffnet Browser und will Website **http://evernote.com/settings** öffnen (**http-request**)



4. Webserver sendet **html** String zurück (**http-response**)

CLIENT

INTERNET

SERVER

5. Client Webbrowser verarbeitet HTML-Datei, rendert Website und fragt weitere Daten (z.B. Bilder, CSS, …) an
…
Will der Client neue Seiten, dann wieder request-response Zyklus

2. Routing: Server überprüft welche Route der Client angefragt hat. Hier **/settings**

3. Server generiert Seite (dynamisch) – Prüft, ob Nutzer eingeloggt, sucht Nutzerprofil, Lädt Profilinformationen, etc.

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
- ...

Quelle: https://developer.mozilla.org/de/docs/Web/HTTP/Methods

AGENDA

HTTP und serverseitig generierte Webseiten

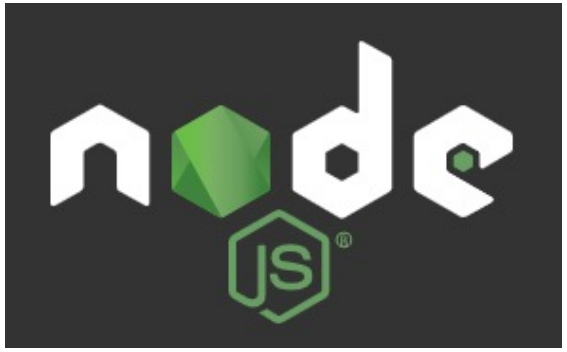
Walkthrough Web-App Template – Struktur einer Web App

Request – Response Lebenszyklus - Zusammenspiel von View und Controller

Templates und Schleifen

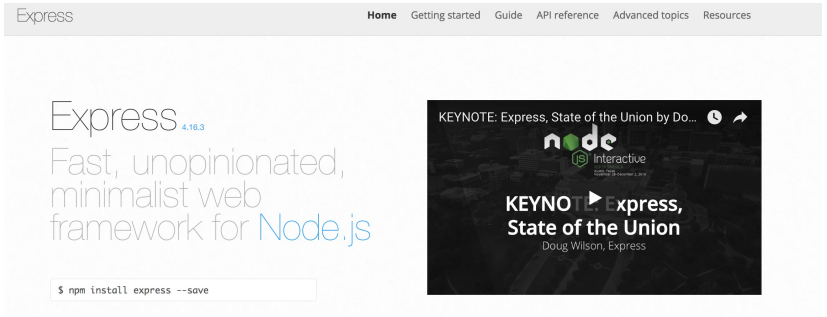
NODE.JS UND EXPRESS SIND FRAMEWORKS ZUR ENTWICKLUNG DYNAMISCHER WEBAPPLIKATIONEN

Node.js



- Framework, das JS Code auf dem Server ausführen kann
- Rudimentäre Unterstützung von http-Kommunikation
- Erweiterbard durch packages (z.B. Express)

Express



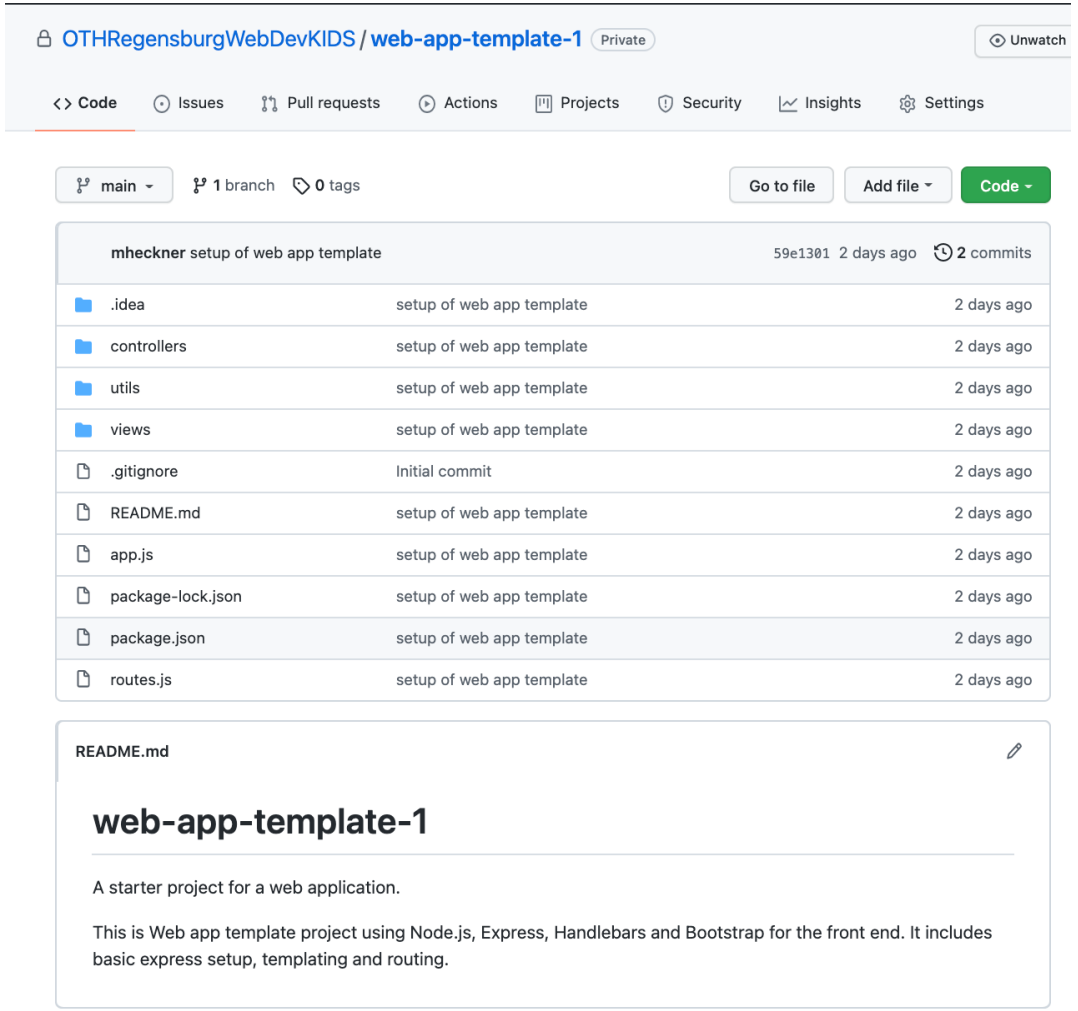
- Package für Node.js
- “Fast, unopinionated minimalist web framework for Node.js”
- Baut auf Node.js auf
- Ermöglicht die Erstellung und das Ausliefern von Webapplikationen

Handlebars

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

STARTERPROJEKT

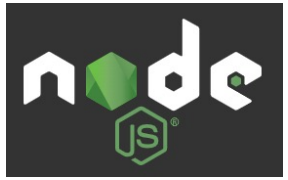
- Für jedes Projekt steht ein Startertemplate zur Verfügung (man muss nicht bei 0 loslegen)
- web_app_template_1 enthält enthält alle notwendigen Elemente für den Start der Entwicklung der eigenen App
- Auf dieser Basis lassen sich zahlreiche interessante (und komplexe) Projekt verwirklichen
- In replit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

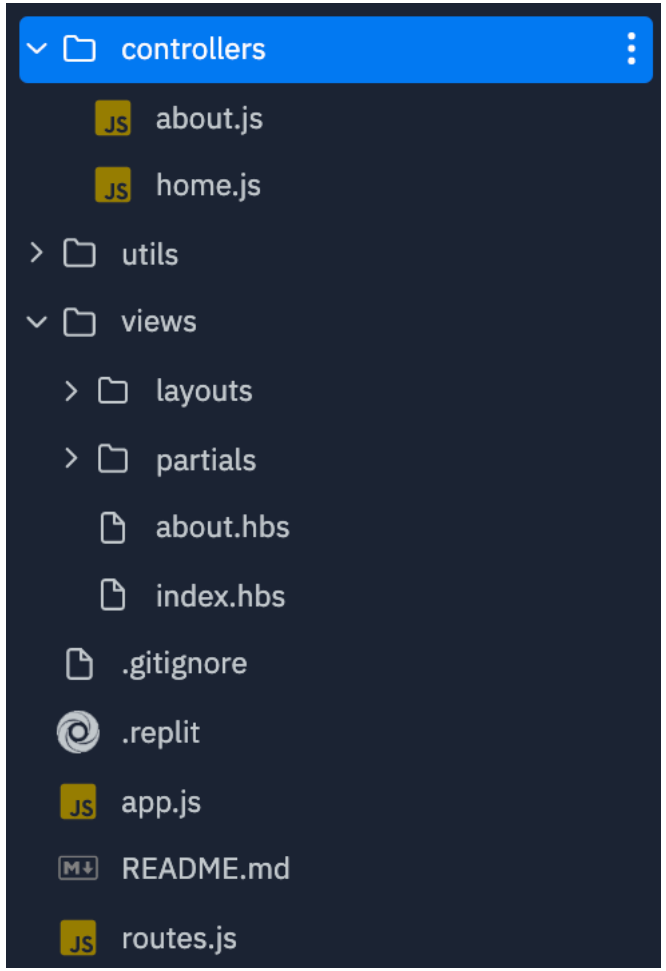


Template: https://github.com/OTHRegensburgWebDevKIDS/web-app-template-1

BACKEND

- Entwickelt in JavaScript (aufbauend auf Node.js und Express)
- Besteht aus
  - `app.js` – Startpunkt
  - `routes.js` – Welche URLs unterstützt werden
  - `controllers` – Objekte die Anfragen an die Routes verarbeiten
  - `views` – Das Frontend Seiten der Web-App





APP.JS IST DIE ZENTRALE KONFIGURATION DER WEB APP

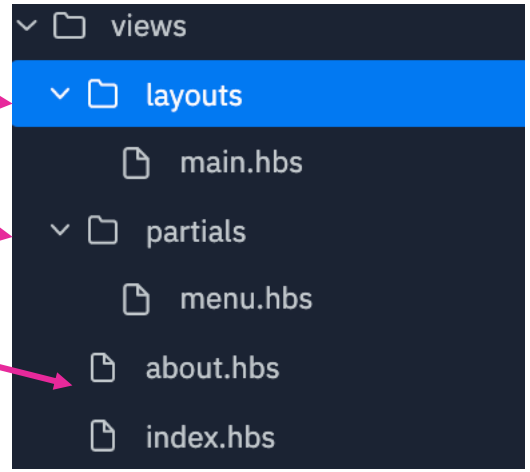
- Importiert node-packages (= Bibliotheken wie express und handlebars)
- Konfiguriert view-Engine (handlebars)
- Importiert den Router
- Wird über die Kommandozeile gestartet
- Rest erstmal nicht wichtig…

app.js

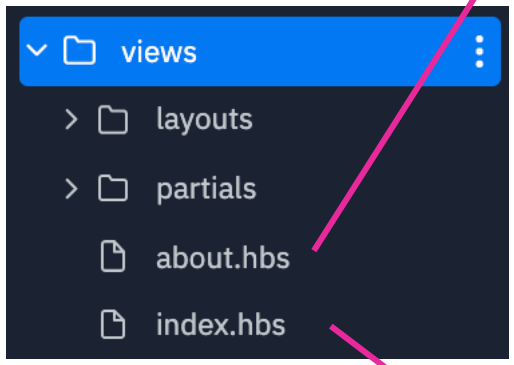
```
1  const express = require("express");
2  const logger = require("./utils/logger");
3  const handlebars = require("express-handlebars");
4
5  const dotenv = require("dotenv");
6  dotenv.config();
7
8  const app = express();
9
10 app.engine('.hbs', handlebars.engine({extname: '.hbs'}));
11 app.set('view 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;
```

FRONTEND

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



DAS STARTERPROJEKT VERFÜGT ÜBER 2 VIEWS



Web App Template About

Web app template

Content goes here...

Web App Template About

Welcome to the Web app template!

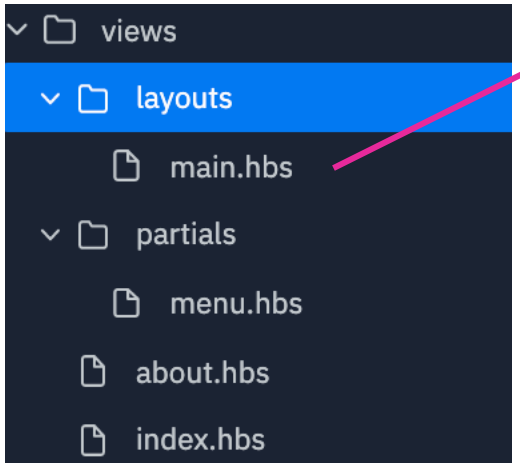
Web app template

A web app template.

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)

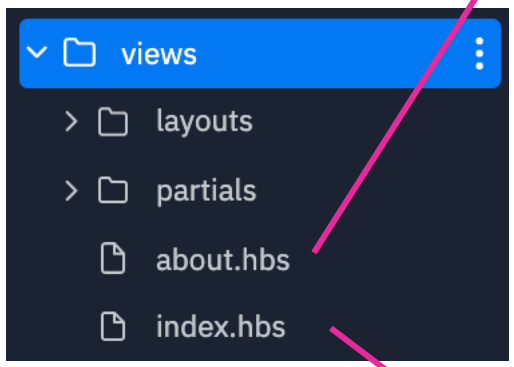


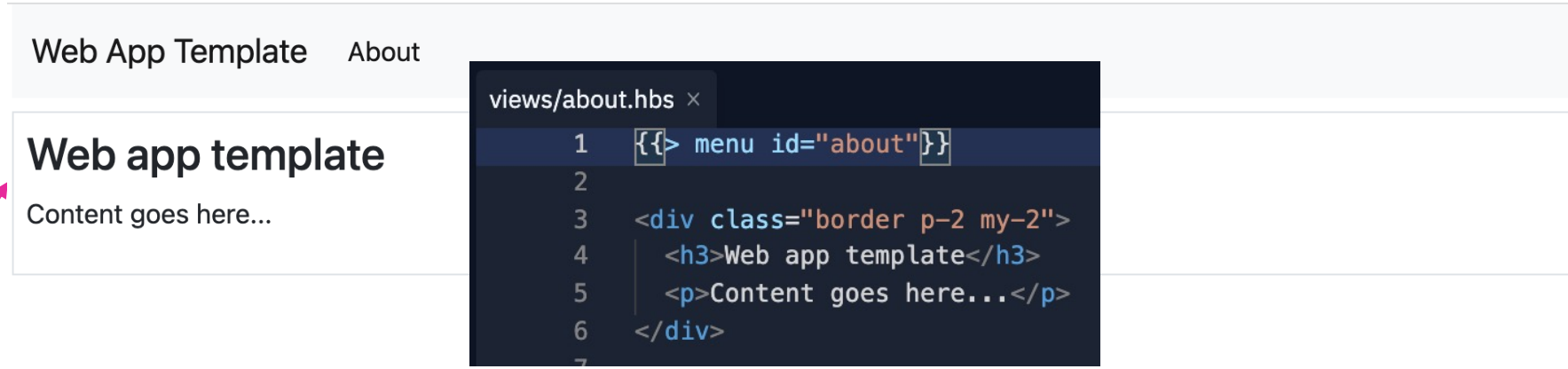
views/layouts/main.hbs

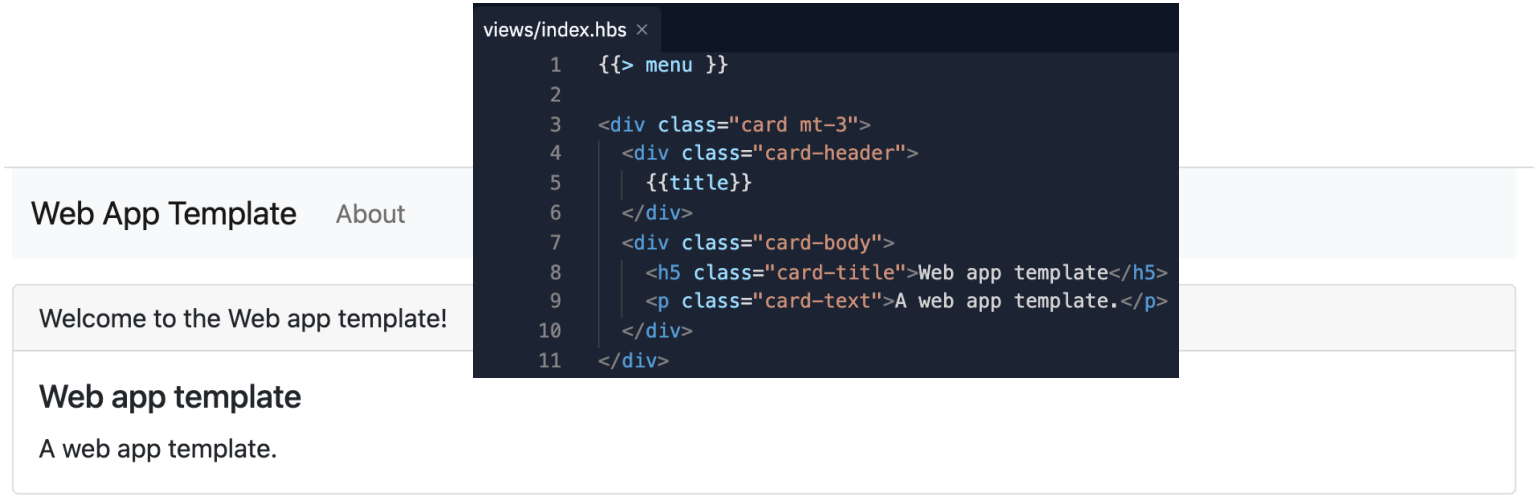
```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>{{title}}</title>
6      <meta charset="UTF-8">
7      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.
       integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jI
       crossorigin="anonymous">
8      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bund
       integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+
       crossorigin="anonymous"></script>
9      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
10   </head>
11   <body>
12     <section class="container">
13       {{{body}}}
14     </section>
15   </body>
16 </html>
```

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

ANZEIGE DER VIEWS IM FRONTEND







VIEWS BINDEN PARTIALS EIN (HIER MENÜ)

views/partials/menu.hbs

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">Web App Template</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
    data-bs-target="#navbarTogglerDemo02"
            aria-controls="navbarTogglerDemo02" aria-expanded="false" aria-label="Toggl
      <span class="navbar-toggler-icon"></span>
    </button>
    <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>
  </div>
</nav>

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

Web App Template About

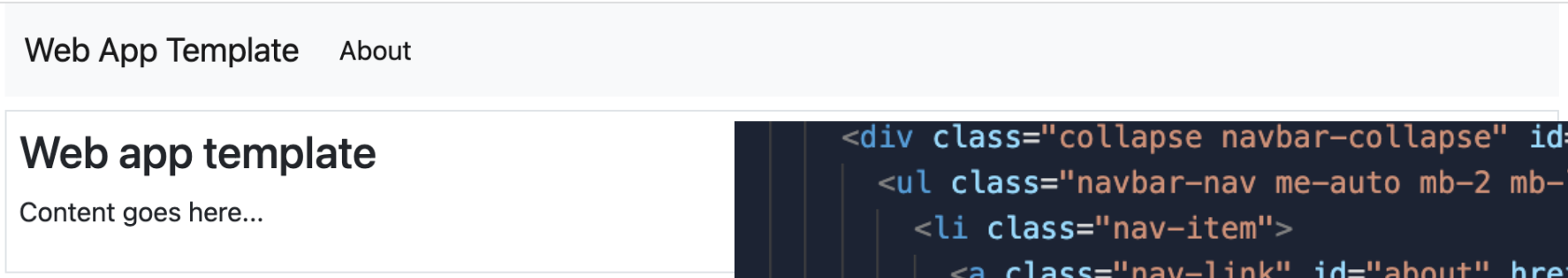
Web app template

Content goes here...

views/about.hbs

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

<div class="border p-2 my-2">
  <h3>Web app template</h3>
  <p>Content goes here...</p>
</div>
```

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



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

views/about.hbs

```
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>
```

```
      <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 jQuey 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)

DREI DATEIEN GENERIEREN EINEN VIEW

views/partials/menu.hbs

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">Web App Template</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
    data-bs-target="#navbarTogglerDemo02"
            aria-controls="navbarTogglerDemo02" aria-expanded="false" aria-la
      <span class="navbar-toggler-icon"></span>
    </button>
    <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>
  </div>
</nav>

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

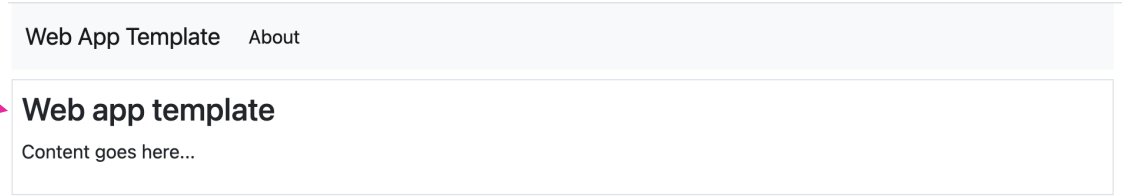
views/layouts/main.hbs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>{{title}}</title>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
    rel="stylesheet"
    integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
    crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
    crossorigin="anonymous"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  </head>
  <body>
    <section class="container">
      {{{body}}}
    </section>
  </body>
</html>
```

views/about.hbs

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

<div class="border p-2 my-2">
  <h3>Web app template</h3>
  <p>Content goes here...</p>
</div>
```

Gerenderter View



ZUSAMMENSPIEL ZWISCHEN BACKEND UND FRONTEND

- Entwicklung einer neuen App
  - Erweitern, Anpassen und Ersetzen von Code
    - Im router und in den Controllern
    - In den Handlebars Template

FAZIT

- main.hbs legt das Hauptlayout bzw. Die Hauptstruktur der Seite fest (hier steht der HTML-Header und Bootstrap wird eingebunden
- Die Views (z.B. about) werden in `{{{body}}}` eingesetzt
- Views können weitere partials (z.B. menu.hbs) integrieren, die auch in mehreren Views wiederverwendet warden können
- Alle Templates können angepasst und erweitert werden um eigene Web-Apps zu erstellen
- Neue views und partials können angelegt werden, um die Web-App zu erweitern

AGENDA

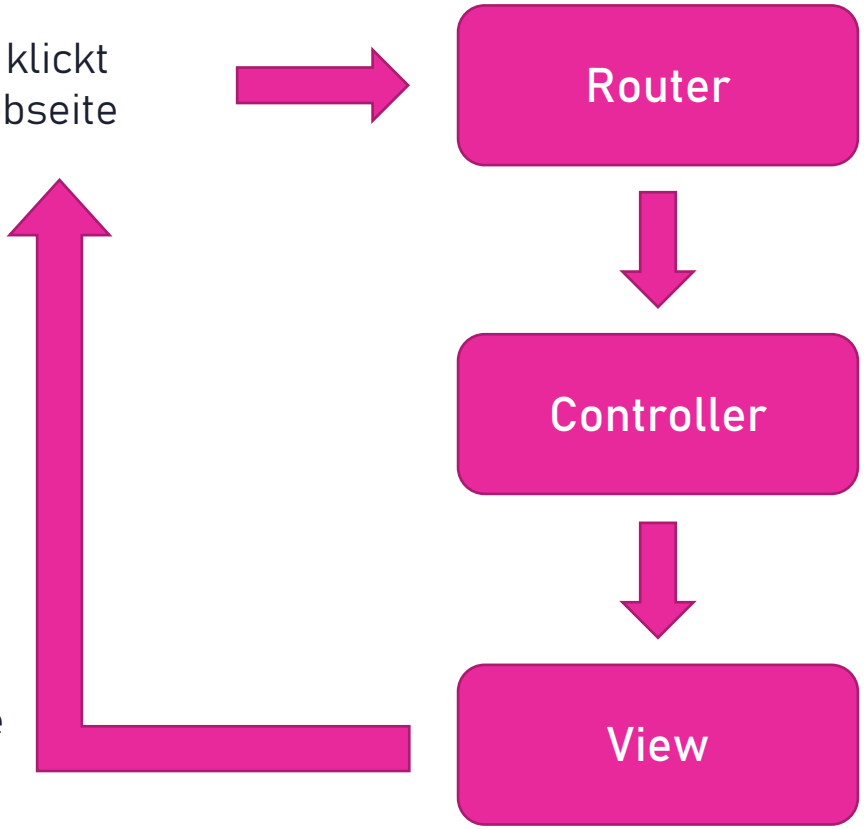
HTTP und serverseitig generierte Webseiten

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

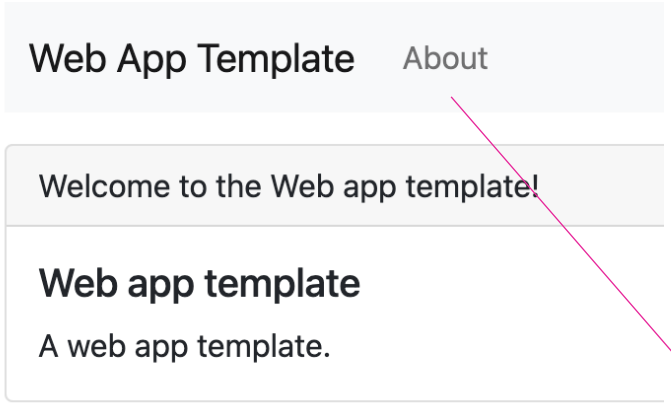
Findet passendes Controller-Objekt

Methode des Controllers wird aufgerufen, um Request zu verarbeiten

Controller schickt Daten an View, um Response zu erstellen

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

REQUEST - USER KLICKT LINK AUF DER WEBSEITE




```
<li class="nav-item">
  <a class="nav-link" id="about" href="/about">About</a>
</li>
```

Requests erzeugt durch:

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

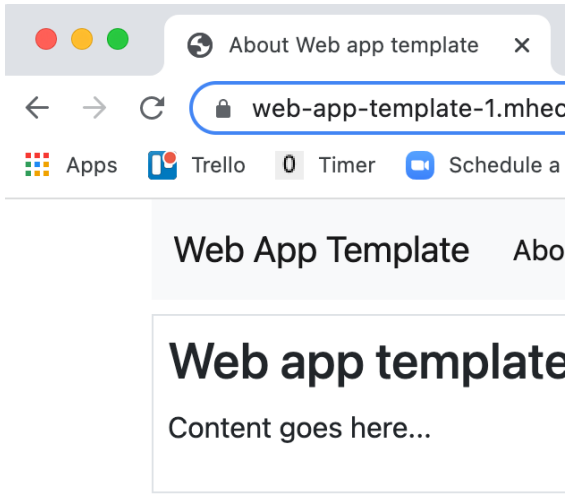
CONTROLLER – METHODE WIRD AUFGERUFEN, UM REQUEST ZU VERARBEITEN

controllers/about.js

```
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
```

Controller schickt Daten an View, um Response zu erstellen

response wird an den Client zurückgesendet (`title` erscheint im Tab)



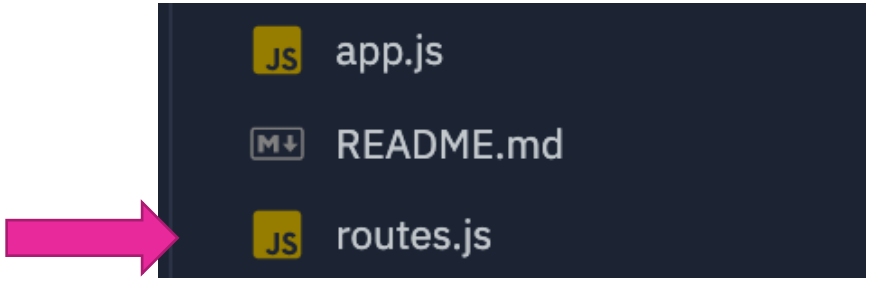
Der Wert des Attributs `title` des Objekts `viewData` wird in den View eingesetzt
(das macht handlebars)

views/layouts/main.hbs

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>{{title}}</title>
```

ROUTER – FINDET PASSENDES CONTROLLER-OBJEKT

request landet beim Router



routes.js

```
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

Router “matcht” die Anfrage auf die Controller-Objekte und leitet den request an die Controller-Methoden weiter (hier: die Methode `index` des about-Controller)

DAS ABOUT CONTROLLER OBJEKT

- Eine Methode `index` mit zwei Parametern:
  - Request – Objekt mit Details zur Anfrage des Clients
  - Response – Objekt, das benutzt wird um Anfrage an den Client zurückzusenden

controllers/about.js

```
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
```

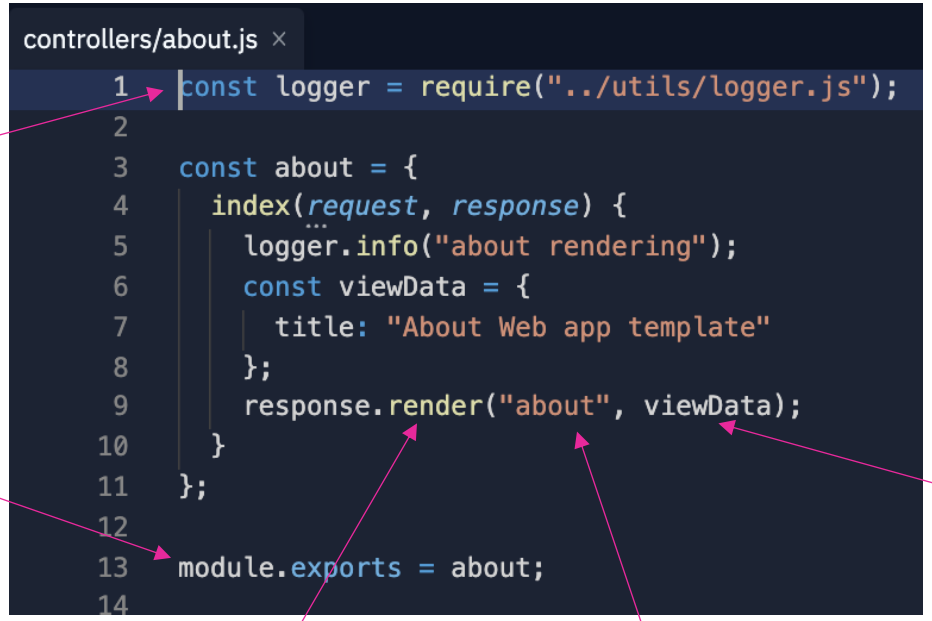
Erzeugt Logausgabe auf der Konsole (in replit)

Erzeugt Objekt `viewData` mit einer Eigenschaft `title`

DER CONTROLLER SENDET DIE DATEN AN DEN VIEW UM EINE RESPONSE ZU ERZEUGEN

Importieren des Loggers (später)

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



Render erzeugt die response (HTML-Code) und verschickt diese anschließend zurück an den Client

Name des zu rendernden Views

Daten die für das rendern des Views benötigt werden (werden in den View “injected” bzw. eingesetzt)

ZUSAMMENARBEIT ZWISCHEN BACK-END UND FRONTEND

controllers/about.js

```
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
```

about.hbs

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

<div class="border p-2 my-2">
  <h3>Web app template</h3>
  <p>Content goes here...</p>
</div>
```

menu.hbs

```
<nav class="navbar navbar-expand-lg n
  <div class="container-fluid">
    ...
  </div>
</nav>
```

main.hbs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>{{title}}</title>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/b
    integrity="sha384-1BmE4kWBq78iYhFldvKuhfTA
    <script src="https://cdn.jsdelivr.net/npm/
    integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1w
    <script src="https://ajax.googleapis.com/a
  </head>
  <body>
    <section class="container">
      {{{body}}}
    </section>
  </body>
</html>
```

`main.hbs` ist ein Wrapper für alle Views, die Inhalte des Views `about.hbs` werden in `{{body}}` eingesetzt.

AGENDA

HTTP und serverseitig generierte Webseiten

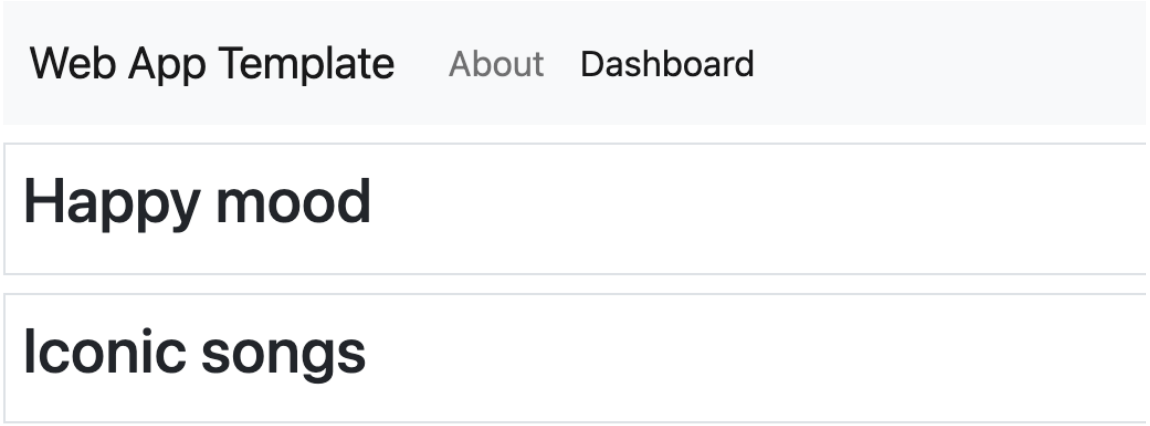
Walkthrough Web-App Template – Struktur einer Web App

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 beispielweise in Tabellen, Dropdowns oder Aufzählungen verwendet. Hier: Darstellung mehrerer Playlists
- Handlebars bietet dafür eine Schleife



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

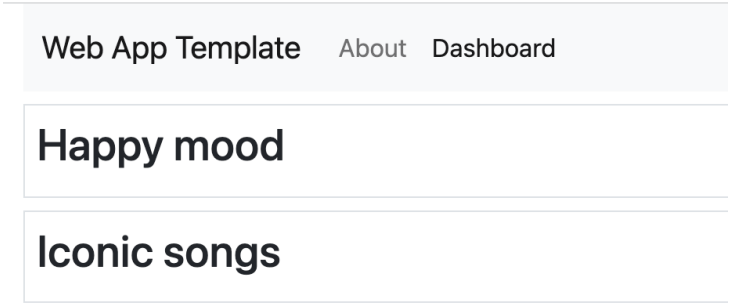
```
{{> menu id="dashboard"}}

<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"}];
```



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, nächste Woche mehr dazu)