

# WEBTECHNOLOGIEN

## 04 – NODE.JS UND DATENBANKEN, MODEL-VIEW- CONTROLLER

PROF. DR. MARKUS HECKNER

# AGENDA

Datenbanken und Web-Apps

Model-View-Controller

URL-Parameter

# WARUM DATENBANKEN

- Webanwendungen verarbeiten oft strukturierte Massendaten (Profile, Likes, Posts, Playlisten, Kommentare, etc.)
- Die Daten müssen aus einer Datenbank in die Webanwendung und die Webanwendung muss die Daten einfügen, aktualisieren und löschen können (CRUD – Create Read Update Delete)
- Zur Verwaltung dieser Daten stehen zahlreiche Datenbankmanagementsysteme (DBMS) zur Verfügung, z.B.
  - MySQL
  - Oracle 12c
  - **PostgreSQL**
  - SQLite
  - MongoDB (nicht-relational)
  - ...

# DATENBANKEN – HEUTE SQL MIT POSTGRES SQL



- Relationales DBMS
- Open Source
- Packages für den Zugriff über Node.js verfügbar
- Kostenlose Datenbank bei Heroku verfügbar

# IN DIESER SESSION NUR EINE TABELLE

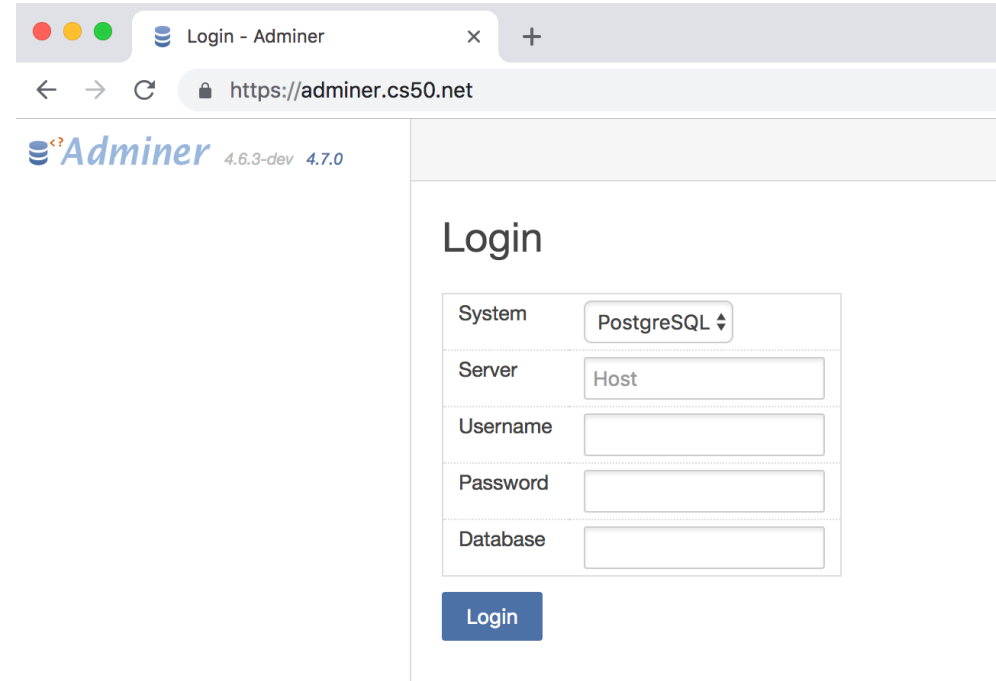
```
CREATE TABLE dbdemo_playlists (  
    ID SERIAL PRIMARY KEY,  
    TITLE VARCHAR  
);
```

```
INSERT INTO dbdemo_playlists (TITLE) VALUES ('Happy Mood');  
INSERT INTO dbdemo_playlists (TITLE) VALUES ('Iconic songs');
```

# ZUGRIFF AUF DIE POSTGRESQL DATENBANK MIT ADMINER

- Weboberfläche zum Management von Datenbanken
- Direkte Eingabe von SQL-Queries möglich
- Zugangsdaten zur Datenbank von Heroku (vgl. Lab)

<https://adminer.cs50.net/>



The screenshot shows a web browser window with the title "Login - Adminer" and the URL "https://adminer.cs50.net". The Adminer logo and version "4.6.3-dev 4.7.0" are visible. The login form is titled "Login" and contains the following fields:

System	PostgreSQL ▾
Server	Host
Username	<input type="text"/>
Password	<input type="password"/>
Database	<input type="text"/>

Below the form is a blue "Login" button.

# DB DEMO – ÜBERSICHT ALLER PLAYLISTEN

[Playlist 1](#) [Dashboard](#) [About](#)

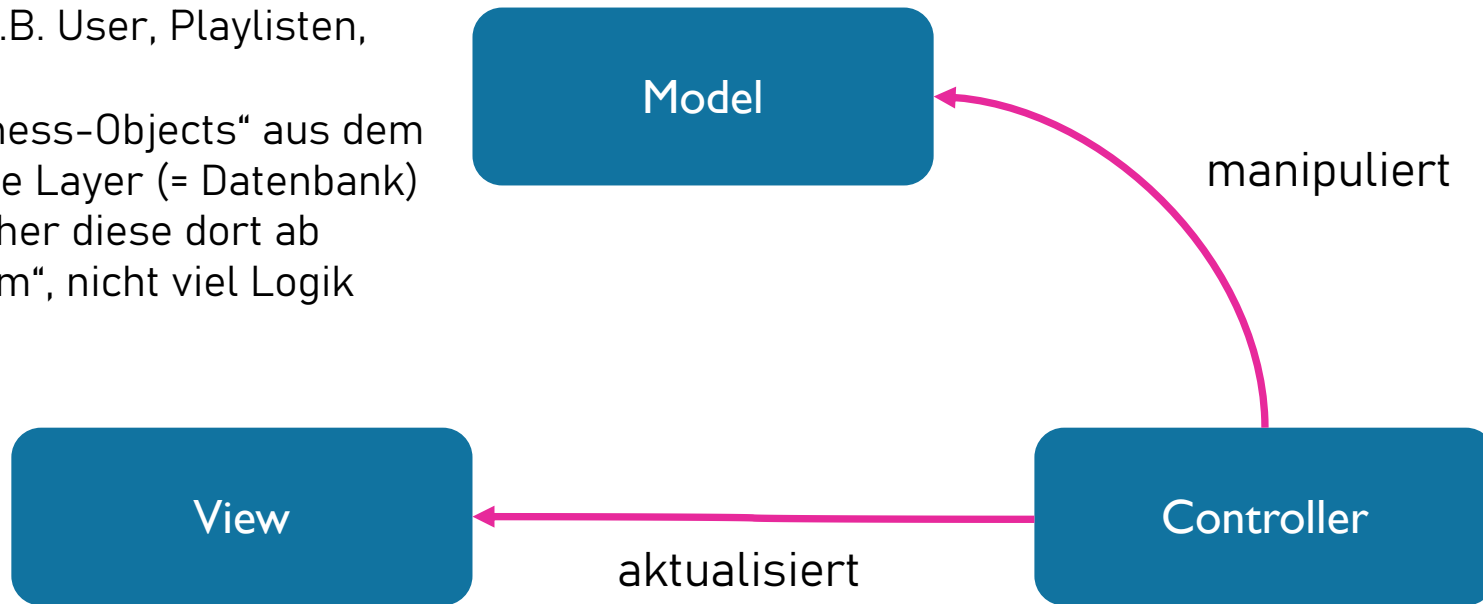
## Dashboard

Happy Mood

Iconic songs

# MODEL VIEW CONTROLLER IST EIN WEIT VERBREITETES SOFTWARE-ENTWURFSMUSTER

- Persistiert Daten aus „Business-Objects“ (z.B. User, Playlisten, Songs, ...)
- Holt „Business-Objects“ aus dem Persistence Layer (= Datenbank) oder speicher diese dort ab
- Eher „dumm“, nicht viel Logik

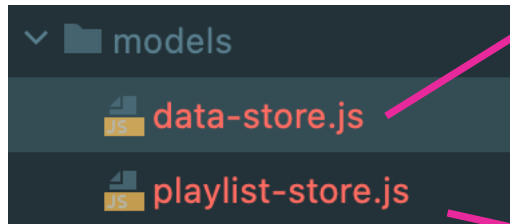


- Darstellung der Web-App im Browser

- Holt Daten aus den Models
- Speichert Daten in den Models
- Fügt Daten in die Views ein und senden Daten diese an Client zurück



# ZUSAMMENSPIEL ZWISCHEN DATA-STORE.JS UND DEM MODEL PLAYLIST-STORE.JS



Model benötigt Zugriff auf die Datenquelle  
(hier: PostgreSQL) – Das Modul data-store.js stellt diesen  
Zugriff her für alle Models her

Model zur Abfrage und Ablegen von Daten zur Playlist

# DAS MODEL STEHT DEN CONTROLLERN ZUR VERFÜGUNG UM DATEN ABZURUFEN UND ZU SPEICHERN

playlist-store.js

```
const datastore = require("../data-store.js");
const datastoreClient = datastore.getDataStore();
const logger = require("../utils/logger.js");

const playlistStore = {
  async getAllPlaylists() {
    const query = 'SELECT * FROM playlist2_playlists';
    try {
      let result = await datastoreClient.query(query);
      return result.rows;
    } catch (e) {
      logger.error("Error fetching all playlists", e);
    }
  },
};

module.exports = playlistStore;
```

} Model benötigt Zugriff auf die Datenquelle mittels des Moduls data-store.js

Objekt playListStore ist das Model und bietet über seine Methoden Zugriff auf die Daten

SQL-Query-String

Absenden des Queries mithilfe des datastoreClient-Objekts – Rückgabe ist Objekt result mit einem Array rows (= Ergebnis der SQL-Abfrage)

Rückgabe des Ergebnisses an den Controller

Error Handling – Logging des DB Fehlers

# ASYNC UND AWAIT ERLEICHTERN ASYNCHRONE PROGRAMMIERUNG

```
const datastore = require("../data-store.js");
const datastoreClient = datastore.getDataStore();
const logger = require("../utils/logger.js");

const playlistStore = {
  async getAllPlaylists() {
    const query = 'SELECT * FROM playlist2_playlists';
    try {
      let result = await datastoreClient.query(query);
      return result.rows;
    } catch (e) {
      logger.error("Error fetching all playlists", e);
    }
  },
};

module.exports = playlistStore;
```

`getAllPlaylists` enthält ein `await`, d.h. das Keyword `async` muss vor der Funktionsdefinition hinzugefügt werden!

Methode `query` ist asynchron, d.h. gibt nicht sofort ein Ergebnis zurück – Aber mittels `await` verhält sich der Code synchron – d.h. bis das Ergebnis von `query` zurückgegeben wird kann der Node Server andere Dinge tun, aber die Ausführung in `getAllPlaylists` geht erst weiter, wenn `query` ein Ergebnis zurückliefert!



Asynchrone Programmierung mit JS ist eine größere Baustelle (Stichworte Callbacks, Promises) – Für diesen Kurs reicht die Verwendung von `async` / `await`!

# DATA-STORE.JS STELLT VERBINDUNG ZUR DB HER

- Stellt mittels eines DB-Connection-Strings die Verbindung zur DB her
- Ist eine Schnittstelle zur Datenbank
- Hier PostgreSQL, beliebige andere relationale und nicht-relationale DBMS integrierbar

```
let pg = require("pg");
const logger = require("../utils/logger.js");
const conString = process.env.DB_CON_STRING;

const dbConfig = {
  connectionString: conString,
  ssl: { rejectUnauthorized: false }
}

if (conString == undefined) {
  logger.error("ERROR: environment variable DB_CON_STRING not set.");
  process.exit( code: 1);
}

let dbClient = null;

const datastore = {
  getDataStore() {
    if (dbClient !== null) {
      return dbClient;
    } else {
      dbClient = new pg.Client(dbConfig);
      dbClient.connect();
      return dbClient;
    }
  },
  async endConnection() {
    await dbClient.end();
  }
}
```

# DATA-STORE.JS (1/2)

Liest String für Zugriff auf die Datenbank aus der Datei .env (bzw. in replit aus den Secrets)

```
let pg = require("pg");
const logger = require("../utils/logger.js");
const conString = process.env.DB_CON_STRING;

const dbConfig = {
  connectionString: conString,
  ssl: { rejectUnauthorized: false }
}

if (conString == undefined) {
  logger.error("ERROR: environment variable DB_CON_STRING not set.");
  process.exit( code: 1);
}
```

Config

Error-Handling

# DATA-STORE.JS (2/2)

Gibt dbClient für das Absenden von SQL-Abfragen  
an die Models zurück

```
let dbClient = null;

const datastore = {
  getDataStore() {
    if (dbClient !== null) {
      return dbClient;
    } else {
      dbClient = new pg.Client(dbConfig);
      dbClient.connect();
      return dbClient;
    }
  },
  async endConnection() {
    await dbClient.end();
  }
}
```

Mehr ist erstmal nicht wichtig!

# AGENDA

Datenbanken und Web-Apps

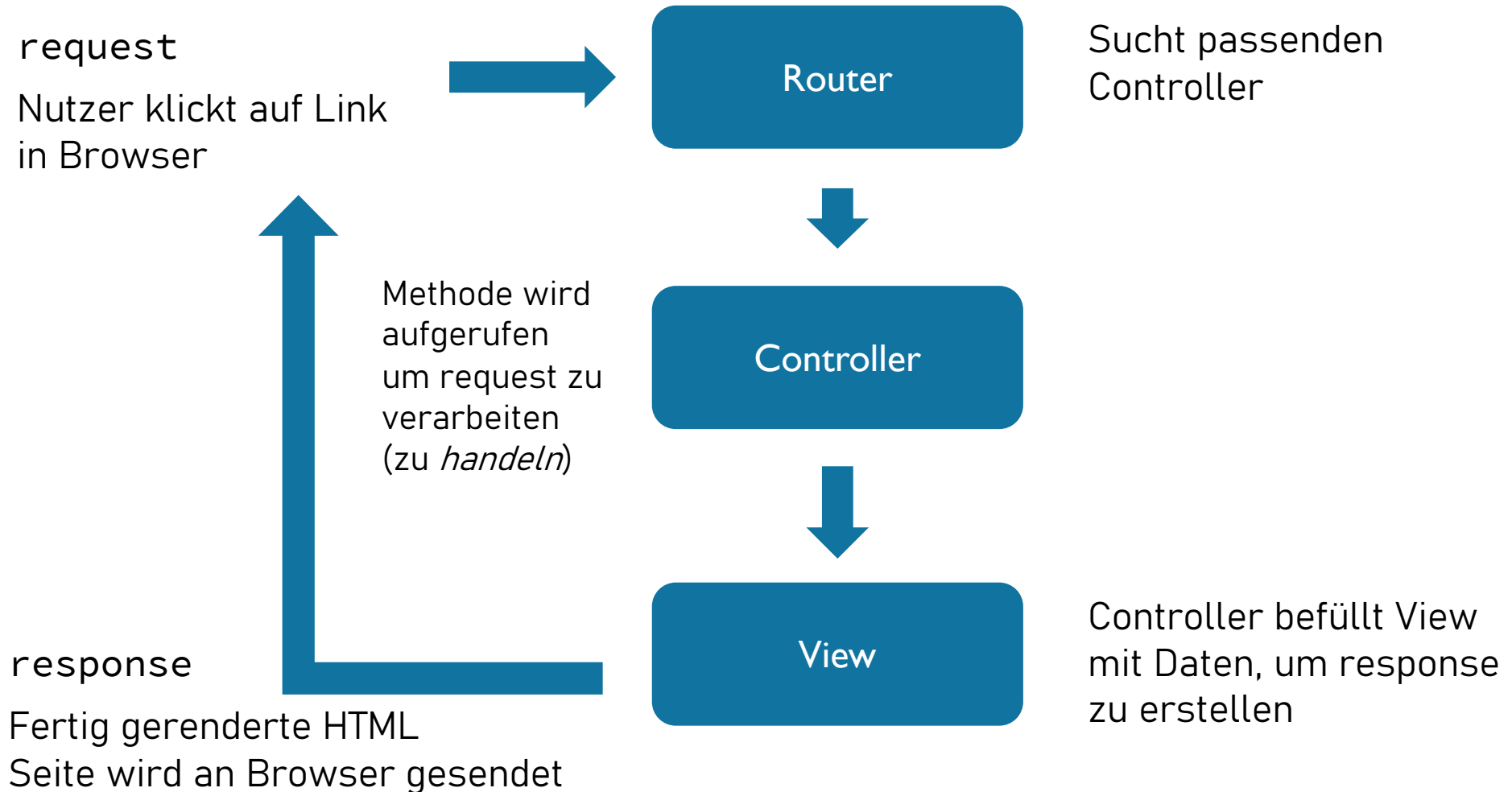
Model-View-Controller

URL-Parameter

# ROUTER CONTROLLER VIEW VS. ROUTER CONTROLLER MODEL VIEW

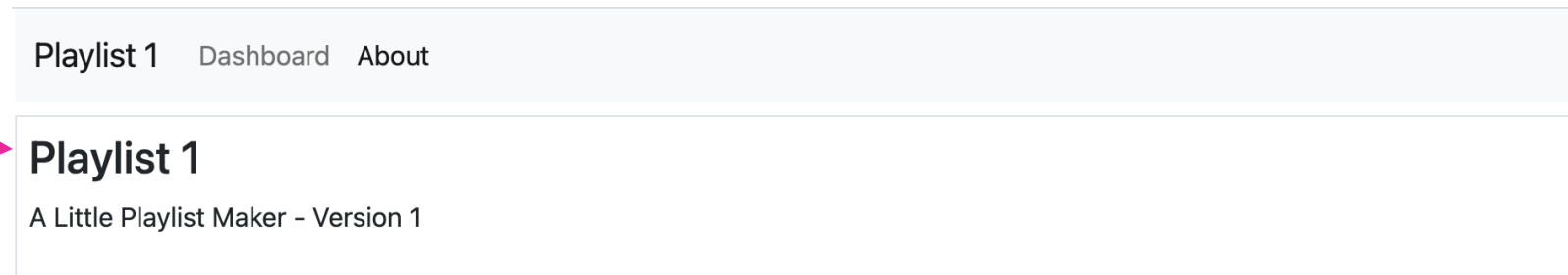
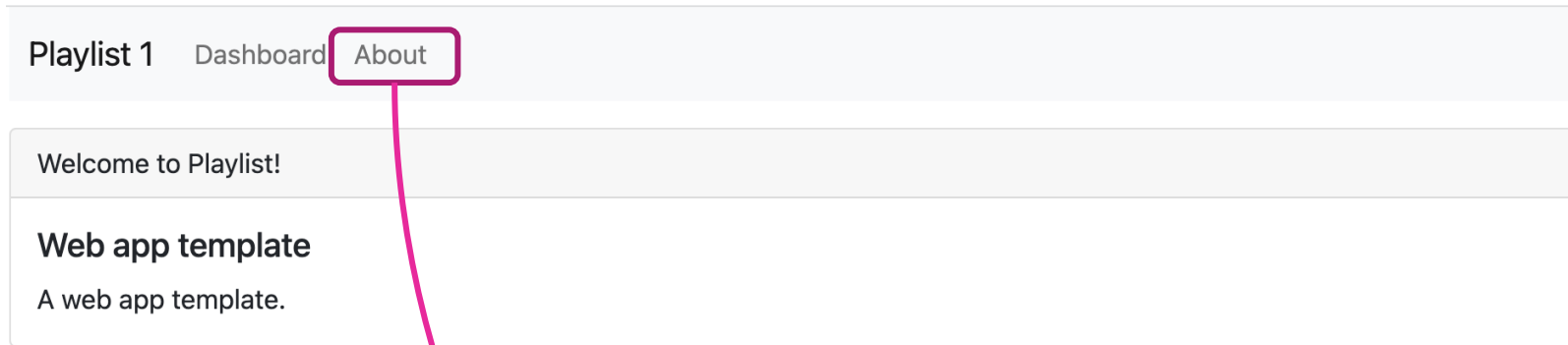


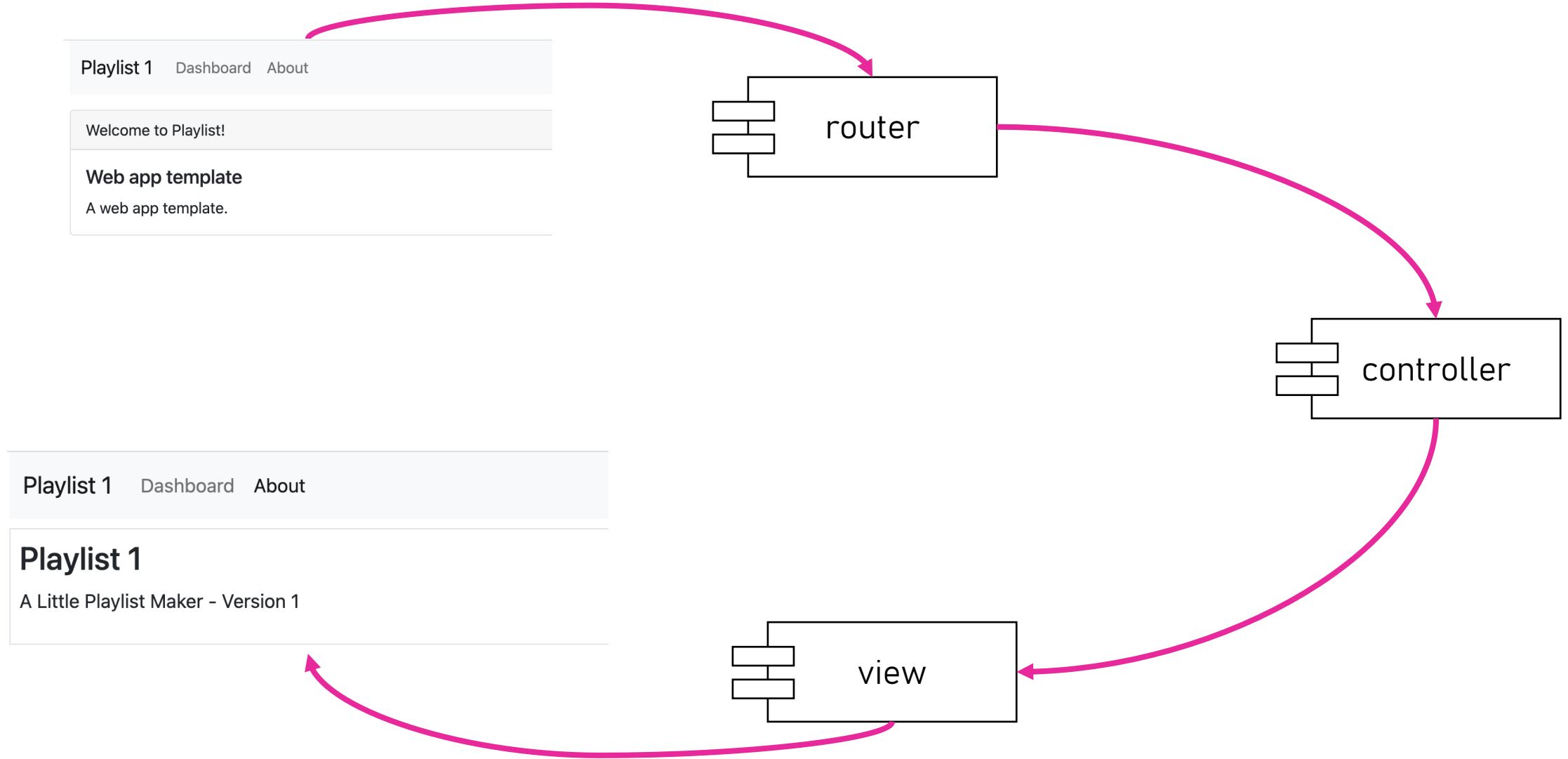
# WIEDERHOLUNG: DER WEG VOM ROUTER ÜBER DEN CONTROLLER ZUM VIEW



# ROUTER CONTROLLER VIEW

MVC





Playlist 1 Dashboard About

Welcome to Playlist!

Web app template

A web app template.

```
rel="stylesheet" integrity="sha384-ipt"
<a class="nav-link" id="about" href="/about">About</a>
<div class="collapse navbar-collapse" id="navbarTogglerDemo02">
  <ul class="nav-item">
    <li class="nav-link" id="dashboard" href="/dashboard">Dashboard</li>
    <li class="nav-link" id="about" href="/about">About</li>
  </ul>
</div>
</nav>
```

Playlist 1 Dashboard About

Playlist 1

A Little Playlist Maker - Version 1

```
<a class="nav-link" id="dashboard"
</li>
<li class="nav-item">
  <a class="nav-link" id="about" href="
</li>
</ul>
</div>
</nav>
<script>
  $("#about").addClass("active");
</script>
<div class="border p-2 my-2">
  <h3>Playlist 1</h3>
  <p>A Little Playlist Maker - Version 1</p>
</div>
</section>
</body>
</html>
```



router

routes.js

```
router.get( path: "/", home.index);
router.get( path: "/about", about.index);
router.get( path: "/dashboard", dashboard.index);
```

GET request /about

about.index(req, res);



controller

about.js

```
const about = {
  index(request, response) {
    logger.info( message: "about rendering");
    const viewData = {
      title: "About Playlist 1"
    };
    response.render("about", viewData);
  }
};
```

response.render("about", viewData);

```
const viewData = {
  title: "About Playlist 1"
};
```

about.hbs

```
{{> menu id="about"}}
<div class="border p-2 my-2">
  <h3>Playlist 1</h3>
  <p>A Little Playlist Maker - Version 1</p>
</div>
```



view

main.hbs

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>{{title}}</title>
  <meta charset="UTF-8">
  <link href="https://cdn.jsdelivr.net"
  <script src="https://cdn.jsdelivr.net"
  <script src="https://ajax.googleapis"
</head>
<body>
  <section class="container">
    {{{body}}}
  </section>
</body>
</html>
```

{{{body}}}

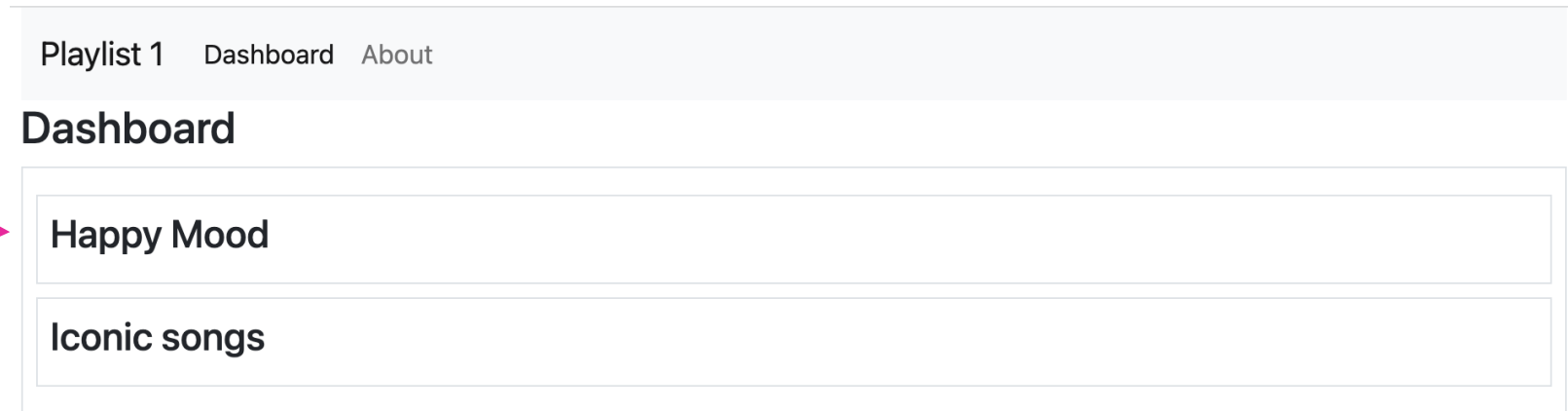
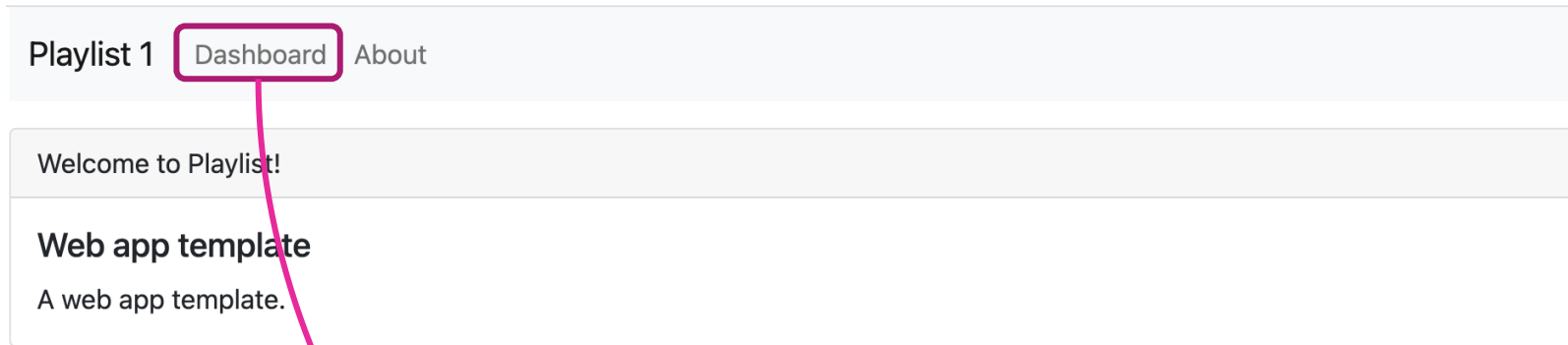
response

{{> menu id="about"}}

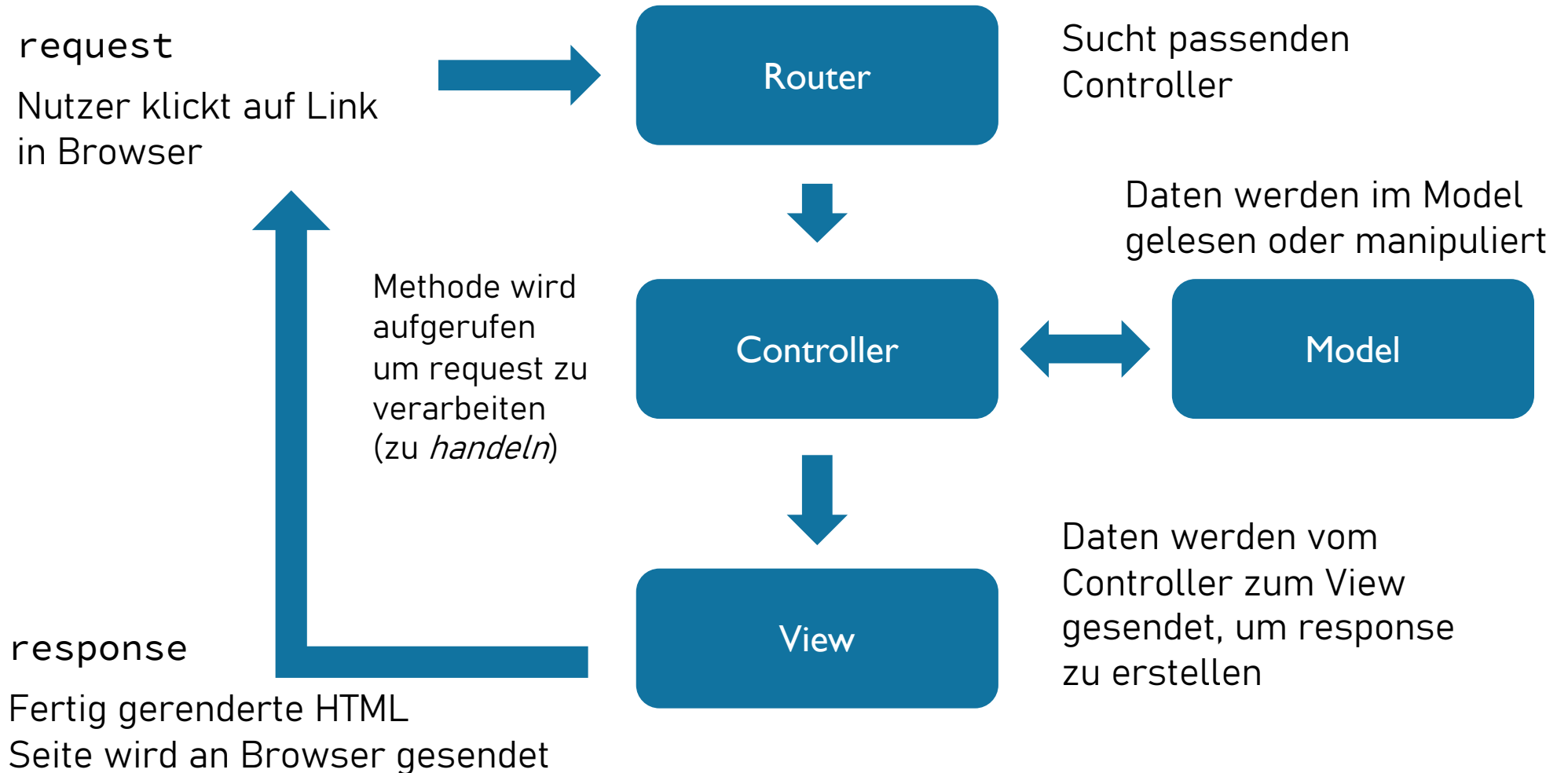
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  ...
</div>
</nav>
```

# ROUTER CONTROLLER MODEL VIEW

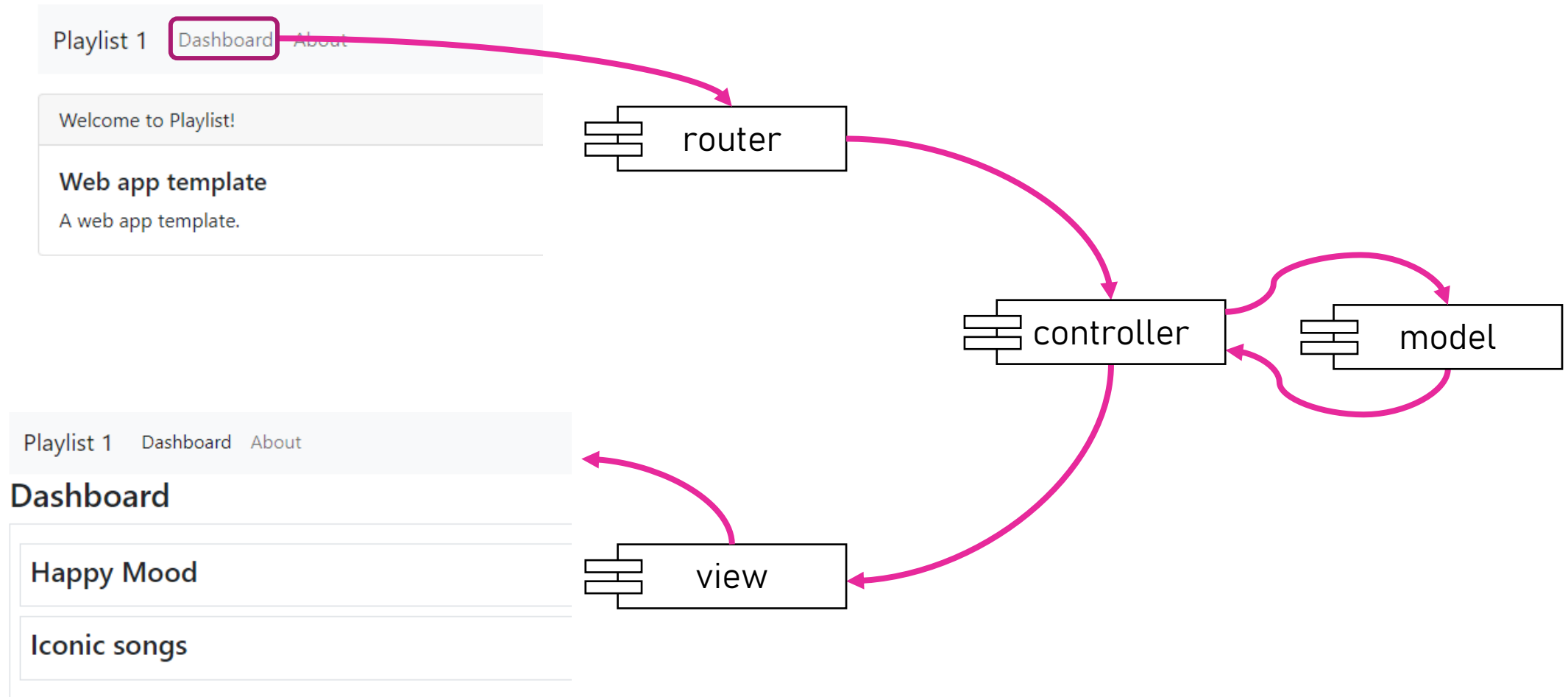
MVC



# DER WEG VOM ROUTER ÜBER DEN CONTROLLER UND DAS MODEL ZUM VIEW



# ROUTER CONTROLLER MODEL VIEW



Playlist 1 Dashboard About

Welcome to Playlist!

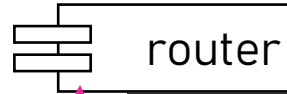
## Web app template

A web app template.

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

Dashboard.html

GET request /dashboard



router

```
router.get( path: "/", home.index);
router.get( path: "/about", about.index);
router.get( path: "/dashboard", dashboard.index);
```

dashboard.index(req, res)



controller

```
const playlistStore = require("../models/playlist-store.js");

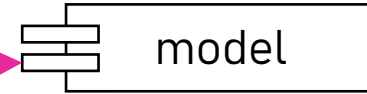
const dashboard = {
  async index(request, response) {
    logger.info( message: "dashboard rendering");
    const playlists = await playlistStore.getAllPlaylists();
    const viewData = {
      title: "Dashboard",
      playlists: playlists
    };
    logger.info( message: "about to render", playlists );
    response.render("dashboard", viewData);
  }
};

module.exports = dashboard;
```

dashboard.js

```
const datastore = {
  getDataStore() {
    if (dbClient !== null) {
      return dbClient;
    } else {
      dbClient = new pg.Client(dbConfig);
      dbClient.connect();
      return dbClient;
    }
  },
  async endConnection() {
    await dbClient.end();
  }
}
```

data-store.js



model

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

const playlistStore = {
  async getAllPlaylists() {
    const query = 'SELECT * FROM playlist_playlists';
    try {
      let result = await datastoreClient.query(query);
      return result.rows;
    } catch (e) {
      logger.error( message: "Error fetching all playlists", e );
    }
  }
};
```

playlist-store.js

response.render("dashboard", viewData);



view

```
<!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">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" rel="stylesheet">
</head>
<body>
  <section class="container">
    {{body}}
  </section>
</body>
</html>
```

main.hbs

```
{{> menu id="dashboard"}}
<h3>Dashboard</h3>

<div class="border p-2 my-2">
  {{#each playlists}}
    <div class="border p-2 my-2">
      <h4>{{title}}</h4>
    </div>
  {{/each}}
</div>
```

dashboard.hbs

```
<div class="header navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">Playlist 1</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbar-toggler-demo2" aria-controls="navbar-toggler-demo2" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbar-toggler-demo2">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link" id="dashboard" href="/dashboard">Dashboard</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" id="about" href="/about">About</a>
        </li>
      </ul>
    </div>
  </div>
</div>
```

menu.hbs

```
{
  title: 'Dashboard',
  playlists: [
    { id: 1, title: 'Happy Mood', user_id: 'markus.heckner@gmail.com' },
    { id: 2, title: 'Iconic songs', user_id: 'markus.heckner@gmail.com' }
  ]
}
```

viewdata

MVC



# CONTROLLERS UND MODELS IM DETAIL

## Controller

- Funktionen werden vom Router aufgerufen (Nutzer greift auf eine Seite zu) oder von anderen Controllern (z.B. Controller zeigt Dashboard nach Login (später))
- Holen Daten aus den Models
- Speichern Daten in den Models
- Verwenden utils (falls nötig – später)
- Fügen Daten in die Views ein und senden Daten diese an Client zurück

## Models

- Persistieren Daten aus „Business-Objects“ (z.B. User, Playlisten, Songs, ...)
- Holen „Business-Objects“ aus dem Persistence Layer (= Datenbank) oder speichern diese dort ab
- Eher „dumm“, nicht viel Logik

## Utils

- Verarbeiten Daten aus den Models (z.B. für Analytics, Konvertierungen)

# AGENDA

Datenbanken und Web-Apps

Model-View-Controller

URL-Parameter

# TYPISCHE URL FÜR DAS ANZEIGEN ALLER OBJEKTE EINES BESTIMMTEN TYP

- URL enthält den Namen des Objekts (eigtl. playlists)
- Keine weiteren Bestandteile in der URL

/dashboard

oder alternativ:

/playlist

Playlist 1   Dashboard   About

## Dashboard

Happy Mood

[View](#)





Iconic songs

[View](#)

# TYPISCHE URL FÜR DIE ABFRAGE EINES BESTIMMTEN OBJEKTS

- URL enthält den Namen des Objekts (eigtl. playlists)
- ID (Primärschlüssel) des Objekts als Teil der URL
- ID kann im Controller ausgelesen werden und an das Model weitergegeben werden für Abfragen, Löschen, Manipulieren

## /playlist/1

Playlist 1 Dashboard About		
Happy Mood		
Song	Artist	
Valerie	Amy Winehouse	
22	Taylor Swift	
Happy	Pharrell Williams	
a	a	

Router:

```
router.get('/playlist/:id', playlist.index);
```

Controller:

```
const playlistId = request.params.id;
```

# URLS KÖNNEN AUCH MEHRERE PARAMETER ENTHALTEN

- URL enthält den Namen des Objekts (eigtl. playlists)
- Hier: Löschen des Songs mit der ID 2 aus der Playlist mit der ID 1
- URL-Parameter werden als Keys im Objekt `params` des `requests` angelegt

`/playlist/1/deletesong/2`

Router:

```
router.get('/playlist/:id/deletesong/:songid', playlist.deleteSong);
```

Controller:

```
const playlistId = request.params.id;  
const songId = request.params.songid;
```

# FAZIT

- Datenbanken ermöglichen in Verbindung mit Webanwendungen die Verwaltung von größeren Datenmengen (z.B. Nutzer, Songs, Playlisten, Posts, Wetterstationen, Wettermesswerte, etc.)
- Wir verwenden PostgreSQL, im Prinzip lässt sich nahezu jede relationale und nichtrelationale Datenbank mit Node.js und Express einsetzen
- Models speichern Daten und rufen Daten ab, Controller verwenden die Models um Daten zu speichern und abzurufen, Views sind das Frontend der Web-App und werden vom Controller mit Daten befüllt und an den Client zurückgeschickt
- In der URL lassen sich Parameter integrieren, die von der Web-App ausgelesen werden können (vgl. z.B. `playlist/1` zum Abruf der Playlist mit der ID 1)