

WEBTECHNOLOGIEN

06 – SESSIONS

PROF. DR. MARKUS HECKNER

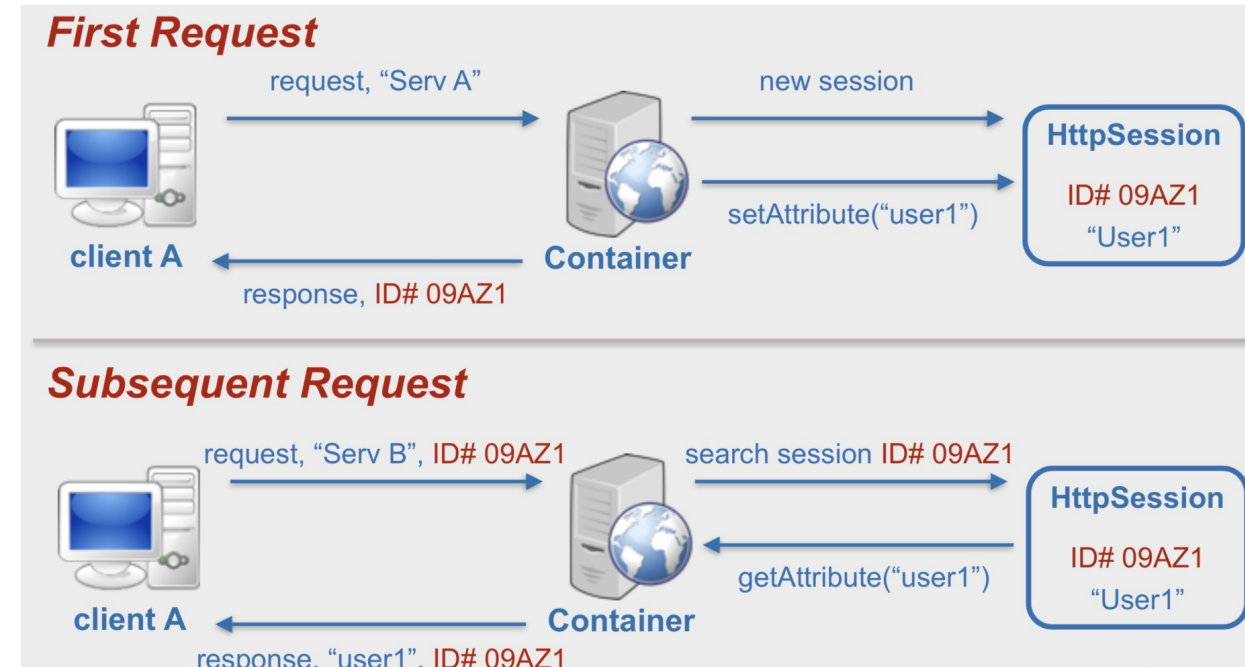
HTTP IST ZUSTANDSLOS

- Alle Anfragen vom Client an den Server werden als voneinander unabhängige Transaktionen betrachtet – Der Server weiß nicht, dass mehrere Anfragen zu einem bestimmten Client gehören
- Wie ist es möglich die Anfragen einem bestimmten Client zuzuordnen (z.B. Artikel im Warenkorb für einen bestimmten Nutzer merken, wenn er weiter auf der Seite surft)?

SESSION TRACKING

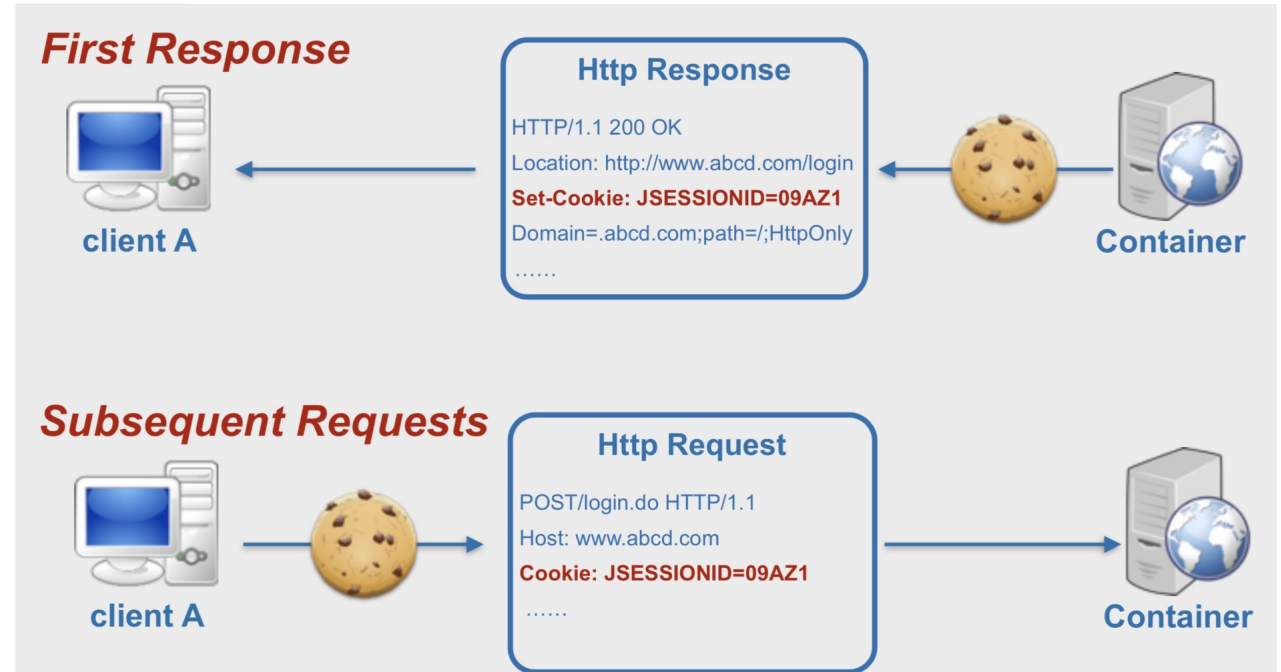
- User ruft demo.com auf
 - Server generiert (bei der ersten Anfrage) eine session-id und
 - Speichert diese in einem Session-Objekt ab
 - Gespeichert in Memory, Datei, Datenbank, etc.
 - Kann alles enthalten (Einkaufskorb, Spielstand, Userinformationen, etc.)
 - Session-ID wird der response hinzugefügt
- Ab jetzt:
 - Enthält jede Anfrage des Clients an den Server die Session-ID
 - Diese wird vom Server verwendet, das Session-Objekt zu finden, um dann beispielsweise Daten aus der Datenbank für einen spezifischen User zu laden oder einen Einkaufskorb anzuzeigen
- Und immer so weiter
- Keine Daten (außer Session-ID) auf dem Client gespeichert

SESSIONS



ÜBERMITTELN DER SESSION-ID IN COOKIES

- Cookies werden http-requests und responses im Head hinzugefügt
- **Set-Cookie:** in der response des Servers: Browser speichert Cookie und schickt es zukünftig bei jedem Request (Cookie:)
- Enthalten die Session-ID



SESSIONS IN NODE UND EXPRESS

- Frameworks abstrahieren vom darunterliegenden Mechanismus
- Frameworks können auch andere Mechanismen für die Session-Verwaltung verwenden (z.B. JSON Web Tokens – JWT, URL-Rewriting, versteckte Formularfelder)
- Express bietet dazu das package `express-session`
- Wir verwenden Session-Cookies (derzeit aktuell so implementiert) merken es aber nicht (API abstrahiert)

express-session

npm v1.17.2 downloads 4.9M/month ci success coverage 100%

Installation

This is a [Node.js](#) module available through the [npm registry](#). Installation is done using the `npm install` command:

```
$ npm install express-session
```

API

```
var session = require('express-session')
```



NEUE VIEWS FÜR SESSIONS - ÜBERBLICK

signup.hbs

Signup

First name

Last name

Email

Password

Register

partials/welcomemenu.hbs

Playlist 4 Signup Login About

Welcome to Playlist!

Playlist 4

A simple demo app to manage song playlists.

login.hbs

Log-in

Email

Password

Login

NEUE VIEWS FÜR SESSIONS - SIGNUP

signup.hbs

Signup

First name

Last name

Email

Password

Register

```
<form action="/register" method="POST">
  <div class="mb-3">
    <label for="inputFirstName" class="form-label">First name</label>
    <input type="text" class="form-control" id="inputFirstName" name="firstName">
  </div>
  <div class="mb-3">
    <label for="inputLastName" class="form-label">Last name</label>
    <input type="text" class="form-control" id="inputLastName" name="lastName">
  </div>
  <div class="mb-3">
    <label for="inputEmail" class="form-label">Email</label>
    <input type="email" class="form-control" id="inputEmail" name="email">
  </div>
  <div class="mb-3">
    <label for="inputPassword" class="form-label">Password</label>
    <input type="password" class="form-control" id="inputPassword" name="password">
  </div>
  <button type="submit" class="btn btn-primary">Register</button>
</form>
```

NEUE VIEWS FÜR SESSIONS - LOGIN

login.hbs

Log-in

Email

Password

Login

```
<form action="/authenticate" method="POST">
  <div class="mb-3">
    <label for="inputEmail" class="form-label">Email</label>
    <input type="email" class="form-control" id="inputEmail" name="email">
  </div>
  <div class="mb-3">
    <label for="inputPassword" class="form-label">Password</label>
    <input type="password" class="form-control" id="inputPassword" name="password">
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```


ROUTEN ZUR DARSTELLUNG DER FORMULARE

```
router.get("/", home.index);  
router.get("/login", accounts.login);  
router.get("/signup", accounts.signup);
```

views/partials/welcomemenu.hbs

Playlist 4 Signup Login About

Welcome to Playlist!

Playlist 4

A simple demo app to manage song playlists.

views/signup.hbs

Signup

First name

Last name

Email

Password

Register

views/login.hbs

Log-in

Email

Password

Login

ROUTER LEITET ANFRAGEN AN CONTROLLER ZUR ANZEIGE DER FORMULARE WEITER

SESSIONS

```
router.get("/", home.index);  
router.get("/login", accounts.login);  
router.get("/signup", accounts.signup);
```

controllers/home.js

```
const home = {  
  index(request, response) {  
    logger.info("home rendering");  
    const viewData = {  
      title: "Welcome to Playlist!"  
    };  
    response.render("index", viewData);  
  },  
};
```

controllers/accounts.js

```
signup(request, response) {  
  const viewData = {  
    title: "Signup for the Service"  
  };  
  response.render("signup", viewData);  
},
```

controllers/accounts.js

```
login(request, response) {  
  const viewData = {  
    title: "Login to the Service"  
  };  
  response.render("login", viewData);  
},
```

ROUTEN FÜR DEN LOGIN UND DIE REGISTRIERUNG WERDEN BEIM ABSENDEN DER FORMULARE AUFGERUFEN

SESSIONS

Neuen Nutzer in der
Datenbank anlegen

```
router.post("/register", accounts.register);  
router.post("/authenticate", accounts.authenticate);
```

Prüfen, ob eine Kombination
aus Nutzernamen
und Passwort gültig ist
Falls korrekt, Session-ID
erzeugen, Session-Objekt
anlegen und Nutzer in der
Session speichern

ACCOUNTS.REGISTER

signup.hbs

Signup

First name

Last name

Email

Password

Register

User aus dem POST-request auslesen –
User-Objekt hat alle Attribute der name-
Attribute aus dem Formular (firstName,
lastName, email, password)

```
async register(request, response) {  
  const user = request.body;  
  await userstore.addUser(user);  
  logger.info("Registering user", user);  
  response.redirect("/");  
},
```

Speichern des Users in der Datenbank
mithilfe des Models userstore.

USER-STORE.ADDUSER

```
async addUser(user) {  
  const query = 'INSERT INTO playlist_users (email, password, first_name, last_name) VALUES($1, $2, $3, $4)';  
  const values = [user.email, user.password, user.firstName, user.lastName];  
  try {  
    await datastoreClient.query(query, values);  
  } catch (e) {  
    logger.error("Error adding user", e);  
  }  
},
```

ACCOUNTS.AUTHENTICATE

login.hbs

Log-in

Email

Password

Login

```
async authenticate(request, response) {  
  let user = await userstore.authenticateUser(request.body.email, request.body.password);  
  if (user) {  
    request.session.user = user.id;  
    logger.info("User successfully authenticated and added to session", user);  
    response.redirect("/dashboard");  
  } else {  
    response.redirect("/login");  
  }  
},
```

Überprüfen, ob Nutzer authentifiziert werden kann (d.h. ob Kombination aus email und password existiert)

Falls nein, weiterleiten zur Loginseite (nochmal versuchen)

Falls ja, Session-ID erzeugen, User im Objekt session anlegen – Ab jetzt schicken Client und Server bei jedem Request / Response ein Session Cookie mit der Session-ID

USER-STORE.AUTHENTICATEUSER

```
async authenticateUser(email, password) {  
  const query = 'SELECT * FROM playlist_users WHERE email=$1 AND password=$2';  
  const values = [email, password];  
  try {  
    let dbRes = await datastoreClient.query(query, values);  
    if (dbRes.rows[0] !== undefined) {  
      return {id: email};  
    } else {  
      return undefined;  
    }  
  } catch (e) {  
    console.log("Error authenticating user", e);  
  }  
},
```

ACCOUNTS.GETCURRENTUSER

```
async getCurrentUser(request) {  
  const user = request.session.user;  
  return await userstore.getUserById(user);  
}
```

helper-Funktion, die den aktuellen User aus der Session ausliest und die Details des Users aus dem Model userstore lädt – Ist kein gültiger User in der Session gibt der userstore undefined zurück

USER-STORE.GETUSERBYID

```
async getUserById(id) {
  logger.info(`Getting user ${id}`);
  const query = 'SELECT * FROM playlist_users WHERE email=$1';
  const values = [id];
  try {
    let dbRes = await datastoreClient.query(query, values);
    logger.info(`Getting user ${dbRes.rows[0].email}`);
    if (dbRes.rows[0] !== undefined) {
      return {id: dbRes.rows[0].email, firstName: dbRes.rows[0].first_name, lastName: dbRes.rows[0].last_name};
    } else {
      return undefined;
    }
  } catch (e) {
    console.log("Error getting user", e);
  }
},
```

DASHBOARD.INDEX

```
async index(request, response) {  
  const loggedInUser = await accounts.getCurrentUser(request);  
  const playLists = await playlistStore.getUserPlaylists(loggedInUser.id);  
  const viewData = {  
    title: "Playlist Dashboard",  
    playlists: playLists,  
  };  
  logger.info("about to render dashboard", playLists);  
  response.render("dashboard", viewData);  
},
```

Holt den eingeloggten User (Zusammenspiel aus Accounts-Controller und User-Store-Model)

Holt die playLists für diesen User

Rendert das Dashboard mit den playLists des Users

DASHBOARD.ADDPLAYLIST

```
async addPlaylist(request, response) {  
  const loggedInUser = await accounts.getCurrentUser(request);  
  const newPlayList = {  
    userid: loggedInUser.id,  
    title: request.body.title,  
  };  
  logger.debug("Creating a new Playlist", newPlayList);  
  await playlistStore.addPlaylist(newPlayList);  
  response.redirect("/dashboard");  
}
```

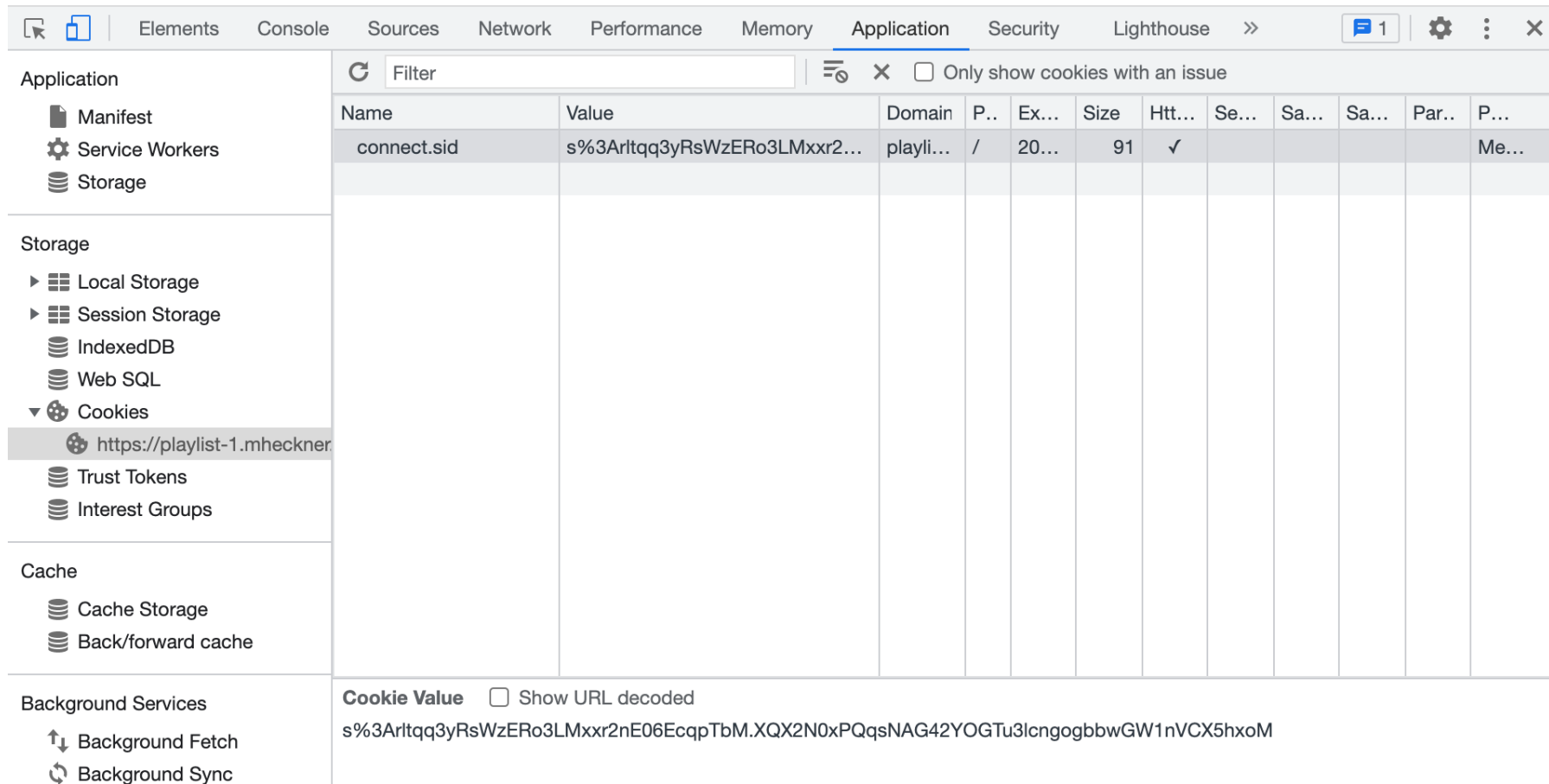
Holt den eingeloggten User

Erstellt Objekt
newPlayList
mit title (aus den
Formulardaten) und
der
id des aktuellen
Users

Leitet zum Dashboard
weiter

Fügt newPlayList
Der Datenbank hinzu

SESSION-COOKIE KANN NACH LOGIN IM BROWSER ANGESEHEN WERDEN



The screenshot shows the Chrome DevTools Application tab. The left sidebar is expanded to 'Cookies' for the URL 'https://playlist-1.mheckner.de'. The main pane displays a table of cookies. One cookie is visible: 'connect.sid' with a long alphanumeric value. Below the table, the 'Cookie Value' is shown in its raw, URL-encoded form.

Name	Value	Domain	P..	Ex...	Size	Http...	Se...	Sa...	Sa...	Par...	P...
connect.sid	s%3Arltqq3yRsWzERo3LMxxr2...	playli...	/	20...	91	✓					Me...

Cookie Value ☐ Show URL decoded
s%3Arltqq3yRsWzERo3LMxxr2nE06EcqpTbM.XQX2N0xPQqsNAG42YOGTu3lcngogbbwGW1nVCX5hxoM

LOGOUT

Session für den zuvor eingeloggten User wird zerstört – Beim nächsten Request Dieses Users ist das Objekt session nicht mehr gesetzt – User ist ausgeloggt

```
logout(request, response) {  
    request.session.destroy();  
    response.redirect("/");  
},
```

Weiterleitung auf die Startseite

FAZIT

- Server erzeugt Session-ID und Session-Objekt und speichert darin beliebige Informationen
- Session-ID wird zwischen Client und Server bei jedem request und jeder response mitgeschickt
- Server kann anhand der Session-ID Daten für den Client aus dem Session-Objekt laden
- In der Playlist-App arbeiten Controller und Model zusammen, um Nutzer zu registrieren, einzuloggen und ausuloggen
- Nach Logout löscht der Server das Session Objekt und schickt keine weiteren Session Cookies mehr – Der User ist ausgeloggt

QUELLEN

Express Sessions. <https://www.npmjs.com/package/express-session>

Express Route Parameters.

<http://expressjs.com/en/guide/routing.html#route-parameters>

W3schools.com. SQL Injection. Online verfügbar unter:

https://www.w3schools.com/sql/sql_injection.asp