

NodeJS Part 6

Sessions & Cookies

jan.schulz@devugees.org

Agenda

1. Stateless VS Stateful
2. Cookies & Sessions
3. Session Authentication
4. Task

1. Introduction

- Stateless



1. Introduction

- Stateless



She does not know what you ordered last week.



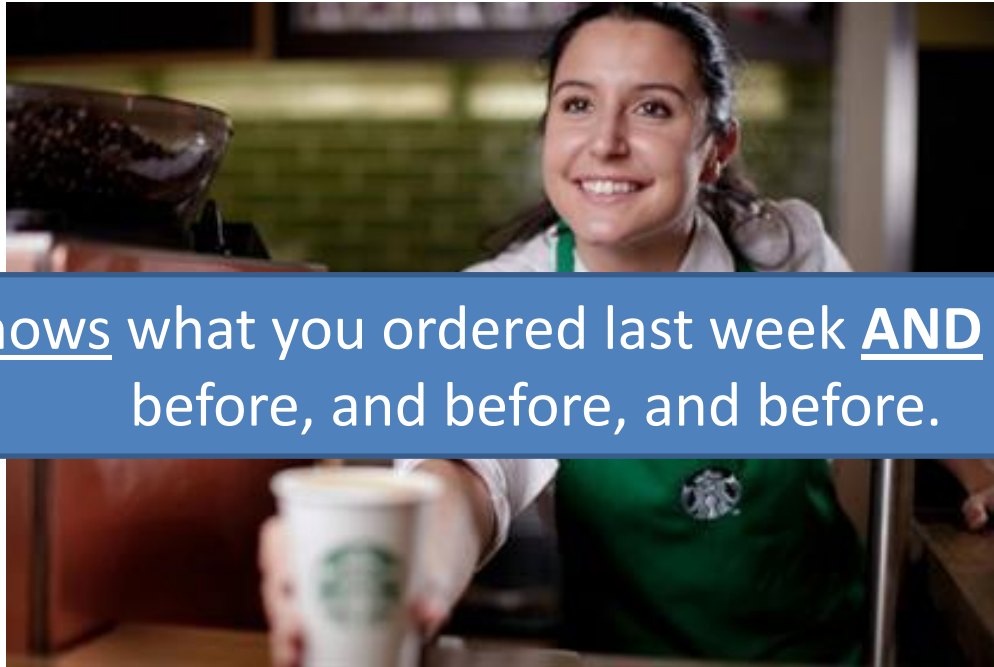
1. Introduction

- Stateful



1. Introduction

- Stateful



She knows what you ordered last week AND the week before, and before, and before.

1. Introduction

- **Request:** HTTP packet sent from client to server („One coffee please“)

1. Introduction

- **Request:** HTTP packet sent from client to server („One coffee please“)
- **Response:** HTTP packet sent back from server to client after it received a Request („There you go – here it is“)

1. Introduction

- **Request:** HTTP packet sent from client to server („One coffee please“)
- **Response:** HTTP packet sent back from server to client after it received a Request („There you go – here it is“)
- **Transaction:** One pair of Request and Response („One coffee please“ – „There you go – here it is“)

1. Introduction

- **Request:** HTTP packet sent from client to server („One coffee please“)
- **Response:** HTTP packet sent back from server to client after it received a Request („There you go – here it is“)
- **Transaction:** One pair of Requests and Responses („One coffee please“ – „There you go – here it is“)
- **Session:** A set of transactions
 - „One coffee please“ – „There you go – here it is“)
 - „One latte please“ – „Okay – here it is“)
 - „One espresso please“ – „Okay.“)

2. Cookies & Sessions

- **Cookies:** is an ID that is sent in the HTTP-Header and identifies a session (saved mostly client-side).
- **Session:** A set of transactions, saved server-side mostly in the memory, as a file or on the database.
- No JavaScript necessary. When a browser receives a reponse, it automatically saves the cookie in all future requests.
- Getting rid of cookies: delete them in browser configs.

3. Session Authentication (1/4)

USER

HTTP
SERVER

POST /login

```
{  
  username: 'hallo',  
  password: 'world'  
}
```



3. Session Authentication (2/4)

USER

HTTP
SERVER

POST /login

```
{  
  username: 'hallo',  
  password: 'world'  
}
```



RESPONSE /login

Header: Cookie-ID: 123

```
{  
  
}
```



username + password
exists. create a
session object and send
back cookie.

3. Session Authentication (3/4)

USER

HTTP
SERVER

GET /content

Header: Cookie-ID: 123

{

}



3. Session Authentication (4/4)

USER

HTTP
SERVER

GET /content

Header: Cookie-ID: 123

{

}



RESPONSE /content

Header: Cookie-ID: 123

{

<div>secret content</div>

}



4. Task



4. Task

Create a small Too-late-comer website with Webpack, JQuery, Bootstrap and MongoDB. The user can keep track of students who come too late.

- 1) It has three navigation points.
 - 1) Logout
 - 2) New Later-Comer (a form where a new too-late comer can be saved)
 - 3) History (shows a history of people coming too late)

By default, 3) is the landing page

4. Task

1) New Too Later Comer GUI

The image shows a graphical user interface (GUI) window titled "Too-Late-Comer". The window has a light gray background and a dark gray border. Inside the window, there are three main components centered vertically and horizontally:

- A white rectangular input field with a dark gray border containing the text "Username".
- A second white rectangular input field with a dark gray border containing the text "Password", positioned directly below the first field.
- A dark gray rectangular button with a dark gray border and a slight shadow, containing the text "Login", positioned below the password field.

4. Task

1) New Too Later Comer GUI

[New Too Later Comer](#) | [History](#) | [Logout](#)

Max Mustermann

12.02.2018

9:35

Add

4. Task

1) History

New Too Later Comer <u>History</u> Logout				
Name	Date	Time	Minutes too late	Action
Max Mustermann	11.02.2018	09:30	15	Delete
Julia Müller	11.02.2018	09:45	30	Delete

4. Task

1) Login/Logout

Too-Late-Comer

You are logged out now.

Login

4. Task

2. Extend the /login POST that it looks up username and password on a DB of your choice -> Mongo or MySQL. Also add an information that logs WHEN the user has logged in -> date and time.

4. Task

3. Add a signup form to the start page

Too-Late-Comer

Email

Password

Login

Email

Password

Repeat Password

Sign Up

4. Task

3. Implement an signup process on the backend:

- a) Change your user model that username will be replaced by email. Add a field `activationCode` (varchar/string) and a field `active` (boolean) to your database.
- b) Create a POST route `/signup` that accepts an email, a password and the repeated password. Check if the body parameters exist and if the password equals the repeated password. Otherwise, return an error.
- c) Create a new record on the database with an `activationCode` of a randomstring with length 20 and `active` set to false
- d) When the user is created, send an email to the email address using the code on the next slide. In the mail, the user should see a text like:

„Thanks for your registration. Please verify your account by clicking on the following link: [http://localhost:3000/?activate=\\${ACTIVATIONLINK}](http://localhost:3000/?activate=${ACTIVATIONLINK}) „

4. Task

```
//nodemailer.js

var nodemailer = require('nodemailer');

function sendMail(recipientAddress, subject, body) {
  var smtpConfig = {
    host: 'smtp.gmail.com',
    port: 465,
    secure: true,
    auth: {
      user: 'devugeesshop1234@gmail.com',
      pass: 'devugees2018'
    }
  };
  nodemailer.createTransport(smtpConfig);
  var mailOptions = {
    from: '" TooLate App" <devugeesshop1234@gmail.com>',
    to: recipientAddress,
    subject: subject,
    text: body,
    html: body
  };
  transporter.sendMail(mailOptions, function(err, info) {
    if(err) console.log('mail was not delivered');
  });
}

module.exports.sendMail = sendMail;
```

4. Task

3. Implement the activation process on the backend:

- a) Create a GET route `/activate` with one parameter **activationLink**. Look for a user on the database with the **activationLink** and if one is found, set active to true. Return a success result (i.e. `error = 0`) if that worked.

4. Task

4. Implement the activation process on the frontend:

a) When the page loads, check if your URL looks like this:

<http://localhost:3000/?activate=123456>

b) Take a look at `window.location.href` and parse out 123456. Then, send this information to the backend and if the activation was successful, tell the user by showing an alert message „Your account is active now“.

5. Task

Make a copy of toolate-app and create a JWT version of it -> toolate-app-jwt.

Note: When the user logs out, its sufficient to remove the token from the localStorage.