Online Multimedia

Winter Semester 2019/20

Tutorial 12 and 13 – Repetition

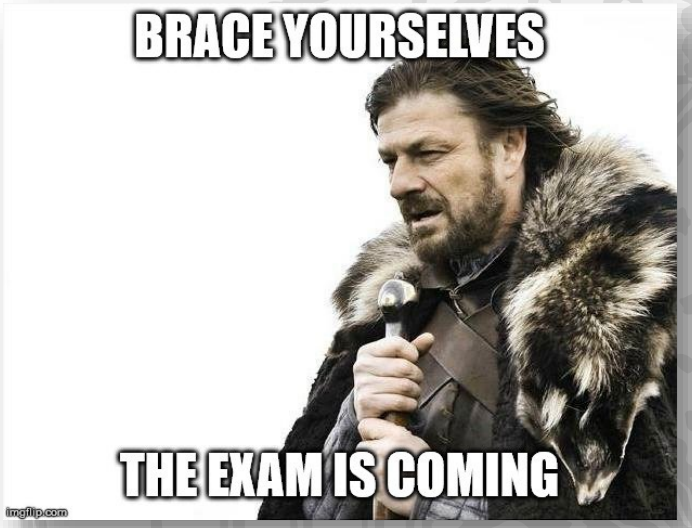
Today's Agenda

- The OMM Technology Landscape
- Javascript
- React
- REST APIs
- NodeJS
- Databases

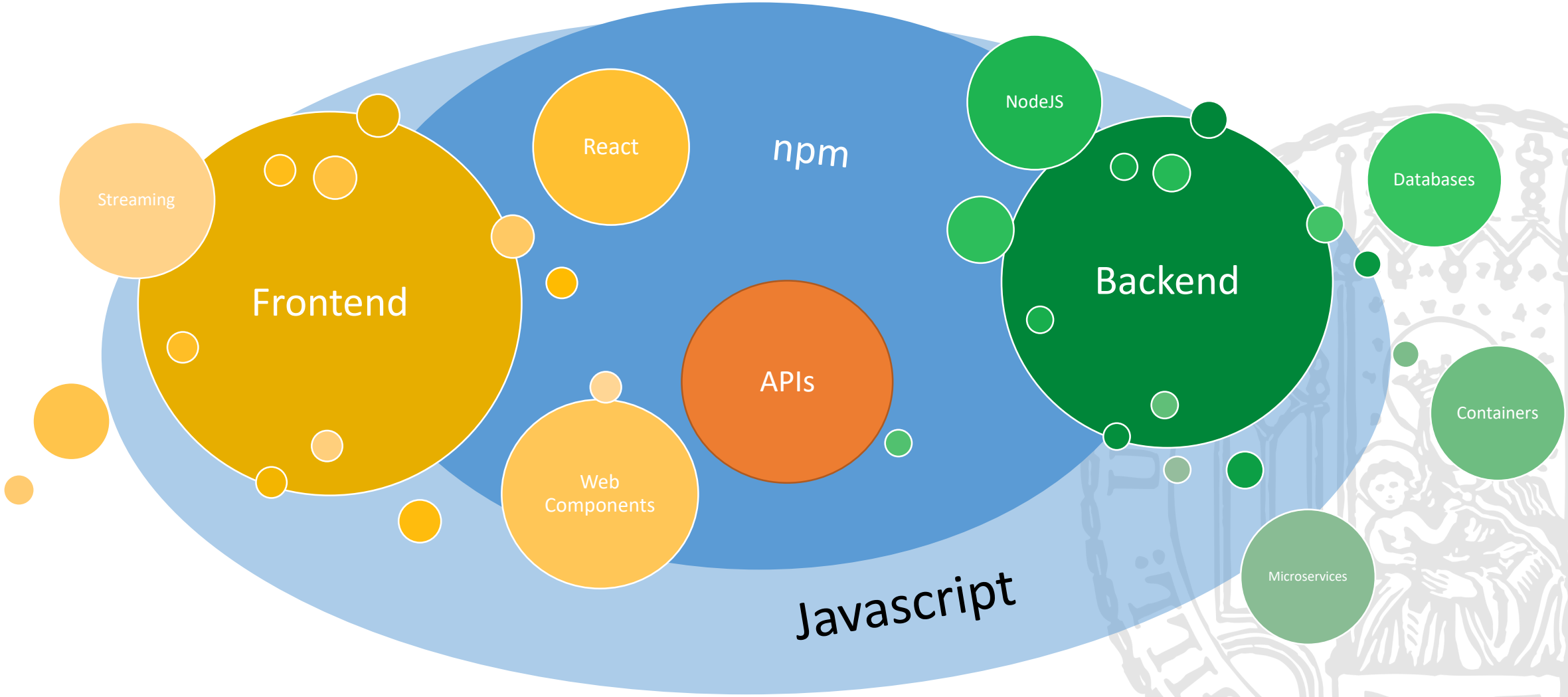
Exam

- The exam will take place on **Feb. 11th**
- You have to register via Uni2Work until **Feb. 5th**
- If you do not register you cannot participate in the exam

- The exam will be **open book**
- You **can** bring printed or handwritten notes, slides, etc.
- You **cannot** bring electronic devices



The OMM Technology Landscape



The OMM Technology Landscape

In which technolog(y|ies) would you implement…?

- Persisting tickets on a train booking website
- A highly interactive UI
- Restricting access to a system to devices within a company network only
- A login, registration and password reset UI that you want to use in multiple projects
- Sketch a structuring concept of an application with server + client app(s)
- Include components of other open source projects into yours

Javascript

Key Take-Aways:

- An understanding for asynchronous code execution
- Some functional programming experience
- A notion of the challenges that come with dynamic typing and many many dependencies

What concepts do you recognize?

javascript/someScript.js

```
const textField = document.querySelector('input[type="text"]')

textField.addEventListener('input',e => {
    let newContent = e.target.value;

    fetch(`https://www.somesearchengine.de?query=${newContent}`)
        .then(responseJson => {
            responseJson.json()
                .then(responseData => {
                    responseData.searchResults.forEach(aResult => {
                        const newEl = document.createElement('li')
                        newEl.innerHTML = aResult
                        document.getElementById('resultsList').appendChild(newEl)
                    })
                })
                .catch(error => {
                    console.error('Request to server failed',error)
                })
        })
        .catch(error => {
            console.error('Parsing JSON failed',error)
        });
})
```

- Promises
- Callbacks
- DOM Manipulation
- Event Handling
- Var, let, const
- Ajax

…

And where are the mistakes?

javascript/someScript.js

```
const textField = document.querySelector('input[type="text"]')

textField.addEventListener('input',e => {
    let newContent = e.target.value;

    fetch(`https://www.somesearchengine.de?query=${newContent}`)
        .then(responseJson => {
            responseJson.json()
                .then(responseData => {
                    responseData.searchResults.forEach(aResult => {
                        const newEl = document.createElement('li')
                        newEl.innerHTML = aResult
                        document.getElementById('resultsList').appendChild(newEl)
                    })
                })
                .catch(error => {
                    console.error('Request to server failed',error)
                })
        })
        .catch(error => {
            console.error('Parsing JSON failed',error)
        });
})
```

javascript/someWebPage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <input type="text">
    <include href="someScript.js"></include>
    <ul id="resultsList">

    </ul>
</body>
</html>
```

Response json of: https://www.somesearchengine.de?query=Online Multimedia

```
{
    "data": {
        1: "Online Multimedia - LMU",
        2: "The Online Multimedia Crash Course - YouTube",
        3: "How to master your multimedia project online?"
    }
}
```

And where are the mistakes?

javascript/someScript.js

```
const textField = document.querySelector('input[type="text"]')

textField.addEventListener('input',e => {
    let newContent = e.target.value;

    fetch(`https://www.somesearchengine.de?query=${newContent}`)
        .then(responseJson => {
            responseJson.json()
                .then(responseData => {
                    responseData.searchResults.forEach(aResult => {
                        const newEl = document.createElement('li')
                        newEl.innerHTML = aResult
                        document.getElementById('resultsList').appendChild(newEl)
                    })
                })
                .catch(error => {
                    console.error('Request to server failed',error)
                })
        })
        .catch(error => {
            console.error('Parsing JSON failed',error)
        });
})
```

javascript/someWebPage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <input type="text">
    <include href="someScript.js"></include>
    <ul id="resultsList">

    </ul>
</body>
</html>
```

Response json of: https://www.somesearchengine.de?query=Online Multimedia

```
{
    "data": {
        1: "Online Multimedia - LMU",
        2: "The Online Multimedia Crash Course - YouTube",
        3: "How to master your multimedia project online?"
    }
}
```

Javascript

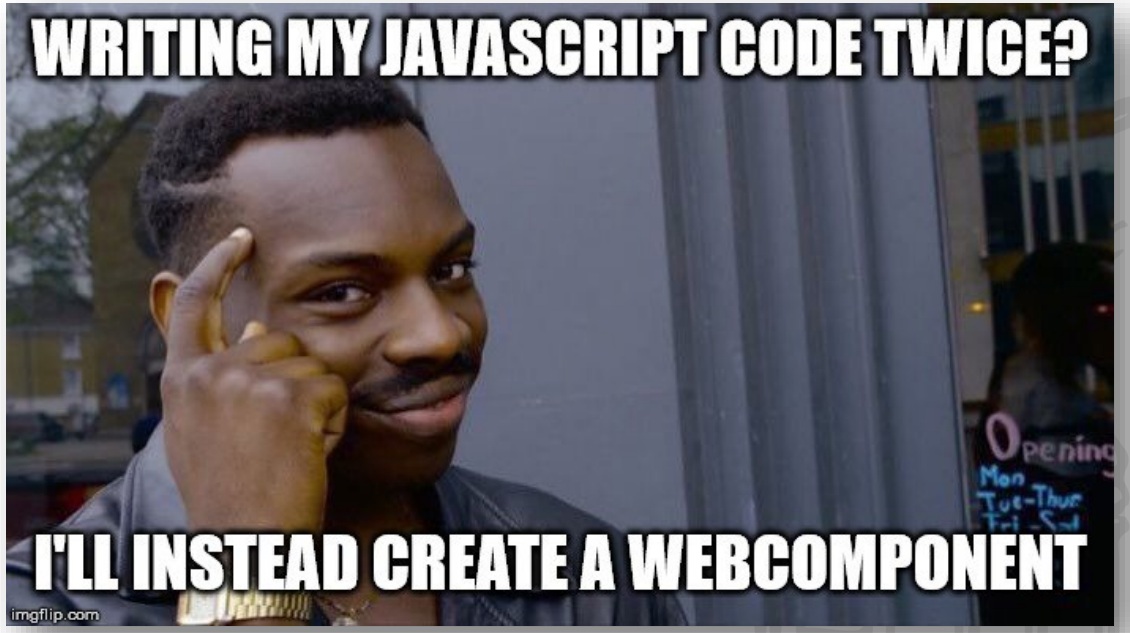
How would you implement…?

- a website making 3 API calls in parallel, and only continuing when all 3 responses are fetched
- A conversion of the left JS object into the right without a for loop?

```
[
    {name:"Max Mustermann"},
    {name:"Thomas Tester"},
    {name:"Fred Feuerstein"}
]
```

```
[
    {firstname:"Max",lastname:"Mustermann"},
    {firstname:"Thomas",lastname:"Tester"},
    {firstname:"Fred",lastname:"Feuerstein"}
]
```

Web Components



Web Components

Custom Elements

```
<my-element></my-element>
```

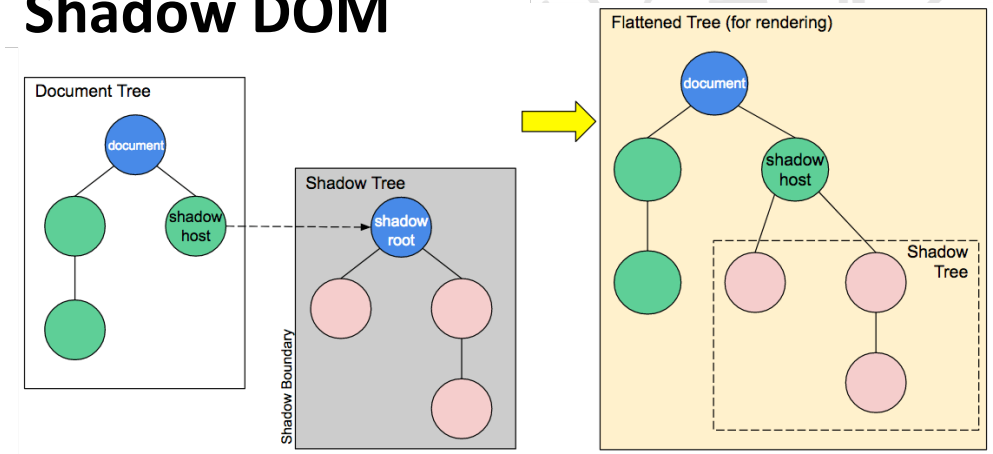
HTML Templates

```
<template>
  This content is rendered
  later on.
</template>
```

HTML Imports

```
<link rel="import"
  href="my-code.html">
```

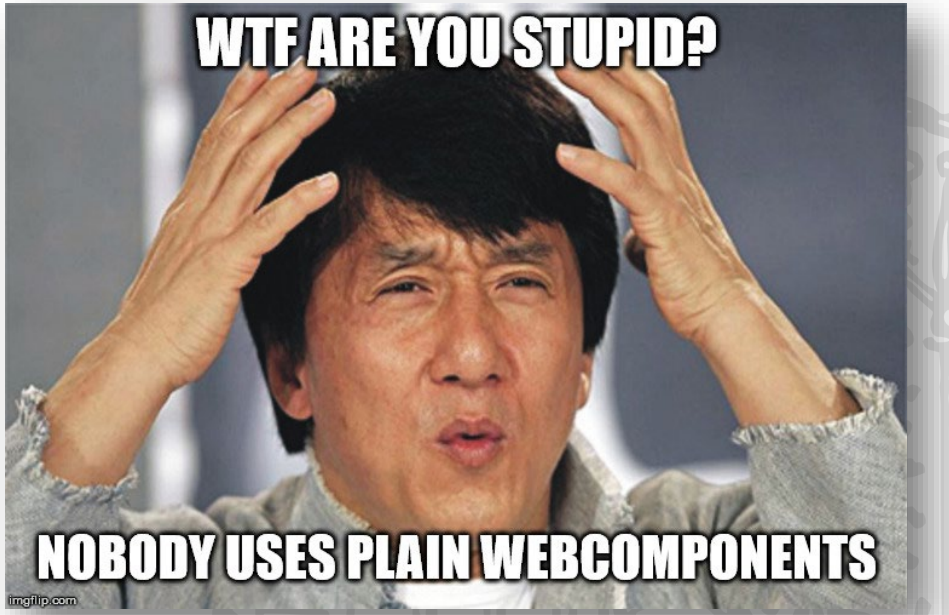
Shadow DOM



https://mdn.mozillademos.org/files/15788/shadow-dom.png

React

- Web Components require lots of boilder plate code
- React is a library to complement Web Components



React

counter-react/src/App.tsx

```tsx
import React from 'react';
import './App.css';
import OmmCounter from './components/omm-counter/omm-counter';

const App: React.FC = () => {
  return (
    <div className="app">
      <div className="header">
           <h1>Counter - React</h1>
      </div>
      <OmmCounter />
    </div>
  );
}

export default App;
```

counter-react/src/components/omm-counter.tsx

```tsx
import React, {Component} from 'react'

export class OmmCounter extends Component {
  state = { count: 0 }
  inc = () => {
     this.setState({count: (this.state.count + 1)}) }
  dec = () => {
     this.setState({count: (this.state.count - 1)}) }
  render() {
    return (
      <div>
        <span>{ this.state.count }</span>
        <div>
          <button onClick={this.inc}>+</button>
          <button onClick={this.dec}>-</button>
        </div>
      </div>
    )
  }
}
```

Web Component vs. React Component

```
<template> <!-- Defines element markup -->
    <img src=""/>
    <p class="caption captionTop"></p>
    <p class="caption captionBottom"></p>
    <style>…</style>
</template>

<script>
(function(window, document, undefined) {
var thatDoc = document;
var thisDoc =  (thatDoc._currentScript || thatDoc.currentScript).ownerDocument;

var template = thisDoc.querySelector('template').content;
var MyElementProto = Object.create(HTMLElement.prototype);

    MyElementProto.createdCallback = function() {

        var shadowRoot = this.createShadowRoot();
        var clone = thatDoc.importNode(template, true);
        shadowRoot.appendChild(clone);


 …     // same for captionBottom and memeTemplateUrl
        if (this.hasAttribute('captionTop')) {
            var captionTop = this.getAttribute('captionTop');
            this.setCaptionTop(captionTop);
        }
 …
};

    // Sets new value to the attributes
    MyElementProto.setCaptionTop = function(val) {
        this.captionTop = val;
        this.captionTopParagraph = shadowRoot.querySelector('p.captionTop‘);
        this.captionTopParagraph.textContent = this.captionTop;
    };
 …    // same for captionBottom and memeTemplateUrl

    window.MyElement = thatDoc.registerElement('my-meme‘, {
        prototype: MyElementProto
    });
})(window, document);
</script>
```

webcomponent-example/my-meme.html

```
import React from 'react';
import './Meme.css';

class Meme extends React.Component {
    render() {
        return (
            <div className="meme">
                <img src={this.props.imgSrc}/>
                <p className="captionTop">{this.props.captionTop}</p>
                <p className="captionBottom">{this.props.captionBottom}</p>
            </div>
        )
    }
}

export default Meme;
```

webcomponent-example/meme.jsx

Breakout #1

In *react-breakout* you find a basic React app that renders a list of dog names and ages

1. Improve the duplicate code at the creation of *Dog* elements in *App.tsx* with something better

2. Add a „+“ button to each dog. Implement that clicking this button increases a dog‘s age (the change should be propagated to the „data store“ in *App.tsx* of course)

Timeframe: 10 Minutes

React Functional Components

Class-based component

```
class MyComponent extends React.Component {
    render () { return (<div>TSX</div>) }
}
```

Functional component

```
const c: React.FC = () => {
    return (<div>TSX</div>)
}
```

- No class needed for each component
- The whole component is just a function, finally returning its DOM
- But: no class => not state. Thus React provides the **useState** hook:

  An array containing the current value and a setter method

React Functional Components

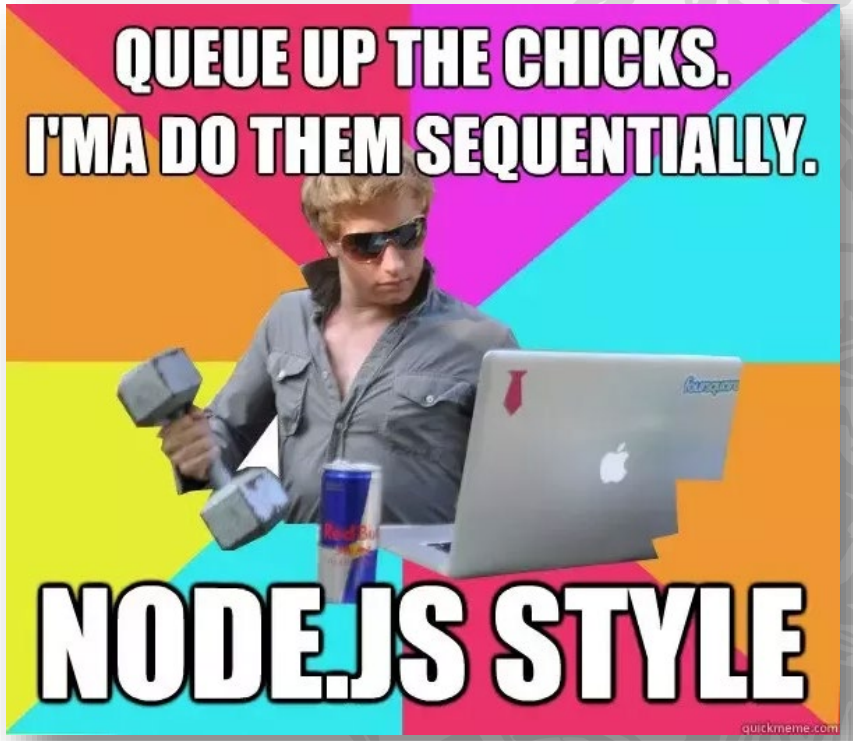
```jsx
import React, {Component} from 'react'

class OmmCounter extends Component {
  state = { count: 0 }
  inc = () => { this.setState({count: (this.state.count + 1)}) }
  dec = () => { this.setState({count: (this.state.count - 1)}) }
  render() {
    return (
      <div><span>{ this.state.count }</span>
          <button onClick={this.inc}>+</button>
          <button onClick={this.dec}>-</button>
        </div></div>
    )  }}
export default OmmCounter;
```

react/functional-component/omm-counter-classbased.jsx

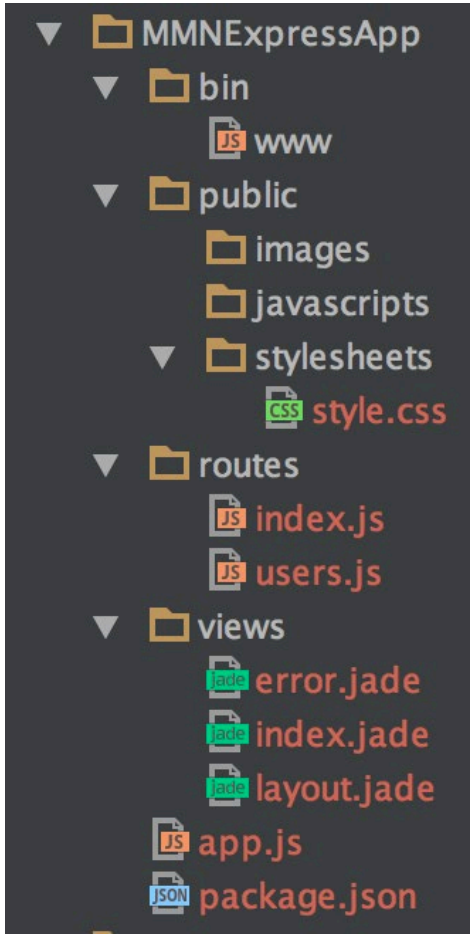
```jsx
import React, { useState } from 'react‘;

const OmmCounter: React.FC = () => {
  const [getter, setter] = useState({counter: 0});
  const inc = () => {setter({counter: getter.counter + 1})};
  const dec = () => {setter({counter: getter.counter - 1})};
  return (
    <div><span className="counter-state">{ getter.counter }</span>
      <button onClick={inc}>+</button>
      <button onClick={dec}>-</button>
    </div></div>
  )};
export default OmmCounter;
```

react/functional-component/omm-counter-functional.jsx

NodeJS


QUEUE UP THE CHICKS.
I'MA DO THEM SEQUENTIALLY.
NODE.JS STYLE
quickmeme.com

NodeJS Express Apps



- What are those files doing?

Node Request Flow

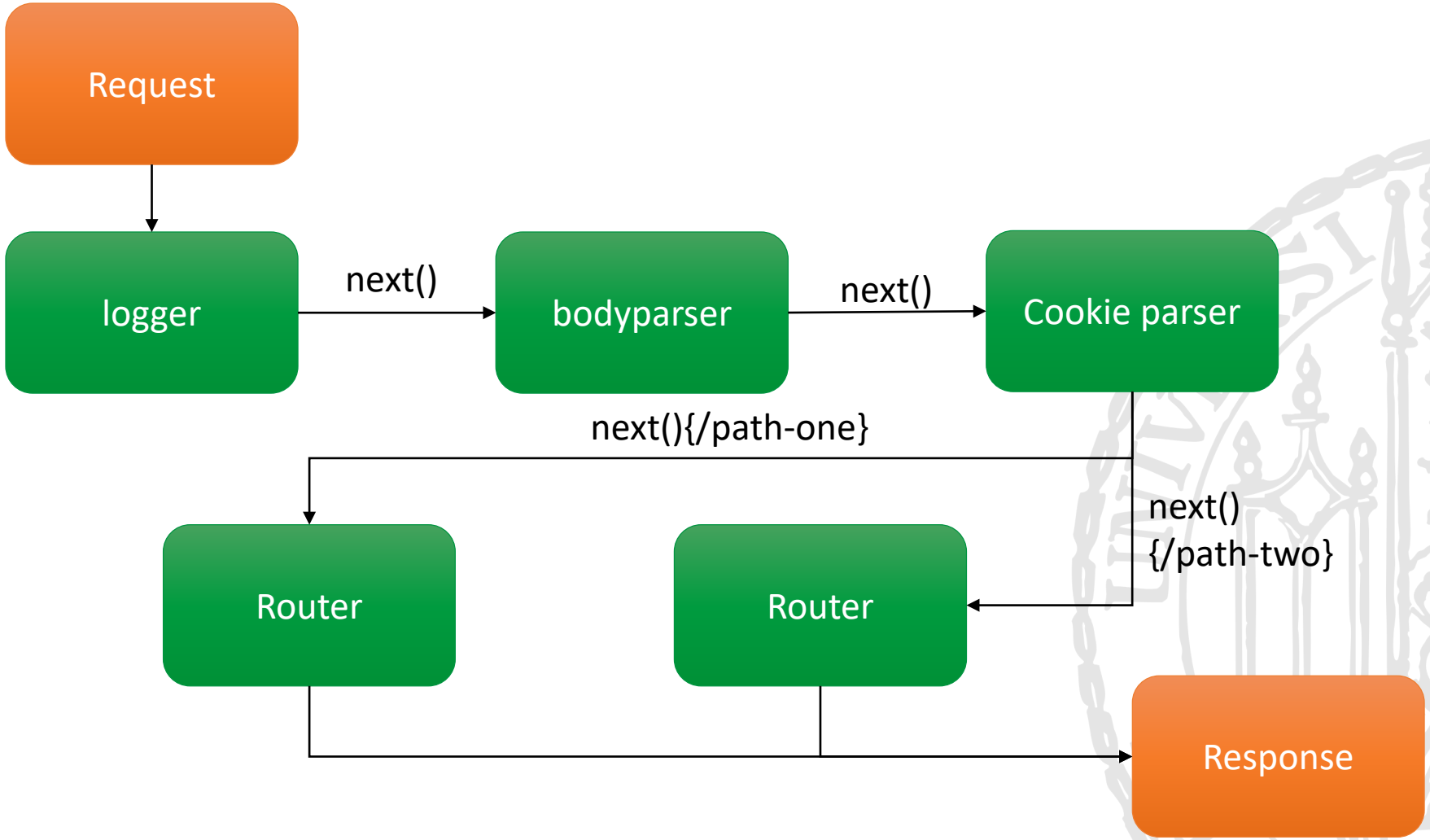
POST Request to
localhost:3000/images/landscape

```
var app = express();

var indexRouter = require('./routes/index');
var imagesRouter = require('./routes/images');

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use(fileUpload());

app.use('/', indexRouter);
app.use('/images', imagesRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  …
  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

```
var express = require('express');
var router = express.Router();

router.post('/:category', (req,res) => {
    …
    console.log(`location: ${req.body.location}, category: ${req.params['category']}`)

    res.status(201);
    res.setHeader('Location',`/images/${req.params['category']}`);
    res.send();
});

module.exports = router;
```

Middleware Visualization



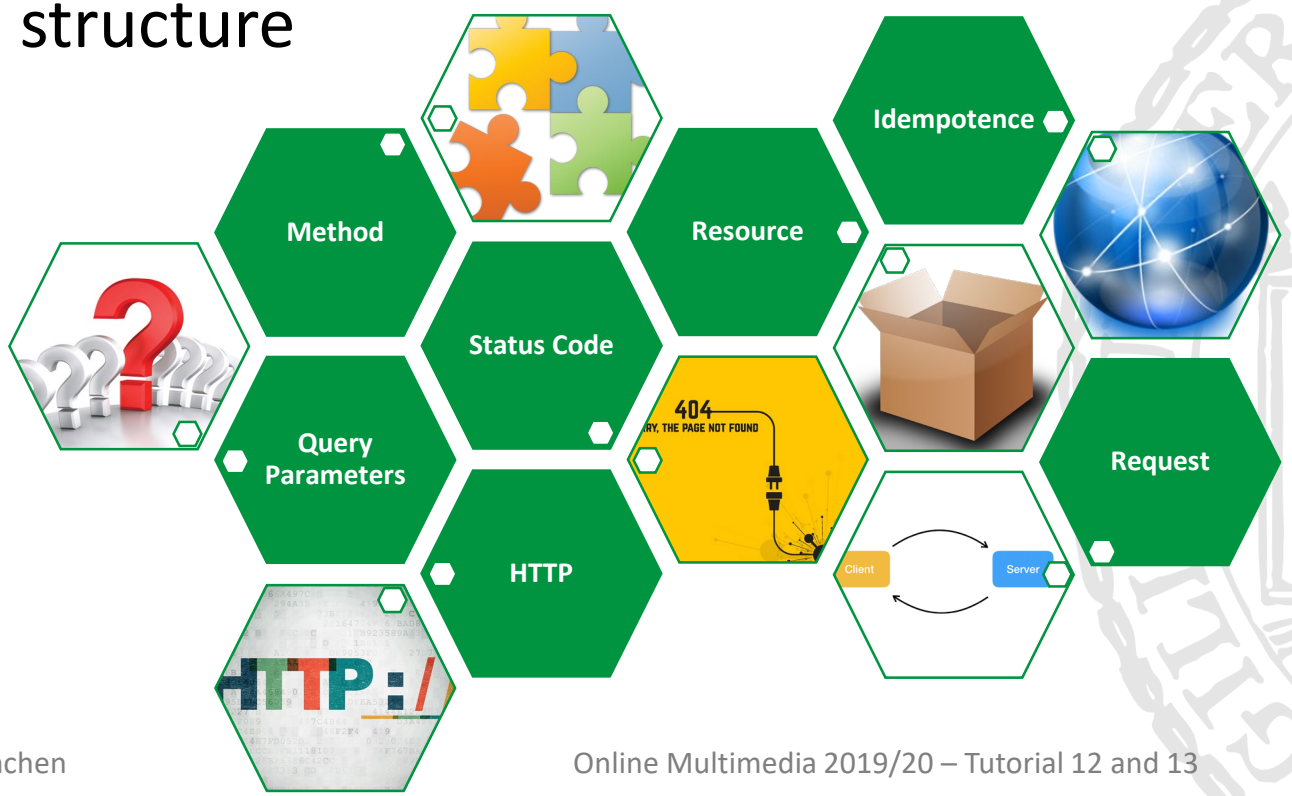
Middlewares in NodeJS

- A middleware is a function that sits between the request and the response ("in the middle")
- Usually more than one middleware per route (middleware chain)

```
app.use('/', (req, res, next) => {
  // do something with req.object
  // then either send the response or call:
  next();
});
```

NodeJS: Building APIs

- APIs are the interface between server and client
- A well-designed REST architecture is an important step towards good system + code structure



Breakout #2

Build your own API: The Mountaineering Diary

Users should be able to:

- Show all activities in a region
- Filter activities by type (hike, climb, alpine tour, …)
- Add new activities, edit and delete existing ones. Activities consist of an author, some text content, and a picture url

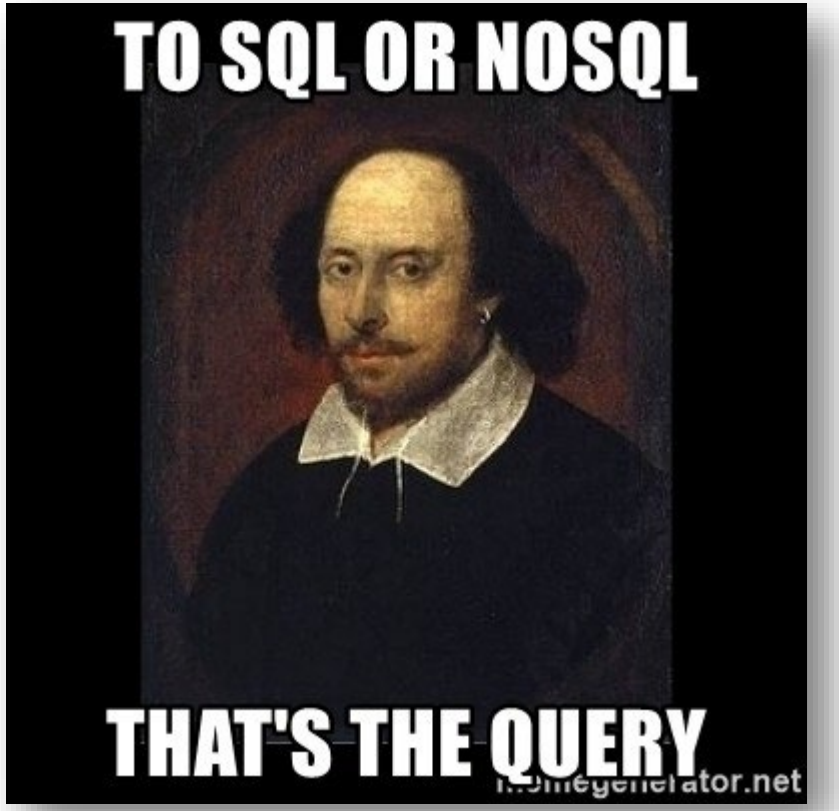
1.: Imagine the mountaineering diary website and design the backend‘s API

2.: Create an empty Express project, and implement your API (fake data is fine so far)

Timeframe: 7 + 13 Minutes

Databases in Web Applications

SQL vs. NoSQL – What‘s the Difference?



Pro-Document-Based:

- No ORM mapping
- No tons of empty fields
- Scalability

Pro-Relational:

- Structure can be guaranteed
- No duplicate data

Questions?

Please let us know your questions soon!

We will try to incorporate them in the next repetition tutorial session.