2020_projektarbeit_ws2020_by_go

Test Accounts / Profile

Twitter

Projektarbeit WS2020 by GO (@ByWs2020)

Facebook

Projektarbeitwszwanzigzwanzig bygo

Instagram

Projektarbeitws2020 bygo (@projektarbeitws2020bygo)

Repo - github

ozwoldFH/2020_projektarbeit_ws2020_by_go

Was?

Im zweiten Zwischenbericht gibt es nicht viel zu erzählen, da eher weniger gemacht wurde. Es wurden einige Tutorials abgeklappert und viele verschiedene Technologien implementiert. Mehr dazu wird im Bereich "Wie?" erklärt.

Nächste Schritte wäre es statt Testdaten mit der wirklichen Facebook und Instagramm Graph API zu arbeiten. Der User soll sich einloggen können und dann nochmals bei Facebook und Instagramm. Dann sollten die Posts automatisch erscheinen und man kann Funktionen ausführen wie Liken/Daumen hoch oder ähnliches.

Warum?

Warum werden hier so viele verschiedene Technologien verwendet? Zum einen ist das nicht wirklich viel und zum anderen will ich sehen, ob ich mich im Nachhinein mich mit dem Programm noch auskenne. Ich hatte immer Probleme den Zusammenhang verschiedener Libraries zu sehen und jetzt bin ich einem großen Projekt gefolgt und kann durchaus die Vorteile sehen.

Wie?

Hier werden folgende Technologien verwendet:

```
npm install react react-dom -save
```

- React für User Interfaces

- ReactDOM um nur die ".render" Methode aufzurufen. Der nimmt React Component und "glueed" es auf ein DOM fürn den Web-Browser

`npm install webpack webpack-cli` webpack -> link & link2

- Im Prinzip packt es mehrere JavaScript Teile zu einem Packet zusammen.
- webpack-CLI -> cmd Version

`npm install babel-loader @babel/node @babel/core @babel/preset-env @babel-preset-react @babel/plugin-proposal-class-properties`

- wird verwendet, um neueste Funktionen in ES zu nutzen und Babel wandelt es automatisch so um, dass alte Browser damit zurechtkommen.
- babel-load -> für Babel und Webpack
- @babel/node -> CLI? Wird das überhaupt gebraucht?
- @babel/core -> Babel Compiler
- @babel/preset-env -> eine Voreinstellungen/Konfiguration, damit es für ein bestimmet "Target" Browser funktioniert. Babel plugins und polyfills werden automatisch gewählt-> Brauchen wir das?
- @babel-preset-react -> damit JSX (in React) und Babel funktioniert -> muss nicht unbedingt genutzt werden
- @babel/plugin-proposal-class-properties -> Vereinfacht uns Properties für Klassen zu schreiben: link

`npm install nodemon`

- Bei Veränderungen müssen wir den Node Server immer wieder neustartet. Dieses package monitored unsere Files und sobald wir speichern klicken, wird der Server automatisch neugestartet.#

`npm install eslint babel-eslint eslint-plugin-react`

- eslint ist dafür da, um Fehler im Code besser zu finden
- babel-elisnt -> damit es mit babel funktioniert
- eslint-plugin-react -> damit eslint die empfohlene Settings für React einstellt.

`npm install --save prop-types`

- ist nicht mandatory, aber eine Empfehlung -> ähnlich wie TypeScript, können wir sagen, welche Typen (string, isRequired) notwendig sind, für ein bestimmtes React Component

`"start": "node index.js"` -> `"start": "nodemon --exec babel-node server.js --ignore public/"` geändert: nodemon, damit es automatisch ausführt. exec babel-node, damit Babel genutzt werden kann mit JSX und ignore, damit public folder ignoriert wird bei changes.

- Nodemon soll ja der Server nur neustarten, wenn im "src/" Folder etwas geändert wird.

`npm install ejs` fürs Frontend, damit wir unser html einfach ausplitten können. Es ist ein Template Engine.

`npm install json-loader`

- damit webpack json versteht.
- Brauchen wir anscheinend nicht, da es eigentlich ab Version 2 oder größer schon bereits das inkludiert hat.
- Brauch ich doch -> bin später auf ein Fehler gekommen, den man mit webpack einfach lösen kann

`npm install node-sass-middleware`

- gibt's zwei Arten wie wir sass umwandeln -> mit webpack oder mit nodeJS
- Mit dem obigen Kommando, machen wir das gleich über nodeJS (perfomanter).

`npm install axios`

- Promise based HTTP client for the browser and node.js
- Brauchen wir, um vom Client auf die API eine Anfrage zu schicken.

`webpack.config.js`

- hier wird unser entry point eingestellt
- wo unser ganzer "bundle" hinkommen soll
- Wir verwenden hier auch unser "babel-loader", damit webpack unser Babel Code in JSX auch richtig umwandelt.

`babel.config.js`

- Config für Babel, damit er die Presets für React nimmt und /env, damit alle Browser kompatibel sind.
- Plugin auch, damit wir die Klassen einfacher schreiben können
- -> ich dachte, dass das automatisch geht, aber anscheinend muss man eine Konfig schreiben.

`.eslintrc.js` -> Config für eslint

- Aus einem Tutorial entnommen.
- Wir müssen das hier konfigurieren, da wir Babel verwenden.

`npx babel-node server.js`

- Um den Server zu starten

BUG

ES modules funzt nicht so gut wie mit CommonJS -> link

- Beispiel:
    - CommonJS: import apiRouter from "./api"; sucht automatisch nach index.js
    - ES modules: import apiRouter from "./api/index.js"; index.js muss man angeben
- Template Engines:
    - Handlebars {{}}
    - pug / jade -> whitespace
    - EJS -> sehr ähnlich zum Microsoft Classic ASP.NET -> verwende ich im jetzigen Projekt