full-stack.md

Full-Stack App basics mit React & NodeJs

Schritt 1: Einrichten deiner Node.js-Serverumgebung

Stelle sicher, dass Du Node.js und NPM auf Deinem Computer installiert hast. Du kannst es von nodejs.org herunterladen und installieren. Überprüfe die Installation, indem Du die folgenden Befehle in Deiner Befehlszeile ausführst:

```
node -v
npm -v
```

Schritt 2: Initialisieren eines neuen Node.js-Projekts

a) Öffne Deine Befehlszeile und erstelle das Hauptverzeichnis für Dein Projekt, `my-fullstack-app` und wechsle hinein:

```
PROJ_NAME=my-fullstack-app && mkdir $PROJ_NAME && cd $PROJ_NAME
```

b) Erstelle das Server-Verzeichnis:

```
mkdir server
```

c) Erstelle das Client-Verzeichnis:

```
mkdir client
```

d) Wechsle in das Server-Verzeichnis:

```
cd server
```

5. Initialisiere Dein Node.js-Projekt mit `npm init -y`:

```
npm init -y
```

Schritt 3: Installiere Express.js

Installiere Express.js, um Deinen Node.js-Server zu erstellen:

```
npm install express
```

Schritt 4: Erstelle eine einfache Node.js-Anwendung mit Express

Erstelle eine JavaScript-Datei, z.B. `server.js`, und importiere Express. Konfiguriere dann Express und starte den Server:

```
const express = require('express');
const app = express();
```

```
const port = 8080;

app.get('/', (req, res) => {
  res.send('Hallo von Express!');
});

app.listen(port, () => {
  console.log(`Server läuft auf Port ${port}`);
});
```

Starte den Server mit dem folgenden Befehl:

```
node server.js
```

Dein Express-Server sollte jetzt unter `http://localhost:8080` erreichbar sein.

Um den Server bei Änderungen nicht jedes Mal neu starten zu müssen, empfiehlt es sich `nodemon` zu installieren:

```
npm install --save-dev nodemon
```

Außerdem erstellen wir noch ein Script um den Entwicklungs Server einfacher zu starten:

Im `package.json` File:

```
  "scripts": {
    "dev": "npx nodemon server.js"
  },
```

Starte also den Dev Server mit `npm run dev` und ändere dann die `server.js` Datei ab. Nodemon erkennt die Änderung und startet den Server neu!

Schritt 5: Richte das React-Frontend ein

Wechsle nun ins vorher erstellte Client Verzeichnis

```
cd ../client
```

Initialisiere ein neues React-Projekt mit Create React App:

```
npx create-react-app .
```

Um mit einem leeren Projekt zu starten, lösche die folgenden Dateien:

- public:
  - logo192.ong
  - logo512.png
- src:
  - App.css
  - App.test.js
  - index.css
  - logo.svg
  - reportWebVitals.js
  - setupTests.js

Entferne nun auch alle Referenzen zu diesen Dateien, sodass deine Dateien so aussehen:

`App.js`

```
function App() {
  return (
    <div className="App">
      Hallo Welt
    </div>
  );
}

export default App;
```

`index.js`

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

`public/manifest.json`

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Schritt 6: Kommunikation zwischen Frontend und Backend

Erstelle einen API-Endpunkt in Deiner Express-Anwendung, um Daten an das Frontend zu senden. Zum Beispiel:

```
// In server.js

app.get('/api/nachricht', (req, res) => {
    console.log('GET /api/nachricht ', req.ip);
    res.json({ nachricht: 'Hallo von der API!' });
});
```

Nun können wir diesen API-Endpunkt im Frontend abfragen. Zuerst erstellen wir einen neuen State (über den `useState` (Hook) um dort die Nachricht zu speichern. Danach nutzen wir den `useEffect` Hook um beim Laden + Updated der Komponente die unsere API abzufragen

```
// In src/App.js im React-Frontend

import React, { useState, useEffect } from 'react';

function App() {
  const [nachricht, setNachricht] = useState('');

  useEffect(() => {
    fetch('http://localhost:8080/api/nachricht'
    re)
      .then(response => response.json())
      .then(json => {
        console.log(json)
        setNachricht(json.nachricht);
```

```
      });
  }, []);

  return (
    <div>
      <h1>{nachricht}</h1>
    </div>
  );
}

export default App;
```

Schritt 7: CORS

Im aktuellen Zustand funktioniert die App noch nicht aufgrund von CORS. Das liegt daran, dass wir zwei verschiedene Origins verwenden:

- Frontend: localhost:3000
- Backend: localhost:8080

Was ist CORS?

CORS, oder Cross-Origin Resource Sharing, ist ein Sicherheitsmechanismus, der in Webbrowsern implementiert ist, um zu verhindern, dass Webseiten Ressourcen von einer anderen Domäne (Origin) laden, es sei denn, die andere Domäne erlaubt dies ausdrücklich. Dies ist eine wichtige Sicherheitsmaßnahme, um böswillige Angriffe zu verhindern, jedoch kann es in bestimmten Szenarien problematisch sein, wenn Du eine Full-Stack-App mit separaten Frontend- und Backend-Servern erstellst.

CORS konfigurieren / deaktivieren

Um also in unserem Projekt zwei verschiedene Origins verwenden zu können, gehen wir wie folgt vor:

1. Wir wechseln wieder zu unserem `server` Ordner

```
cd ../server
```

2. Instaliere das `cors` npm paket

```
npm install cors
```

3. Füge die `middleware` in das Projekt hinzu

```
const express = require('express');
const cors = require('cors');
const app = express();
const port = 8080;

app.use(cors());
```

4. Wenn du jetzt das Frontend neu startest, solltest du die Nachricht angezeigt sehen

Diese Einstellungen haben auch den Vorteil, dass du in der Zukunft das Backend und Frontend getrennt hosten kannst:

- Frontend in S3 (mit CloudFront CDN)
- Backend als EC2

Schritt 8: Backend aufräumen

Routes

Für den aktuellen Umfang, ist die Struktur der `server.js` Datei völlig ausreichend. Wenn das Projekt allerdings wächst wird es schnell unübersichtlich. Deshalb macht ist es Standard, dass man seine API in verschiedene Routen / Dateien auslagert

Stelle sicher, dass du dich noch im `server` Ordner befinest

1. Erstelle die Ordner Struktur

```
mkdir -p routes/api
touch routes/api.js
touch routes/api/nachricht.js
```

2. Initialisiere den api Router

In der `routes/api.js`

```
const express = require('express')
const router = express.Router()
const nachricht = require("./api/nachricht")

router.use("/nachricht", nachricht)

module.exports = router
```

`routes/api/nachricht.js`

```
const express = require('express')
const router = express.Router()

router.get('/', (req, res) => {
    console.log('GET /api/nachricht ', req.ip);
    res.json({ nachricht: 'Hallo von der API!' });
});

router.post('/update', (req, res) => {
    console.log('PUT /api/nachricht/update ', req.ip);
});

module.exports = router
```

`server.js`

```
const express = require('express');
const cors = require('cors');
const api = require('./routes/api');
const app = express();
const port = 8080; app.use(cors()); app.use('/api', api);
app.get('/', (req, res) => {
    res.send('Hello von Express');
});


app.listen(port, () => {
    console.log(`Server läuft auf Port ${port}`);
});
```

Jetzt wird kann für jede neue API Route nach dem gleichen Schema eine neue Datei im `routes/api/` Ordner angelegt werden und dann in der `routes/api.js` angepasst werden.

Schritt 9: React Components

Um im Frontend auch die Übersicht zu behalten, erstellen wir auch hier verschieden Unterordner

```
cd ../client/src
mkdir -p components/
touch components/Nachricht.js
touch components/NachrichtUpdate.js
```

Neue `App.js` Datei

```
import React from 'react';
import Nachricht from './components/Nachricht';

function App() {

    return (
        <div>
            <Nachricht />
        </div>
    );
}

export default App;
```

`components/Nachricht.js`

```
import React, { useState, useEffect } from 'react';
import NachrichtUpdate from './NachrichtUpdate';

function Nachricht() {
    const [nachricht, setNachricht] = useState('');

    useEffect(() => {
        fetch('http://localhost:8080/api/nachricht')
            .then(response => response.json())
            .then(json => {
                console.log(json)
                setNachricht(json.nachricht);
            });
    }, []);

    return (
        <div className="Nachricht">
            <h1>Aktuelle Nachricht:</h1>
            <h2>{nachricht}</h2>
            <NachrichtUpdate />
        </div>
    );
}

export default Nachricht;
```

`components/NachrichtUpdate.js`

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

function NachrichtUpdate() {

    const [nachricht, setNachricht] = useState('');

    function handleUpdate(e) {
        e.preventDefault();
        console.log('Update Nachricht: ', nachricht);
    }

    return (
        <div className="NachrichtUpdate">
            <h2>Update Nachricht</h2>
```

```
            <input type="text" value={nachricht} onChange={e => setNachricht(e.target.value)} />
            <button onClick={handleUpdate}>Update</button>
        </div>
    );
}

export default NachrichtUpdate;
```