Dokumentation

Einleitung

Die Website „Myths & Forges“ stellt eine Plattform zum freien Erstellen fiktiver Charaktere dar, deren Anwendungszweck im Unterhaltungs- und/oder betrieblichen Kontext finden können.
Auf der Startseite werden alle frei publizierten Charakter-Sheets in Cardviews angezeigt. Registriert man sich, ist es möglich Sheets zu favorisieren und sich für den späteren Gebrauch vorzumerken. Außerdem ist es möglich, eigene Sheets zu erstellen und diese privat oder öffentlich zu speichern.
Wenn man an mehreren Bögen eines Users Gefallen findet, kann man diesem direkt folgen und sieht so sehr schnell aktuelle Neuerscheinungen.

Installation

Das Projekt ist über ein **docker-compose Kommando** zu starten. Es gibt hierbei drei Images (Frontend, Backend und Database). Um einen korrekten build zu gewährleisten, ist es wichtig zu prüfen, ob eine Datei namens „**.env**“ im Server Ordner des Repositories zu finden ist. Wenn nicht, sollte die Datei „**.env.example**“ in „**.env**“ umbenannt werden.
Anschließend kann man die Build- und Startbefehle (während des Entwickelns in der Konsole über „**docker-compose up –build**“ im **Hauptverzeichnis**) einleiten.
Sobald Docker den Container erstellt hat, ist die Seite über „**localhost:20091/**“ bereits erreichbar.

Verzeichnisstruktur

Client

- ➔ node_modules/
- ➔ public/
- ➔ src/
  - ➔ actions/ *(Redux)*
    - Übergabe gewünschter Operationen an Reducer
    - Entgegennahme eventueller Antworten
  - ➔ api/ *(Axios)*
    - Verbindungsaufbau zu Backend über Axios
  - ➔ components/
    - Sichtbare, modulare Frontendkomponente
      - Auth: Authentifizierung, Login, Registrierung
      - Form: Formular zum Erstellen/Verändern von Sheets
      - Home
      - NavBar
      - Sheets: Grid Container mit Sheet-Elementen
    - Aktionsauslösung durch Logik und User-Interaktion
  - ➔ constants/
  - ➔ images/
  - ➔ reducers/ *(Redux, Thunk)*
    - Vermittlung zwischen Actions und API
  - ➔ App.js
    - Hauptkomponente und Importpunkt der Components
  - ➔ globalStyles.js
  - ➔ index.js
    - Bündelung der Komponente und Funktionalitäten
      - App.js
      - ReactDom
      - Provider
      - Thunk
    - Startpunkt der FrontendApp
  - ➔ Dockerfile

Server/

- ➔ controllers/
  - Implementierung der Aktionen und Datenbankzugriffe über 3 Routen

- ➔ models/
  - Mongoose Schemata für die zu speichernden Objekttypen

- ➔ node_modules/

- ➔ routes/
  - Angaben für die Weiterleitung an Controller-Funktionen über den Router
  - „/sheets“, „/user“, „/following“

- ➔ .env
  - Angabe von Port und Datenbank-URL in Environmental Variables

- ➔ .env.example
  - Kopie der „.env“-Datei falls nicht ins Repository übernommen

- ➔ Dockerfile

- ➔ index.js
  - Implementierung und Konfiguration der Express-App
  - Angabe der Routen
  - Implementierung der Datenbankverbindung über Mongoose

- ➔ package-lock.json
- ➔ package.json
- ➔ swagger.json

Database

- ➔ Lokale MongoDB

Docker-Compose.yml

- ➔ Konfigurationen für die Inbetriebnahme des Docker Containers
  - Frontendport: 20091:3000
  - Backendport: 20093:5000
  - Datenbankport: 20092:27017

Technologien

Frontend:

- React
  - JavasScript Bibliothek
  - Aufbau modularer Benutzeroberflächen
  - Deklarative Syntax
  - Material-UI
    - React-Komponentenframework
    - Enthält vorgefertigte UI-Komponente und Stile
  - React Router
    - Bibliothek zur Erleichterung der Navigation
    - Ermöglicht Anzeige unterschiedlicher Komponente über URLs

- Redux
  - Container-System für React-Anwendungen
  - Zustandsverwaltung
  - Einfache Verwaltung von Datenflüssen
  - Zentrale Speicherung von Anwendungsstatus
  - Redux Thunk
    - Middleware für Redux
    - Integration von asynchronen Prozessen in Redux-Aktionen

- Axios
  - JavaScript Bibliothek
  - Senden von HTTP Anfragen an den Server und Empfang von Antworten
  - Ruft API's auf

Backend:

- Node.js
  - Laufzeitumgebung für JavaScript
  - Ermöglicht JS Ausführung auf Servern

- Express.js
  - Node.js-Framework
  - Ermöglicht einfaches Definieren und Anfragen von Routen

- MongoDB
  - NoSQL-Datenbank
  - Speichern von JSON-ähnlichen Objekten
  - Häufige Anwendung mit Node.js

- Mongoose
  - Object Data Mapping Bibliothek für MongoDB
  - Erleichtert Arbeit mit der Datenban in Node.js

Code-Beispiele

Components

/client/components/Home/home.js

```
1   import React, { useState, useEffect } from "react";
2   import { useDispatch } from "react-redux";
3   import { Container, Grow, Grid } from '@mui/material';
4
5   import { getSheets } from '../../actions/sheets';
6
7   import Form from "../Form/form";
8   import Sheets from "../Sheets/sheets";
9
10  const Home = () => {
11      const [currentId, setCurrentId] = useState(0);
12      const dispatch = useDispatch();
13
14      useEffect(() => {
15        dispatch(getSheets());
16      },
17      [currentId, dispatch]); // As soon as currentId is changed, start dispatch anew
18
19      return(
20          <Grow in>
21              <Container>
22              <Grid container justify="space-between" alignItems="stretch" spacing={3}>
23                  <Grid item xs={12} sm={7}>
24                  <Sheets setCurrentId={setCurrentId} />
25                  </Grid>
26                  <Grid item xs={12} sm={4}>
27                  <Form currentId={currentId} setCurrentId={setCurrentId}/>
28                  </Grid>
29              </Grid>
30              </Container>
31          </Grow>
32      )
33  }
34
35  export default Home;
```

Die Home-Component bündelt hier die Sheets- und Form-Components (Zeile 7,8) und stellt anschließend in einem Grid alle von der Sheets-Component gesammelten Sheet-Objekte in einem Raster dar (Zeile 23-25).
Neben dieser Sammlung wird die Form-Component gerendert. Diese ermöglicht das Erstellen oder Bearbeiten von Sheets (Zeile 26-28).
Durch den „useState“-Hook wird das inkrementelle durchlaufen der Sheets-Objekte aus der Datenbank ermöglicht, worauf jeder Sheet einzeln angezeigt werden kann.
Weiterhin wird über einen „dispatch“-Aufruf im „useEffect“-Hook die Route zur „getSheets“-Funktionalität aufgerufen. Über die API Schnittpunkte werden anschließend die Sheet-Daten in die Components transportiert.

Redux Aktionen

/client/src/components/Home/home.js

```
6   import useStyles from "./styles"
7   import { createSheet, updateSheet } from "../../actions/sheets"
8
9   const Form = ({ currentId, setCurrentId }) => {
10      const [sheetData, setSheetData] = useState({
11          name: '',
12          description: '',
13          profession: '',
14          creator: '',
15          age: '',
16          home: '',
17          isPublic: false,
18          SelectedFile: '',
19          tags: ''
20      })
21      const sheet = useSelector((state) => currentId ? state.sheets.find((v) => v._id === currentId) : null);
22      const classes = useStyles();
23      const dispatch = useDispatch();
24
25      useEffect(()=>{
26          if(sheet) setSheetData(sheet);
27      }, [sheet])
28
29      const handleSubmit = (event) => {
30          event.preventDefault();
31
32          if(currentId){
33              dispatch(updateSheet(currentId, sheetData));
34          } else {
35              dispatch(createSheet(sheetData));
36          }
37          clear();
38      }
```

Redux Aktionen beginnen in den Components durch die Logik oder User-Interaktion.
In diesem Beispiel importiert die Form-Component die Aktionen „createSheet“ und „updateSheet“ (Zeile 7)
Falls im Redux store eine „currentId“ für einen Sheet vorhanden ist, wird die Variable „sheet“ mit ihm verbunden (Zeile 21).
Wenn für einen neuen oder vorhandenen Sheet der Submit Button betätigt wird, prüft die „handleSubmit“ Funktion ob es einen bereits existierenden Sheet gibt, auf den referenziert wird und löst darauf aufbauend die „createSheet“ oder „updateSheet“ Aktion aus (29-38).

/client/src/actions/sheets.js

```
14  export const createSheet = (sheet) => async (dispatch) => {
15      try {
16          const { data } = await api.createSheet(sheet);
17          dispatch({ type: CREATE, payload: data});
18      } catch (error){
19          console.log(error);
20      }
21  }
22
23  export const updateSheet = (id, _sheet) => async (dispatch) => {
24      try {
25          const {data} = await api.updateSheet(id, _sheet);
26          dispatch({ type: UPDATE, payload: data });
27      } catch (error) {
28          console.log(error);
29      }
30  }
```

Durch den Aufruf der Aktionen werden Try-Catch Blöcke aktiviert, welche eine asynchrone Anfrage an die API weiterleiten und danach ggf. auf eine Antwort warten.
Im „dispatch“ werden dabei die Zustände beibehalten, übermittelt und erneuert.

API-Anfragen

/client/src/api/index.js

```
1   import axios from 'axios';
2
3   const sheetsURL = 'http://localhost:20093/sheets';
4   const userURL = 'http://localhost:20093/user';
5   const followingURL = 'http://localhost:20093/following';
6
7   export const fetchSheets = () => axios.get(sheetsURL);
8   export const createSheet = (newSheet) => axios.post(sheetsURL, newSheet);
9   export const updateSheet = (id, updatedSheet) => axios.patch(`${sheetsURL}/${id}`, updatedSheet);
10  export const deleteSheet = (id) => axios.delete(`${sheetsURL}/${id}`);
11  export const likeSheet = (id) => axios.patch(`${sheetsURL}/${id}/likeSheet`); // Change me into a followingURL call later
```

Aktionen leiten über Redux-Thunk ihre benötigten Routen und Funktionalitäten an die API weiter. Über eine der drei Routen „.../sheets“, „.../user“ oder „.../following“ werden die passenden Axios-Operationen ausgewählt und die Verbindung zum Backend etabliert.

React-Router

/client/src/App.js

```
17  const App = () => {
18    return (
19      <BrowserRouter>
20        <ThemeProvider theme={theme}>
21          <Container maxWidth="lg">
22            <NavBar />
23            <Routes>
24              <Route exact path="/" element={ <Home /> } />
25              <Route exact path="/auth" element={ <Auth /> } />
26            </Routes>
27          </Container>
28        </ThemeProvider>
29      </BrowserRouter>
30    );
31  }
32
33  export default App;
```

Der React-Router ermöglicht es hier, die Komponente unter der Navigationsleiste zu wechseln. Über den Abschnitt „<Routes>...</Routes>“ (Zeile 23-26) werden etwaige Zielrouten, ergo Komponentenkompositionen, über URL-Pfade sortiert und abrufbar gemacht.