Temat projektu: “Kinomaniak” - aplikacja do zarządzania seansami w kinie.

Autor projektu:

- **Paweł Kamieński Ćw5**
- **Paweł Orzel Ćw1**

1. Architektura komponentów

Architekturą, która została użyta w projekcie jest architektura standardowa, z racji niepowodzenia zaimplementowania Architektury Flux.

Opis komponentów:

- **Layout.js -** komponent funkcyjny odpowiedzialny za renderowanie w kontenerze layoutu strony.
- **Navbar.js -** komponent funkcyjny odpowiedzialny za renderowanie górnego paska nawigacyjnego.
- **Home.js -** komponent funkcyjny odpowiedzialny za renderowanie zawartości strony głównej.
- **App.js -** komponent funkcyjny odpowiedzialny za zarządzanie routingiem całego projektu.
- **Errorpage.js -** komponent funkcyjny odpowiedzialny za renderowanie błędu w momencie wpisania URL nie znajdującego się w puli obsługiwanych adresów URL przez routing.
- **withRouter.js -** komponent funkcyjny służący do obsługiwania zmiennych właściwości routingu.
- **Seanse.js -** komponent stanowy odpowiedzialny za wyświetlanie seansów pobierający dane z pliku serwerowego seanse.json.
- **Filmy.js -** komponent stanowy odpowiedzialny za wyświetlanie filmów pobierający dane z pliku serwerowego filmy.json.
- **Addseans.js** - komponent stanowy odpowiedzialny za dodawanie seansów.
- **Editseans.js** - komponent stanowy odpowiedzialny za edycję seansów.
- **SelectData.js** - komponent funkcyjny odpowiedzialny za formularz wyboru daty(dnia i miesiąca).
- **Statystyki.js** - komponent funkcyjny odpowiedzialny za wyświetlanie statystyk.
- **Addfilm.js -** komponent stanowy odpowiedzialny za dodawanie filmów.
- **Editfilm.js -** komponent stanowy odpowiedzialny za edycję filmów.
- **Addticket.js -** komponent stanowy odpowiedzialny za dodawanie biletów do seansu.

2. Ścieżki i komponenty związane z routingiem.

Komponentami związanymi z routingiem są **App.js** i **withRouter.js** pierwszy z nich definiuje ścieżki routingowe natomiast drugi obsługuje możliwość używania zmiennych właściwości routingu. Wszystkie ścieżki użyte w projekcie zostały przedstawione na rysunku poniżej.

```
function App(){
  return (
    <React.Fragment>
    <NavBar/>
    <Layout>
    <Router>
      <Routes>
          <Route path="/" element = {<Home/>}/>
          <Route path="/filmy" element={<Filmy/>} />
          <Route path="/Addfilm" element={<Addfilm/>} />
          <Route path="/seanse" element={<Seanse/>} />
          <Route path="/filmy/edytuj/:id" element = {<Editfilm/>}/>
          <Route path="/statystyki" element={<Statystyki/>} />
          <Route path="/filmy/dodajseans/:id" element = {<Addseans/>} />
          <Route path="/seanse/edytuj/:id" element = {<Editseans/>}/>
          <Route path="/seanse/kupbilet/:id" element = {<Addticket/>}/>
          <Route path="*" element = {<Errorpage/>} />
      </Routes>
    </Router>
    </Layout>
    </React.Fragment>
  );
}
```

Ścieżka	Komponent	Opis
"/"	Home	Ścieżka strony startowej aplikacja.
"/filmy"	Filmy	Ścieżka wyświetlająca listę dostępnych filmów wraz z formularzem.
"/Addfilm"	Addfilm	Ścieżka nieaktywna.
"/seanse"	Seanse	Ścieżka służąca do wyświetlania seansów.
"/filmy/edytuj/:id"	Editfilm	Ścieżka wyświetlająca formularz edycji filmu o id umieszczonym jako parametr ścieżki.
"/statystyki"	Statystyki	Ścieżka służąca do wyświetlania statystyk filmów w danym dniu w postaci wykresu słupkowego.
"/filmy/dodajseans/:id"	Addseans	Ścieżka wyświetlająca formularz dodania seansu dla filmu o konkretnym id zawartym jako parametr ścieżki.
"/seanse/edytuj/:id"	Editseans	Ścieżka służąca do wyświetlania formularza edycji seansu o id znajdującym się jako parametr ścieżki.
"/seanse/kupbilet/:id"	Addticket	Ścieżka wyświetlająca formularz kupna biletu na seansu o id zawartym jako parametr ścieżki.
"*"	Errorpage	Ścieżka, która wyświetla zawartość komponentu Errorpage w sytuacji, gdy zostanie podana nieprawidłowa ścieżka nieznajdująca się w puli

		ścieżek projektu.

3. Dane w store

Projekt nie posiada danych w store z racji niewykorzystania w projekcie Architektury Flux.

4. API serwera.

Serwerem, który został wdrożony w projekcie jest serwer Node.js. Plik **serwer.js** przechowuje funkcje, które realizują wszystkie żądania wykorzystywane w projekcie (GET, POST , PUT ,DELETE). Adresem serwera jest: **http://localhost:7777.** Serwer przechowuje dane w 4 plikach z rozszerzeniem .json:

- **sale.json** – przechowuje listę dostępnych sal. Każdy z obiektów w pliku zawiera atrybut „nr_sali" oraz „pojemnosc".
- **filmy.json** – przechowuje listę dostępnych filmów.Każdy z obiektów w pliku zawiera atrybut „filmId", „tytul", „czas_trwania" oraz „plakat_url".
- **bilety.json** – przechowuje listę wszystkich zakupionych biletów w kinie. Każdy z obiektów w pliku zawiera atrybut „biletId", „nr_miejsca" oraz „id_seansu".
- **seanse.json** – przechowuje listę wszystkich seansów bez podziału na filmy.Każdy z obiektów w pliku zawiera atrybut „seansId", „seansdata", „seanshour", „seansfilm", „seanssala", „seansliczbasprzedanychbiletow", „seansliczbadostepnychbiletow" oraz „seansnumeryzajetychmiejsc". Atrybut „seansfilm" zawiera w sobie obiekt, który składa się z atrybutów tj. „id", „filmname", „filmtime". Atrybut „seanssala" zawiera w sobie obiekt, który składa się z atrybutów tj „nr_sali" i „pojemnosc".Atrybut „seansnumeryzajetychmiejsc" zawiera tablice obiektów , gdzie każdy obiekt zawiera „id_miejsca" oraz boolean „zajete".Długość tej tablicy jest zależne od pojemności sali na jakiej odbywa się dany seans.Dzięki temu atrybutowi możemy sprawdzić czy dane miejsce jest zajęte w konkretnym seansie.

GET:

- "/filmy" – pobiera wszystkie filmy z pliku filmy.json.
- "/bilety" – pobiera wszystkie bilety z pliku bilety.json.
- "/seanse" – pobiera wszystkie seanse z pliku seanse.json.
- "/filmy/:id" – pobiera film o wskazanym w żądaniu id z pliku filmy.json.
- "/sale/:id" – pobiera sale o wskazanym w żądaniu id z pliku sale.json.
- "/seanse/:id" – pobiera seans o wskazanym w żądaniu id z pliku seanse.json.

POST:

- "/filmy" – dodaje nowy film do pliku filmy.json."
- "/seans" – dodaje nowy seans do pliku seanse.json.
- "/bilety" – dodaje nowy seans do pliku seanse.json.

DELETE:

- ”/filmy/:id” – usuwa film o wskazanym w żądaniu id z pliku filmy.json.Wraz ze wskazanym filmem usuwają się również seanse filmu oraz bilety na te seanse.

PUT:

- ”/filmy/:id” – aktualizuje nowymi danymi film o wskazanym w żądaniu id w pliku filmy.json.
- ”/seanse/:id” – aktualizuje nowymi danymi seans o wskazanym w żądaniu id w pliku seanse.json.

5. Wybrane przez autorów, szczególnie ciekawe fragmenty kodu.

- Rysowanie wykresu słupkowego na podstawie danych: “tytuł”,”Liczba_Sprzedanych_Biletów”.

```
<BarChart
  style={{ margin: "auto" }}
  width={550}
  height={550}
  data={seanse}
>
  <XAxis dataKey="tytul" />
  <YAxis />
  <Legend />
  <Bar dataKey="Liczba_Sprzedanych_Biletow" barSize={20} fill="#8884d8" />
</BarChart>
```

- Utworzenie graficznej wizualizacji krzesełek sali kinowej w formularzu kupna biletu.

```
<table style = {{margin: "auto"}}>
    {
        (() => {
            console.log(this.inputRef)
            if (!(tablica_miejsc instanceof Array)) {
                return;
            }

            let container = [];

            const chunk = this.chunk(tablica_miejsc, 10);
            for (const arr of chunk) {
                let trBody = [];
                for (const miejsce of arr) {
                    if (miejsce.zajete == true) {
                        trBody.push(<td className="p-1"><button style = {{margin:"15px"}} className="btn btn-danger"
                        disabled >{miejsce.id_miejsca}</button></td>)
                    } else {
                        trBody.push(<td className="p-1"><button style = {{margin:"15px"}} className="btn btn-success"
                        value={miejsce.id_miejsca} onClick={this.onButtonClick}>{miejsce.id_miejsca}</button></td>)
                    }
                }
                container.push(<tr>{trBody}</tr>)
            }
            return container;
        })()
    }
</table>
```

6. Wypunktowane elementy techniczne, które zostały zrealizowane w projekcie wraz z krótkim komentarzem odnośnie realizacji: jak zrealizowano i w którym pliku.

- **Własna walidacja danych wprowadzanych przez użytkownika (w każdym przypadku wprowadzania danych).**

  Stworzona została własna funkcja, która na początku działania ustawia wszystkie dane na false, następnie każda zmienna odpowiadająca za dany input jest walidowana zgodnie z wymaganiami określonymi w treści projektu np. dla zmiennej “tytul” filmu funkcja walidująca sprawdza długość wprowadzanej nazwy oraz czy zmienna zaczyna się od wielkiej litery(za pomocą funkcji match()). Własna walidacja została wykonana w komponentach tj. Addfilm, Addseans, Addticket, Editfilm, Editseans.

  Status: Zrealizowane.

- **Obowiązkowa weryfikacja typu danych (PropTypes) przekazywanych do wszystkich komponentów (nie stosujemy typu 'any').**

  Dla komponentów (przykładowo komponent SelectData), które przekazują wyłącznie informację innym komponentom została utworzona weryfikacja, która wymaga wykorzystania props. Zostało to zrealizowane w każdym komponencie zawierającym props.

  Status: Zrealizowane.

- **Weryfikacja typu danych (PropTypes) własną funkcją.**

  Dla komponentu SelectData, który przekazuje dzień i miesiąc została utworzona funkcja, która sprawdza wartość dla currentDay oraz currentMonth, aby była większa od 0.

  Status: Zrealizowane.

- **Dwukierunkowa komunikacja pomiędzy komponentami:**

  Komunikacja między komponentem SelectData oraz Statystyka, komponent SelectData otrzymuje props od komponentu Statystyka. Komunikacja odbywa się również między komponentem Film i AddFilm, dzięki czemu mamy wyświetlony formularz dodawania filmu w komponencie Film.

  Status: Zrealizowane.

- **Modyfikacja danych odbywa się tylko w jednym komponencie.**

  Status: Niezrealizowane.

- **Operacje modyfikacji danych za pomocą 4 rodzajów żądań http.**

  W podpunkcie dokumentacji „API serwera" można zauważyć, że zdefiniowano 4 rodzaje funkcji do żądań http.Wszystkie znajdują się w pliku serwerowym **server.js**.Żądanie PUT jest wykorzystywane w komponentach, które przechowują formularze do edycji danych np. Editseans, Editfilm. Żądanie GET jest wykorzystywane w komponentach, które wyświetlają dane serwerowe np. Filmy, ale również w tych które edytują dane np. Editseans. Żądanie DELETE zostało zrealizowane jedynie w komponencie Filmy z uwagi na to, że tylko usuwanie filmów zostało określone w wymaganiach projektowych. Żądanie POST zostało wykorzystane w komponentach, które zawierają formularze do dodania danych do bazy serwera np. w komponentach Addfilm, Addseans, Addticket.

  Status: Zrealizowane.

- **Żądania do serwera są zapisane w jednym oddzielnym pliku.**

  Wszystkie żądania do serwera są zapisane w jednym oddzielnym pliku jakim jest **server.js**.

  Status: Zrealizowane.

- **Routing (ścieżki 'routes', w tym jedna z parametrem).**

  Routing został zrealizowany w komponencie **App.js**. Zostały utworzone cztery ścieżki, których komponenty korzystają z parametrów umieszczonych w tych ścieżkach.

```
function App(){
  return (
    <React.Fragment>
    <NavBar/>
    <Layout>
    <Router>
      <Routes>
          <Route path="/" element = {<Home/>}/>
          <Route path="/filmy" element={<Filmy/>} />
          <Route path="/Addfilm" element={<Addfilm/>} />
          <Route path="/seanse" element={<Seanse/>} />
          <Route path="/filmy/edytuj/:id" element = {<Editfilm/>}/>
          <Route path="/statystyki" element={<Statystyki/>} />
          <Route path="/filmy/dodajseans/:id" element = {<Addseans/>} />
          <Route path="/seanse/edytuj/:id" element = {<Editseans/>}/>
          <Route path="/seanse/kupbilet/:id" element = {<Addticket/>}/>
          <Route path="*" element = {<Errorpage/>} />
      </Routes>
    </Router>
    </Layout>
    </React.Fragment>
  );
}
```

Status: Zrealizowane.

- **Wykorzystanie dwóch zmiennych właściwości routingu.**

  W celu używania zmiennych właściwości routingu w komponentach stanowych stworzono komponent **withRouter.js**, który to umożliwia.W projekcie wykorzystano właściwość przekierowania (np. **this.props.navigate(”/filmy”)** w komponencie Addseans) oraz właściwość pobierania parametru ze ścieżki (np. **this.props.params.id** w funkcji getSeansFilmByID() komponentu Editseans).
  Status: Zrealizowane.

- **Wykorzystanie architektury Flux.**

  Status: Niezrealizowane.

7. Dodatkowe biblioteki użyte w aplikacji: link oraz zdanie opisu biblioteki i celu użycia.

- **react-moment 1.1.1** – biblioteka użyta do obsługi dat.
  https://www.npmjs.com/package/react-moment
- **react-recharts** – biblioteka użyta do tworzenia wykresu w komponencie Statystka.js.
  https://www.npmjs.com/package/recharts
- **react-mui** – biblioteka użyta do stworzenia inputa do komponentu SelectData.js
  https://mui.com/
- **react-bootstrap** – biblioteka użyta do utworzenia formularzy, wyświetlania filmów w kartach oraz ogólnie layoutu na stronie.
  https://react-bootstrap.github.io/
- **react-icons** – biblioteka ,z której użyto m.in. ikone kosza, dzięki której odbyło się usuwanie filmów
  https://react-icons.github.io/react-icons/
- **axios** – biblioteka użyta do komunikacji aplikacji z serwerem.
  https://www.npmjs.com/package/axios

8. Podział prac w zespole.

Paweł Kamieński - funkcjonalność kupienia biletu, ogólny layout strony, dodawanie , wyświetlanie i edycja filmów (wraz z walidacją), utworzenie serwera Node.js, wykonanie routingu.
Paweł Orzel - komponenty dotyczące seansów wraz z walidacją, wyświetlanie statystyk.