



Wyższa Szkoła Informatyki i Zarządzania
Kolegium Informatyki Stosowanej, Informatyka

Dominik Chmielowski, w65529

***Aplikacja webowa do przeglądania popularnych seriali
z wykorzystaniem frameworka Vue.js***

Rzeszów, 06.02.2024

Spis treści

Opis funkcjonalność	5
Struktura Projektu i Funkcjonalności	9

```

import { Forbidden } from "@errors/Forbidden";
import { NotFound } from "@errors/NotFound";
import { ServerError } from "@errors/ServerError";
import { ServerUnavailable } from "@errors/ServerUnavailable";

class ClientApi {
  private baseUrl: string;

  constructor(baseUrl: string) {
    this.baseUrl = baseUrl;
  }

  public async get(endpoint: string, params: string = ""): Promise<any> {
    const url = params ? `${this.baseUrl}/${endpoint}?${params}` : `${this.baseUrl}/${endpoint}`;
    const response = await fetch(`${url}`);

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  public async post(endpoint: string, data: object): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${endpoint}`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  public async put(endpoint: string, data: object): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${endpoint}`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  public async delete(endpoint: string, data: object): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${endpoint}`, {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  private errorHandler(status: number, message: string = "") {
    if (status === 404) {
      throw new NotFound(message);
    } else if (status === 403) {
      throw new Forbidden(message);
    } else if (status === 500) {
      throw new ServerError(message);
    } else if (status === 502) {
      throw new ServerUnavailable(message);
    } else {
      throw new Error(message);
    }
  }
}

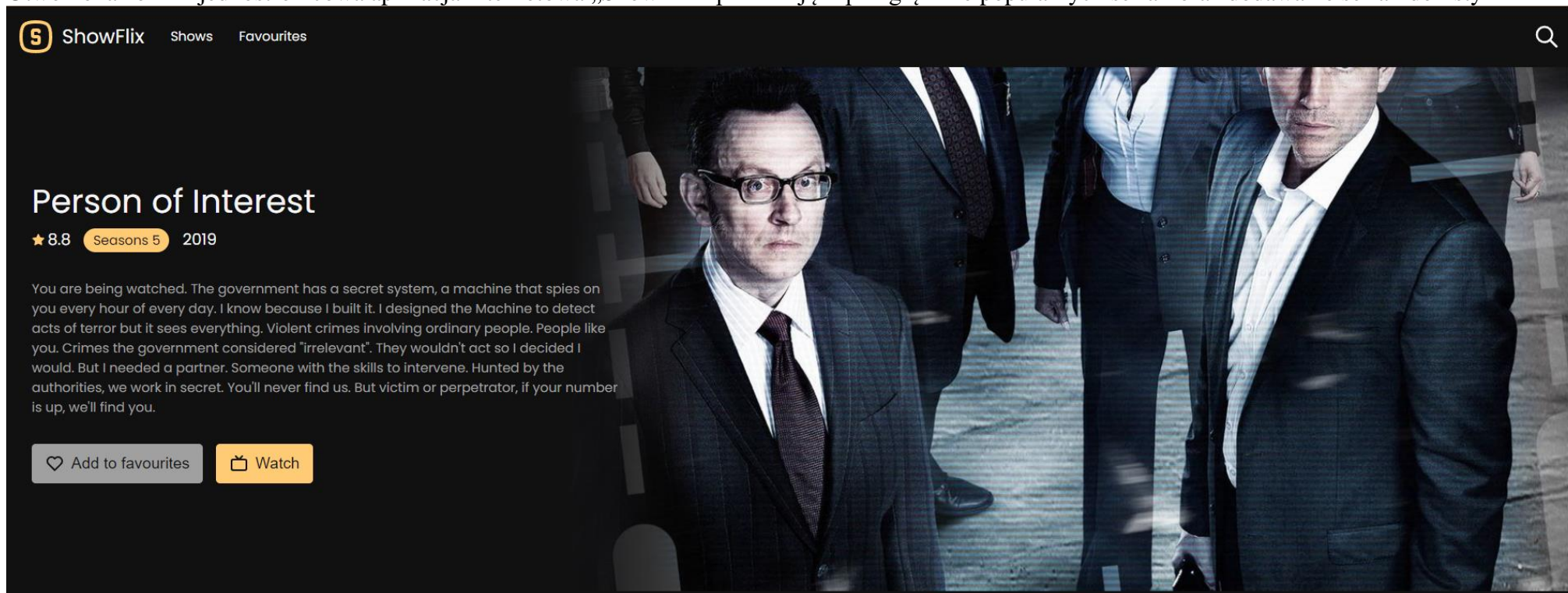
export default ClientApi;

```

Serwis Główny do zarządzania API	10
Serwis Show wykorzystujący serwis ClientApi do obsługi żądań w celu poprawienia z api seriali	11
Zastosowane technologie.....	12

Opis funkcjonalność

Utworzona została jednostronicowa aplikacja internetowa „ShowFlix” pozwalająca przeglądanie popularnych seriali oraz dodawanie seriali do listy



Aplikacja umożliwia sprawdzenie opisów, ocen i roku premiery

Show


Vikings

★ 8.6 Seasons 6 2019

Vikings transports us to the brutal and mysterious world of Ragnar Lothbrok, a Viking warrior and farmer who yearns to explore - and raid - the distant shores across the ocean.

Remove from favourites Watch

Favo

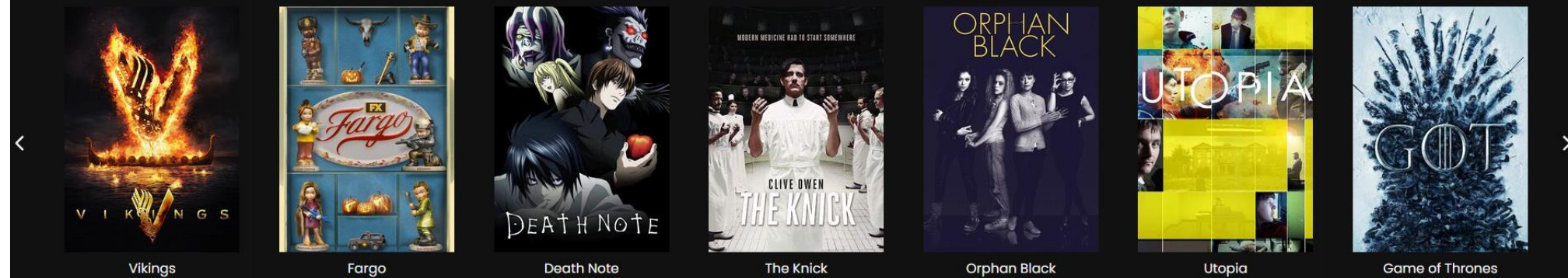
A dramatic scene of a Viking longship engulfed in intense orange and yellow flames. The ship's dragon-headed prow is visible on the right, and its stern on the left. The fire is particularly fierce around the central mast area, which is obscured by the flames. The ship is on a dark, choppy sea at night, with the fire reflecting on the water's surface.

Przeglądać serie można poprzez karuzele podobnie jak listy polubionych seriali

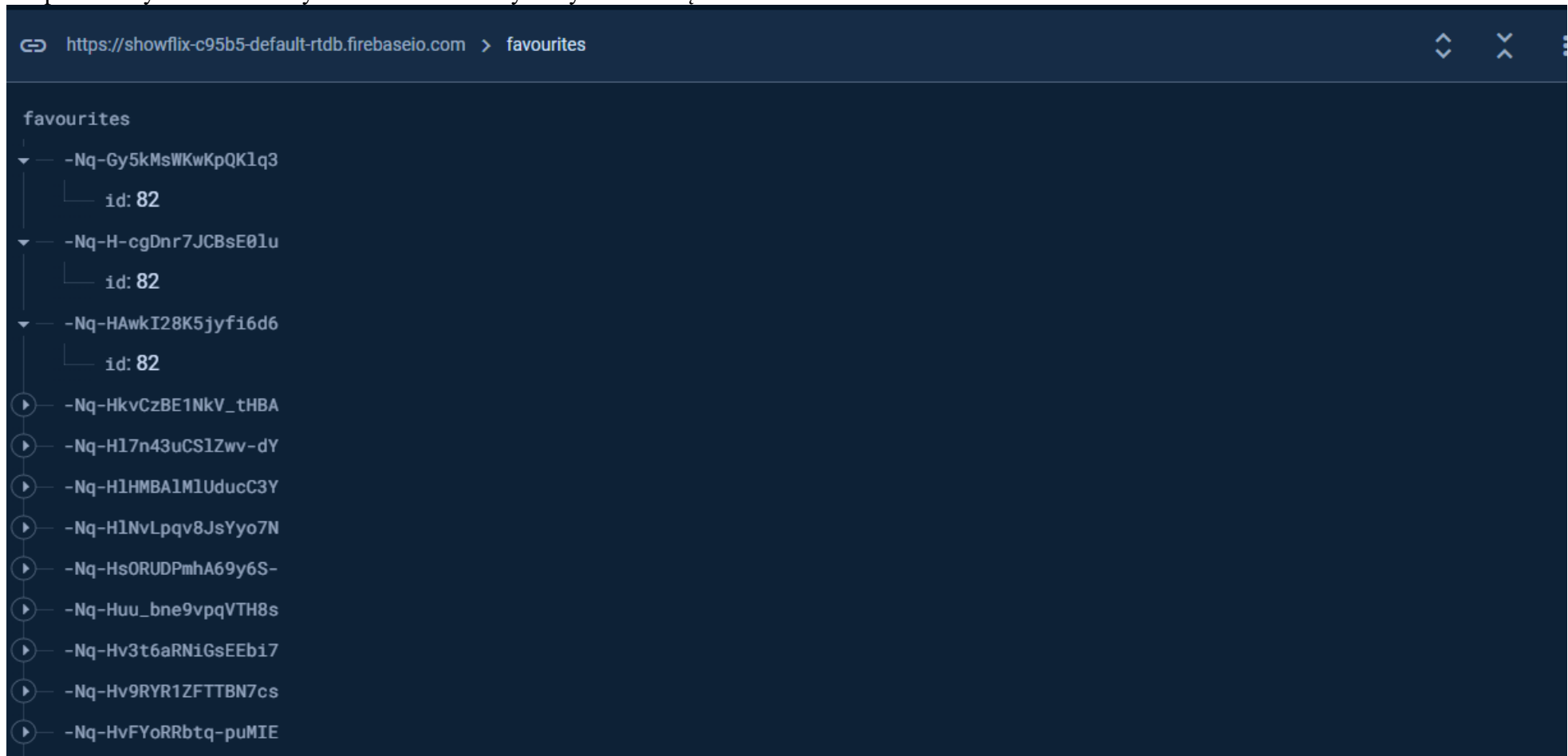
Shows



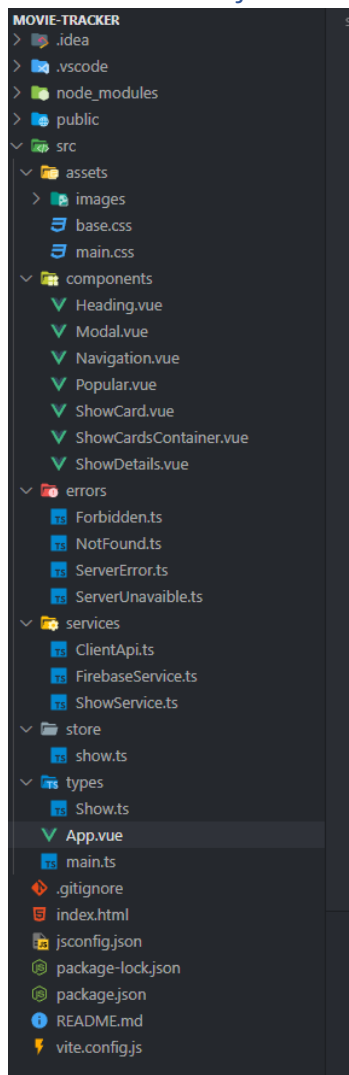
Favourites



Do przechowywania ulubionych seriali zostało wykorzystane narzędzie firebase:



Struktura Projektu i Funkcjonalności



Projekt został podzielony na:

- Components – główny folder zawierający wszystkie komponenty służące do renderowania elementów HTML,
- Errors – w tym folderze znajdują się podstawowe klasy errorów
- Services – znajdują się wszystkie potrzebne serwisy do obsłużenia żądań do endpointów
- Store – folder przechowujący plik do zarządzania globalnym statem
- Types – do przechowywania typów

Serwis Główny do zarządzania API

```
import { Forbidden } from "d/errors/Forbidden";
import { NotFound } from "d/errors/NotFound";
import { ServerError } from "d/errors/ServerError";
import { ServerUnavailable } from "d/errors/ServerUnavailable";

class ClientApi {
  private baseUrl: string;

  constructor(baseUrl: string) {
    this.baseUrl = baseUrl;
  }

  public async get(endpoint: string, params: string = ""): Promise<any> {
    const url = params ? `${this.baseUrl}/${endpoint}?${params}` : `${this.baseUrl}/${endpoint}`;
    const response = await fetch(`${url}`);

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  public async post(endpoint: string, data: object): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${endpoint}`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  public async put(endpoint: string, data: object): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${endpoint}`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  public async delete(endpoint: string, data: object): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${endpoint}`, {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      this.errorHandler(response.status, response.statusText);
    }

    return response.json();
  }

  private errorHandler(status: number, message: string = "") {
    if (status === 404) {
      throw new NotFound(message);
    } else if (status === 403) {
      throw new Forbidden(message);
    } else if (status === 500) {
      throw new ServerError(message);
    } else if (status === 503) {
      throw new ServerUnavailable(message);
    } else {
      throw new Error(message);
    }
  }
}

export default ClientApi;
```

Serwis Show wykorzystujący serwis ClientApi do obsługi żądań w celu poprawienia z api seriali

```
class ShowService {
  private clientApi: ClientApi;
  private baseUrl: string;

  constructor(baseUrl: string) {
    this.baseUrl = baseUrl;
    this.clientApi = new ClientApi(this.baseUrl);
  }

  public async getShows(): Promise<any> {
    try {
      const response = await this.clientApi.get("shows");
      return response;
    } catch (error) {
      throw error;
    }
  }

  public async getShowSeasons(id: number): Promise<any> {
    try {
      const response = await this.clientApi.get(`shows/${id}/seasons`);
      return response;
    } catch (error) {
      throw error;
    }
  }
}

export default new ShowService("http://api.tvmaze.com");
```

Zastosowane technologie

Framework Vue:

Vue.js to progresywny framework do tworzenia interfejsów użytkownika. Jest to popularna technologia do budowania dynamicznych stron internetowych i aplikacji webowych. Vue.js umożliwia tworzenie komponentów interfejsu użytkownika, zarządzanie stanem aplikacji i integrację z innymi bibliotekami i narzędziami. Jest bardzo elastyczny i łatwy do nauki, co sprawia, że jest często wybierany do projektów front-endowych.

Pinia:

Pinia to biblioteka do zarządzania stanem w aplikacjach Vue.js. Jest to narzędzie, które pozwala na efektywne zarządzanie stanem aplikacji, unikając pewnych problemów związanych z korzystaniem z innych rozwiązań, takich jak Vuex. Pinia wprowadza bardziej deklaratywny i modularny sposób zarządzania stanem, co ułatwia skalowanie aplikacji.

Firebase:

Firebase to platforma od Google, która zapewnia wiele narzędzi i usług do budowania aplikacji mobilnych i webowych. W projekcie wykorzystuje się Firebase do wielu celów, takich jak autentykacja użytkowników, przechowywanie danych w czasie rzeczywistym, hostowanie aplikacji, analiza i wiele innych. Firebase dostarcza gotowe rozwiązania do wielu typowych zadań backendowych, co przyspiesza proces tworzenia aplikacji.

TypeScript:

TypeScript to nadzbiór języka JavaScript, który wprowadza statyczne typowanie. Działa on na bazie JavaScript, ale dodaje statyczne typy, co pomaga w wykrywaniu błędów w trakcie pisania kodu oraz poprawia czytelność i zrozumiałość kodu. W połączeniu z Vue.js, TypeScript pomaga w tworzeniu bardziej niezawodnych i skalowalnych aplikacji.

Biblioteka Tabler Icons:

Tabler Icons to biblioteka ikon, która zawiera zestaw ikon w formie gotowych komponentów SVG lub kodu ikonowego, które można łatwo wykorzystać w aplikacjach Vue.js. Te ikony są często używane do dekoracji interfejsu użytkownika i ułatwiają dostęp do różnych symboli i grafik, które mogą być wykorzystane w projektach UI.

Podsumowanie i wnioski

Projekt został pomyślnie ukończony, co zaowocowało stworzeniem kompletnie działającej strony internetowej, która wykazuje głęboką ekspertyzę w obszarze HTML, CSS oraz JavaScript. Ten projekt zawiera wiele zaawansowanych funkcji, takich jak animacje, integrację z API, dynamiczne generowanie treści na stronie, a także różnorodne komponenty, takie jak wyświetlanie seriali i możliwość dodawania ulubionych programów. Co więcej, zastosowanie technologii takich jak Vue.js i TypeScript sprawiło, że projekt jest skalowalny i łatwy do zarządzania. Korzystanie z Firebase umożliwiło zbieranie i przetwarzanie danych, a użycie paczki ikon TablerIcons dodatkowo wzbogaciło projekt pod względem wizualnym. Podczas projektu nauczyłem się skutecznego wykorzystywania Firebase do zbierania, przechowywania i przetwarzania danych, co znacząco usprawniło zarządzanie informacjami na stronie internetowej oraz umożliwiło tworzenie dynamicznych funkcji.