1. Identyfikacja zagadnienia biznesowego

W ostatnich latach portale udostępniające materiały wideo na żądanie bardzo się rozwinęły. Wiele firm wprowadza swoje rozwiązania próbując przyciągnąć nowych użytkowników. Następstwem tego jest duża liczba serwisów z filmami i serialami. Twórcy filmowi podpisują ekskluzywne kontrakty z tymi dostawcami wideo, przez co każdy portal posiada w ofercie inną treść. Do tego dochodzą klasyczne wydania w postaci premier kinowych.

Wszystko to, skutkuje pojawieniem się problemu znajdowania nowych premier filmowych w przystępnym dla przeciętnego użytkownika formacie. Struktura i sposób udostępnianych informacji przez dostawców jest często modyfikowany. Dodatkowo pojawiają się nowe portale co utrudnia proces agregacji i uaktualniania spisu nowych premier.

Proponowane rozwiązanie zawiera wykorzystanie użytkowników portalu społecznościowego reddit.com w celu sprawdzania nowych wydań wideo. Posty udostępniane na subredditach – podstronach portalu - dotyczące filmów oraz seriali będą regularnie przeszukiwane, zapisywane do bazy danych, a następnie wyświetlane w przejrzystym dla użytkownika formacie na stronie.

Stworzony serwis internetowy polegałby na entuzjastach w celu utrzymania aktualnego spisu materiałów, jednak filtrowałby niepotrzebne informacje – aktualności filmowe to tylko mała część wszystkich postów – i wyświetlał je na stronie z informacjami, kiedy dany serial/film wyszedł, oraz gdzie go można znaleźć.

2. Wymagania systemowe i funkcjonalne

Aplikacja wymagać będzie uruchamiania czasowych procesów wyszukujących na portalu społecznościowym informacji o nowych wydaniach filmowych. W przypadku awarii lub rutynowego wyłączenia, kolejne terminy uruchomienia powinny być zapisane w bazie, aby nie nadużywać API portalu społecznościowego. Wszystkie znalezione informacje również powinny być zapisywane w bazie. Pozwoli to na niezależność od dostępności portalu społecznościowego.

Do wyszukiwania postów, potrzebny będzie token dostępu do API reddita, który również powinien być odświeżany co pewien określony czas.

Sama aplikacja zbudowana będzie w oparciu o MVC, z rozdzielonym backendem oraz frontendem. Komunikować się one będą między sobą za pomocą REST API z wykorzystaniem JSON jako format przesyłanych danych.

Podstawowe informacje jakie powinny być wyświetlane to tytuł filmu, data, kiedy został wypuszczony oraz na jakich platformach można go znaleźć.

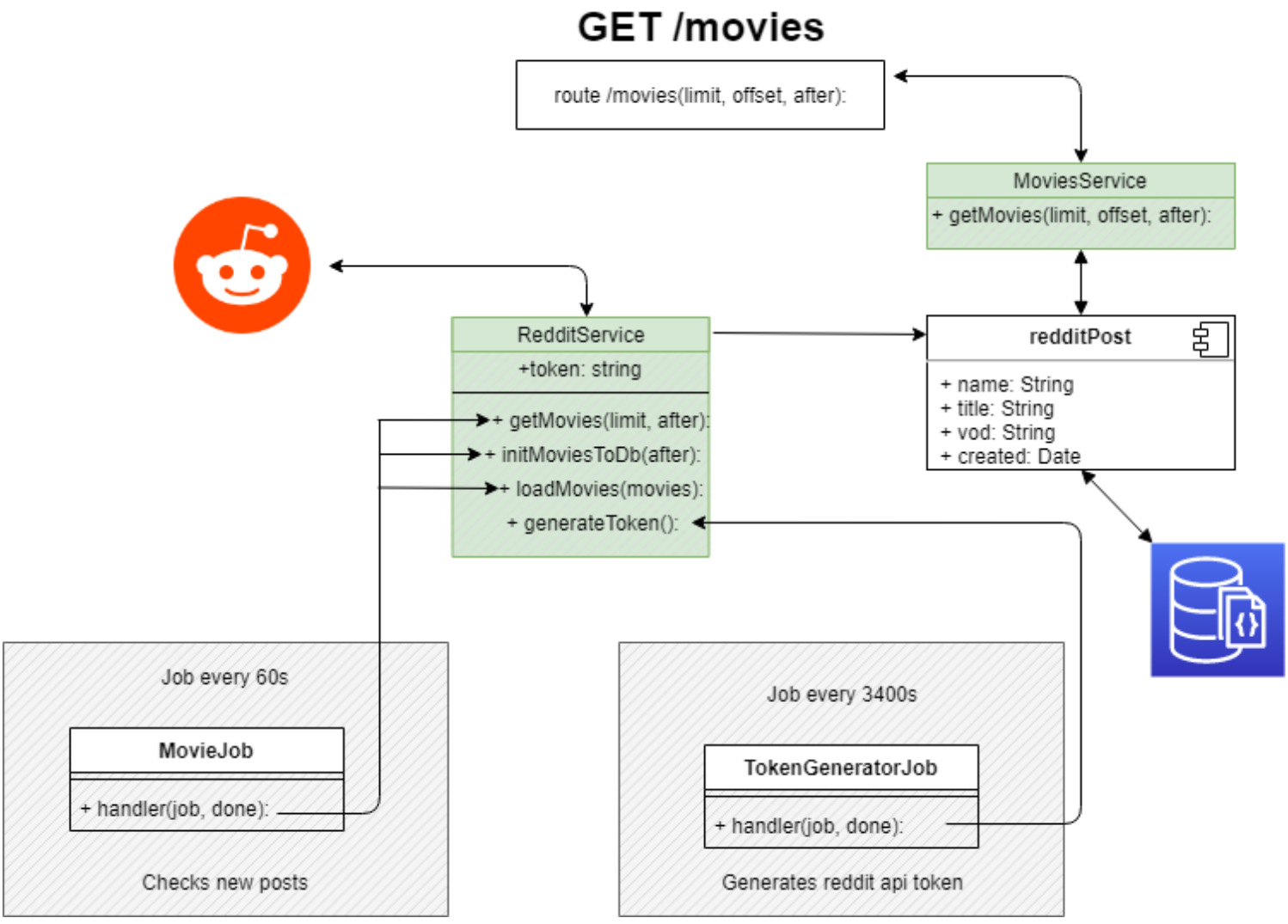
Cała aplikacja będzie działać na modelach JavaScript, a modele będą się zmieniać wraz z rozwojem aplikacji. Dodatkowo niepotrzebne będą skomplikowane powiązania obiektów, gdyż tak naprawdę jedyne co nas interesuje to przekształcone w odpowiedni format posty o nowych wydaniach. Z tych powodów dobrym wyborem będzie nierelacyjna baza danych.

3. Analiza zagadnienia i jego modelowanie

3.1. Backend

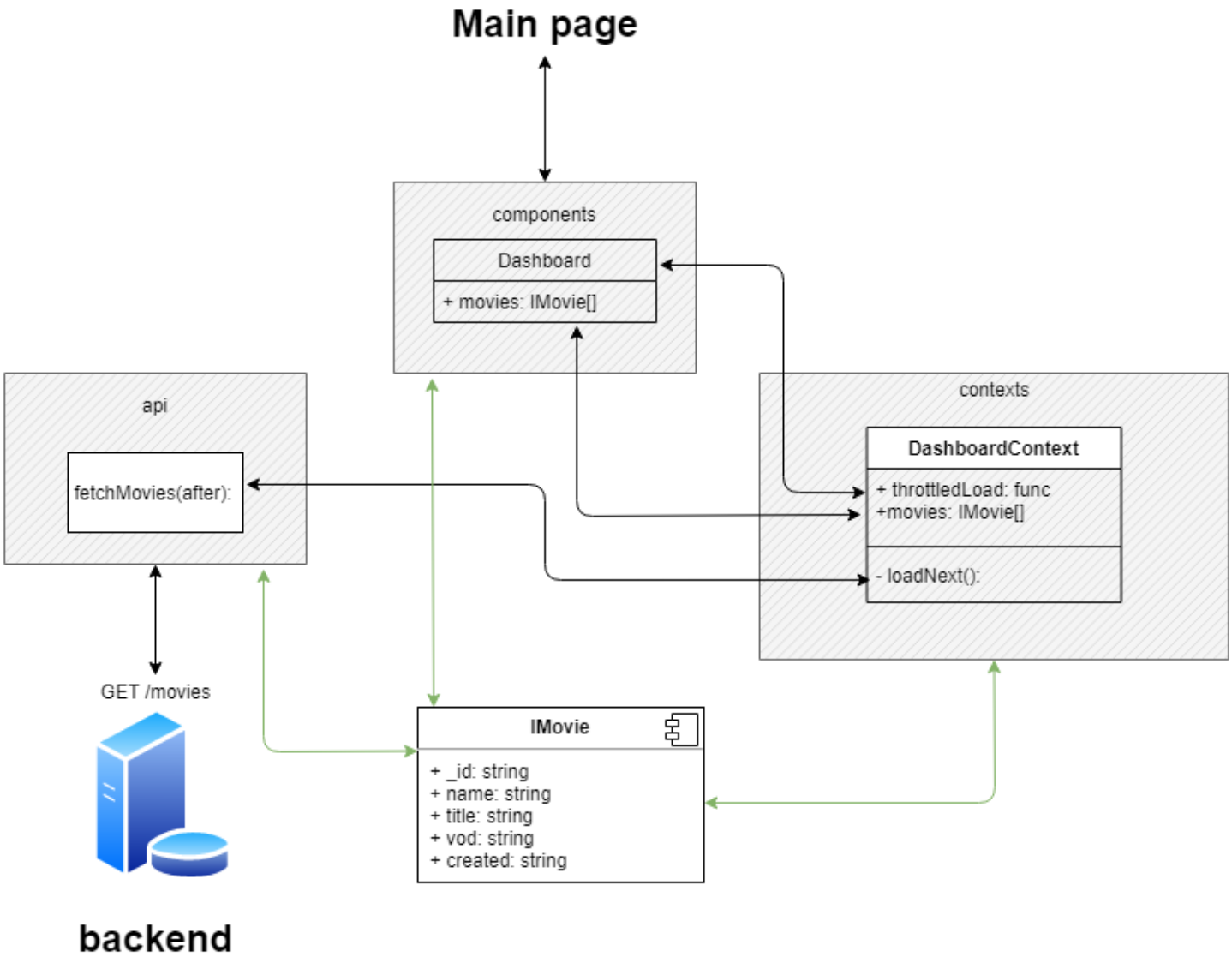
W tej części aplikacji potrzebne będą następujące elementy:

- Uzyskiwanie autoryzacji do API reddit.com.
- Wyszukiwanie nowych postów na portalu reddit.com.
- Transformacja tych postów na użyteczne dane.
- Zadania czasowe uruchamiające serwisy autoryzacji oraz wyszukiwania postów.
- Kontroler i serwis odpowiadający za udostępnienie API z zapisanymi wydaniami filmowymi.
- Zapisywanie i odczytywanie informacji z bazy.



3.2. Frontend

W tej części aplikacji wymagana będzie początkowo jedna strona, wyświetlająca najnowsze wydania filmowe. Posty powinny być ładowane dynamicznie w przyjemny dla użytkownika sposób.



4. Implementacja

4.1. Backend

Napisany został w języku Typescript w środowisku node.js połączonym z express[1]. Do automatycznego formatowania kodu podłączony został formatter od Google: GTS[2]. Do ogólnego połączenia struktury backendu,

[1] https://github.com/expressjs/express
[2] https://github.com/google/gts

użyty został framework typedi[3] pozwalający na wstrzykiwanie zależności. Efektem jest uzyskanie łatwego dostępu do odpowiednio przygotowanych obiektów w miejscach, które tego wymagają.

Przykład inicjalizacji kontenerów modeli dostępu do bazy danych i loggera.

```
    models.forEach(m => {
      Container.set(m.name, m.model);
    });

    Container.set('logger', LoggerInstance);
```

dependencyInjector.ts

Pozwala to na wstrzykiwanie zależności np. w konstruktorze.

```
constructor() {
    this.logger = Container.get('logger');
    this.redditPostModel = Container.get('redditPostModel');
  }
```

services/movies.ts

Logowanie informacji do konsoli odbywa się poprzez logger winston[4]. Czasowe wykonywanie zadań: wygenerowania tokenu dostępu oraz sprawdzenia i pobrania nowych postów z portalu społecznościowego reddit.com, wywoływane jest przez bibliotekę Agenda[5].

```
export default ({agenda}: {agenda: Agenda}) => {
    const movieJob = new MovieJob();
    agenda.define('load-movies', {concurrency: 1}, movieJob.handler);
    agenda.define(
      'generate-token',
      {priority: 'high', concurrency: 1},
      new TokenGeneratorJob().handler
    );

    agenda.on('ready', () => {
      agenda
        .every('3400 seconds', 'generate-token')
        .then(() => agenda.every('60 seconds', 'load-movies'))

        .then(() => agenda.start());

    });
  };
```

jobs.ts

Jako baza danych użyte zostało MongoDB Atlas[6], a modele obsługuje mongoose[7].

[3] https://github.com/typestack/typedi
[4] https://github.com/winstonjs/winston
[5] https://github.com/agenda/agenda
[6] https://www.mongodb.com/cloud/atlas
[7] https://github.com/Automattic/mongoose

4.2. Frontend

Podobnie jak w przypadku backendu, użyty został język Typescript z podłączonym formaterem GTS. Podstawą tej części aplikacji jest framework React. Strona składa się z jednego adresu głównego bez żadnych ścieżek. Do implementacji wyglądu użyta została biblioteka komponentów Material-UI[8]. Na głównej stronie serwowany jest komponent Dashboard. Dostęp do załadowanych materiałów wideo z backendu dostarczony został przez provider DashboardContext.

Do wczytywania kolejnych (starszych) filmów zaimplementowano Infinite scroll. Przesunięcie strony na sam dół wywołuje funkcje ładującą starsze widea.

```
const onBottomHit = throttledLoad;

useEffect(() => {
  if (initialLoad) {
    onBottomHit();
    setInitialLoad(false);
  }
}, [onBottomHit, initialLoad]);

useEffect(() => {
  const onScroll = () => {
    if (hasMoreData && isBottom(contentRef)) {
      onBottomHit();
    }
  };
  document.addEventListener('scroll', onScroll);
  return () => document.removeEventListener('scroll', onScroll);
}, [onBottomHit, hasMoreData]);
```

Ilość wywołań ograniczona jest przez funkcje *throttle* z biblioteki lodash[9].

```
const throttledLoad = _.throttle(() => {
    loadNext();
  }, 500);

  const loadNext = () => {
    const [lastMovie] = movies.slice(-1);

    if (lastMovie && !lastMovieIds.includes(lastMovie._id)) {
      setLastMovieIds([...lastMovieIds, lastMovie._id]);

      fetchMovies(lastMovie.name).then(nextMovies =>
        setMovies([...movies, ...nextMovies])
      );
    }
  };
```

[8] https://github.com/mui-org/material-ui
[9] https://github.com/lodash/lodash

5. Podsumowanie

W obecnym stanie aplikacja dostarcza podstawowych funkcjonalności ustalonych w poprzednich punktach. Stworzony został prosty interfejs wyświetlający nowe wydania filmowe oraz serwis automatycznie zbierający potrzebne dane. Kolejnym krokiem w rozwoju jest dodanie wyszukiwania premier nowych seriali. Wymagać to będzie implementacji parsowania potrzebnych informacji z poddomeny zrzeszającej fanów seriali.