Sprawozdanie końcowe z projektu

Aplikacja gromadząca dane ze sklepów internetowych z grami - *GameScrapper*

Przemysław Woźny 137347
Jakub Florczak 137121
Bartosz Porębski 136303
Andrzej Skrobak 136310

WIiT, Informatyka (WE) semestr 6

15.09.2020 r.

1. Opis aplikacji

GameScraper to aplikacja umożliwiająca obserwowanie różnic w cenach gier dostępnych na trzy konsole to grania tej generacji :

- XboxOne
- PS4
- Nintendo Switch

Dane dotyczące gier “scrapowane” są z dostępnych dla Polskich użytkowników sklepów, bądź jak w przypadku Nintendo z sklepu nintendo store uk i następnie przewalutowywane wykorzystując najnowszy kurs walut.

Użytkownik może oznaczyć obserwowaną grę aby otrzymać powiadomienie mailowe gdy cena gry się zmieni.

2. Podział prac

Opis zadania	Osoby realizujące zadanie
Stworzenie dokumentacji	Bartosz Porębski, Przemysław Woźny
Stworzenie scrapera pobierającego dane z platformy PS4 korzystającego z Scrapy	Przemysław Woźny
Stworzenie scrapera pobierającego dane z platformy Nintendo Eshop UK korzystającego z Scrapy-splash	Przemysław Woźny
Stworzenie scrapera pobierającego dane z platformy Xbox korzystającego z Selenium	Przemysław Woźny
Utworzenie komend Django które po wywołaniu dodają nowe gry do bazy danych i aktualizują dane dotyczące gier już zapisanych.	Przemysław Woźny
Stworzenie panelu logowania (frontend)	Bartosz Porębski, Jakub Florczak
Stworzenie panelu rejestracji (frontend)	Bartosz Porębski, Jakub Florczak
Stworzenie Api służącego autoryzacji użytkownika	Bartosz Porębski, Jakub Florczak
Autoryzacja użytkownika w aplikacji frontendowej	Bartosz Porębski, Jakub Florczak
Zaprojektowanie głównego układu strony i stworzenie paska nawigacji	Jakub Florczak
Stworzenie panelu bocznego	Jakub Florczak, Przemysław Woźny
Utworzenie endpointu udostępniającego informacje dotyczące gier	Przemysław Woźny, Andrzej Skrobak
Stworzenie głównego komponentu do wyświetlania zescrapowanych gier	Bartosz Porębski, Jakub Florczak
Utworzenie endpointu pozwalającego na paginację zwracanych danych	Przemysław Woźny
Stworzenie komponentu paginacji stron	Bartosz Porębski, Jakub Florczak
Dodanie do endpointu możliwości filtrowania konsoli.	Przemysław Woźny

Stworzenie komponentu do wyświetlania przefiltrowanych gier. Filtry: PS4, Xbox, Nintendo	Bartosz Porębski, Jakub Florczak
Stworzenie endpointu zwracającego informacje dotyczące historii cen danej gry	Przemysław Woźny
Stworzenie widoku pojedynczej gry wraz z wykresem historii jej cen	Bartosz Porębski, Jakub Florczak
Umożliwienie subskrybcji gier przez użytkownika (frontend)	Bartosz Porębski
Stworzenie planisty który automatycznie aktualizuje ceny i wysyła maile na adresy użytkowników których obserwowane gry zmieniły ceny.	Przemysław Woźny
Stworzenie modelu bazy danych	Andrzej Skrobak

3. Wykorzystywane technologie

Do utworzenia projektu wykorzystujemy znane nam technologie. Ze względu na pewność w kompatybilności wszystkich używanych bibliotek uznaliśmy że do tworzenia backendu najlepszym wyborem będzie język Python w wersji 3.7 w połączeniu z frameworkiem Django. Do frontendu natomiast zdecydowaliśmy się na język JavaScript wraz z biblioteką React w wersji v16.3.1.

- Python
    - Intuicyjny i przejrzysty, wysokopoziomowy język posiadający dużą ilość bibliotek pozwalających na szybką rozbudowę projektu w kolejne funkcjonalności. Wybraliśmy go głównie ze względu na framework Django.

- Django/Django Rest-Framework
    - Wysokopoziomowy web framework wykorzystujący język Python. Pozwala on na szybkie, zautomatyzowane tworzenie aplikacji internetowych i REST Api.

    - Dużą zaletą Django jest zautomatyzowana komunikacja z bazą danych za pomocą Modeli, które w istocie są layoutem twojej bazy danych z dodatkowymi meta-danymi.

- SQLite
    - System zarządzania bazą danych oraz biblioteka C implementująca szybkie i małej wielkości, ale o dużej niezawodności, w pełni funkcjonalne silniki bazy danych SQL.
    - Wybór na SQLite padł głównie ze względu na prostotę naszej bazy i wbudowaną funkcjonalność z Web Frameworkiem Django.

- Scrapy
    - Szkielet do tworzenia aplikacji napisany w języku programowania Python i służący do pisania robotów internetowych, które przeszukują strony internetowe i wydobywają z nich określone dane.

- Splash
    - Usługa renderowania JavaScript. To lekka przeglądarka internetowa z interfejsem API HTTP, zaimplementowana w Pythonie przy użyciu Twisted i QT5. Pomaga nam aby aplikacja była w pełni asynchroniczna.
    - Wykorzystywana w połączeniu ze Scrapy do przeszukiwania stron wykorzystujących JavaScript do renderowania części strony. (W naszym wypadku wykorzystywana wraz z frameworkiem Scrapy.

- Selenium
    - Usługa automatyzująca przeglądarkę, pozwalająca nam pobierać dane ze stron ładujących dane za pomocą JavaScript.
    - Wybraliśmy selenium ze względu na problemy biblioteki Splash w renderowaniu strony xbox.com

- ChromeDriver
    - Narzędzie do automatycznego korzystania z przeglądarki Chrome, wykorzystywana przez Selenium

- React.js
    - Biblioteka do języka JavaScript umożliwiająca w prosty sposób połączenie logiki aplikacji internetowej wraz z jej wyglądem. Dużym jej atutem jest reagowanie na zmiany danych w aplikacji i odświeżanie tylko tych komponentów, które tego wymagają.

- Redux
    - Biblioteka języka JavaScript umożliwiająca utworzenie i korzystanie ze współdzielonego ogólnego stanu aplikacji

Narzędzia pomocne w procesie tworzenia aplikacji to :

- Zintegrowane środowiska programistyczne:
    - Pycharm
        - Wykorzystywany do programowania w języku Python, pomocny również w zarządzaniu nad kontrolą wersji.

    - Visual Studio Code
        - Wykorzystywany do programowania w językach Python i JavaScript.

- Kontrola Wersji
    - Git
        - Rozproszona kontrola wersji wykorzystywana do śledzenia zmian w kodzie źródłowym aplikacji
    - Github
        - Serwis internetowy wykorzystywany do projektów programistycznych korzystających z Git-a jako kontroli wersji.
- Docker

- Otwarte oprogramowanie służące do wirtualizacji na poziomie systemu operacyjnego, w naszym wypadku wykorzystywane do usługi Splash.

4. Architektura

Aplikacja posiada architekturę opartą o mikroserwisy, czyli każda klasa cechuje się zasadą pojedynczej odpowiedzialności. Pozwala to na uniknięcie błędów lawinowych oraz umożliwia łatwe oraz szybkie naprawy błędów. Jednak nasza decyzja była podyktowana głównie prostotą rozwoju aplikacji, gdyż niedostępność jednego mikroserwisu nie uniemożliwia dalszego działania innym i pozwala skalować tylko te elementy, które wymagają więcej zasobów.

5. Funkcjonalności

Zadaniem aplikacji jest zbieranie informacji dotyczących gier na stronach głównych producentów konsol na rynku (Nintendo Switch, Xbox One, PS4).

- Dane są zbierane i przekazywane do bazy danych za pomocą narzędzi Scrapy, Splash i Selenium.
- Następnie przekazywane są do bazy danych, która przechowuje historię cenową gier, jak i konta użytkowników i informacje czy ceny zostały zaktualizowane.

Przechowywane dane są serializowane za pomocą automatycznego serializera danych wbudowanego w Django. Następnie przerabiane są na endpointy REST Api.

- Utworzone są następujące Endpointy:
  - GET /api/games - zwraca wszystkie dane dotyczące przechowywanych gier
  - GET /api/price?id={id} - zwraca historię ceny gry o wybranym identyfikatorze

- GET /api/game_info?id={id1,id2} - zwraca wszystkie dane dotyczące przechowywanych gier o danych identyfikatorach
- POST /api/acc_games - przesyła informację o grze którą dany użytkownik dodał do obserwowanych (informacja o grze i użytkowniku przesyłana jest w ciele żądania http).
- DEL /api/del_games/<id> - usuwa gry o podanym id(informacja o danym użytkowniku przesyłana w ciele żądania http).

Dane następnie są przekazywane za pomocą endpointów do użytkowników w formie graficznego widoku strony. Dzięki temu użytkownik przed dodaniem gry do obserwowanych jest w stanie sprawdzić jej cenę, dokładną historię ceny jak i może bezpośrednio przejść do strony gry aby dostać większą ilość informacji.

Każdy użytkownik chcący korzystać z naszej strony musi być zalogowany.

6. Szczegóły implementacyjne

- Projekt polega na utworzeniu bazy danych, którą porównywać będziemy z danymi pobieranymi przez scrapery. Do tego potrzebna nam była baza danych, a korzystając z Django mogliśmy stworzyć modele które opisują bazę danych.
  Nasze modele to :
  - **GamesModel** posiadający pola:
    - id (klucz podstawowy) - unikalny identyfikator gry,
    - title (pole tekstowe) - tytuł gry,
    - price (pole wartości zmiennoprzecinkowych) - cena gry,
    - console (pole tekstowe) - nazwa konsoli,
    - img (pole tekstowe) - link do zdjęcia okładki na serwerze zewnętrznym,
    - link (pole tekstowe) - link do strony gry,
    - date (pole typu data) - data aktualizacji rekordu.
  - **HistoryModel** posiadający pola:

        - id (klucz podstawowy)
        - game_id (klucz obcy - GamesModel) - unikalny identyfikator gry,
        - title (pole tekstowe) - tytuł gry,
        - price (pole wartości zmiennoprzecinkowych) - cena gry,
        - date (pole typu data) - data aktualizacji rekordu.
    - **AcountGames** posiadający pola:
        - id (klucz podstawowy)
        - account_id (klucz obcy - User [wbudowany model Django **auth.models.User**]) - unikalny identyfikator użytkownika,
        - game_id (klucz obcy - GamesModel) - unikalny identyfikator gry,
    - **GamesChanged** posiadający pole:
        - id (klucz podstawowy)
        - game_id (klucz obcy - GamesModel) - unikalny identyfikator gry,

  Każdy z modeli ma ważną funkcjonalność:
    - GamesModel - przechowuje informacje na temat gier,
    - HistoryModel - przechowuje zmiany cen gier,
    - AccountGames - wiąże użytkowników z obserwowanymi grami,
    - GamesChanged - informuje o zmienionych cenach gier.

- Dane w backendzie zbierane są do bazy danych za pomocą scraperów które są przypisane do komend “crawl” i “xbox” (**crawl.py** i **xbox.py**)
    - **xbox.py** wykorzystuje selenium do zbierania danych. Selenium oczekuje 15 sekund, lub aż załaduje się element . Gdy element zostanie załadowany, dane są sukcesywnie zbierane za pomocą funkcji:

.find_element_by_class_name(‘klasa”)**.get_attribute**(‘innerHTML’)

      Następnie dane są przypisane do zmiennych i przekazywane do wcześniej utworzonych modeli które zapisywane są w bazie danych.

    - **crawl.py** wykorzystuje scrapy i splash do zbierania danych. Scrapy pozwala na tworzenie tak zwanych pająków, w naszym wypadku to “pająki” **ps4.py** i **nintendo.py**. Klasy NintendoSpider(scrapy.Spider) i PS4Spider(scrapy.Spider) posiadają funckje **start_request()** i **parse()**, które rozpoczynają działanie “pająków”. W funkcji parse() korzystamy z :

```
.css('sciezka_do_danych::text/::attr(href)').get()
```

      Następnie dane są przypisane do zmiennych i przekazywane do wcześniej utworzonych modeli które zapisywane są w bazie danych.

- Do komunikacji z częścią webową naszej aplikacji potrzebowaliśmy REST Api, dlatego skorzystaliśmy z biblioteki pomocniczej do frameworku Django o nazwie Django-Rest-framework. Każdy model jest dzięki niej serializowany i pozwala to na szybkie utworzenie endpointów (opisane w punkcie 5).

- Aplikacja aby działała potrzebuje codziennie aktualizować ceny gier, w tym wypadku wykorzystaliśmy bibliotekę **apscheduler** która tworzy planistę który pozwoli nam na automatyczne sprawdzanie ceny codziennie o godzinie 22.30. Scheduler wykorzystuje wbudowaną w pythona bibliotekę os, która wywołuje komendę “runcom”, która wywołuje wcześniej wspomniane komendy (“crawl”, “xbox”). Runcom również wysyła maile za pomocą Django, do użytkowników których obserwowane gry zmieniły ceny.

- Aplikacja Webowa - implementacja została wykonana z wykorzystaniem biblioteki react.js. Charakteryzuje się ona podziałem aplikacji webowej na komponenty. W naszym przypadku głównymi komponentami są: górny pasek nawigacji i boczny pasek nawigacji.
    - Górny pasek nawigacji plik Topbar.js - główną metodą używaną w tym pliku jest conditionalAuthDisplay, która w zależności od tego czy użytkownik jest uwierzytelniony

zwraca inne funkcjonalności paska górnego. Komponent ten ma dodatkowo przewidziany wrapper stylizujący StyledTopBar napisany przy użyciu biblioteki styled-components
    - Pasek boczny nawigacji Sidebar.js - wyświetlany tylko gdy użytkownik jest uwierzytelniony. Posiada on zbiór przycisków przekierowujących do widoków wyświetlających przefiltrowane gry.
    - Komponent Content - komponent służący do routingu. Może zwracać on ścieżki publiczne i prywatne dostępne tylko dla uwierzytelnionych użytkowników. W komponencie tym renderowane są również: pasek boczny i pasek górny.
    - Komponent MyGames - pobiera i wyświetla gry przypisane do konta danego użytkownika. Renderowany przy linku /filtered/MyGames.
    - Komponent FilteredGamesView - pobiera i wyświetla dane dotyczące gier w zależności od przekazanego do niego id poprzez link /filtered/:id pobiera odpowiednio przefiltrowane gry. W przypadku nie podania id wyświetli on pierwsze 50 gier z bazy danych.
    - Komponent CardView - służy pobraniu szczegółowych danych gry i wyświetleniu ich wraz z wykresem historii cen tej gry
    - Komponent PrivateRoute - komponent służący do routingu sprawdzający czy użytkownik jest uwierzytelniony. Jeżeli nie komponent przekieruje go do strony logowania
    - Komponent Register - służy do rejestracji. Pobiera dane od użytkownika i waliduje je a następnie wysyła on metodę post do serwera.
    - Komponent Login - służy do logowania. Pobiera dane od użytkownika i przesyła do serwera, po otrzymaniu poprawnej odpowiedzi przekierowuje go do uwierzytelnionych widoków.
- Token uwierzytelniający użytkownika przechowywany jest w globalnym stanie aplikacji stworzonym przy użyciu biblioteki redux.

7. Problemy implementacyjne

W przypadku aplikacji webowej największym problemem było dostosowanie przepływu danych między komponentami, gdyż nie mieliśmy informacji na temat tego jakie komponenty gier są w danej chwili wyrenderowane w głównym widoku. Działo się tak przez to, że były one ładowane dynamicznie i nie mieliśmy bezpośredniego dostępu do danych, które przechowywały. Udało nam się rozwiązać ten problem poprzez przypisanie ID danej karty do jej przycisku dzięki czemu mógł on przekazać je do następnego widoku jakim są szczegóły danej gry.

W przypadku backendu aplikacji problemy były związane głównie z dostosowaniem endpointów dla aplikacji webowej. Przy wykorzystywaniu endpointu “GET /api/game_info?id={id1,id2}” potrzebne było przyjmowanie kilku identyfikatorów aby można było pobierać dane dla dużej ilości gier. Problemem jest to że Django nie wspiera takich funkcji, dlatego musieliśmy przerobić serializer aby przyjmował kilka identyfikatorów. Wykorzystałem do tego wbudowaną w pythona funkcję split(), która podzieliła identyfikatory po przecinkach. Gdy przygotowana była lista identyfikatorów wykorzystałem sumowanie querysetu, a następnie dany set zwracany był jako endpoint.

8. Użytkowanie aplikacji

Wideo z użytkownaniem programu.

https://github.com/flopczak/GameScraper

Film pokazuje możliwości naszej aplikacji ze strony panelu użytkownika. Gdy gra dodana jest do obserwowanych a cena zostanie zaktualizowana użytkownik otrzymuje takie powiadomienie mailowe:

Sale on games

From: <229b951b0a-1c9ea2@inbox.mailtrap.io>
To: <user2212@gmail.com>
Show Headers

2020-09-21 19:53, 7.6 KB

HTML | HTML Source | Text | Raw | Analysis | Check HTML | Tech Info

Hi there,

These games that you are subscribing to are on sale:

Super Smash Bros. Ultimate Fighters Pass Vol. 2 - Digital Download

135.0 zł

Check Offer