POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Patryk Zdral
Nr albumu s20117

Strumienie RSS jako źródło istniejących oraz nowych treści w Internecie

Praca magisterska napisana pod kierunkiem:

Dr. inż. Mariusz Trzaska

Wrocław, luty, 2021

Streszczenie

Praca dotyczy zagadnienia pozyskiwania treści w sieci opierając się o technologię RSS. w pracy przeanalizowano aktualnie popularne rozwiązania będące implementacjami czytników RSS oraz problematykę przy tworzeniu narzędzi tego typu. Na podstawie dokładnej analizy został przygotowany projekt prototypu. Następnie stworzono implementację interaktywnego prototypu w nowoczesnych technologiach.

Stworzona aplikacja webowa w ramach pracy dostarcza użytkownikowi szereg możliwości zapewniających komfort agregacji informacji. Prototyp ma na celu przedstawienie oryginalnego podejścia do tematyki kanałów RSS i problematyki związanej z tym tematem.

Podziękowania

Autor pracy chciałby serdecznie podziękować promotorowi dr. Mariuszowi Trzasce za poświęcony czas, cenne uwagi i pomoc na każdym etapie pisania pracy.

Spis treści

1. Wstęp

RSS (ang. RDF *Site Summary* lub *Really Simple Sindication*) [1] jest standardowym formatem danych, który używany jest do dostarczenia treści użytkownikowi. Ten format zawiera często zmieniające się treści takie jak wpisy z blogów czy wiadomości ze świata. Technologia RSS bazuje na języku znaczników XML. Jej pierwsza wersja została stworzona w 1999 roku przez Dan'a Libby. Dzięki RSS agregacja danych w jednym miejscu stała się możliwa i zmniejsza problem rozproszonych informacji w sieci. Aktualnie najnowsza wersja tego standardu to 2.0. i mimo wielu lat istnienia na rynku dalej jest powszechnie wykorzystywana. Najpopularniejszym czytnikiem online był niegdyś Google Reader, ale w 2013 roku Google zdecydował zamknąć system. Wraz z zamknięciem platformy wielu ludziom może wydawać się, że RSS przestał być używany, ale rzeczywistość jest inna, co pokazują statystyki [2]. w dalszym ciągu istnieje wiele innych rozwiązań wykorzystujących tę technologię i dostarczających możliwość szybkiego i bezproblemowego przeglądania wiadomości. Dla formatu do dziś nie stworzono alternatywy, a z dostępnych rozwiązań. korzystają setki tysięcy ludzi. Autor niniejszej pracy skupił się na analizie aktualnych systemów bazujących na technologii RSS i problematyki związanej z tym tematem. w wyniku tego powstał autorski prototyp strony webowej będący implementacją systemu opartego o kanały RSS.

1.1. Systemy do czytania i agregacji newsów

Systemy do czytania i agregacji newsów bazujące na kanałach RSS dają użytkownikom wiele korzyści. Przede wszystkim jest to oszczędność czasu. Oczywiście możliwe jest śledzenie wielu stron na raz, ale niekoniecznie jest to wygodny sposób, z powodu mocnego rozproszenia informacji w sieci. Platformy tego typu pozwalają agregować wiadomości w jednym miejscu. Od momentu zamknięcia najpopularniejszego czytnika RSS – Google Readera (lipiec 2013), systemy będące niegdyś prostymi platformami bazującymi na RSS znacznie się rozwinęły. w wyniku rosnących potrzeb użytkowników wprowadzono wiele zaawansowanych funkcjonalności i aktualnie nie są to tylko miejsca, w których zbiera się informacje. Inteligentne algorytmy śledzące ruch użytkownika, czy zaawansowane filtry umożliwiają dostosowanie zawartości do użytkownika i tym samym eliminowanie zbędnych, nieinteresujących go artykułów. Integracja z portalami społecznościowymi, lub nawet wbudowane blogi w portale, pozwalają w prosty sposób na wymianę informacje między ludźmi i śledzenie aktywności innych użytkowników. w rozdziale 2. zostaną bliżej przedstawione najpopularniejsze platformy.

1.2. Cel pracy

Celem niniejszej pracy jest analiza problematyki związanej z tworzeniem systemów wykorzystujących kanały RSS do agregacji wiadomości. w wyniku analizy biznesowej i technicznej istniejących systemów, powstanie funkcjonalny prototyp będący autorską wersją czytnika RSS.

1.3. Rezultaty pracy

Głównym rezultatem stworzonej pracy jest ocena i analiza platform bazujących na technologii RSS oraz opracowanie założeń własnego systemu, umożliwiającego zbieranie wiadomości. Autor konkretnie skupił się na rozwiązaniach zwanych czytnikami RSS.

Dodatkowym rezultatem, który ma znaczący wpływ na jakość pracy, jest prototyp aplikacji webowej. Został on stworzony w nowoczesnych technologiach, zgodnie z dobrymi praktykami programistycznymi. Prototyp umożliwia przeglądanie i agregację treści dostępnych w Internecie

bazując na kanałach RSS. Stworzony system webowy, na podstawie zaimplementowanych algorytmów i wyborów użytkownika, w sposób inteligentny, dostosowuje artykuły i tworzy sugestie potencjalnie interesujących. Umożliwia również proponowanie nowych treści.

1.4. Organizacja pracy

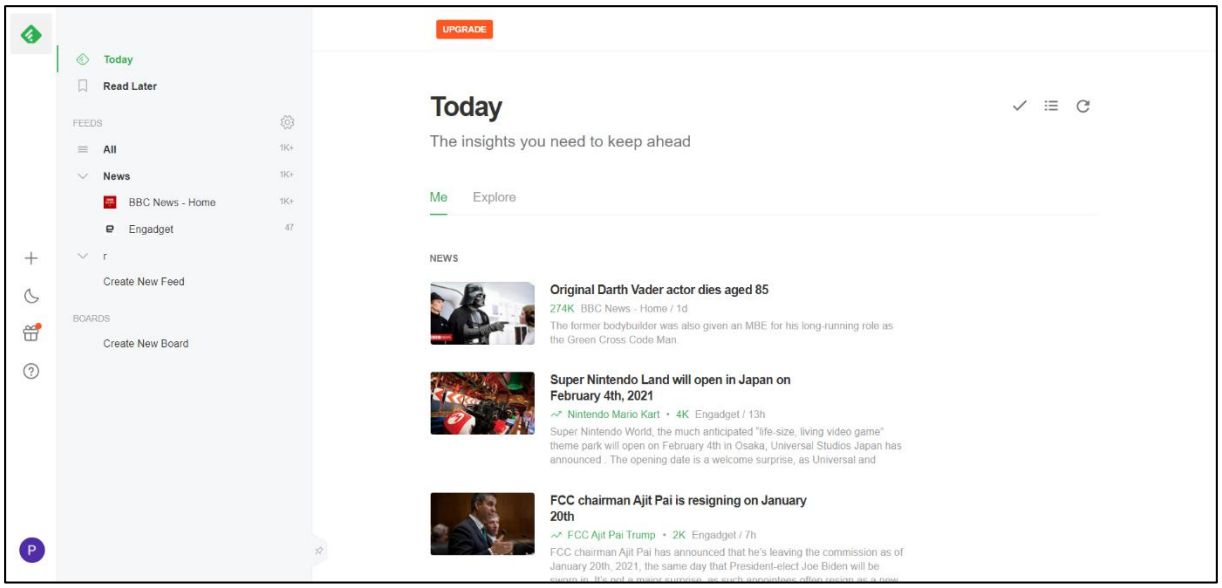
Praca została podzielona na siedem rozdziałów. Pierwsze dwa rozdziały przedstawiają aktualnie najpopularniejsze rozwiązania będące czytnikami RSS i problematykę związaną z tworzeniem takich systemów. w trzecim rozdziale na podstawie analizy aktualnych rozwiązań zaproponowano autorską implementację systemu tego typu. Kolejny rozdział przedstawia użyte technologie przy implementacji prototypu. w piątym rozdziale opisano proces implementacji prototypu oraz sposób jego testowania. w dwóch ostatnich rozdziałach przedstawiono zalety, wady i plany rozwojowe stworzonego prototypu. Przeanalizowano także elementy, które muszą zostać dopracowane, aby produkt mógł być wdrożony do użytku komercyjnego. Ostatnim elementem pracy jest jej podsumowanie i wysunięcie wniosków.

2. Przegląd istniejących rozwiązań

W tym rozdziale opisano najpopularniejsze implementacje czytników RSS. Każdy system został krótko opisany oraz przedstawiono jego zalety i wady.

2.1. Feedly

Feedly [3] jest zdecydowanie najpopularniejszym czytnikiem RSS na rynku [4]. Atrakcyjny i intuicyjny interfejs graficzny sprawia, że jego użytkowanie jest bardzo proste i przyjemne. Jedynym, co użytkownik musi zrobić, aby zacząć korzystać z aplikacji jest dodanie ulubionych strumieni RSS z tych portali do swojej kolekcji. Aplikacja dostępna jest w wersji web, na iOSa i Androida. Darmowa wersja tego portalu pozwala na subskrypcję aż do 100 kanałów, co z pewnością wystarczy większej ilości użytkowników. Feedly udostępnia możliwość logowania przez wszystkie najpopularniejsze portale takie jak Facebook, Twitter, Google. Daje możliwość dzielenia się artykułami na portalach społecznościowych poprzez zdefiniowane przyciski. Rysunek 1. przedstawia wygląd opisywanej aplikacji po wejściu na stronę główną.



Rysunek 1. Strona główna aplikacji Feedly

Zalety:

- prosty i atrakcyjny interfejs graficzny;
- integracja z portalami społecznościowymi;
- wieloplatformowość – dostępna wersja na przeglądarki i system Android, iOS;
- bardzo dobra aplikacja na systemy mobilne.

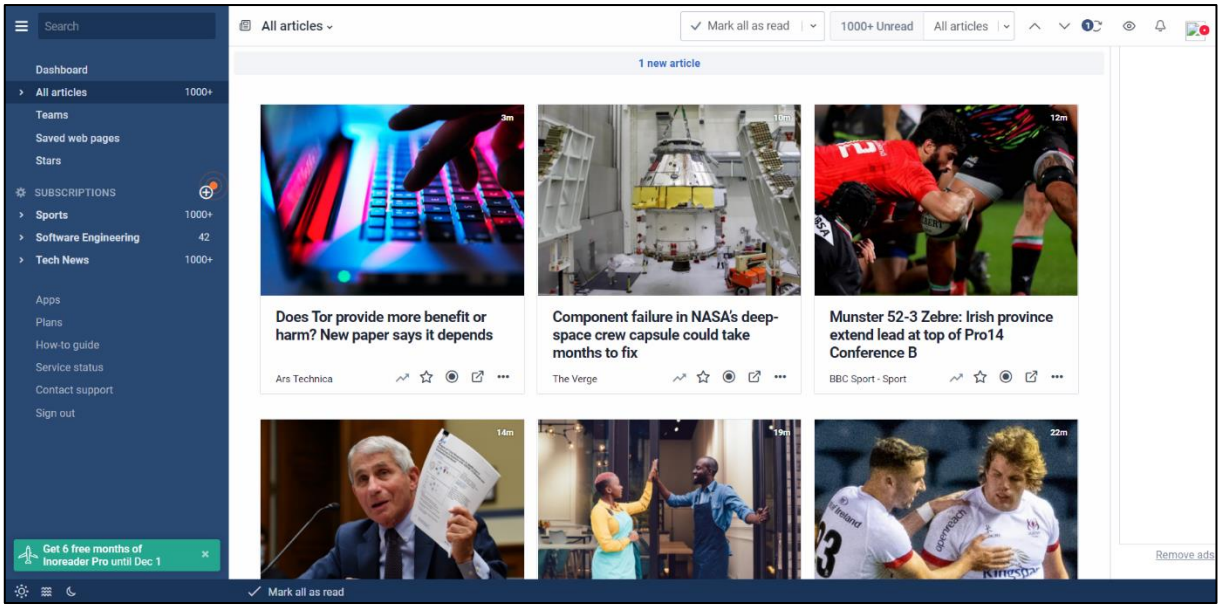
Wady:

- zaawansowane funkcjonalności dostępne dopiero w wersji pro np. Feedly Teams, notatki pod artykułami, podkreślanie ważnych fragmentów, alerty słów kluczowych z Google;
- pomimo wielu próśb użytkowników w dalszym ciągu brak notyfikacji push;

- brak wsparcia offline;
- możliwość wyszukiwania po słowie klucz jedynie w wersji płatnej.

2.2. Inoreader

Inoreader [5] jest bardzo funkcjonalnym czytnikiem i praktycznie darmowym. w przeciwieństwie do Feedly nie są wymagane żadne opłaty, aby śledzić nieograniczoną liczbę kanałów RSS oraz przeszukiwać subskrypcje. Inoreader ma także nieograniczone w czasie archiwum, w przeciwieństwie do wielu innych systemów. GUI aplikacji jest nieco odmienne od innych czytników i przypomina bardziej to znane np. z aplikacji Trello. Na rysunku nr 2 ukazana jest strona główna Inoreader.



Rysunek 2. Strona główna aplikacji Inoreader

Zalety:

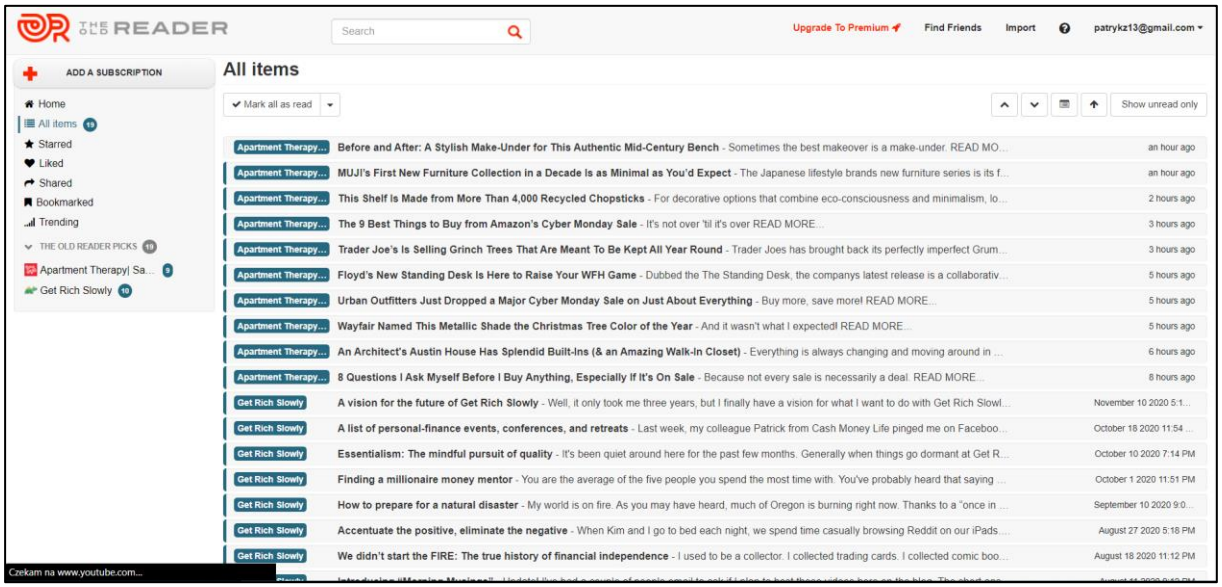
- darmowe wyszukiwanie artykułów;
- nieograniczona liczba subskrypcji kanałów;
- szybkość działania;
- intuicyjny przegląd artykułów, co pozwala na szybkie znalezienie interesujących artykułów.

Wady:

- reklamy w darmowej wersji;
- zaawansowane funkcje niedostępne w płatnej wersji.

2.3. The Old Reader

The Old Reader [6] jest systemem stworzonym tylko na przeglądarki. Jest świetnym rozwiązaniem, jeśli użytkownicy lubią dzielić się materiałami ze znajomymi. The Old Reader po połączeniu konta z Google lub Facebookiem udostępnia szereg społecznościowych funkcjonalności. Na rysunku 3. widnieje *screen* początkowej strony prezentowanej aplikacji.



Rysunek 3. Strona główna aplikacji The Old Reader

Zalety:

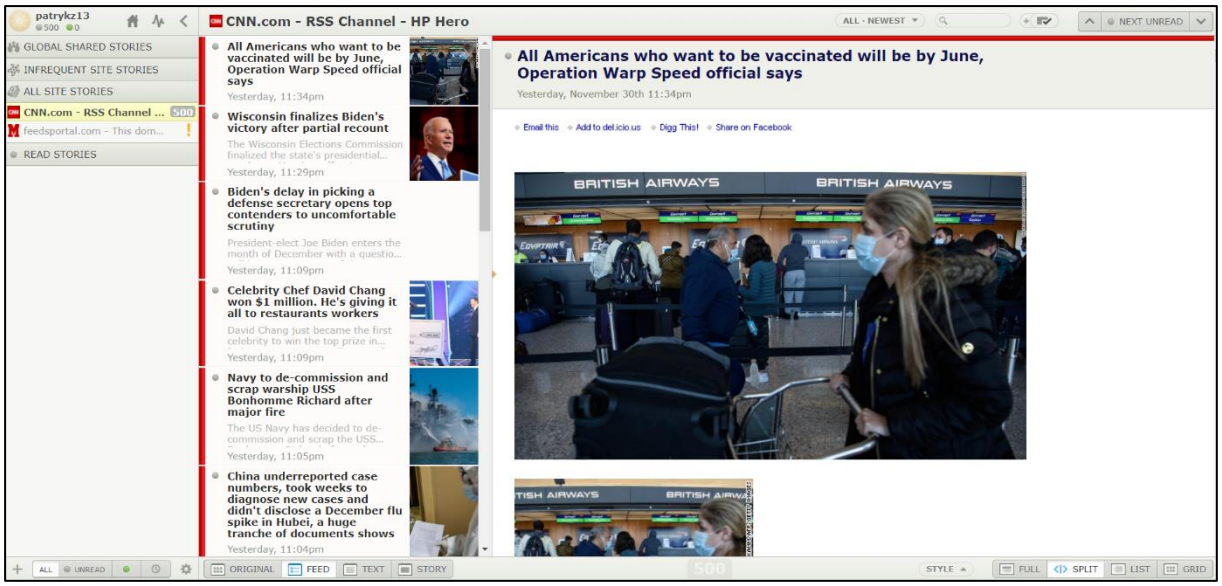
- po połączeniu konto z Google/Facebookiem, możliwość dzielenia się artykułami ze znajomymi, śledzenia co lubią, wspólnych dyskusji;
- podobny do nieaktywnego Google Readera;
- łatwy w użytkowaniu.

Wady:

- dostępna tylko wersja web;
- mało konfigurowalny – łatwe dodawanie lub usuwanie kanałów RSS, ale już nie da ich np. filtrować;
- płatny w wersji powyżej 100 subskrypcji i w przypadku potrzeby wyszukiwania pełno tekstowego.

2.4. Newsblur

Newsblur [7] jest aplikacją RSS, która pozwala w darmowej wersji na subskrypcję do 64 stron. Aplikacja cechuje się możliwością zaawansowanego filtrowania. Istnieje możliwość automatycznego wyróżniania lub ukrywania artykułów na podstawie określonych kryteriów. Newbslur ma także rozwinięte funkcje sieci społecznościowych, dzięki którym można udostępniać ulubione artykuły w sieciach społecznościowych lub bezpośrednio przez Newsblur. Dostępna jest także opcja śledzenia blogów innych ludzi oraz prowadzenie swojego własnego bloga. Dla zwizualizowania załączono zrzut ekranu, który przedstawia stronę główną aplikacji (rysunek 4.).



Rysunek 4. Strona główna aplikacji Newsblur

Zalety:

- zaawansowany mechanizm filtrowania artykułów;
- wysoka jakość rekomendowanych artykułów dzięki zaimplementowanych algorytmom odkrywania;
- skróty klawiszowe ułatwiają szybką pracę;
- wieloplatformowość – IOS, Android, aplikacja webowa;
- szybkość użytkowania.

Wady:

- brak możliwości posiadania nieprzeczytanych artykułów starszych niż 14 dni;
- cześć funkcjonalności dostępna tylko w wersji płatnej;
- przestarzały interfejs graficzny.

2.5. Feedbin

Feedbin [8] jest najmniej popularnym systemem wśród wszystkich innych wyżej wymienionych. Nie zmienia to jednak faktu, iż jest portalem wartym uwagi. System wyróżnia się nowoczesnym interfejsem graficznym, który został profesjonalnie zaprojektowany. Tryb pełnoekranowy, synchronizacja w czasie rzeczywistym i integracja z Twitterem to aspekty, które wyróżniają Feedbina. z pewnością może to przyciągnąć wielu użytkowników. Na załączonym rysunku (rys. 5) można ujrzeć stronę, która jest wyświetlana zaraz po wejściu do aplikacji.

Rysunek 5. Strona główna aplikacji Feedbin

Zalety:

- wysokiej jakości, atrakcyjny, intuicyjny interfejs graficzny;
- wspierane przez wiele innych aplikacji jak np. Reeder, ReadKit, SubToMe;
- wspiera zbieranie feedu w postaci JSON;
- integracja z Twitterem.

Wady:

- plan darmowy dostępny tylko przez 14 dni;
- brak możliwości wymiany informacji w aplikacji.

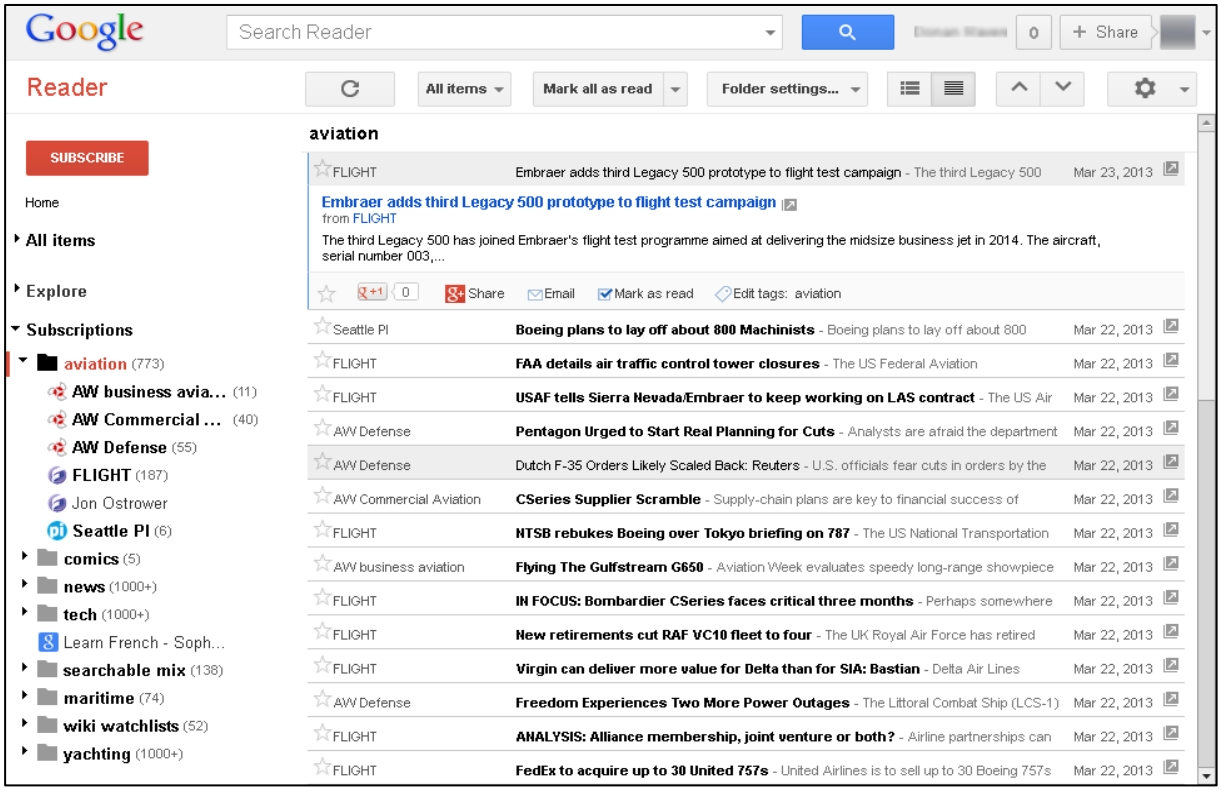
3. Propozycja rozwiązania

W niniejszym rozdziale zostanie przedstawiona propozycja prototypu aplikacji webowej bazując na analizie istniejących rozwiązań i przemyśleń autora.

3.1. Wnioski po analizie istniejących rozwiązań

W rozdziale drugim zostały omówione najpopularniejsze czytniki RSS. Na rynku istnieje znacznie więcej rozwiązań. Być może niektóre nawet mogłyby okazać się lepsze po bliższym zapoznaniu. Po analizie pięciu przykładowych systemów można dojść do wniosku, że żaden z nich nie jest idealny. Niektóre mają więcej wad, inne mniej. Nad każdą platformą przedstawioną w drugim rozdziale pracowały, przez długi czas, zespoły doświadczonych programistów. Dzięki intensywnym i długoterminowym pracom, aplikacje zawierają bogaty zasób zaawansowanych funkcjonalności.

Szaty graficzne i nowoczesne możliwości niektórych portali wskazują na to, że systemy są cały czas rozwijane, a nie tylko utrzymywane. Dobrym przykładem jest najpopularniejsze Feedly, którego interfejs wygląda bardzo atrakcyjnie i nowocześnie. w przypadku NewsBlur widać, że GUI jest przestarzałe. Mimo przeciętnej jakości oprawy graficznej system spełnia swoją rolę i ma kilka tysięcy aktywnych użytkowników [9]. Wszystkie portale posiadają dosyć podobny wygląd. Artykuły są po prawej stronie, a panel użytkownika po lewej. Widać, że pod względem graficznym strony trzymają się jednego schematu. Być może wzorowały się na prekursorze, czyli niefunkcjonującym aktualnie Google Readerze (rys. 6).



Rysunek 6. Wygląd aplikacji Google Reader; Źródło: [10]

3.2. Autorskie rozwiązanie

Autor w swojej pracy nie ma na celu stworzenia aplikacji pozbawionej wad omówionych platform. Planowana jest implementacja prototypu będącego wizją własnego podejścia do tematyki tworzenia czytników RSS. Postanowiono stworzyć funkcjonalny model aplikacji webowej działający w najczęściej używanych przeglądarkach. Został wykonany przy użyciu nowoczesnych technologii. Należy w tym miejscu podkreślić, iż autor pracy nie miał wcześniej doświadczenia z tego typu rozwiązaniami. Użycie ich na potrzeby projektu rozwinęło jego umiejętności i poszerzyło horyzonty programistyczne. Postanowiono nie wzorować się na interfejsach graficznych innych aplikacji, lecz wdrożyć swoją własną koncepcję.

Stworzony prototyp powinien zawierać wszystkie podstawowe funkcjonalności czytnika RSS. Patrząc z perspektywy standardowego użytkownika korzystającego ze stron, które są czytnikami, najważniejszą możliwością jest tworzenie list subskrypcji do wybranych kanałów RSS i gromadzenie treści w jednym miejscu. Będąc zasubskrybowanym do wybranego kanału możliwe jest przeglądanie aktualnych artykułów z tego źródła. w przypadku każdego czytnika RSS z rozdziału 2. jest to pierwsza widoczna funkcjonalność.

Niewątpliwie, ludzie mający mniej doświadczenia z kanałami RSS także powinni mieć możliwość odnalezienia się w systemie. Chodzi o osoby nie mające ulubionych strumieni RSS. Konsumenci potrzebują podpowiedzi jakie artykuły bądź strony są warte uwagi. Stworzony prototyp uwzględnia taki typ użytkowników i udostępnia rozwiązanie. Pierwszy panel, który jest widoczny zaraz po wejściu na stronę, pozwala na przeglądanie losowo wyświetlanych, popularnych i ciekawych artykułów. Istnieje możliwość dodania artykułu do ulubionych, odłożenia go do przeczytania na później lub usunięcia tak, żeby nigdy później się nie pokazał. Dostępna jest funkcja oceniania artykułów, na podstawie, której zawartość jest bardziej dopasowana do danego użytkownika. Inny panel, który może być użyteczny, przedstawia popularne artykuły z ostatniego tygodnia, miesiąca i roku. Zawiera także sugerowane i ciekawe wiadomości na podstawie zmodyfikowanego algorytmu *Slope One* [11].

W artykule [12] zostało wspomniane, że czytniki RSS działają w podobny sposób jak subskrypcja e-mailowa. Po podłączeniu się do źródeł otrzymywane są notyfikacje, dopóki jest się zasubskrybowanym. Niedziałający Google Reader znacząco przypomina, po względem graficznym, serwis *webmail* - Gmail. Aby przystąpić do korzystania aplikacji RSS należy podłączyć się do kilku źródeł i wypróbować, czy będzie użyteczna taka forma agregacji informacji. Jest to bez wątpienia najważniejsza wymagana funkcjonalność, która powinna się znaleźć w prototypie.

3.3. Analiza wymagań

W tym podrozdziale zostaną opisane wymagania funkcjonalne, niefunkcjonalne oraz dotyczące bezpieczeństwa aplikacji.

3.3.1 Wymagania funkcjonalne

Wymagania funkcjonalne określają co ma konkretnie realizować system, czyli definiują jego funkcje. Zostały one przedstawione w postaci tabelarycznej (tab.1).

Tabela 1. Wymagania funkcjonalne

Kod	Opis wymagania
WFU1	Do skorzystania z serwisu wymagana jest wcześniejsza rejestracja. Potwierdzenie rejestracji następuje poprzez kliknięcie w link przesłany na maila.

WFU2	Dostęp do treści jest możliwy tylko po zalogowaniu się serwisu.
WFU3	Możliwość zresetowania hasła.
WFU4	Personalizacja profilu – ustawienie nazwy użytkownika, opisu, zawodu, wybranie paczek z preferowanymi kategoriami.
WFU5	Paczki z kanałami są wyświetlone w zdefiniowanej sekcji. Definicja kategorii paczek odbywa się w widoku personalizacji profilu.
WFU6	Przeglądanie treści z kanałów RSS bazując na strumieniach danych zebranych w bazie danych. Dane o strumieniach bazują na informacjach zebranych od wszystkich użytkowników.
WFU7	Dodawanie artykułów do sekcji ulubione/do przeczytania/nie interesuje mnie. Ulubione i odłożone na później treści są dostępne w osobnym widoku.
WFU8	Ocenianie artykułów.
WFU9	Dodawanie swoich własnych kanałów RSS, usuwanie wcześniej dodanych strumieni, przeglądanie artykułów w zasubskrybowanych kanałach.
WFU10	Przeglądanie sugerowanych i najbardziej popularnych artykułów.

3.3.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne opisują ograniczenia usług i funkcji systemu. Określają również jakie cechy powinien posiadać tworzony projekt. Tego typu wymagania często mają bezpośredni wpływ na wybór architektury.

Stworzony prototyp ma mieć zaimplementowane proste mechanizmy bezpieczeństwa. Podstawowa wersja nie wymaga wdrażania skomplikowanych mechanizmów uwierzytelniania. w pierwszej iteracji tworzonego produktu, planowane jest tylko logowanie poprzez zarejestrowane konto w portalu w sposób klasyczny. Nie przewidziano logowania przez portale społecznościowe (Facebook, Twitter, Google). w tabeli 2. przedstawiono wymagania niefunkcjonalne zdefiniowane w projekcie.

Tabela 2. Wymagania niefunkcjonalne

Kod	Opis wymagania
WNFU1	Kod prototypu napisany w języku angielskim
WNFU2	Angielska wersja interfejsu graficznego użytkownika
WNFU3	Obsługa najpopularniejszych przeglądarek internetowych: Chrome 86, Safari 14, Firefox 84
WNFU4	Przejrzysty, czytelny graficzny interfejs użytkownika
WNFU5	Dostęp do wszystkich zasobów tylko po zalogowaniu
WNFU6	Uwierzytelnianie przy użyciu loginu i hasła
WNFU7	Potwierdzenie rejestracji użytkownika poprzez link wysłany po wypełnieniu formularza rejestracyjnego
WNFU8	Resetowanie zapomnianego hasła użytkownika poprzez link wysłany na skrzynkę mailową użytkownika.

3.4. Charakterystyka użytkowników

Aplikacja tworzona jest z myślą o użytkownikach zainteresowanych agregacją informacji ze świata. Potencjalnymi odbiorcami są ludzie, którzy pozyskują informacje z wielu źródeł jednocześnie. Prototyp umożliwia im wykonywanie tych codziennych czynności szybciej i efektywniej. Dodatkową korzyścią dla użytkownika są sugerowane artykuły, wyświetlane na podstawie jego preferencji. Warunkiem koniecznym, aby skorzystać ze wszystkich głównych funkcjonalności, jest zalogowanie się. w związku z tym zdecydowano się nie udostępniać żadnych paneli użytkownikom niezalogowanym. Zarówno użytkownicy mający doświadczenie z kanałami RSS jak i początkujący powinni móc efektywnie korzystać z aplikacji. w systemie wyróżniono dwa typy użytkowników.

a) Użytkownik niezalogowany – ma możliwość rejestracji i zalogowania się;

b) Użytkownik zalogowany – ma dostęp do wszystkich funkcji wymienionych w tabeli wymagań funkcjonalnych (tab.1).

3.5. Uproszczony model systemu

Przygotowano uproszczony model przedstawiający wysokopoziomowe spojrzenie na stworzony system. Wyróżniono trzy główne części systemu. Komunikacja pomiędzy wszystkimi wyróżnionymi trzema częściami odbywa się poprzez rozwiązanie architektoniczne oparte o wzorzec REST (ang. *Representional State Transfer*). REST [13] jest zbiorem dobrych praktyk tworzenia architektury oprogramowania. Do komunikacji najczęściej używa się, w jego przypadku, protokół http. Aplikacje pisane w oparciu o ten wzorzec wyróżnia kilka założeń:

- jednorodny interfejs – definiuje interfejs pomiędzy klientem a serwerem. Interfejs upraszcza integrację i architekturę systemów;
- bezstanowość – serwer nie zapamiętuje aktywności użytkownika jego API;
- separacja klient-serwer – klient nie ma wpływu na to, co dzieje się po stronie serwera. Sytuacja wygląda podobnie w drugą stronę. REST API nie może ingerować w to, co się dzieje po stronie użytkownika;
- buforowalność – dane przesyłane do serwera zawierają informacje, czy powinny zostać zapamiętane;
- warstwowość – architektura jest budowana warstwowo. Przykładowo warstwa dostępu do danych nie powinna nic wiedzieć o warstwie logiki biznesowej, a warstwa logiki biznesowej o warstwie prezentacji.

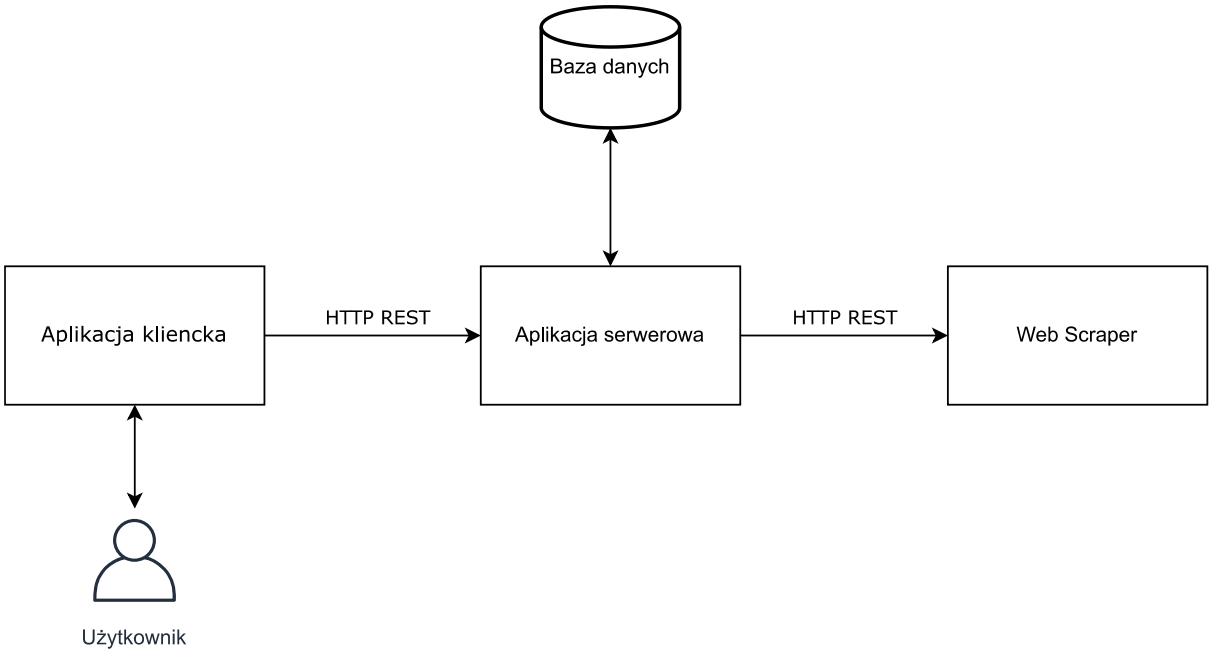
Części systemu:

a) aplikacja kliencka – *frontend* platformy napisany w jednym z nowoczesnych *frameworków*. Podsystem odpowiada za warstwę wizualną systemu;

b) główna aplikacja serwerowa – w tej części zostanie zaimplementowany podsystem przetwarzający wszystkie zapytania od użytkowników i większość logiki biznesowej;

c) aplikacja serwerowa odpowiedzialna za *scrapowanie* zdjęć – mała aplikacja pod względem objętości kodu. Jej zadaniem jest pozyskiwanie zdjęć z wybranych stron internetowych.

Diagram przedstawiony na rysunku 7. pokazuje uproszczony model działania systemu.



Rysunek 7. Model systemu w uproszczeniu

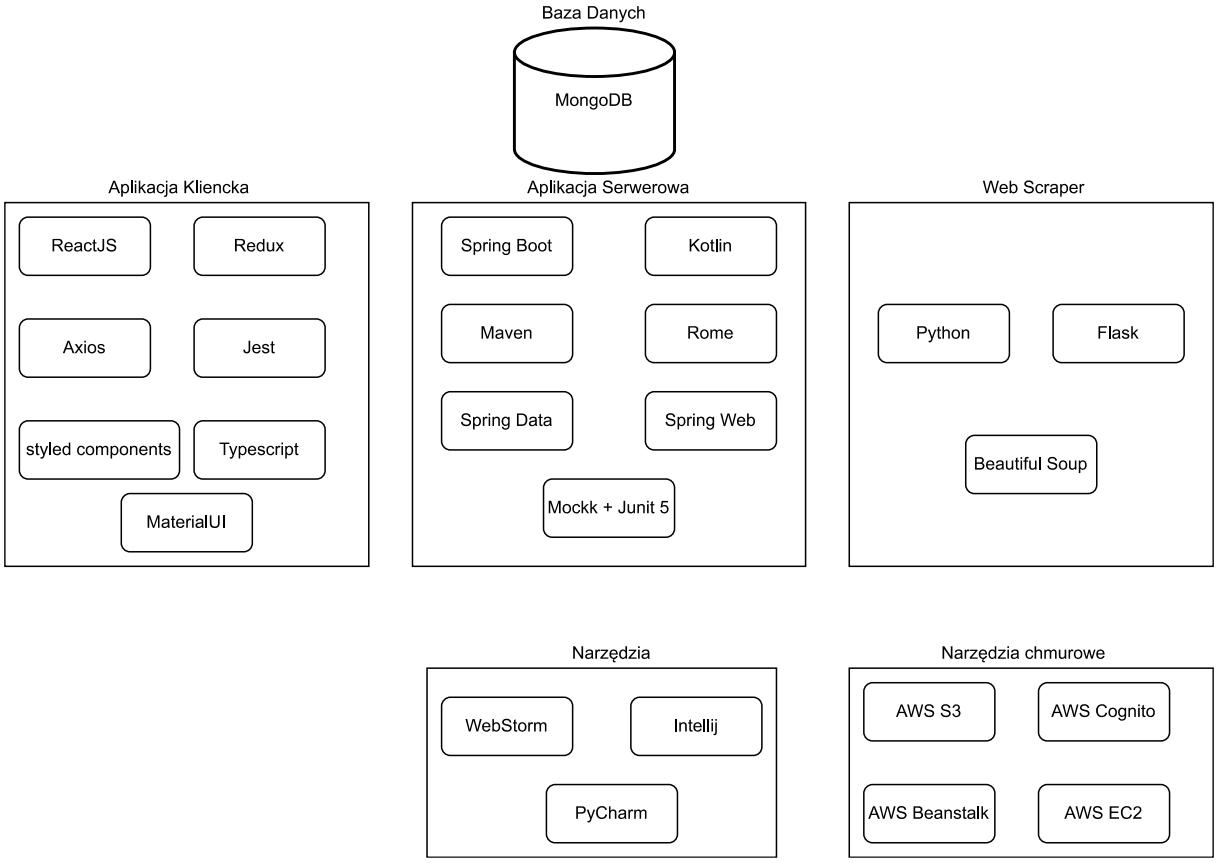
Punktem wejścia do aplikacji jest GUI (ang. *Graphical user interface*), czyli graficzny interfejs użytkownika. Na diagramie przedstawiony jest pod nazwą „Aplikacja kliencka”. Użytkownik wszelkie zmiany w aplikacji dokonuje manualnie poprzez ten interfejs. Aplikacja kliencka komunikuje się z aplikacją serwerową poprzez REST *endpointy*, aby pobierać, zapisywać, modyfikować lub usuwać dane. Aby zmaksymalizować wydajność działania niektóre informacje są przechowywane w bazie danych. Zawiera ona informacje o użytkownikach, artykułach i źródłach. Aplikacja wykonuje zapytanie do *web scrapera*, jeśli nie jest w stanie uzyskać zdjęcia na podstawie informacji zawartych bezpośrednio w kanale RSS. *Web scrape*r jest prostą aplikacją będącą REST API. Jej zadaniem jest wyciąganie źródeł ze stron internetowych w celu uzyskania zdjęcia reprezentującego dany artykuł.

4. Opis narzędzi zastosowanych w pracy

W tym rozdziale zostały omówione narzędzia i technologie wykorzystane do realizacji pracy. Autor wybrał powszechnie używane i nowoczesne rozwiązania. Najważniejsze z nich zostały dokładniej opisane w podrozdziałach 4.1-4.9. Miały one szczególny wpływ na sposób wykonania prototypu.

4.1. Mapa technologii

Aby lepiej zwizualizować projekt stworzono diagram (rys.8) przedstawiający rozłożenie technologii pomiędzy poszczególne części w systemie.



Rysunek 8. Mapa technologii w aplikacji

4.2. Spring Boot

Aby opisać czym jest Spring Boot warto najpierw wspomnieć o Spring Frameworku [14]. Spring jest *frameworkiem* do tworzenia aplikacji webowych. Powstał jako alternatywa dla niegdyś bardzo popularnej Javy Enterprise Edition. Jest rozwiązaniem dużo lżejszym i prostszym w użyciu niż Java EE. Rod Johnson, twórca technologii, zaadoptował rzeczy z Javy EE i maksymalnie je uprościł. Niepotrzebne elementy zostały usunięte. Najważniejszą funkcjonalnością Springa jest bardzo dobrze działający kontener IoC (ang. *Inversion of Control*). Wzorzec ten został zaimplementowany poprzez

wstrzykiwanie zależności (ang. *Dependency Injection*). Polega on na przeniesieniu odpowiedzialności za tworzenie obiektów na zewnątrz. Spring posiada ten kontener i zarządza beanami (obiektami) w aplikacji.

Spring Boot jest rozszerzeniem dla dzieła programistów Pivotala. Został stworzony z myślą o mikroserwisach. Pozwala na uniknięcie dużych ilości dodatkowego kodu zwanego też z angielskiego *boilerplate code*. Dzięki wbudowanemu serwerowi (domyślnie Apache Tomcat) zniknął problem z utrzymaniem zewnętrznego. Wprowadzenie Spring Boota odmieniło świat Javy. Pisanie aplikacji stało jeszcze prostsze i szybsze. Obecnie technologa wspiera, oprócz Javy, też inne języki programowania. Istnieje możliwość tworzenia aplikacji w oparciu o Groovy. Wraz z wejściem narzędzia w wersji 5. zaczął być wspierany także Kotlin.

Aby ułatwić początkową pracę nad projektem, twórcy dostarczyli startery, czyli zestawy bibliotek i konfiguracji. Nazwa każdego pakietu zaczyna się od `spring-boot-starter`. Zależność do startera, która została przykładowo przedstawiona na listingu 1. należy dodać do pliku konfiguracyjnego.

Listing 1. Zależność w pliku pom.xml do startera zawierającego konfiguracje MongoDB

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Do działania aplikacji, oprócz pliku konfiguracyjnego, wystarczy dodać jedynie klasę startującą cały serwer.

Listing 2. Klasa napisana w języku Kotlin, uruchamiająca serwer Springa

```
@SpringBootApplication
class RssReaderApplication

fun main(args: Array<String>) {
    runApplication<RssReaderApplication>(*args)
}
```

Jak widać na listingu 2. klasa startująca serwer wygląda prawie tak jak *main* znany z Javy SE (list. 3).

Listing 3. Klasa main startująca aplikację w Java SE

```
class Main {
    public static void main(String[] args) {
        …
    }
}
```

4.3. Maven

Maven jest najpopularniejszym narzędziem [16] służącym automatyzacji budowania projektów. Użycie narzędzia takiego jak Maven, znacznie przyspiesza cały proces i zdejmuje odpowiedzialność z programistów. Bardzo ważnym atutem jest eliminacja błędów, które mogłyby powstać, w momencie samodzielnego budowania projektu przez programistę. Maven przejmuje całą odpowiedzialność za takie procesy. Aby skorzystać z dobrodziejstw tego narzędzia wymagana jest odpowiednia struktura plików

i konfiguracja *pom.xml*. We wspomnianym pliku xml zdefiniowana jest konfiguracja projektu mavenowego.

Dostępne funkcjonalności narzędzia to m.in.:

- zarządzanie bibliotekami i ich wersjami w tworzonym systemie;
- zautomatyzowane kompilowanie kodu;
- uruchamianie testów aplikacji;
- generowanie pliku wykonywalnego z aplikacją;
- generowanie dokumentacji;
- niezależność od środowiska (projekt powinien tak samo działać np. na systemie Windows i MacOS).

Listing 3. Przykładowa struktura pliku pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.pjatk.pzdral</groupId>
    <artifactId>example</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>…dependencies>

</project>
```

Na listing 3. przedstawiono strukturę pom.xml w jednej z najprostszych form. Przykładowy kod zawiera pola:

- *groupId* – twórca projekt;
- *artifactId* – nazwa aplikacji;
- *version* – wersja aplikacji;
- *properties* – właściwości np. wersje bibliotek;
- *dependencies* – zależności do bibliotek.

4.4. Kotlin

Kotlin [15] to dosyć nowy język programowania stworzony w 2011 roku. Jest zaimplementowany i rozwijany przez programistów JetBrains, w środowisku programistów Javy, znanych jako twórcy bardzo popularnego IDE - Intellij. Język ten jest statycznie typowany i działa na maszynie wirtualnej

Javy. Kotlin został zaprojektowany tak, żeby działać z innym językami uruchamianymi na JVM (ang. *Java Virtual Machine*). Jest w pełni kompatybilny z Javą, gdyż ostatecznie jest do niej kompilowany.

Kotlin wprowadza wiele użytecznych funkcjonalności względem języka firmy Oracle m.in.:

- eliminację możliwości wystąpienia wyjątku, który jest niewygodny dla programistów – *NullPointerException*;

Kod z listingu nr 4 nie skompiluje się.

Listing 4. Próba przypisania *null* do zmiennej nie *nullowej*

```
val n: Int = null
```

Aby zmienna akceptowała *nulle* trzeba oznaczyć ją operatorem (?) (list.5)

Listing 5. Przypisanie *null* do oznaczonej zmiennej

```
val n: Int? = null
```

Oprócz tego są inne możliwości jako operator bezpiecznego wywołania funkcji (?.). Metoda zostanie wywołana, tylko jeśli zmienna nie jest *nullem*. Istnieje także operator (!!), który pozwoli uniknąć konieczności sprawdzania wartości *null*, jeśli ma się całkowitą pewność, że zmienna nigdy nim nie będzie.

- zdefiniowanie funkcji rozszerzającej możliwości klasy, tzw. *extension functions.*

Przykładowa funkcja przedstawiona jest na listingu 6. Funkcja dodaje metodę *swap* do kolekcji *MutableList<Int>*

Listing 6. Dodanie metody swap w kolekcji MutableList<Int>

```
fun MutableList<Int>.swap(val1: Int, val2: Int) {
    val tmpVal = this[val1]
    this[val1] = this[val2]
    this[val2] = tmpVal
}
```

- korutyny (ang. *Couritines*) – funkcjonalność wprowadzona w Kotlinie, która znacznie ułatwia pisanie asynchronicznego kodu, poprzez możliwość pisania go w sposób sekwencyjny;
- *data classes* – klasy używane jako kontenery danych i niezawierające żadnej logiki. Ta struktura danych pozwala uniknąć wiele zbędnego kodu (list.7);

Listing 7. Wygląd przykładowej data class

```
data class FeedSource(val source: String?, val link: String?, val
description: String?)
```

- mocno rozwinięte programowanie funkcyjne, funkcje wyższego rzędu, przeciążanie operatorów, *lazy evaulation.*

Z powodu ścisłego powiązania z Javą migracja na Kotlin jest dosyć prosta. Wielu programistów decyduje się na ten krok ze względu na udogodnienia jakie wprowadził język programistów z JetBrainsa. Dowodem na to jest to, że stał się oficjalnym językiem programowania dla platformy Android.

4.5. MongoDB

MongoDB [17] to nierelacyjna dokumentowa baza danych stworzona przez firmę 10gen. Została napisana w języku C++. w bazie dokumenty są składowane jako pliki JSON (ang. *JavaScript Object Notation*). Zastosowanie tego lekkiego formatu danych pozwala na ułatwione przetwarzanie informacji przez inne aplikacje. Najważniejsze cechy tej bazy danych to:

- zapytania Ad hoc – w przypadku mongo językiem zapytań jest JavaScript;
- agregacja – dostarczony jest *framework* do agregacji;
- indeksowanie – Mongo pozwala na zakładanie indeksów na bazodanowych dokumentach. Działa to podobnie jak w bazach relacyjnych;
- rozproszenie – skalowanie danych poprzez rozproszenie danych w kolekcji;
- replikacja – dzięki zastosowaniu funkcji replikacji, można osiągnąć bardzo wysoką dostępność. w przypadku awarii głównej repliki, repliki zapasowe przejmują odpowiedzialność za operacje I/O;
- obsługa JavaScript przez serwer.

Aktualnie jest to najpopularniejsza nierelacyjna baza danych na rynku [18].

4.6. ReactJS

ReactJS [25] jest biblioteką (przez wielu mylnie nazywaną *frameworkiem*) języka programowania JavaScript. Używany jest do tworzenia interfejsów graficznych, czyli warstwy prezentacyjnej. Został stworzony przez Facebooka. React nazywany jest biblioteką ze względu na to, że nie narzuca żadnych rozwiązań np. przechowywania stanu aplikacji. Po wybraniu tej technologii można bezproblemowo dodać swoje inne ulubione rozwiązania. Zalety jakie niesie ze sobą korzystanie z Reacta to przykładowo:

- łatwiejsze tworzenie interfejsów graficznych;
- komponenty wielokrotnego użytku;
- poprawa wydajności aplikacji;
- bardzo duża społeczność użytkowników.

Listing 8. Prosty komponent stworzony w React.js

```
import React from 'react'

function ExampleComponent() {
    return (
        <div>
            <h1>Simple View</h1>
        </div>
    );
}
```

Na listingu 8. został przedstawiony komponent funkcyjny wyświetlający napis „*Simple View*". Komponent jest funkcją lub klasą wyświetlającą jakąś część aplikacji. Na cały projekt składa się wiele małych komponentów. Tworzenie i ich składanie jest podstawową czynnością w React. w funkcji zwracany jest kod JSX (JavaScript XML). JSX jest nakładką składniową rozszerzającą ECMAScript. Kod przypomina HTML, ale tak naprawdę to nadal JavaScript. Składnia jest przetworzona przez Babel

i ostatecznie w kodzie jest wiele wywołań funkcji `React.createElement()`. Do `createElement()` przekazany jest kod HTML.

React.js jest projektem stale rozwijanym, posiadającym coraz większą rzeszę zwolenników. Nowe, wartościowe funkcjonalności np. React *hooki* ułatwiają pracę programistów.

4.7. TypeScript

TypeScript [20] jest językiem programowania będącym semantycznie nadzbiorem JavaScript. Został stworzony przez Microsoft do tworzenia aplikacji internetowych. Poprawny kod napisany w JS jest również poprawny w TypeScript. Końcowo każda aplikacja napisana w języku Microsoftu jest transpilowana do Javascriptu.

Głównymi powodami, jakie skłaniają wielu programistów do wybrania właśnie TypeScriptu, są typowane zmienne, argumenty i funkcje. Na listingu 9. zaprezentowano przykład kodu, który zawierają deklarowanie typowanych zmiennych.

Listing 9. Przykładowy kod w TypeScript

```
const name: string = "Patryk";
let num1: number = 50;
let num2: number = 42.50;
let sum = score1 + score2;
```

Oprócz typowania dostarczone są także inne użyteczne funkcjonalności znane z języków zorientowanych obiektowo jak Java, m.in.:

- interfejsy;
- klasy abstrakcyjne;
- moduły;
- typy generyczne;
- opcjonalne parametry funkcji;
- wyliczeniowy typ danych *enum*.

4.8. Python

Python [21] to wysokopoziomowy język programowania mający szerokie spektrum zastosowań. Można go użyć do tworzenia na przykład serwisów internetowych, skryptów, aplikacji desktopowych, a nawet gier. Jedną z jego głównych cech jest czytelność i klarowność kodu. Dzięki temu uważany jest za język prosty w nauce nadający się dla początkujących [22]. Wielu ludzi zaczynających przygodę z programowaniem decyduje się rozpocząć ją od Pythona. Cechuje się on bardzo rozbudowaną liczbą darmowych bibliotek na licencji Open Source. Język ten cieszy się wielką popularnością wśród wielu młodych programistów. Aktualnie jest jednym z najpopularniejszych języków na świecie i z pewnością warto się go nauczyć. Społeczność programistów Pythona jest znana ze swojej otwartości i chęci pomocy. w związku z tym w przypadku jakichkolwiek problemów zawsze można liczyć na ich wsparcie [23].

Do rozpoczęcia pracy z Pythonem niepotrzebna jest żadna klasa opakowująca kod.

Listing 10. Przykładowy kod napisany w Pythonie

```
print('Pjatk')

phrase = 'example'
for letter in phrase:
    print(letter)
```

Na listingu 10. przedstawiono przykładowy kod, którego wykonanie wyświetli napis „Pjatk" i przeiteruje po słowie „*example*". w wielu innych językach stworzenie analogicznego kodu mogłoby zająć znacznie więcej linijek kodu.

4.9. Flask

Flask [24] to mikro *framework* służący do pisania aplikacji webowych w Pythonie. Jego określenie „mikro" wynika z tego, że nie wymaga żadnych określonych narzędzi lub bibliotek, aby działać. Każdą dodatkową bibliotekę lub rozszerzenie programista dobiera samodzielnie. Dzięki takiemu podejściu Flask jest uważany za mocno elastyczną technologię i jest to jedna z jego głównych zalet. Warto rozważyć użycie tego rozwiązania, kiedy nie wiadomo jakie biblioteki będą użyte w projekcie. Flask także idealnie nadaje się w przypadku tworzenia małych aplikacji z ograniczoną liczbą funkcjonalności. z tego powodu w pracy zostało użyte to narzędzie do stworzenia *Web Scrapera,* który jest małą zadaniową aplikacją.

Boilerplate kod potrzebny do wystartowania aplikacji w podstawowej wersji jest zniwelowany do minimum. Wystarczy stworzenie obiektu `Flask` i użycie metody `run()`.

Listing 11. Kod startujący serwer Flaska

```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello():
    return "Hello from pjatk"

if __name__ == '__main__':
    app.run()
```

W efekcie wykonania kodu w listingu 11. zostanie wystartowany lokalnie serwer na domyślnym porcie 5000. Po przejściu na adres *http://localhost:5000* zostanie wyświetlony użytkownikowi napis „*Hello from pjatk*".

5. Implementacja prototypu

Rozdział ten zawiera opis rozwiązań implementacyjnych zastosowanych przy realizacji pracy.

5.1. Aplikacja kliencka

Aplikacja kliencka jest napisana w oparciu o bibliotekę React. Zastosowano się do zasad najlepszych wzorców programowania w oparciu o dokumentację techniczną tej technologii [25]. Najważniejszymi częściami każdej aplikacji tego typu są komponenty. Wybierając technologię programistów Facebooka można tworzyć poszczególne komponenty na trzy sposoby:

- klasycznie za pomocą metody `React.createClass`,
- poprzez użycie składni z ES6 `extends React.Component`,
- tworząc komponenty funkcyjne.

Komponenty pobierają dane i renderują je jako HTML w DOM. Wykorzystują właściwości (ang. *properties*) i stany (ang. *states*), które mają wpływ na przekazywane dane. Zarówno właściwości, jak i stany są czystymi obiektami JavaScript.

W projekcie zdecydowano się zastosować najbardziej nowoczesny typ komponentów, czyli komponenty funkcyjne. Wraz z wejściem *hooków* w 2018 roku, bezstanowe komponenty funkcyjne mogą zachowywać się jak klasowe. Kod po ich wdrożeniu jest bardziej czytelny i przejrzysty. *Hooki* są funkcjami, które posiadają akcje odpowiedzialne za cykl życia komponentu. Nie ma już potrzeby tworzyć klas i dziedziczyć po `React.Component`, aby nadpisywać metody odpowiedzialne za stan komponentu. Podejście funkcyjne komponenty wraz z *hookami* jest dużo bardziej nowoczesne i aktualnie polecane przez samych twórców [26].

Listing 12. przedstawia fragment kodu przykładowego komponentu funkcyjnego:

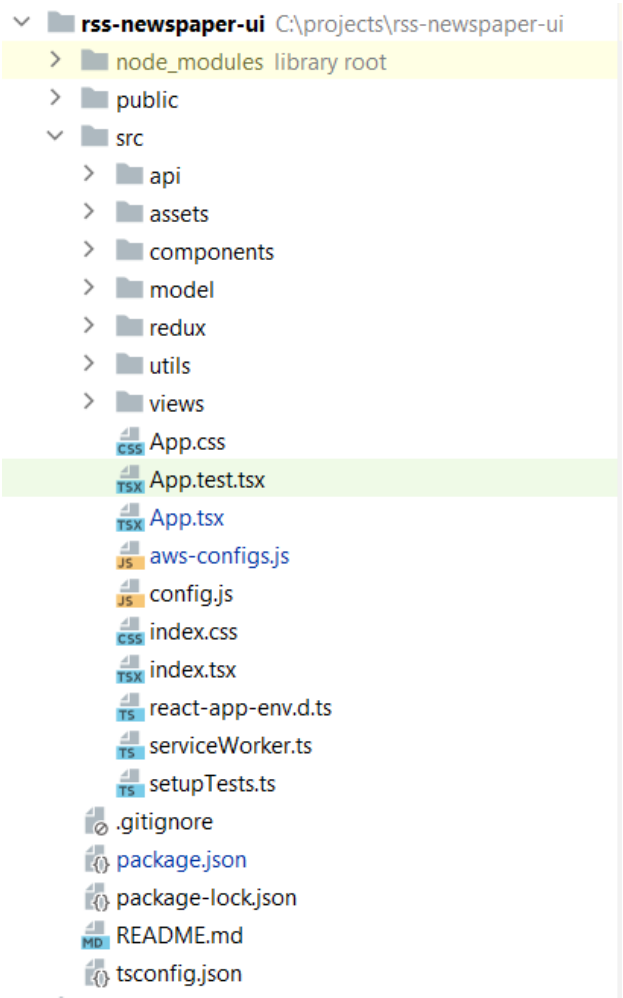
Listing 12. Komponent funkcyjny w React wyświetlający listę artykułów

```
export interface ShowFeedPageProps {
    feedsStore: any;
    location: any
}

export const ShowFeedPage: FunctionComponent<ShowFeedPageProps> = (props:
ShowFeedPageProps) => {
    const [initialized] = useState(false);
    const [data, setData] = useState<RssFeed[]>([]);
    const [url, setUrl] = useState<string>("");

    const getListings = (url: string, category: string) => {
        FetchPageFeedAPI.fetchPageFeed(url, category).then(
            (rssFeeds) => {
                if (Array.isArray(rssFeeds)) {
                    console.log(rssFeeds)
                    setData(rssFeeds);
                    props.feedsStore.setFeeds(rssFeeds);
                }
            }
        ).catch((error) => console.log(error))
    };

    useEffect(() => {
        const url = props.location.state.url;
        setUrl(url);
        const category = props.location.state.category;
        getListings(url, category);
    }, [initialized, props.location.state]);

    return (
        <div>
            <h2 className="center title">
                Feed from {url}
            </h2>
            <br/>
            <div className="grid-container">

                {data.map((item: RssFeed, i: any) => {
                    return (
                        <Article key={i} rssFeed={item}/>
                    );
                })}
            </div>
        </div>


    );
}
```

Zastosowano typową strukturę folderów dla aplikacji pisanej w React [27], widoczną na rysunku 9. Najwyżej w strukturze plików stoi plik *package.json*. Jest to plik konfiguracyjny projektu, który zawiera wszystkie niezbędne informacje, np. wersja, opis, autorzy. Oprócz tego, jego bardzo ważną rolą jest przechowywanie informacji o niezbędnych zależnościach w projekcie. Za pomocą komend dostępnych w *npm* można zainstalować wszystko, co jest niezbędne do działania projektu, a następnie go uruchomić. w prototypie w tym celu używane do tego są komendy `npm install` i `npm start`.

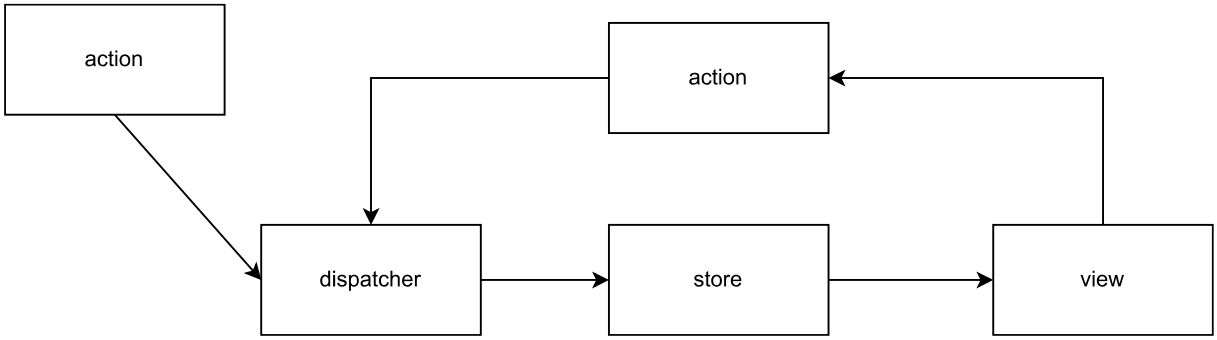
Na rysunku 9. przedstawiono strukturę plików wykorzystaną w aplikacji klienckiej. w folderze *src* znajdują się wszystkie zaimplementowane komponenty w ramach wytwarzania części klienckiej. Plik index.tsx jest puntem wejściowym w systemie. w tym miejscu startuje aplikacja. w podfolderach *src* znajdują się kolejne foldery z plikami:

- ***api*** – wszystkie zaimplementowane interfejsy http;
- ***assets*** – zdjęcia i style;
- ***components*** – komponenty używane w aplikacji;
- ***model*** – obiekty nie zawierające logiki biznesowej a jedynie będące kontenerami na dane;
- ***redux*** – pliki związane z wdrożonym wzorcem Redux;
- ***utils*** – obiekty pomocnicze używane w wielu miejscach;
- ***views*** – komponenty reprezentujące widoki.



Rysunek 9. Struktura plików w aplikacji klienckiej

Do zarządzania stanem został wdrożony wzorzec Redux. Jest to jedna z najpopularniejszych implementacji architektury Flux, czyli architektury polegającej na jednokierunkowym przepływie danych.



Rysunek 10. Diagram działania wzorca Redux

Diagram przedstawiony na rys. 10. rozpoczyna się w klocku *action* z lewej strony. Przepływ danych inicjowany jest akcją. Jest ona, za pomocą *dispatchera*, dostarczona do obiektu *store*, w którym zostaje zapisany nowy stan. *View* to po prostu komponent w aplikacji, który korzysta ze stanu przechowywanego w storze.

Store zaimplementowany w ramach aplikacji przechowuje stan użytkownika. Znajdują się w nim informacje o aktualnie zalogowanej osobie i jej ocenach. Listing 13. zawiera kod tworzący Redux *store*.

Listing 13. Deklaracja Redux Store'a

```
import {createStore} from 'redux';
import {rootReducer} from "./index";
import {composeWithDevTools} from 'redux-devtools-extension';

let store = createStore(rootReducer, composeWithDevTools());

export default store;
```

W ramach zaimplementowanego wzorca dostępnych jest kilka akcji. Przykładowe akcje widoczne są na listingu 14.

Listing 14. Przykładowe reduxowe akcje

```
export const login = (username: string | null, rates: UserRate[]) => {
    return typedAction(LOGIN_SUCCESS, {username, rates});
};
export const addRate = (rate: UserRate) => {
    return typedAction(ADD_RATE, rate);
};
export const logout = () => {
    return typedAction(LOGOUT);
};
```

Reducer odpowiednio reaguje na każdą z akcji. Podjęte operacje zmiany stanu, na podstawie wymienionych akcji w listingu 14. zostały ukazane na listingu 15. Przedstawiono na nim reakcję na zmianę stanu po dodaniu oceny lub zalogowaniu się.

Listing 15. Część kodu reducera użytkownika

```
export function userReducer(
    state = initialState,
    action: UserAction
): UserState {
    switch (action.type) {
        case LOGIN_SUCCESS:
            return {loggedIn: true, username: action.payload.username,
rates: action.payload.rates};
        case ADD_RATE: {
            const val = state.rates.findIndex((rate) => rate.link ===
action.payload.link)
            let newList = [] as UserRate[]
            if(val !== -1) {
                state.rates[val] = action.payload
                newList = state.rates
            } else {
                state.rates.push(action.payload)
                newList = state.rates
            }
            return {...state, rates: newList};
        }
     …
   }
}
```

Do komunikacji z częścią serwerową zastosowano bibliotekę Axios [28]. Jest ona uważana za jedną z najlepszych bibliotek będących implementacją klienta http [29]. w folderze *api* dodano wszystkie interfejsy służące do komunikacji z REST *endpointami*. Wszystkie obiekty korzystająca z Axiosa. Listing 16. przedstawia przykładowy kod prezentujący API:

Listing 16. Kod REST API użytkownika

```
const RATE_ARTICLE_URL = "/rate-article"
const FIND_USER_RATES_URL = "/find-user-rates"


export const UserRateAPI = {

    rateArticle: async function (userRateTo: UserRateTo):
Promise<RssFeed> {
        const response = await axios.post(RATE_ARTICLE_URL, userRateTo);
        return response.data;
    },
    findUserRates: async function (email: String): Promise<UserRate[]> {
        const response = await axios.get(FIND_USER_RATES_URL,{
            params: {
                email: email
            }
        });
        return response.data;
    },
   …
}
```

Każdy widok w aplikacji ma przypisany określony komponent. Natomiast w nim może być zawarte wiele innych komponentów. Fragment kodu z listingu 17. pokazuje widok personalizacji profilu użytkownika.

Listing 17. Kod komponentu funkcyjnego przedstawiającego widok profilu użytkownika

```
export const Profile: FunctionComponent<any> = (props: any) => {
    const classes = useStyles();
    const [userSettings, setUserSettings] = useState<UserSettings>({
        description: "",
        username: "",
        email: null, categories: []
    });
    const {...rest} = props;
    const imageClasses = classNames(
        classes.imgRaised,
        classes.imgRoundedCircle,
        classes.imgFluid
    );
    const state = store.getState();

    useEffect(() => {
        UserSettingsAPI.getUserPreferredCategories(state.username)
            .then((settings) => {
                    setUserSettings(settings)
                }
            ).catch((error) => console.log(error))
    }, []);


    function saveSettings(settings: string[]) {
        const usrSettings: UserSettings = {
            email: state.username,
            categories: settings,
            username: userSettings.username,
            description: userSettings.description
        };
        UserSettingsAPI.saveUserPreferredCategories(usrSettings)
            .then(() => {
                toast('Settings saved successfully', {
                    position: "bottom-center",

                })
            }).catch((error: any) => console.log(error));
    }

    const [age, setAge] = React.useState('Musician');

    const handleChange = (event: React.ChangeEvent<{ value: unknown }>)
=> {
        setAge(event.target.value as string);
    };

    const handleUsernameChange = (event: React.ChangeEvent<{ value:
unknown }>) => {
        setUserSettings({...userSettings, username: event.target.value as
string})
    };

    const handleDescriptionChange = (event: React.ChangeEvent<{ value:
unknown }>) => {
        setUserSettings({...userSettings, description: event.target.value
as string})
```

```
    };

    return (
        <div>
        …
        </div>
    )
};
```

Zgodnie z konwencją, react *hooki* zaczynają się od słowa „*use*”. `UseStyles` jest używany do wstrzykiwania stylów CSS. Do przechowywania stanu w komponencie używa się *hooka* `useState`. Domyślnie *hook* `useEffect` działa tak samo, jak znane z komponentów klasowych `componentDidUpdate` i `componentDidMount`. Wywoływany jest w momencie wyrenderowania komponentu. w tym hooku możliwa jest także akcja w momencie odmontowania (w klasowym akcja *`componentWillUnmount`*). Należy dodać wtedy w `useEffect` akcję w *return*. w przykładowym listingu 17. jest wywoływana akcja tylko na początku życia komponentu. z serwera pobierane są ulubione kategorie użytkownika, a następnie zapisywane w stanie `userSettings`. `w` sekcji return w znajduje się zawsze kod JSX odwzorowujący kod HTML. Kod przypomina HTML, ale faktycznie jest to JavaScript, który jest lukrem składniowym dla funkcji `React.createElement(component, props, ...children)`. Orientacyjny kod z listingu 18. jest kompilowany do innego ukazanego na listingu 19.

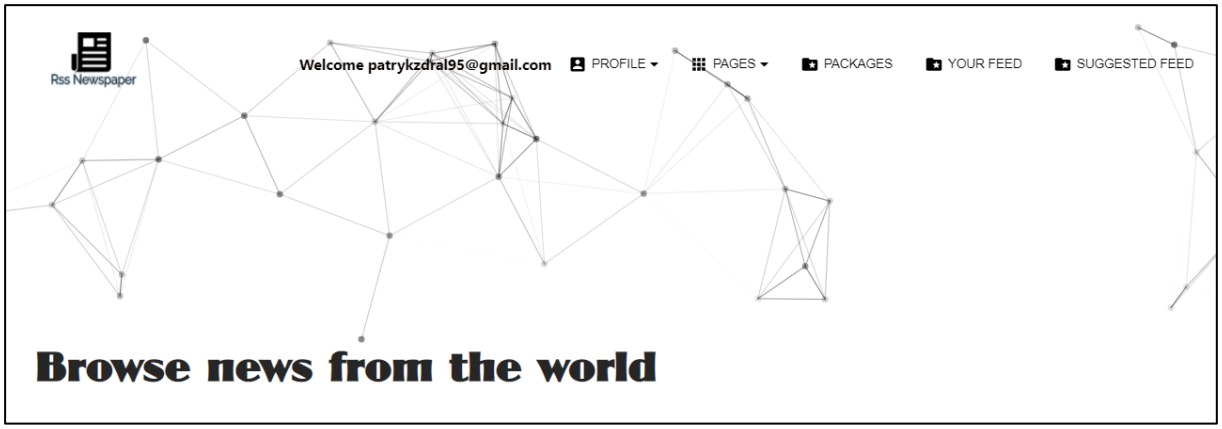
Listing 18. Kod napisany w JSX

```
<RegularButton color="info">Save settings</RegularButton>
```

Listing 19. Kod JavaScript tworzący element JSX

```
React.createElement( RegularButton, {color: info}, 'Save settings' )
```

Prototyp został stworzony w języku angielskim tak, jak to zostało ustalone w wymaganiach niefunkcjonalnych. Do poruszania się po aplikacji używany jest pasek nawigacji (rys.11).



Rysunek 11. Pasek nawigacji

Dostępne widoki to:

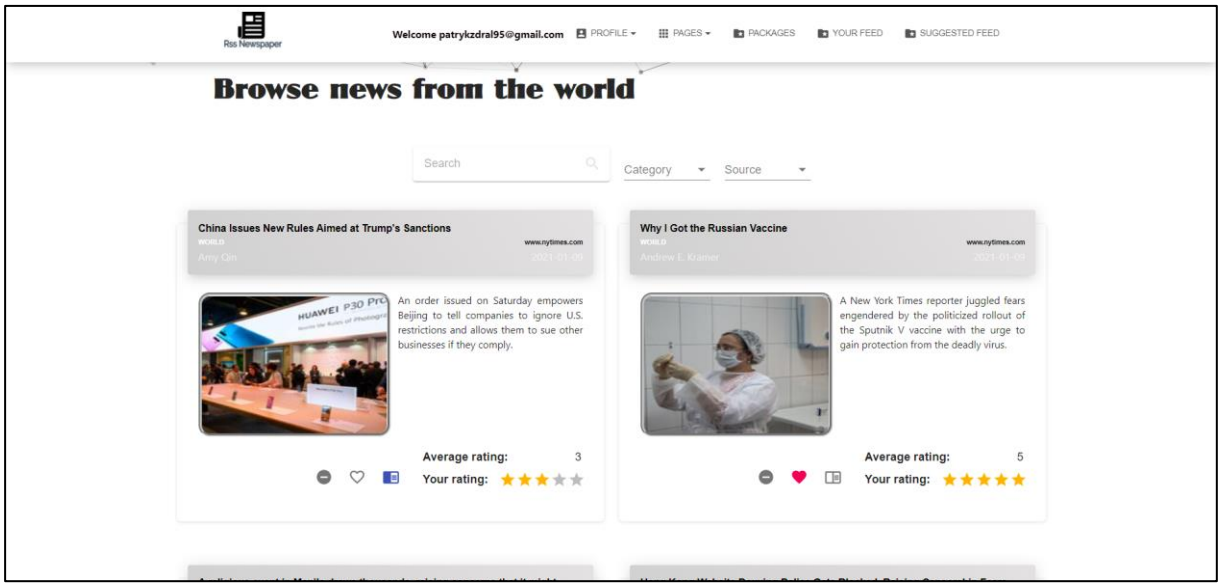
- strona główna – wyświetlana użytkownikowi niezalogowanemu;
- *profile* – profil użytkownika, ustawienie danych osobowych i personalizacja paczek kategorii;
- *pages* – zawiera 3 podstrony

    - *main* – główna strona, zbiór przypadkowych artykułów. Dostępna jest możliwość ich wyszukiwania, sortowania;
    - *favourite pages* – ulubione strony użytkownika;
    - *to read pages* – strony zapisane do przeczytania na później.
- *packages* – wyświetlone artykuły na podstawie wybranych paczek (kategorii) użytkownika;
- *your feed* – strona z kanałami RSS dodanymi przez użytkownika. Istnieje możliwość ich dodawania i usuwania;
- *suggested feed* – sugerowane artykuły.

Użytkownikowi niezalogowanemu zostaje wyświetlony ekran powitalny (rys. 12). z tego poziomu ma możliwość rejestracji i zalogowania się.

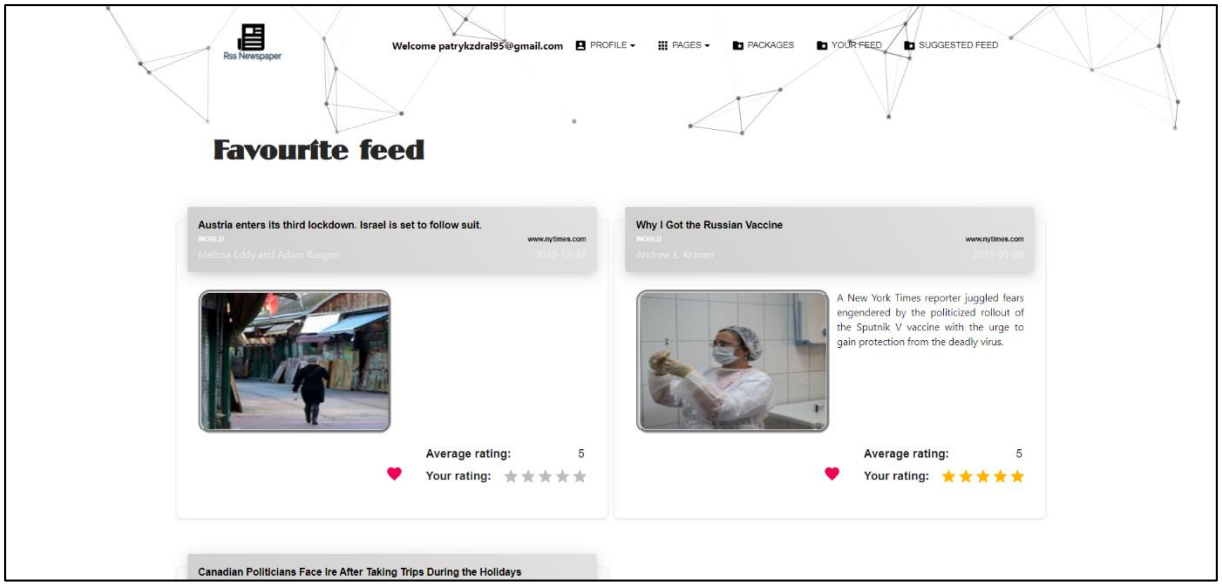
Rysunek 12. Widok użytkownika niezalogowanego

Po wejściu do aplikacji każdej zalogowanej osobie pokazany jest ekran główny (rys. 13). Strona odpowiada widokowi *Main* w zakładce *Pages*. Do każdego artykułu użytkownik może przejść po kliknięciu w tytuł. Bezpośrednio na stronie czytnika ma możliwość oceny artykułu, dodania do ulubionych, przeczytania i ukrycia.



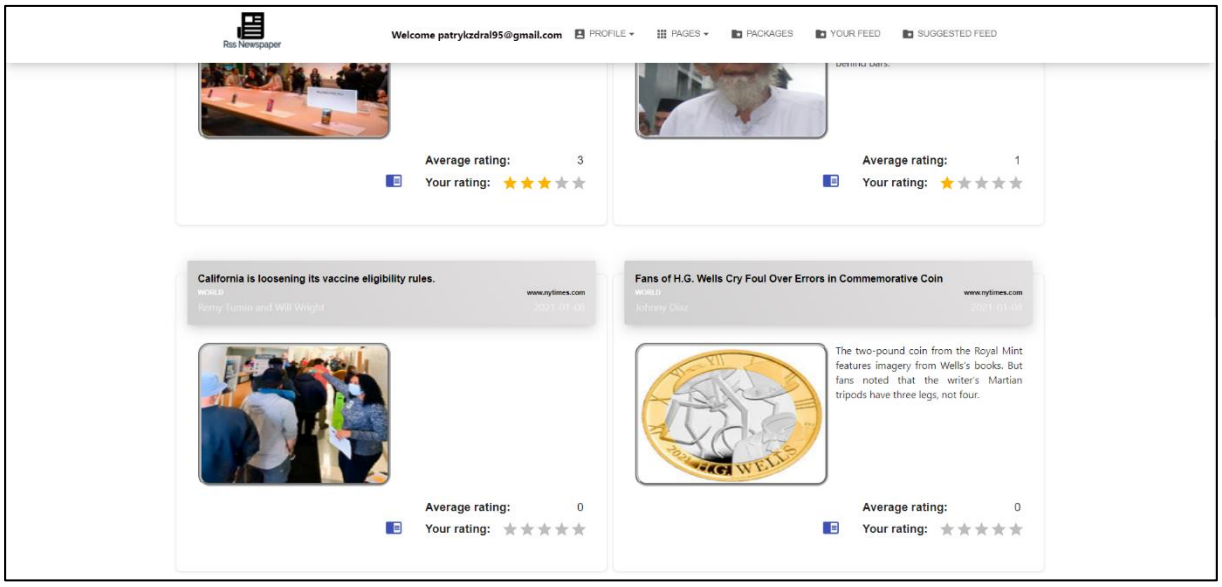
Rysunek 13. Widok strony głównej

Aby dodać artykuł do polubionych należy nacisnąć ikonę w kształcie serca. Strona z ulubionymi artykułami pokazana jest na rysunku 14. Po odkliknięciu znaku serca przynależącego do konkretnego artykułu zostanie on usunięty z listy ulubionych.



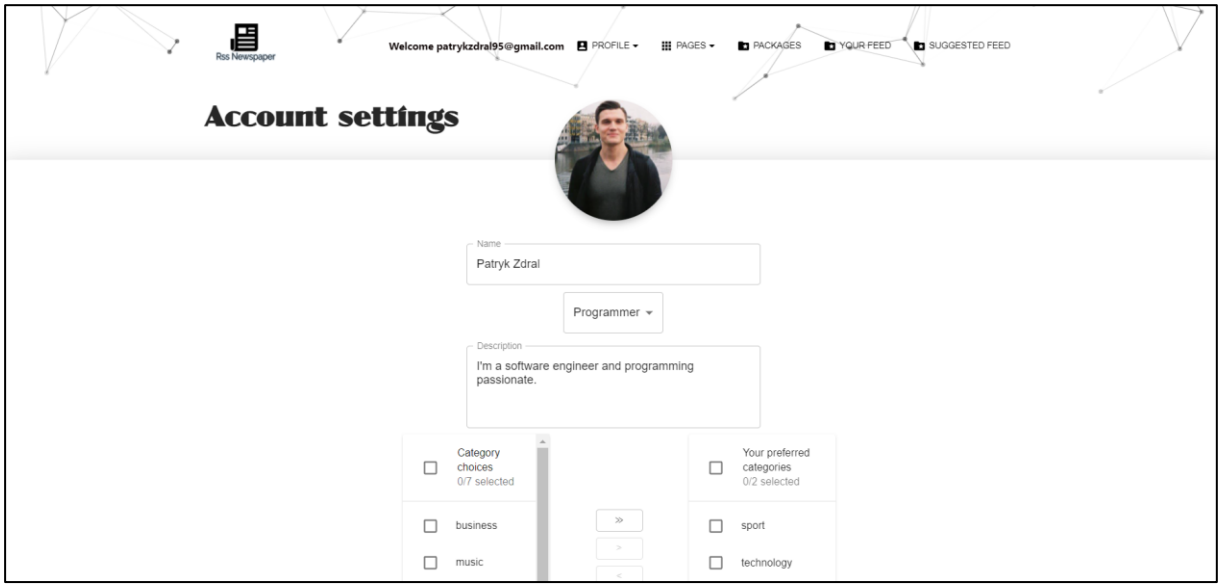
Rysunek 14. Widok ulubionych artykułów RSS

Aby odłożyć artykuł do przeczytania później należy nacisnąć ikonę książki. Strona z takimi artykułami jest pokazana na rysunku 15. Po odkliknięciu znaku książki przynależącego do konkretnego artykułu zostanie on usunięty z listy artykułów do przeczytania

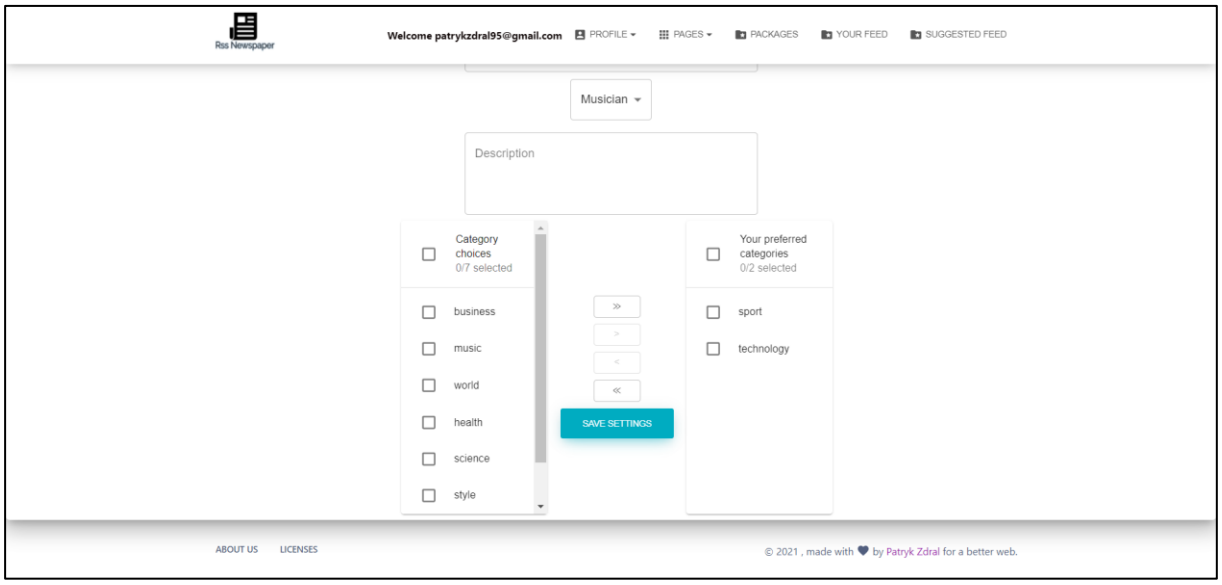


Rysunek 15. Widok artykułów RSS do przeczytania

Każdy użytkownik może personalizować swoje ustawienia. Standardowe pola jak nazwa i opis, nie mają wpływu na działanie aplikacji. Praca użytkownika w chwili ukończenia prototypu także, ale pomysł rozwojowy tego pola został omówiony w rozdziale 6. Ważnym elementem jest ustawienie ulubionych kategorii. Na ich podstawie, algorytmy mogą dostosowywać treści i w widoku „*PACKAGES*" pojawią się tylko paczki kategoryczne interesujące użytkownika. Widok użytkownika został przedstawiony na rysunku 16. i 17.

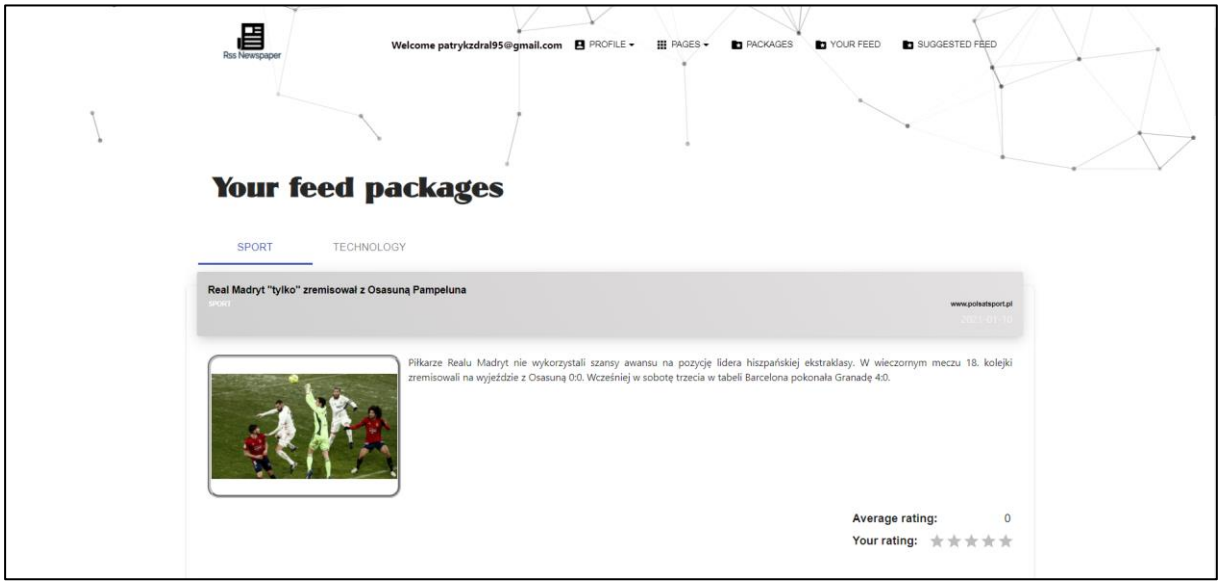


Rysunek 16. Widok profilu użytkownika cz.1/2



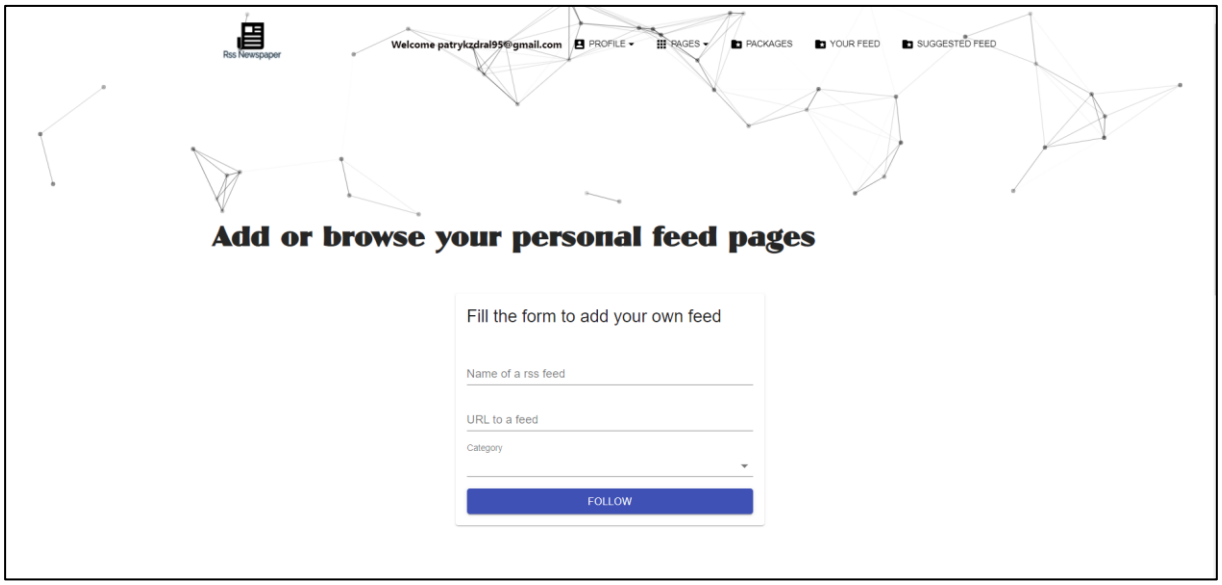
Rysunek 17. Widok profilu użytkownika cz. 2/2

W widoku *Packages* (rysunek 18.) znajdują się wszystkie paczki z kategoriami wybrane przez użytkownika. Na zdjęciu jest przedstawiony wygląd po wybraniu paczki sport i *technology*.

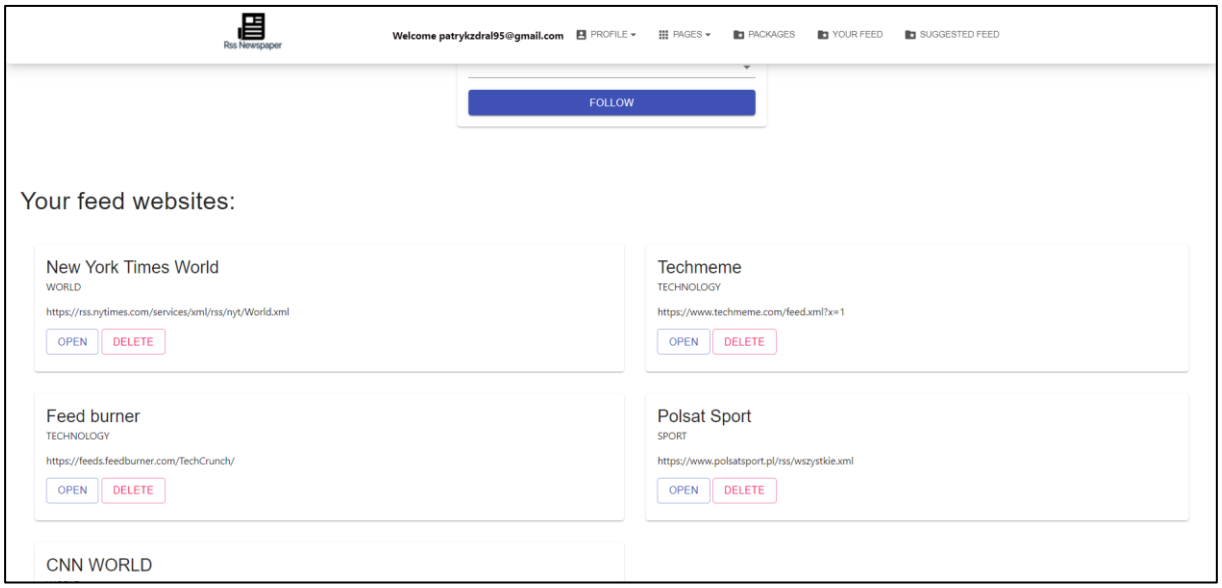


Rysunek 18. Widok paczek użytkownika

Niewątpliwie jednym z najważniejszych widoków jest *Your feed* (rysunek 19. i 20.). Na rysunku 18. widoczny jest formularz służący do dodawania swojego kanału RSS. Formularz zawiera nazwę kanału, adres i wybraną przez użytkownika kategorię. Na rysunku pokazano listę wszystkich dodanych kanałów użytkownika *patrykzdral95@gmail.com*. Można wejść na wybrany kanał po kliknięciu przycisku „*Open*". Zostanie wyświetlona aktualna lista artykułów udostępniona w tym strumieniu. Można także niechciany kanał usunąć.



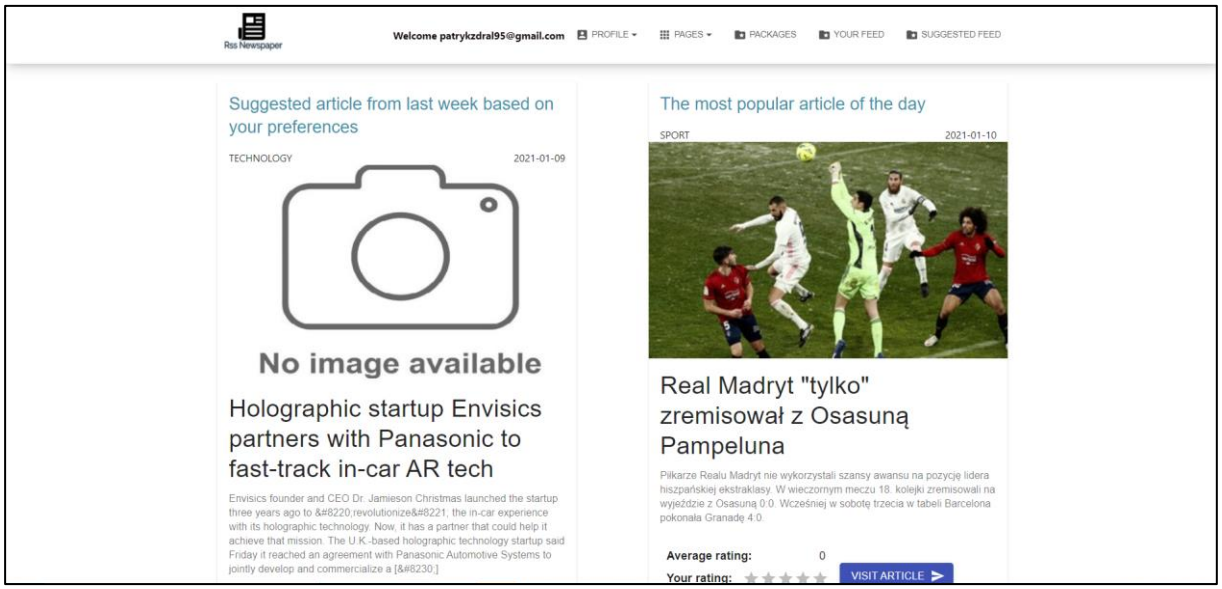
Rysunek 19. Widok "Twój feed" cz. 1/2



Rysunek 20. Widok "Twój feed" cz. 2/2

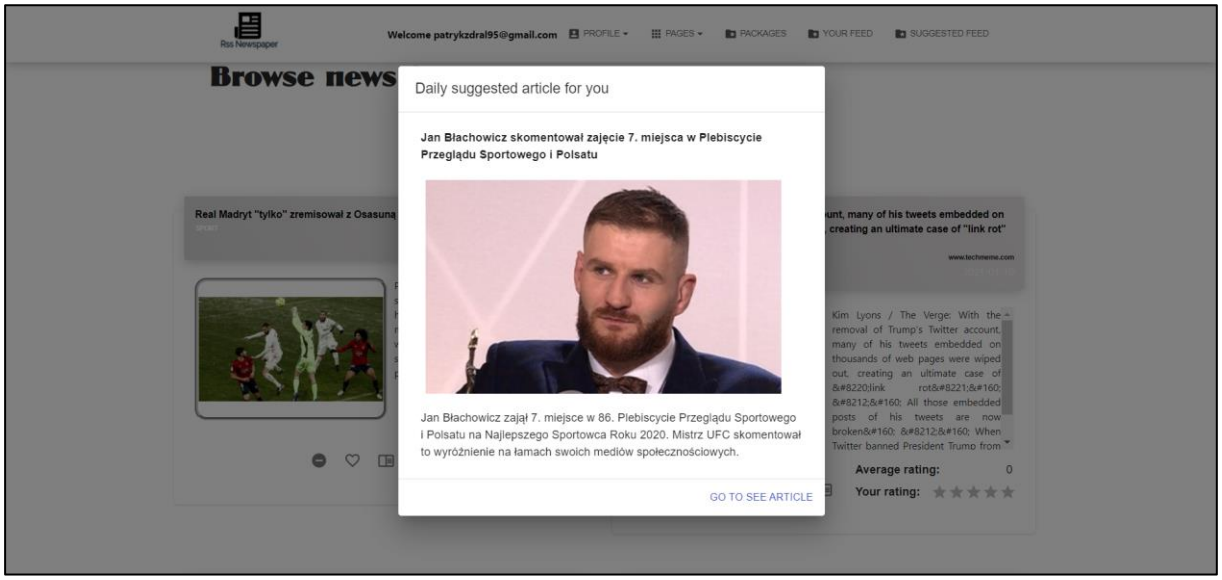
Rysunek 21. przedstawia ekran z sugerowanymi artykułami:

- sugerowany artykuł na podstawie zaimplementowanego algorytmu *Collaborating filteringu* omówionego w rozdziale 5.2.1.;
- najpopularniejszy artykuł dnia;
- najpopularniejszy artykuł tygodnia;
- najpopularniejszy artykuł miesiąca.



Rysunek 21. Widok sugerowanych artykułów

Użytkownikowi po określonym czasie przebywania na jednej stronie może pojawić się *pop-up* zawierający sugerowany artykuł (rys. 22).



Rysunek 22. Pop-up zawierający sugerowany artykuł

5.2. Aplikacja serwerowa

Aplikacja serwerowa została napisana w oparciu o język programowania Kotlin i *framework* Spring Boot. w tej części systemu zawarta jest większość logiki biznesowej. Jedynym punktem wejścia do aplikacji są zabezpieczone *REST endpointy*.

Listing 20. Rest Controller do obsługi zapytań dotyczących ocen użytkownika

```
@RestController
class UserRateArticleController(private val rateArticleUseCase:
RateArticleUseCase,
                                private val findUserRatesUseCase:
FindUserRatesUseCase) {


    @PostMapping("/rate-article")
    fun rateArticle(@RequestBody userRateTo: UserRateTo) {
        rateArticleUseCase.rateArticle(userRateTo)
    }

    @GetMapping("find-user-rates")
    fun findUserRates(@RequestParam("email") email: String):
List<UserRate> {
        return findUserRatesUseCase.findUserRates(email)
    }
}
```

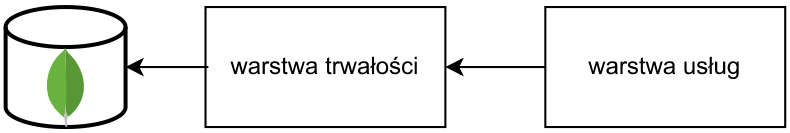
Każdy klasa odpowiedzialna za obsługiwanie zapytań HTTP oznaczona jest adnotacją `@RestController`. Jeśli funkcja służy do obsługiwania zapytań typu POST, to oznaczona jest adnotacją `@PostMapping`. Analogicznie sytuacja wygląda dla GET – `@GetMapping` itd. Dobre praktyki programistyczne mówią, że REST kontrolery nie powinny zawierać żadnej logiki biznesowej. Logika biznesowa powinna być tworzona w warstwie usług. Dlatego widoczne są tylko wywołania serwisów w listingu 20. w funkcjach obsługujących zapytania HTTP np. `rateArticleUseCase.rateArticle(userRateTo)`.

Wszystkie klasy zawierające logikę biznesową mają nazwę zakończoną wyrazem *UseCase* np. `RateArticleUseCase`, `FindUserRatesUseCase`. Serwisy będące przypadkami użycia są oznaczone adnotacją `@Service` lub tworzone w specjalnych klasach konfiguracyjnych oznaczonych `@Configuration`. *Beany* są tworzone poprzez adnotację `@Bean`. Końcowo `@Service` i `@Bean` tworzą ten sam typ *beana* i są jedynie informacją dla programistów. Te adnotacje mówią springowi, że dana klasa ma być zarejestrowana jako bean w kontenerze. Będzie ją można później wstrzyknąć w innej klasie dowolnym sposobem. We wszystkich klasach zastosowano wstrzykiwanie przez konstruktor. Jest to aktualnie preferowany sposób wstrzykiwania przez samych twórców Springa [30]. Zastosowanie tego podejścia posiada kilka zalet. Klasy są łatwiejsze w testowaniu. w klasie serwisowej nie ma kodu związanego ze Springiem jak np. `@Autowired`, więc możemy potem w testach te klasy traktować jak czysto javowe. Kod jest tym samym mniej zależny od Springa a do tego należy dążyć. Wiele źródeł uznaje to jako jedną z podstawowych dobrych praktyk programowania stosując narzędzie Pivotala [31]. Jest to bardzo użyteczne także, jeśli kiedykolwiek zapadnie decyzja o wymianie *frameworku* na inny np. na ostatnio popularnego Quarkusa [32].

Listing 21. Serwis pobierający feed z bazy danych

```
class ReadRandomNotHiddenFeedUseCase(private val rssFeedRepository:
RssFeedRepository) {


    fun readRandomNotHiddenFeed(links: List<String>, page: Int):
FeedResponse {
        val pageableReq: Pageable = PageRequest.of(page, 10,
Sort.by("pubDate").descending())
        val cont = rssFeedRepository.findByNameNotEqual(links,
pageableReq);
        val feedList = cont
                .map {
                    RssFeed(
                            it.title,
                            it.description,
                            it.link,
                            it.author,
                            it.pubDate,
                            it.category.category,
                            FeedSource(it.feedSourceDocument.source,
it.feedSourceDocument.link, it.feedSourceDocument.description),
                            RssImage(it.rssImageDocument?.src,
it.rssImageDocument?.title),
                            Rate(it.rateDocument.averageRate))
                }.toList()
        return FeedResponse(cont.totalPages, feedList)
    }
}
```

W przykładowym listingu kodu nr 21 jest przedstawiona klasa będąca serwisem. w tym przypadku jest to prosty kod klasy do pobierania przypadkowych artykułów z bazy danych. Pobierane jest 10 artykułów posortowanych po dacie dodania. Dodatkowym ograniczeniem dla funkcji jest lista artykułów, które użytkownik ukrył i nie chce zobaczyć. Lista tych artykułów jest przekazana do klasy `RssFeedRepository` będącej pośrednikiem pomiędzy bazą danych, a aplikacją. Jest to także znaczący element architektury. w klasie stworzonej w warstwie usług nie ma bezpośredniego połączenia z bazą danych. Za to odpowiada warstwa trwałości, gdzie znajdują się klasy będące repozytoriami. Są one oznaczone adnotacjami `@Repository`. Do warstwy trwałości należy także klasa `MongoTemplate`. Jest to obiekt pomocniczy służący do odpytywania MongoDB. Klasa jest dostarczona przez bibliotekę Spring Data MongoDB. Na załączonym rysunku (rys. 23.) widnieje podział warstw przedstawiony w formie diagramu.



Rysunek 23. Podział warstw w aplikacji Spring Boot

Przepływ danych zawsze odbywa się w jedną stronę. Baza danych nic wie o będącej wyżej warstwie trwałości, a ona z kolei nic nie wie o części usług. Dzięki takim prostym założeniom jest utrzymany porządek w systemie

Do translacji informacji z kanałów RSS na kod Kotlina użyto bibliotekę Rome [23]. Jest to narzędzie będące API pomiędzy strumieniami RSS, a kodem Javy/Kotlina. Początkowo użyto innego rozwiązania - bibliotekę Rssreader [24]. Okazała się być jednak niewystarczająca. Dzieło programistów Apptastic nie dostarczało kilku ważnych informacji jak np. źródło zdjęcia.

Listing 22. Serwis korzystający z biblioteki Rome do pobierania artykułów RSS

```
class RomeReadFeedUseCase(
    private val rssFeedRepository: RssFeedRepository,
    private val feedSourceRepository: FeedSourceRepository
) {
    fun readFeedUseCase(url: String, category: Category): List<RssFeed> {
        val syndFeed: SyndFeed =
SyndFeedInput().build(XmlReader(URL(url)))
        val feedList = syndFeed.entries
            .stream()
            .map {
                val links = rssFeedRepository.findByLink(it.link)
                val hostName = URL(it.link).host
                val feedSource =
feedSourceRepository.findBySource(hostName) ?:
FeedSourceDocument(hostName, null, null)
                val rssFeedDocument = if (links.isEmpty()) {
                    val src = it.foreignMarkup

                    …
                    rssFeedRepository.save(rssFeedDocument)
                    rssFeedDocument
                } else {
                    links.stream().findFirst().get()
                }
                RssFeed(
                    it.title,
                    it.description?.value.orEmpty(),
                    it.link,
                    it.author,
                    date,
                    it.categories.firstOrNull()?.name.orEmpty(),
                    FeedSource(feedSource.source, feedSource.link,
feedSource.description),
                    RssImage(rssFeedDocument.rssImageDocument?.src,
rssFeedDocument.rssImageDocument?.title),
                    Rate(rssFeedDocument.rateDocument.averageRate)
                )
            }.toList()
        return feedList
    }
}
```

Kawałek kodu z listingu 22. przedstawia jak w prosty sposób można użyć biblioteki Rome, aby wydobyć treści z kanałów RSS. Komenda `SyndFeedInput().build(XmlReader(URL(url)))` zwraca obiekt w, którym jest lista obiektów odwzorowująca artykuły - `syndFeed.entries`. Aby aplikacja działała bardziej efektywnie, wszystkie nieznane artykuły zostają zapisane w bazie danych. w przedstawionym wyżej fragmencie kodu widać, że jeśli dany artykuł nie został odnaleziony w bazie danych przez `RssFeedRepository`, wtedy zostaje zapisany w Mongo. Funkcja końcowo zwraca listę obiektów DTO (ang. *data transfer object*) - `RssFeed`. Obiekty DTO są używane do transferu danych pomiędzy systemami. Obiekt w tej postaci zostaje zwrócony do aplikacji klienckiej. Przykładowy wygląd takiego obiektu przedstawiony jest na listingu 23.

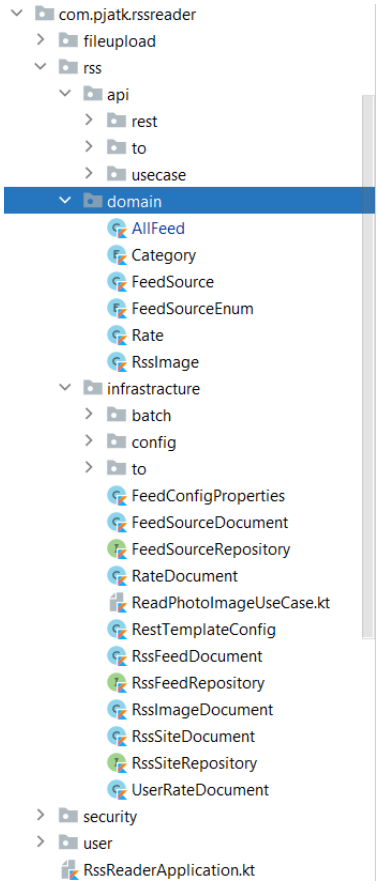
Listing 23. Klasa reprezentująca artykuł RSS

```
data class RssFeed(val title: String?, val description: String?, val
link: String?, val author: String?, val pubDate: LocalDate?, val
category: String?, val feedSource: FeedSource, val rssImage: RssImage?,
val rate: Rate, val clicksNumber: Int = 0)
```

Został użyty typ danych wdrożony w Kotlinie - *data class*. Ta struktura została opisana w rozdziale 4.4.

Na rysunku 24. pokazuje strukturę pakietów w części serwerowej. Ostatnio popularny jest sposób modelowania architektury, gdzie pakiety są dzielone na funkcjonalności lub obiekty domenowe, a nie na warstwy. Powoli odchodzi się od budowania architektury typowo w sposób warstwowy. z tego powodu w prototypie pakiety zostały podzielone na funkcjonalności dotyczące użytkownika, *rss*, *security* i udostępniania plików. w każdym pakiecie wydzielono trzy różne podpakiety:

- *api* – klasy związane z interfejsem aplikacji, mogą to być np. obiekty będące Rest kontrolerami;
- *domain* – obiekty z obszaru biznesowego, które reprezentują coś znaczącego w kontekście danej domeny;
- *infrastracture* – warstwa infrastruktury opisuje sposób, w jaki dane początkowo przechowywane w pamięci są utrwalane w bazach danych lub innym trwałym magazynie. w kontekście tworzonej aplikacji będą to klasy związane z bazą danych MongoDB, a także ściśle związane ze Springiem.



Rysunek 24. Podział pakietów w aplikacji serwerowej

5.2.1 Zbieranie źródeł kanałów RSS

Na etapie implementacji niezbędne było podjęcie decyzji o metodzie zbierania kanałów RSS i zasilania nimi bazy danych. Zdecydowano, że to osoby korzystające z aplikacji będą jednocześnie jednostkami budującymi bazę kanałów RSS. w podrozdziale 5.1.1 zostały opisane widoki i jednym z nich był widok „*Your feed*". Osoba w momencie dodania artykułu do swojego prywatnego zbioru kanałów jednocześnie rozszerza bazę publikacji i ich źródeł. Podczas dodawania kanału wypełnia się trzy pola. Pierwsze to nazwa kanału. Nie ma żadnego wpływu na algorytm. Jest wyłącznie informacją dla konsumenta dodającego konkretny *feed*. Drugim polem jest adres kanału, który jest zapisywany w bazie danych źródeł. Trzecim wymaganym elementem jest kategoria. Ostateczna kategoria skojarzana z kanałem RSS w bazie danych jest definiowana na podstawie najczęściej nadawanej kategorii przez użytkowników. Weźmy pod uwagę przykład przedstawiony w tabeli 4.

Tabela 3. Wybór kategorii kanałów RSS dokonany przez użytkowników

Nazwa użytkownika	Dodany Adres kanału RSS	Nadana kategoria kanałowi RSS
User a	https://rss.nytimes.com/services/xml/rss/nyt/Technology.xml	TECHNOLOGY
User B	https://rss.nytimes.com/services/xml/rss/nyt/Technology.xml	TECHNOLOGY
User C	https://rss.nytimes.com/services/xml/rss/nyt/Technology.xml	TECHNOLOGY
User D	https://rss.nytimes.com/services/xml/rss/nyt/Technology.xml	WORLD

Jako że większość użytkowników wybrała kategorię *TECHNOLOGY*, więc taki gatunek zostanie przypisany kanałowi w bazie danych MongoDB.

Listing 24. Wygląd zapisanego dokumentu rssSiteDocument w MongoDB

```
{
  "_id": "https://rss.nytimes.com/services/xml/rss/nyt/Technology .xml",
  "category": "TECHNOLOGY",
  "suggestedCategories": [
    "WORLD", "TECHNOLOGY", "TECHNOLOGY", "TECHNOLOGY"
  ],
  "_class": "com.pjatk.rssreader.rss.infrastracture.RssSiteDocument"
}
```

Przyjęte rozwiązanie pozwoliło uniknąć ręcznego gromadzenia kanałów w aplikacji co byłoby dosyć uciążliwe i czasochłonne. Potrzebna jest tylko jakaś inicjalna baza informacji, aby cokolwiek mogłoby być wyświetlone pierwszym użytkownikom.

Na podstawie zebranych źródeł, takich jak przedstawione na listingu 24., aktualizowana jest baza danych artykułów (dokumentów *rssFeedDocument*). Co godzinę w osobnym wątku (dzięki adnotacji `@Async` i skonfigurowanemu *`TaskExecutorowi`*) uruchamiany jest program odświeżający. Program przechodzi po wszystkich znanych stronach RSS i zapisuje w bazie danych nowe artykuły dostępne w danym kanale RSS.

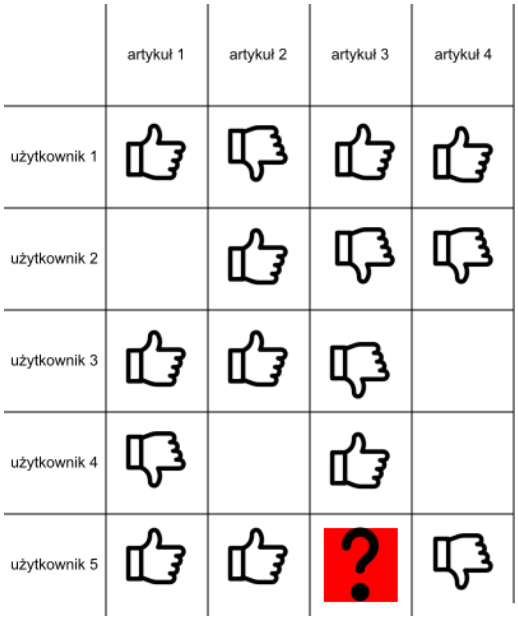
Listing 25. Kod klasy odświeżającej artykuły z kanałów dodanych przez użytkowników

```
open class RefreshExternalFeedBatch(val rssSiteRepository:
RssSiteRepository,
                                    val romeReadFeedUseCase:
RomeReadFeedUseCase) : RefreshFeedBatch {

    companion object {
        @Suppress("JAVA_CLASS_ON_COMPANION")
        @JvmStatic
        private val logger =
LoggerFactory.getLogger(javaClass.enclosingClass)
    }

    @Scheduled(cron = "0 0 */1 * * *")
    @Async
    override fun refreshFeed() {
        logger.info("Started external feed refresh batch")
        val rssSites = rssSiteRepository.findAll()
        rssSites.stream()
                .forEach {
                    romeReadFeedUseCase.readFeedUseCase(it.link,
it.category)
                }
        logger.info("Finished external feed refresh batch")
    }
}
```

Używany serwis `RomeReadFeedUseCase` został omówiony w rozdziale 5.2.

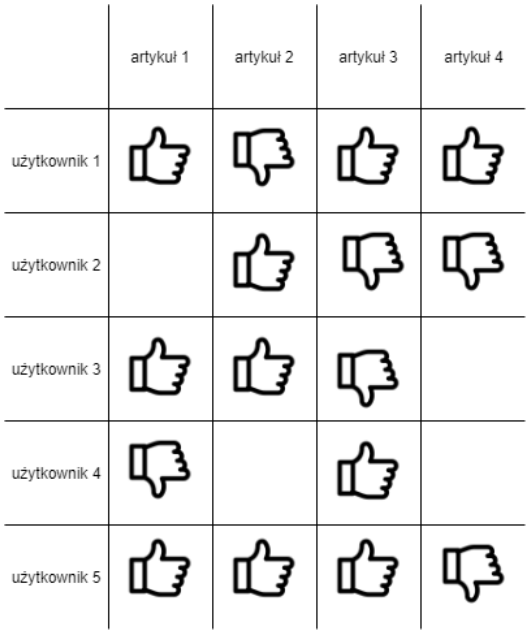
5.2.2 Algorytm Collaborating Filtering

Nierozłącznym elementem aplikacji będących czytnikami RSS są algorytmy rekomendacji. *Collaborative filtering* to najczęściej stosowana technika do tworzenia inteligentnych systemów rekomendacji. Algorytmy z tej kategorii polegają na analizowaniu podobieństw pomiędzy użytkownikami i przedmiotami, które lubią lub nie lubią. Na podstawie takiej analizy możliwe jest przedstawienie rekomendacji. Modele filtrowania mogą polecić element użytkownikowi a na podstawie zainteresowań użytkownika B. Im więcej informacji o użytkownikach jest dostępne tym lepsze rekomendacje mogą podawać. Elementami rekomendacji mogą być różne rzeczy np. filmy, artykuły spożywcze. w przypadku tworzonego systemu łączącymi użytkowników elementami są artykuły wydobywane ze strumieni RSS.



Rysunek 25. Oceny użytkowników przed zastosowaniem algorytmu Collaborating Filtering

Na rysunku 25. chcemy przewidzieć, czy artykuł nr 3 może spodobać się użytkownikowi nr 5. Na podstawie algorytmu *collaborative filteringu* można wywnioskować, że ten artykuł może się mu spodobać [25].



Rysunek 26. Oceny użytkowników po zastosowaniu algorytmu Collaborating Filtering

Zilustrowane na rysunku 25. i 26. wyniki można uzyskać na podstawie konkretnej implementacji systemu rekomendacji. Przedstawiono wyniki uzyskane na podstawie algorytmu *Slope One*. Został stworzony w 2005 roku przez Daniela Lemire i Anne Maclachlan. Jest jedną z najprostszych form

implementacji algorytmu *Collaborative filteringu* w oparciu o oceny. Jego głównym założeniem jest stworzenie liniowej zależności między pozycjami, takiej jak relacja F(x) = a + b.

Jego prostota sprawia, że łatwo wdrożyć go w projekt, a efektywność nie jest dużo gorsza niż inne skomplikowane algorytmy kosztowne obliczeniowo. Jest także dobrą postawą, aby go użyć, a następnie rozszerzyć dzięki czemu możliwe jest poprawienie uzyskanych wyników.

W prototypie użyto zmodyfikowaną wersję algorytmu *Slope One*. Jako podstawową wersję implementacji zastosowano część kodu ze strony *programmersought.com* [26]. w aplikacji nie bez przyczyny została dołączona funkcjonalność dodawania ocen artykułom. Ocenianie pozwala stale rozszerzać bazę danych ocenionych artykułów. Na początku funkcjonowania aplikacji nie ma żadnych ocen od użytkowników, więc *Slope One* nie ma na czym bazować. w takim wypadku przydatne są zaimplementowane funkcjonalności takie jak:

- personalizacja ulubionych kategorii;
- zliczanie kliknięć w artykuły.

Jeśli użytkownik nie spersonalizował swojego profilu, wtedy algorytm wtedy bazuje tylko na *Slope One* i liczbie kliknięć.

Listing 26. Kod odpowiedzialny za znajdowanie sugerowanego artykuł użytkownikowi bez ulubionych kategorii

```
fun findSuggestedArticle(email: String): RssFeed? {
    val userCategories = userSettingsRepository.findById(email)
            .map {
                it.categories
            }.orElse(emptyList())
    val categories = userCategories
            .mapNotNull {
                val category: Category? =
enumValueOrNull(it.toUpperCase())
                category
            }
            .toList()
    return if (categories.isEmpty()) {
        return findPreferedItem(email)
    } else {
      …
    }

}
```

Funkcja `findPreferredItem` z listingu 27. wywołuje algorytm *Slope One* i zwraca rezultat użytkownikowi, ale tylko jeśli predykcja oceny wynosi więcej niż 4.0. w przeciwnym wypadku zwracany jest najpopularniejszy artykuł w danym okresie.

Listing 27. Kod funkcji findPreferredItem

```
private fun findPreferredItem(email: String): RssFeed? {
    val allFeed =
findAllFeedLimited.findAllFeedLimited(LIMITED_NUMBER_OF_FEED)
    val preferences = allFeed
            .map { findPreferenceOnItem.preferenceOnItem(email,
it.link!!) }
            .toList()
    val preference = preferences.maxBy { it.second }
    if (preference!!.second < 4.0) {
```

```
        return allFeed.maxBy { it.clicksNumber }
    }
    return findFeedByLinkUseCase.findFeedByLink(preferences.maxBy {
it.second }!!.first)
}
```

Jeśli użytkownik posiada ulubione paczki kategorii, ścieżka wygląda podobnie, ale bazuje na grupie artykułów z danej kategorii (list. 28).

Listing 28. Fragment kodu odpowiedzialny za znajdowanie sugerowanego artykułu użytkownikowi posiadającemu ulubione kategorie

```
val userSuggestedFeedBasedOnCategories = categories
        .flatMap { findFeedByCategoryUseCase.findFeedByCategory(it) }
        .sortedByDescending { it.pubDate }
        .take(LIMITED_NUMBER_OF_FEED)
if(userSuggestedFeedBasedOnCategories.isNotEmpty()) {
    val preferences =
            userSuggestedFeedBasedOnCategories
            .map { findPreferenceOnItem.preferenceOnItem(email,
it.link!!) }
            .toList()

    val preference = preferences.maxBy { it.second }
    if (preference!!.second < 4.0) {
        return userSuggestedFeedBasedOnCategories.maxBy { it.clicksNumber
}
    }
    findFeedByLinkUseCase.findFeedByLink(preferences.maxBy { it.second
}!!.first)
} else {
    return findPreferedItem(email)
}
```

Najważniejszą częścią stworzonego systemu rekomendacji jest algorytm *Slope One* zaimplementowany z klasie `FindPreferenceOnArticle`. Jak nazwa klasy wskazuje, zwracana jest predykcja oceny konkretnego artykułu. Metoda `findPreferenceOnArticle(userId: String, itemId: String): Pair<String, Double>` przyjmuje id użytkownika i id artykułu, a zwraca dla niego parę - id artykułu i predykcję oceny.

Aby zobrazować sposób działania algorytmu przeanalizowano przykład z tabeli 5.

Tabela 4. Tabela ocen użytkowników z nieznaną oceną

Użytkownik	Ocena artykułu A	Ocena artykułu B	Ocena artykułu C
A	3.0	5.0	**?**
B	4.0	3.0	3.0

Pytanie brzmi jaką ocenę mógłby wystawić użytkownik a artykułowi C? *Slope One* bazuje na różnicy wyników pomiędzy nieznanymi osobami.

Wyliczenie predykcji oceny użytkownika a dla artykułu C wygląda następująco:

$Diff(artykułC\ x\ artykułA) = 3 - 4 = -1$

$Diff(artykułC\ x\ artykułB) = 3 - 3 = 0$

Użytkownik a wystawił ocenę 3.0 artykułowi a i ocenę 5.0 artykułowi B. Dodano różnice do ocen:

$$\text{Predykcja (artykuł C)} = \frac{(-1+3)+(0+5)}{2} = 3.5$$

Predykcja oceny użytkownika a dla artykułu C na podstawie algorytmu *Slope One* wynosi **3.5**.

Po przeniesieniu algorytmu na kod Javy, początkowo tworzona jest tablica różnic w ocenach dla każdej pary przedmiotów. Przedstawia to listing 29.

Listing 29. Część algorytmu *Slope One* wypełniająca mapę diff

```
fun findPreferenceOnArticle(userId: String, itemId: String): Pair<String,
Double> {
    val userIds = findAllUserIds()
    val userRatings = getUserRatings(userId)
    val diffCache = HashMap<String, Double>()
    val counter = HashMap<String, Int>()

    userIds.forEach {
        if (it != userId) {
            val ratings = getUserRatings(it!!)
            userRatings
                    .map { it.link }
                    .forEach { iid ->
                        if (ratings.map { it.link }.contains(itemId) &&
ratings.map { it.link }.contains(iid)) {
                            val diff = ratings.find { it.link == itemId
}!!.rate - ratings.find { it.link == iid }!!.rate
                            if (diffCache.containsKey(iid)) {
                                diffCache[iid] = diffCache[iid]!! + diff
                            } else {
                                diffCache[iid] = diff
                            }
                            if (counter.containsKey(iid)) {
                                counter[iid] = counter[iid]!! + 1;
                            } else {
                                counter[iid] = 1
                            }
                        }
                    }
        }
    }
  …
}
```

Następnie na podstawie tablicy różnic ocen wyliczana jest estymowana ocena artykułu. Jeśli K jest równe zero to oznacza, że nie ma wystarczającej bazy ocen, aby móc przewidzieć ocenę. Wtedy zwracana jest ocena 0.0, która odpowiada braku oceny. Fragment kodu nr 30 przedstawia tę cześć programu.

Listing 30. Część algorytmu Slope One tworząca predykcję oceny artykułu

```
var K = 0
var total = 0.0
userRatings
        .map { it.link }
        .forEach { iid ->
            if (diffCache.contains(iid)) {
                val bias = (diffCache[iid]!! / counter[iid]!!).toFloat()
                val targetUserRating: Double = userRatings.find { it.link
== iid }!!.rate
                val p = targetUserRating + bias
                total += p
                K++
            }
        }
if (K == 0) {
    return Pair(itemId, 0.0)
}
return Pair(itemId, total / K)
```

5.3. Web Scraper

Web scraper to małych rozmiarów aplikacja napisana w Pythonie przy użyciu *frameworka* Flask. Punktem wejściowym do programu są REST *endpointy*. Jest to jedyna możliwa droga interakcji ze *scraperem*.

Listing 31. Rest *Endpoint* do *scrapowania* zdjęcia ze strony CNN

```
@app.route('/get-cnn-image', methods=['POST'])
def get_cnn_image():
    data = request.get_json(silent=True)
    img = download_cnn_image(data['url'])
    if img is not None:
        return jsonify(
            src=img[0],
            title=img[1]
        )
    else:
        return jsonify()
```

Jedynym zadaniem aplikacji jest wyciąganie zdjęć z kodu HTML, czyli tzw. *scrapowanie*. Do tego celu została użyta pythonowa biblioteka BeautifulSoup. Narzędzie pozwala na analizę kodu HTML lub XML, a następnie konwersję w drzewo złożone z atrybutów, wartości, elementów lub *tagów*.

Listing 32. Kod wydobywający zdjęcia ze strony CNN

```
from bs4 import BeautifulSoup

import requests
import re


def download_cnn_image(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    aasClass1 = soup.find_all("img")
    pattern = re.compile("^media__image.*$")
    srcPattern = re.compile("^http.*$")
    srcPattern2 = re.compile("^//cdn.*$")
    image_info = []
    for a in aasClass1:
        photoWithClass = ""
        try:
            photoWithClass = a["class"][0]
            if srcPattern.match(a["src"]):
                return a["src"], a['alt']
            elif srcPattern2.match(a["src"]):
                removedSign = a["src"].replace("//", "http://", 1)
                return removedSign, a['alt']
        except Exception:
            photoWithClass = ''

        if pattern.match(photoWithClass):
            try:
                image_info.append((a["src-mini"], a['alt']))
            except Exception:
                image_info.append((a["src"], "default"))
    return (image_info[:1] or [None])[0]
```

Listing 22. przedstawia kod wydobywający zdjęcie ze strony CNN. Zdjęcie wydobywane jest na podstawie znacznika *img* o odpowiedniej klasie CSS. Tag *img* musi zawierać poprawny atrybut *src*. Jeśli uda się uzyskać zdjęcie to jest ono zwracane w funkcji. w innym wypadku zwracany jest typ `None`, czyli odpowiednik *nulla* z innych języków programowania.

5.4. Baza danych

Podjęto decyzję, aby użyć MongoDB jako bazę danych. Mongo jest jedyną bazą danych użytą w projekcie. w kontekście baz dokumentowych stosuje się termin dokument. Jest to odpowiednik tabeli/encji znanej z klasycznych baz relacyjnych. Istnieje kilka typów używanych dokumentów. Ta dokumentowa baza danych została wybrana ze względu na małą liczbę typów obiektów potrzebnych w aplikacji, ale dosyć mocno zagnieżdżonych. Nie ma potrzeby trzymać obiekty podrzędne jako osobne dokumenty, gdyż odpytywane są zawsze byty nadrzędne. w dokumentach bazodanowych przetrzymywane są także listy. Aby móc coś takiego odwzorować w relacyjnej bazie danych wymagane byłoby wiele operacji *JOIN*. z tego powodu takich rodzaj struktur danych jest dosyć nieefektywny do przechowywania w RDB. Największym atutem baz jak MySQL, Oracle DB jest transakcyjność, która praktycznie nie istnieje w przypadku baz noSQL. w prototypie nie było potrzeby obsługiwania skomplikowanych transakcji, więc problem braku zaawansowanej transakcyjności nie zaistniał. Informacje o dokumentach zapisywanych w bazie danych przedstawiono w formie tabelarycznej (tab. 6 i tab. 7).

Tabela 5. Opis dokumentów wykorzystanych w prototypie

Nazwa dokumentu	**Opis dokumentu**	**Nazwa atrybutu**	**Opis Atrybutu**
RssFeedDocument	Reprezentacja artykułu RSS	*_id*	*Id* dokumentu – reprezentowane jako link do strony z artykułem
		title	Tytuł artykułu
		description	Opis artykułu
		author	Autor artykułu
		pubDate	Data publikacji
		category	Kategoria
		clicksNumber	Liczba kliknięć w artykuł
		rssImageDocument	**Zagnieżdżony dokument** zawiera atrybuty: • *src* – link do zdjęcia; • title – tytuł zdjęcia.
		rateDocument	**Zagnieżdżony dokument** Zawiera atrybuty: • averageRate – średnia ocena od użytkowników; • userRates – lista ocen od użytkowników (para: ocena, email użytkownika.
UserSettingsDocument	Reprezentacja ustawień użytkownika	*_id*	ID dokumentu – reprezentowane jako email użytkownika
		username	Nazwa użytkownika
		description	Opis użytkownika
		categories	Lista ulubionych kategorii użytkownika (pole _id w dokumencie *RssFeedDocument*)
		favouriteFeed	Lista id ulubionych artykułów użytkownika (pole _id w dokumencie RssFeedDocument)
		toReadFeed	Lista id artykułów do przeczytania (pole _id w dokumencie *RssFeedDocument*)

		hiddenFeed	Lista id ukrytych artykułów (pole _id w dokumencie *RssFeedDocument*)
		userFeedPages	**Lista zagnieżdżonych dokumentów:** Lista *feedu* dodanego przez użytkownika Zawiera atrybuty: • *name* – nazwa dodanego feedu; • *url* – link do *feedu*; • *category* – kategoria feedu.
		ratedArticles	**Lista zagnieżdżonych dokumentów:** Lista ocenionych kanałów Zawiera atrybuty: • *name* – nazwa dodanego kanału; • *rate* – ocena artykułu; • *link* – link do artykułu.
RssSiteDocument	Reprezentacja kanału RSS np. strona z *feedem* CNN	*_id*	Link do kanału RSS
		category	Kategoria kanału RSS
		suggestedCategories	Lista sugerowanych przez użytkowników kategorii

Tabela 6. Przykładowe dokumenty w formacie danych JSON

Nazwa dokumentu	**Przykładowy dokument w formacie JSON**

rssFeedDocument	{
	"_id": "https://www.polsatsport.pl/wiadomosc/2021-01-09/jan-blachowicz-skomentowal-zajecie-7-miejsca-w-plebiscycie-przegladu-sportowego-i-polsatu/",
	"title": "Jan Błachowicz skomentował zajęcie 7. miejsca w Plebiscycie Przeglądu Sportowego i Polsatu",
	"description": "Jan Błachowicz zajął 7. miejsce w 86. Plebiscycie Przeglądu Sportowego i Polsatu na Najlepszego Sportowca Roku 2020. Mistrz UFC skomentował to wyróżnienie na łamach swoich mediów społecznościowych.",
	"feedSourceDocument": {
	"_id": "www.polsatsport.pl"

	},
	"author": "",
	"pubDate": {
	"$date": "2021-01-08T23:00:00.000Z"
	},
	"category": "SPORT",
	"rssImageDocument": {
	"src":
	"https://ipla.pluscdn.pl/dituel/cp/iv/ivnpg3zkoshp5thqvf332zc2131jtk1x.jpg",
	"title": ""
	},
	"clicksNumber": 3,
	"rateDocument": {
	"averageRate": 5,
	"userRates": [
	{
	"email": "patrykzdral95@gmail.com",
	"rate": 5
	}
	]
	},
	"_class":
	"com.pjatk.rssreader.rss.infrastracture.RssFeedDocument"
	}

rssSiteDocument	{
	"_id": "https://www.polsatsport.pl/rss/wszystkie.xml",
	"category": "SPORT",
	"suggestedCategories": [
	"SPORT"
	],
	"_class":
	"com.pjatk.rssreader.rss.infrastracture.RssSiteDocument"
	}

userSettingsDocument	{ "_id": "patrykzdral95@gmail.com", "username": "Patryk Zdral", "description": "I'm a software engineer and programming passionate.", "categories": ["sport"], "favouriteFeed": ["https://www.nytimes.com/2020/12/26/world/austria-enters-its-third-lockdown-israel-is-set-to-follow-suit.html"], "toReadFeed": ["https://www.nytimes.com/2020/12/26/us/neverland-ranch-sold.html", "https://www.nytimes.com/2021/01/08/world/americas/ronnie-brunswijk-suriname.html"], "hiddenFeed": [], "userFeedPages": [{ "name": "New York Times World", "url": "https://rss.nytimes.com/services/xml/rss/nyt/World.xml",

<table>
<tr>
<td></td>
<td><pre>
      "category": "WORLD"
    },
    {
      "name": "Techmeme",
      "url": "https://www.techmeme.com/feed.xml?x=1",
      "category": "TECHNOLOGY"
    },
    {
      "name": "Feed burner",
      "url": "https://feeds.feedburner.com/TechCrunch/",
      "category": "TECHNOLOGY"
    }
  ],
  "ratedArticles": [
    {
      "rate": 5,
      "link": "https://www.polsatsport.pl/wiadomosc/2021-
01-09/jan-blachowicz-skomentowal-zajecie-7-miejsca-w-
plebiscycie-przegladu-sportowego-i-polsatu/"
    }
  ],
  "_class":
"com.pjatk.rssreader.user.infrastracture.UserSettingsDocu
ment"
}
</pre></td>
</tr>
</table>

5.5. Testy systemu

System został dokładnie przetestowany. Jako że jest to prototyp, a nie aplikacja produkcyjna skupiono się na dokładnych testach manualnych i testach jednostkowych najważniejszych klas. Wysokie pokrycie kodu testami nie miało priorytetowego znaczenia. w pełnym systemie, który miałby być wdrożony na produkcję wymagane byłyby także zaawansowane testy integracyjne i E2E.

Każda dodana funkcjonalność była testowana na podstawie manualnych scenariuszy testowych, tzw. *smoke testów*.

Tabela 7. Scenariusz *smoke testu* dodania i usunięcia kanału RSS

Test	Dodanie i usunięcie kanału RSS	
Kroki	Działanie	Oczekiwany rezultat
1.	Wybranie w zakładce opcji „*Your feed*”, będąc zalogowanym w aplikacji	Przejście do poprawnego widoku pozwalającego na dodawanie własnych kanałów i przeglądanie dodanych wcześniej
2.	Wypełnienie formularza poprawnymi danymi i kliknięcie przycisku *Follow* np. • *name*: CNN World, • *url*: http://rss.cnn.com/rss/edition_world.rss, • *category*: WORLD	Nowo dodany kanał RSS powinien być widoczny w liście kanałów użytkownika

3.	Kliknięcie przycisku *delete* na świeżo dodanym artykule i odświeżenie strony	Usunięty kanał nie powinien być widoczny

Przy pomocy testów jednostkowych testowano tylko logikę biznesową, czyli środkową warstwę – serwisy. Kod ściśle związany z Kotlinem jest już bardzo dobrze przetestowany i nie ma potrzeby sprawdzać poprawności jego działania. Testy jednostkowe okazały się w szczególności przydatne w momencie sprawdzania poprawności algorytmu *Collaborating Filteringu*. Przykładowy test zawarty jest w listingu 33.

Listing 33. Przykładowy test algorytmu Slope One

```
@Test
fun shouldReturnCorrectSlopeOneEstimation() {
    // given
    val itemA = "example.com"
    val itemB = "example2.com"
    val itemC = "example3.com"
    val john = UserSettingsDocument("x", "username", null,
listOf("TECHNOLOGY"),
        mutableListOf(), mutableListOf(), mutableListOf(),
mutableListOf(),
        mutableSetOf(
            RatedArticleDocument(3.0, itemA),
            RatedArticleDocument(5.0, itemB)))

    val mark = UserSettingsDocument("y", "username", null,
listOf("TECHNOLOGY"),
        mutableListOf(), mutableListOf(), mutableListOf(),
mutableListOf(),
        mutableSetOf(
            RatedArticleDocument(4.0, itemA),
            RatedArticleDocument(3.0, itemB),
            RatedArticleDocument(3.0, itemC)))
    val userSettingsRepository = mockk<UserSettingsRepository> {
        every { findAll() } returns listOf(
            john, mark
        )
        every { findById("x") } returns Optional.of(
            john
        )
        every { findById("y") } returns Optional.of(
            mark
        )
    }
    val findPreferenceOnArticle =
FindPreferenceOnArticle(userSettingsRepository)

    // when
    val preference  =
findPreferenceOnArticle.findPreferenceOnArticle("x", itemC)

    // then
    assertThat(preference.second).isEqualTo(3.5)
}
```

W teście przedstawionym w listingu nr 33 zastosowano konwencję *given/when/then*. w sekcji *given* są argumenty potrzebne do przeprowadzenia testu. w sekcji *when* znajduje się wywołanie testowanej klasy, a w sekcji *then* asercje sprawdzające poprawność testu. w teście sprawdzono wynik ze strony *girlincomputerscience.blogspot.com* [36]. Dla wejściowego zestawu danych autor otrzymał wynik 3.5. Zaimplementowany algorytm także zwrócił ten wynik.

6. Zalety, wady i plany rozwojowe

W prototypie udało się zrealizować założenia opisane w rozdziale nr 3. Tak jak planowano utworzono funkcjonalny prototyp czytnika RSS. Aplikacja chociaż z pozoru prosta dostarczyła sporo problemów natury logicznej i technicznej. w rozdziale zostały opisane zalety i wady rozwiązania. Dodatkowo przedstawiono plany rozwojowe systemu, które nie udało się wdrożyć ze względu na ograniczenia czasowe.

6.1. Zalety

Niewątpliwą zaletą aplikacji jest jej funkcjonalność. Zapewnione są w niej wszystkie podstawowe i potrzebne elementy czytnika RSS. Dzięki tym aspektom mogłaby pozyskać szerokie grono odbiorców. Zaletą projektu jest ciekawe podejście do rozwiązania pewnych problemów, które pojawiły się w czasie implementacji. Automatyczne tworzenie bazy artykułów i stron wraz ze zwiększającą się liczbą użytkowników z pewnością jest prawidłową koncepcją. Wdrożony został zmodyfikowany algorytm *Collaborating Filteringu*. Nie jest on rewolucyjnym rozwiązaniem, ale przedstawił oryginalne podejście do tego naukowego tematu.

Szata graficzna mimo tego, że nie jest na poziomie „topowych" aplikacji internetowych to reprezentuje przyzwoity poziom. Interfejs graficzny jest przejrzysty i schludny. Strona jest responsywna. Oznacza to, że jest w stanie wyświetlać treści na urządzeniach, takich jak komputery, telefony, czy tablety. w dzisiejszych czasach RWD (ang. *Responsive Web Design*) jest jedną z najbardziej pożądanych cech jakie powinny mieć aplikacje webowe [38]. Wynika to z tego, że większość ludzi przegląda informacje na urządzeniach mobilnych.

6.2. Wady

Największą wadą rozwiązania technicznego okazał się stworzony *Web Scraper*. Aplikacja miała posłużyć do *scrapowania* zdjęć z kanałów RSS, które nie udostępniają ich w wiadomościach. Pomysł wydawał się sensowny, lecz, niestety, końcowo nie okazał się zbyt dobry. w praktyce nie dało się stworzyć uniwersalnego *scrapera*. Każda strona ma całkowicie inną strukturę HTML i kodu CSS. z tego powodu niezbędne było tworzenie indywidualnego *scrapera* dla każdego kanału RSS. Jest to działanie czasochłonne i wymagające. Na świecie istnieje bardzo dużo kanałów RSS i w praktyce niemożliwe jest stworzenie *scrapera* dla każdego źródła. Program na pewnym etapie implementacji przestał być rozwijany. Podsystem został na etapie roboczym i umożliwia wydobywanie zdjęć z zaledwie kilku źródeł. w przyszłości na pewno należałoby pomyśleć o rozwiązaniu tego problemu.

Kolejną wadą jest interfejs graficzny, który jest schludny, ale na pewno wymaga poprawy. Profesjonalny projektant UX/UI w zespole mógłby rozwiązać ten problem. Interfejs graficzny odstaje od innych nowoczesnych systemów.

System ze względu na to, że jest prototypem, nie jest przemyślany pod kątem wrażeń użytkownika (tzw. *user experience*). Postawiono na funkcjonalności, które działają, ale od strony użytkownika należy nad nimi popracować, aby korzystanie z systemu było bardziej komfortowe.

Brak odpowiedniego przetestowania (opisanego w rozdziale 5.5) także jest słabą stroną projektu. Produkcyjne systemy aktualnie wymagają dużo skomplikowanych testów.

6.3. Plany rozwojowe

W ramach realizacji pracy część pomysłów nie udało się wdrożyć, ze względu na ograniczone ramy czasowe. w trakcie implementacji narodziło się u autora kilka planów rozwojowych, które zostały opisane w kolejnych akapitach.

W prototypie wdrożono funkcjonalność oceniania artykułów, która ma usprawnić selekcję publikacji. Ocenianie wpływa także na efektywność działania algorytmów rekomendacji. Całość będzie działać w prawidłowy sposób pod warunkiem, że użytkownicy będą oceniać artykuły. Niestety, w praktyce może to wyglądać nieco inaczej. Odbiorcy mogą zignorować funkcję oceniania przez co algorytm nie będzie zwracał pożądanych wyników. Istnieją jednak inne sposoby, aby wywnioskować, czy artykuł wart uwagi. Oprócz określania popularności na podstawie odwiedzin może zostać dodane inne kryterium. Mierzenie czasu przeglądania danego tekstu, może pozwolić określić, jak bardzo artykuł był ciekawy. Do tego użyty może zostać *iFrame*. Artykuł wyświetlony w *iFrame* będzie wciąż istniał w aplikacji, a więc taka kalkulacja jest możliwa.

Aktualnie po kliknięciu w tytuł/kafelek artykułu zostajemy przekierowani na jego stronę. Istnieje tym samym możliwość, że użytkownik straci zainteresowanie czytnikiem i pozostanie na stronie, do której został przekierowany. Ważnym zadaniem narzędzia jest utrzymanie uwagi konsumenta jak najdłużej na źródłowej stronie. Rozwiązaniem tego jest umieszczenie treści artykułu w tzw. *iFrame*. *IFrame* to wbudowana ramka używana wewnątrz strony do załadowania w niej innego dokumentu HTML. Ładowanymi dokumentami HTML w ramce byłyby artykuły z kanałów RSS. z pewnością takie rozwiązanie zwiększyłoby utrzymanie zainteresowania aplikacją.

W profilu użytkownika istnieje możliwość wyboru zawodu. w aktualnej wersji wybrany zawód nie ma wpływu na algorytm rekomendacji. w przyszłości powinien jednak uwzględnić ten parametr. Można to zaimplementować kojarząc dany zawód z konkretnymi zainteresowaniami. Na przykład użytkownik jest z zawodu programistą i możliwe, że ma zainteresowania związane z nowoczesnymi technologiami. w panelu profilu można także wybierać ulubione kategorie. w przyszłości paczki (kategorie) nie będą tylko ogólnymi kategoriami, lecz konkretnymi ścieżkami zainteresowań np. nie kategoria *science*, a zamiast tego kategoria *astronomy*, która byłaby podkategorią *science*.

W planach rozwojowych jest także przeprojektowanie interfejsu graficznego tak, żeby był bardziej profesjonalny i atrakcyjny wizualnie. Potrzebna do tego jest pomoc profesjonalnego projektanta. Jeśli zaistnieje potrzeba wdrożenia wersji produkcyjnej aplikacji, to należy pomyśleć o bardziej zaawansowanym systemie autoryzacji i autentykacji.

7. Podsumowanie i wnioski

Zainteresowania autora pozyskiwaniem treści w Internecie były inspiracją do realizacji tematu niniejszej pracy. Autor na co dzień korzysta z narzędzi w niej omówionych oraz innych, alternatywnych takich jak Google News. Najważniejszym elementem pracy była analiza problematyki tworzenia systemów do czytania i agregacji newsów. Wybrane aplikacje zostały przez autora dogłębnie przestudiowane. Na tej podstawie wyciągnięto wnioski i przygotowano prototyp nowej aplikacji, która wizualizuje inne spojrzenie na tematykę agregacji danych w oparciu strumienie RSS.

Na podstawie analizy tematyki pracy został stworzony prototyp aplikacji będącej implementacją czytnika RSS. Postanowiono wdrożyć w nim swoją wizję realizacji systemu takiego typu. Wszystkie rozwiązania poszczególnych zdefiniowanych zadań były elementem indywidualnej analizy biznesowej i technicznej. Po zagłębieniu się w tematyce autor zdał sobie sprawę, że system będzie tylko namiastką możliwości jakie dają inne zaawansowane czytniki. Okazało się, że sama funkcja dodawania kanałów i przeglądania treści, mimo tego, że dalej jest najważniejsza, to jest tylko jedną z wielu dostępnych możliwości. Przez wiele lat działania czytniki stały się znacznie bardziej zaawansowanymi aplikacjami. Stworzony prototyp jest dobrym punktem startowym do dalszego rozwoju systemu i zgłębiania wiedzy z zakresu tematyki RSS. w przyszłości, po odpowiednim nakładzie pracy, może z powodzeniem konkurować z innymi rozwiązaniami.

Zrealizowane zostały postawione w pracy wymagania. Autor, oprócz wiedzy biznesowej, znacznie rozwinął swoje umiejętności techniczne, dzięki użyciu nieznanych mu wcześniej technologii. Tematyka pracy, po głębszym badaniu, okazała się tematem rozległym. Sam algorytm rekomendacji, który został wdrożony mógłby być dalej rozwijany przez długi czas. Serwisy takie jak Netflix, czy Spotify przez wiele lat pracowały nad przygotowaniem dobrze funkcjonującego silnika rekomendacji [39].

Bibliografia

[1]. *Dokumentacja RSS*, https://validator.w3.org/feed/docs/rss2.html. Dostęp 16.01.2021.

[2]. *Statystyki popularności RSS*, https://trends.builtwith.com/feeds/RSS. Dostęp 16.01.2021.

[3]. *Feedly*, https://feedly.com/. Dostęp 26.12.2020.

[4]. *It is time for an RSS Revival*, https://www.wired.com/story/rss-readers-feedly-inoreader-old-reader/. Dostęp 23.12.2020.

[5]. *Inoreader*, https://www.inoreader.com/. Dostęp 26.12.2020.

[6]. *The Old Reader*, https://theoldreader.com/. Dostęp 26.12.2020.

[7]. *Newsblur*, https://newsblur.com/. Dostęp 26.12.2020.

[8]. *Feedbin*, *https://feedbin.com/*. Dostęp 26.12.2020.

[9]. *Notka o aktywnych użytkownikach na stronie głównej Newsblur*, https://newsblur.com/. Dostęp 23.12.2020.

[10]. *Zdjęcie aplikacji Google Reader,* https://miro.medium.com/max/875/1*t6-St04EotUIZh4EOwE5oA.png. Dostęp 31.01.2021.

[11]. *Slope One Predictors for Online Rating-Based Collaborative Filtering*, https://epubs.siam.org/doi/10.1137/1.9781611972757.43. Dostęp 25.12.2020.

[12]. *RSS Feed vs Email Subscription*, https://www.mojomedialabs.com/blog/rss-feed-vs.-email-subscription-whats-the-difference. Dostęp 16.01.2021.

[13]. *REST API Tutorial*, https://restfulapi.net/. Dostęp 24.12.2020.

[14]. *Spring w akcji. Wydanie V*, Cragi Walls, wyd. Helion 2019, ISBN: 978-83-283-5606-1

[15]. *Kotlin w Akcji*, Dmitry Jemarov, Sveltana Isakova, wyd. Helion 2018, ISBN: 978-83-283-4720-5

[16]. *Which build tool do you use for your main project?*, https://snyk.io/blog/jvm-ecosystem-report-2018-tools/, Dostęp 23.12.2020.

[17]. *Przewodnik po MongoDB. Wydajna i skalowalna baza danych. Wydanie III*, wyd. Helion 2020, ISBN: 978-83-283-6533-9

[18]. *DB-Engines Ranking*, https://db-engines.com/en/ranking. Dostęp 23.12.2020.

[19]. *React i Redux. Praktyczne tworzenie aplikacji WWW. Wydanie II*, Kirupa Chinnathambi, wyd. Helion 2019, ISBN: 978-83-283-4726-7

[20]. *TypeScript. Od początkującego do profesjonalisty*, Adam Freeman, wyd. 2020, ISBN: 978-83-283-6531-5

[21]. *Python. Nowoczesne programowanie w prostych krokach.* Wydanie II, Bill Lubanovic, wyd. Helion 2020, ISBN: 978-83-283-6842-2

[22]. *What programming language should i learn first*, https://www.freecodecamp.org/news/what-programming-language-should-i-learn-first-19a33b0a467d/. Dostęp 23.12.2020.

[23]. *5 reasons why learning Python is the best decision*, https://medium.com/datadriveninvestor/5-reasons-why-i-learned-python-and-why-you-should-learn-it-as-well-917f781aea05. Dostęp 23.12.2020.

[24]. *Flask. Tworzenie aplikacji internetowych w Pythonie. Wydanie II*, Miguel Grinberg, wyd. Helion 2020, ISBN: 978-83-283-6383-0

[25]. *Dokumentacja React.Js*, https://reactjs.org/docs/getting-started.html#versioned-documentation. Dostęp 27.12.2020.

[26]. *React hooks*, https://reactjs.org/docs/hooks-faq.html. Dostęp 28.12.2020.

[27]. *Notka o strukturze folderów w ReactJs*, https://reactjs.org/docs/faq-structure.html. Dostęp 29.12.2020.

[28]. *Dokumentacja i repozytorium biblioteki Axios*, https://github.com/axios/axios. Dostęp 29.12.2020.

[29]. *Najlepsze biblioteki http w języku programowania JavaScript,* https://syndicode.com/blog/best-js-http-requests-libraries-for-2019/. Dostęp 23.01.2021

[30]. *Dlaczego warto używać wstrzykiwanie przez konstruktor w Springu*, https://spring.io/blog/2007/07/11/setter-injection-versus-constructor-injection-and-the-use-of-required. Dostęp 04.01.2021.

[31]. *Dobre praktyki w Springu*, https://www.endoflineblog.com/spring-best-practices. Dostęp 21.01.2021.

[32]. *Porównanie Quarkusa ze Springiem*, https://simply-how.com/quarkus-vs-spring. Dostęp 04.01.2021.

[33]. *Repozytorium i dokumentacja biblioteki Rome*, https://rometools.github.io/rome/. Dostęp 04.01.2021.

[34]. *Repozytorium biblioteki Rssreader*, https://github.com/w3stling/rssreader. Dostęp 04.01.2021.

[35]. *Algorytm Slope One zaimplementowany w Javie*, https://www.baeldung.com/java-collaborative-filtering-recommendations. Dostęp 05.01.2021.

[36]. Przedstawienie Algorytmu Slope One, https://www.programmersought.com/article/9375179769/. Dostęp 19.01.2021

[37]. *Blog omawiający algorytm Slope One,* https://girlincomputerscience.blogspot.com/2010/11/slope-one.html?m=1. Dostęp 11.01.2021.

[38]. *Artykuł o RWD,* https://grupad.pl/rwd-czyli-czym-jest-responsywnosc-i-dlaczego-jest-tak-wazna/. Dostęp 14.01.2021.

[39]. *How Netflix's Recommendation engine works?* https://medium.com/@springboard_ind/how-netflixs-recommendation-engine-works-bd1ee381bf81. Dostęp 16.01.2021.

Spis rysunków

Spis tabel

Spis Listingów