POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

Instytut Automatyki, Robotyki i Inżynierii Informatycznej

Jakub Kolasiński

Kacper Kociubiński

Dawid Kudela

Dokumentacja projektu

PODSTAWY TELEINFORMATYKI

Temat: NewsScraper

ROK AKADEMICKI 2018/2019

1. **Temat projektu**

   NewsScraper - mobilna metawyszukiwarka

2. **Problematyka projektu**

   Web scraping jest to technika, dzięki której oprogramowanie komputerowe wydobywa dane z serwisu internetowego. System aby uzyskać dostęp do strony internetowej może wykorzystywać bezpośrednio protokół HTTP, lub przy użyciu przeglądarki internetowej.

   Termin web scrapingu zazwyczaj odnosi się do zautomatyzowanego procesu przeglądania i pobierania danych. W tym celu wymagana jest implementacja osobnego bota, lub crawlera, który co zadany czas przegląda dane serwisy i pobrane z nich dane zapisuje w lokalnej bazie danych, tak aby można było je wykorzystać w później analizie oraz przetwarzaniu.

3. **Uzasadnienie wyboru tematu**

   Wybór tej tematyki projektu był u nas motywowany chęcią rozwinięcia posiadanych indywidualnych umiejętności z dziedziny programowania. Zaprojektowanie i realizacja metawyszukiwarki pozwoliła nam na łatwy podział zdań opierający się o konkretne technologie. Dzięki czemu dla trzech członków zespołu mogliśmy wyodrębnić trzy aplikacje - klient iOS, klient Android oraz serwer Web API. Każda z realizowanych części była bezpośrednio związana z zainteresowaniami poszczególnych osób w grupie oraz ich umiejętnościami obsługi technologii czy języków. Kluczowy moduł systemu - web scraper, implementowany był po części przez każdego z członków zespołu projektowego.

   Kolejną kluczową przesłanką była możliwość sprawdzenia się w budowaniu aplikacji o charakterze komercyjnym. Głównym zagadnieniem systemu nie było rozwiązanie algorytmicznego problemu, a zaprezentowanie możliwości ułatwienia poszukiwania informacji na dany temat w Internecie, co chcieliśmy kierować bezpośrednio do szerokiej ilości użytkowników. Mogliśmy dzięki temu stanąć przed wyzwaniem zaimplementowania prostych rozwiązań personalizacyjnych takich jak własna zapisana lista tagów, na osobistym koncie użytkownika.

4. **Ogólny podział prac**

LP	Imię i nazwisko	Zadanie
1	Jakub Kolasiński	Aplikacja iOS, Web API, Web scraper
2	Dawid Kudela	Aplikacja Android, Web scraper
3	Kacper Kociubiński	Web scraper, Web API, baza danych

5. **Szczegółowy podział prac**

a) **zadania o charakterze merytorycznym**

- **Kacper Kociubiński:**
    1. Stworzenie mechanizmu przeszukiwania węzłów HTML dla stron Reddit, Twitter, Wykop.
    2. Implementacja parsera informacji do określonego schematu: autor, informacja, bezpośredni link, zdjęcie, pokrewne hashtagi.
    3. Połączenie i komunikacja z bazą danych.
    4. Automatyzacja polegająca na cyklicznym odwiedzaniu zadanych serwisów oraz informowanie zainteresowanych klientów o wynikach danego przeszukiwania.
    5. Optymalizacja algorytmów przeszukiwań stron przez wykorzystanie wielowątkowości.
    6. Implementacja obsługi wielu klientów.
    7. Programowa możliwość dodania kolejnych stron (na podstawie węzłów HTML).
    8. Obsługa błędów i sytuacji wyjątkowych związanych z pobranymi danymi ze stron www.

- **Dawid Kudela:**

    Aplikacja Android:

    1. Utworzenie projektu, opartego o architekturę Jetpack.
    2. Utworzenie widoku listy newsów.

3. Zaimplementowanie komunikacji z serwerem.
4. Dodanie narzędzia Firebase.
5. Wyświetlanie newsów otrzymanych z serwera.
6. Odświeżanie listy newsów.
7. Utworzenie widoków logowania i rejestracji.
8. Utworzenie widoku do zarządzania subskrybowanymi tagami.
9. Wyświetlanie najpopularniejszych tagów.

Narzędzie Google Analytics:

10. Zapoznanie się ze strukturą danych w module Google Analytics.
11. Zbieranie danych o użytkownikach, oraz o subskrybowanych przez nich tagach.
12. Eksport danych do BigQuery.
13. Stworzenie modułu pobierającego tagi z BigQuery.
14. Wyznaczenie najpopularniejszych tagów z ostatnich trzech dni.
15. Wysyłanie najpopularniejszych tagów do użytkownika.

- **Jakub Kolasiński:**

Web API:

1. Inicjalizacja Web API.
2. Inicjalizacja nierelacyjnej bazy danych MongoDB.
3. Utworzenie odpowiednich punktów dostępowych do usługi sieciowej.
4. Połączenie z bazą danych.
5. Obsługa operacji na bazie danych, związanych z konkretnymi punktami dostępowymi.
6. Stworzenie systemu logowania opartego o tokeny.

Aplikacja iOS:

7. Inicjalizacja aplikacji mobilnej działającej w systemie iOS.
8. Wykorzystanie utworzonego Web API w aplikacji.

9. Utworzenie interfejsu graficznego (widoki).
10. Dodanie narzędzi: Firebase Crashlytics oraz Google Analytics.
11. Połączenie stworzonych modułów w działającą aplikację.

b) **zadania o charakterze organizacyjnym**

1. Wybranie zagadnienia i stworzenie posmysłu aplikacji.
2. Określenie architektury i budowy aplikacji.
3. Podział implementacji na zadania oraz rozdzielenie ich pomiędzy członków zespołu.
4. Ustalenie osobowego harmonogramu prac.
5. Wzajemna kontrola i przyrostowe prezentowanie wyników swojej pracy.
6. Złączenie osobistych modułów do jednolitej i działającej aplikacji.
7. Opisanie pracy zespołu w formie sprawozdania grupowego.

6. Harmonogram realizacji projektu

- **Kacper Kociubiński**

Data	**Nr zadania merytorycznego**	**Nr zadania organizacyjnego**
03.04.	1	1, 2, 3, 4
17.04.	2, 3, 4, 8	5
15.05.	5, 6,	5, 6
29.05.	7,	7

- **Jakub Kolasiński**

Data	Nr zadania merytorycznego	Nr zadania organizacyjnego
03.04.	1, 2, 7	1, 2, 3, 4
17.04.	3,4,5	5
15.05.	6,8	5, 6
29.05.	9, 10, 11	7

- **Dawid Kudela**

Data	Nr zadania merytorycznego	Nr zadania organizacyjnego
03.04.	1, 2	1, 2, 3, 4
17.04.	3, 4, 8, 10	5
15.05.	5, 6, 9	5, 6
29.05.	7, 11, 12, 13, 14	7

7. Funkcjonalność aplikacji

Aplikacja pozwala na przeglądanie listy newsów, zebranych z wielu serwisów internetowych. Newsy pobierane są z serwisów na podstawie tagów. Newsy oznaczane są tagami w procesie dodawania ich przez użytkowników, do serwisu internetowego. Dzięki tagom system może pobierać dokładnie te informacje, które interesują użytkownika oraz wyświetlić je w uporządkowanej formie.

System udostępnia możliwość zarządzania obserwowanymi tagami, w celu wyboru przez użytkownika tylko interesujących go tagów, a co za tym idzie, przeglądaniu interesujących go newsów.

Z uwagi na konieczność szybkiego dostępu do danych z aplikacji klienckiej, newsy pobierane są automatycznie co 10 minut. Z tego względu użytkownik domyślnie dostaje odświeżoną listę newsów w konkretnym odstępie czasowym. Jeśli jednak użytkownik ma potrzebę odświeżenia newsów w danej konkretnej chwili, ma taką możliwość, poprzez stuknięcie przycisku “Odśwież”.

Procedura nagłego odświeżenia newsów wywołuje natychmiastowe uruchomienie modułu odwiedzającego witryny internetowe i pobierającego newsy. Skutkiem takiej operacji jest otrzymanie przez użytkownika aktualnych newsów, jednak proces ten trwa o wiele dłużej niż domyślnie, z uwagi na czas jaki potrzebny jest do odwiedzenia stron i pobrania odpowiednich danych.

8. Technologie wybrane do realizacji zadania

Implementacja Web API:

Visual Studio 2019 - C# - ASP.NET Core

Wybraliśmy język programowania wysokiego poziomu jakim jest C# ze względu na szeroką gamę funkcjonalności jakie są wbudowane bezpośrednio w jego standard. Wraz ze środowiskiem Visual Studio stanowi bardzo silne narzędziem do programowania w wielu architekturach. Dzięki jego popularności wiele małych problemów mogliśmy rozwiązać w raz z użyciem dokumentacji lub bezpośrednio na gotowych implementacji zaczerpniętych od community tego języka.

Implementacja przeszukiwania węzłów HTML:

C# - pakiet NuGet HTML Agility Pack

Biblioteka rozszerzająca podstawowy standard o łatwą i zrozumiałą obsługę kodu HTML z odwzorowaniem jego elementów na obiekty. Umożliwiło nam to szybkie odwzorowanie węzłów HTML na konkretne pola w wykorzystywanych przez nas

klasach, iterację po ustalonej hierarchii strony www oraz uprościło parsowanie danych, ze względu na możliwość wybierania z węzła jedynie konkretnych atrybutów.

Aplikacja kliencka iOS:

Xcode 10 - Swift

W celu udostępnienia użytkownikowi wyboru platformy, przez którą będzie mógł korzystać z interfejsu graficznego systemu jakim jest aplikacja mobilna, NewsScraper został zaimplementowany na systemy iOS i Android.

Do implementacji aplikacji działającej w systemie iOS użyty został język Swift oraz natywne IDE do tworzenia takich aplikacji, czyli program Xcode. Aplikacje na system iOS w środowisku natywnym można tworzyć zarówno w języku Swift jak i Objective-C, jednak język Swift jest zdecydowanie nowszym, bardziej czytelnym oraz prostszym w użyciu językiem w kontekście tworzenia aplikacji mobilnej oraz tworzenia powiązań między logiką aplikacji a interfejsem.

Dla języka Swift dostępnych istnieje bardzo dużo tutoriali oraz samouczków, dzięki czemu implementacja pewnych rozwiązań staje się prostsza, a sama społeczność programistów Swift'a jest na tyle rozbudowana, że znalezienie pomocy w rozwiązaniu konkretnych problemów z reguły jest bardzo proste.

Aplikacja kliencka Android:

Android Studio 3.4 - Do stworzenia aplikacji mobilnej na system Android zostało wybrane to narzędzie ze względu na swoją popularność i wygodę w stosowaniu. Jest to IDE oferowane przez Jetbrains i ma taką samą strukturę jak inne tego typu programy tej firmy. Ucząc się korzystania z tego narzędzia, będzie łatwiej korzystać także z pozostałych środowisk.

Kotlin - Na Google I/O w 2019 roku język ten został ogłoszony jako preferowany do programowania aplikacji mobilnych na system Android. Warto inwestować swój czas w naukę tego języka, ponieważ staje się on coraz częściej stosowany w dużych firmach (nie tylko w programowaniu na Androida), przykładem jest firma Allegro, która dosyć szybko

zaczęła mocno stawiać na Kotlin. Jest to język statycznie typowany, działający na maszynie wirtualnej Javy, który pozwala pisać znacznie bardziej zwięzły i przejrzysty kod, niż ten implementowany w Javie.

Moduł do najpopularniejszych tagów:

Google Analytics - Narzędzie stosowane do analizy danych o użytkownikach i ich pracy z naszym systemem. Dzięki tej technologii możemy zbierać informacje o tym jacy są nasi użytkownicy(m.in. płeć, wiek, miejsce w którym korzystają z aplikacji), a także obserwować jakie operacje w naszej aplikacji wykonują.

BigQuery - Jest to baza danych oferowana przez firmę Google, do której możemy eksportować potrzebne przez nas dane, a następnie z poziomu serwera łączyć się przy pomocy specjalnego API.

Firebase - Technologia, w której zawarty jest Google Analytics. Aplikacje mobilne w prosty sposób łączą się bezpośrednio z kontem Firebase'a. Po przejściu do wersji premium, możliwe jest eksportowanie danych do bazy danych BigQuery. Firebase oferuje także mnóstwo innych technologii, jak np. ML Kit, Crashlytics, lub system reklamowy AdMob.

9. Metodyka projektowania i implementowania

Użyta została metodyka zwinna, polegająca głównie na implementacji przyrostowo-iteracyjnej. Każdy termin oznacza dla nas koniec pewnego etapu budowania aplikacji oraz możliwość oddania konkretnej, a w szczególności działającej funkcjonalności. Takie podejście pozwala na rozbicie architektury naszego programu na moduły będące kontynuacją większej funkcjonalności, która w końcowym etapie pracy da w pełni działającą aplikację opartą o wcześniej zaimplementowane części.

Z założenia, metodyka zwinna wymaga zdefiniowania problemu, rozbicia go na części oraz rozplanowania oddania konkretnego modułu do pewnego terminu. Dzięki zastosowaniu takiego sposobu pracy możemy połączyć planowanie, projektowanie i implementację w jeden konkretny schemat okraszony konkretnymi datami realizacji.

10. **Krokowe przedstawienie procesów powstawania modułów systemu**

1. Proces organizacji projektu:
    a. Podjęcie problematyki budowy aplikacji mobilnej w architekturze klient - serwer, opartej o działanie web scrapera.
    b. Wykorzystanie bazy danych, implementacja Web API, serwera będącego zapleczem wykonawczym dla web scrapera oraz mobilnej aplikacji spełniającej rolę interfejsu graficznego dla użytkownika.
    c. Wydzielenie systemu na konkretne moduły, stanowiące część bazową oraz szereg rozwijanych od nich funkcjonalności.
    d. Podział prac z wykorzystaniem metodyk zwinnych oraz z uwzględnieniem umiejętności i zainteresowań każdego z członków zespołu.
    e. Kontrola terminowości implementacji zadań wyznaczonych na konkretną datę.
    f. Systematyczne opisywanie rozwijanych funkcjonalności w dokumentacji oraz prezentacja ukończonych prac zgodnie z założeniami na zajęciach.
    g. Połączenie modułów w gotowy system i praca nad nim jako całościowej aplikacji.
    h. Podsumowanie z ukończoną kompletną dokumentacją.
2. Proces powstawania web scrapera:
    a. Mechanizm węzłów HTML:
        i. Odnalezienie węzłów odpowiedzialnych za dostęp do informacji takich jak bezpośredni link do wydarzenia, dołączone zdjęcie, autor, załączony tekst, data publikacji.
        ii. Zbudowanie odpowiednich klas dla przeszukiwania każdej ze stron.
        iii. Implementacja obsługi błędów związanych z brakiem którejś z informacji.
    b. Parser:
        i. Stworzenie klasy News posiadającej identyczne pola dla każdej ze stron - autor, informacja, bezpośredni link, zdjęcie, data publikacji.
        ii. Metody rozdzielające uzyskane dane ze scrapera do tworzenia listy

newsów.

c. Połączenie i komunikacja z bazą danych:
    i. Wykorzystanie nie relacyjnej bazy danych.
    ii. Połączenie bazy z kodem scrapera.
    iii. Wykorzystanie obsługi bazy danych w logice aplikacji i dołączenie list pobranych newsów do bazy.
d. Automatyzacja pobierania stron:
    i. Implementacja listy subskrypcji tagów dla każdego indywidualnego użytkownika.
    ii. Odświeżanie newsów subskrybowanych użytkowników automatycznie co 2 minuty.
    iii. Pozwolenie użytkownikowi na bezpośrednie odświeżenie listy newsów za pomocą przycisku.
e. Optymalizacja algorytmów przeszukiwań:
    i. Strony zostają pobierane asynchronicznie, każda strona w osobnym wątku.
    ii. Skrócony sumaryczny czas pobierania stron i przeszukiwania ich o ok. 60%.
f. Obsługi wielu klientów.
    i. Zastosowanie rozwiązania REST API pozwalającego na równoległy dostęp do zasobów wielu użytkownikom.
g. Wyłapywanie sytuacji wyjątkowych.
    i. Obsługa błędów związana z pobranymi plikami, aby nie pozostawiać pustych pól tam gdzie nie pobrały się dane w wyniku błędu.

3. Proces powstawania Web API:
    a. wybór technologii (ASP.NET Core) oraz bazy danych (MongoDb),
    b. utworzenie klastra z bazą danych w serwisie MongoDb Cloud,
    c. uzyskanie dostępu do bazy danych z poziomu programu w języku C#, z użyciem odpowiedniego pakietu NuGet (mongodb.Driver),
    d. utworzenie w bazie danych 2 kolekcji: News, Users,

e. stworzone zostały następujące punkty dostępowe:

   i. /api/register - POST - rejestracja użytkownika,
   ii. /api/login - POST, DELETE - logowanie/wylogowanie użytkownika,
   iii. /api/user/tags - PUT, GET - udostępnienie/pobranie tagów użytkownika,
   iv. /api/user/news - GET - pobranie newsów użytkownika,
   v. /api/user/popularNews - GET - pobranie popularnych newsów,

f. system logowania został oparty o tokeny, które generowane są przez serwer podczas logowania użytkownika i przekazywane mu w odpowiedzi na wysłane dane logowania. Przy korzystaniu z punktów dostępowych użytkownik identyfikuje się danym tokenem, który jest przesyłany w nagłówku zapytania. Tokeny te przypisane są do użytkowników i przechowywane są w osobnej bazie na danych,

g. implementacja modelu obejmuje utworzenie klas News, Tag, User oraz umożliwienie serializacji i deserializacji obiektów tych klas do formatu JSON w celu integracji z bazą danych,

h. implementacja punktów dostępowych - generacja/identyfikacja tokenu dla użytkownika oraz odpowiednie operacje na bazie danych,

i. podpięcie modułu Scrapera w taki sposób aby działał niezależnie, cyklicznie odwiedzał zadane strony oraz wpisywał do bazy danych pobrane informacje.

4. Aplikacja mobilna iOS:

   a. ustalenie architektury aplikacji - VIP (View-Interactor-Presenter),
   b. wykorzystanie punktów dostępowych Web API z użyciem framework'a AlamoFire,
   c. zamapowanie danych w postaci JSON do obiektów z użyciem framework'a ObjectMapper
   d. utworzenie 4 głównych widoków - logowania, listy newsów, podglądu pojedynczego newsa oraz zarządzania subskrybowanymi tagami,
   e. implementacja logiki interakcji między widokami,

f. utworzenie algorytmu grupującego newsy według tagów oraz wyświetlanie ich w odpowiednich sekcjach,
g. implementacja reszty logiki aplikacji,
h. podpięcie modułu Google Analytics oraz Crashlytics w celu gromadzenia informacji o subskrybowanych przez użytkowników tagach oraz crash'ach aplikacji,
i. podpięcie modułu reklamowego Google AdMob oraz dodanie widoku reklamy do ekranu logowania.

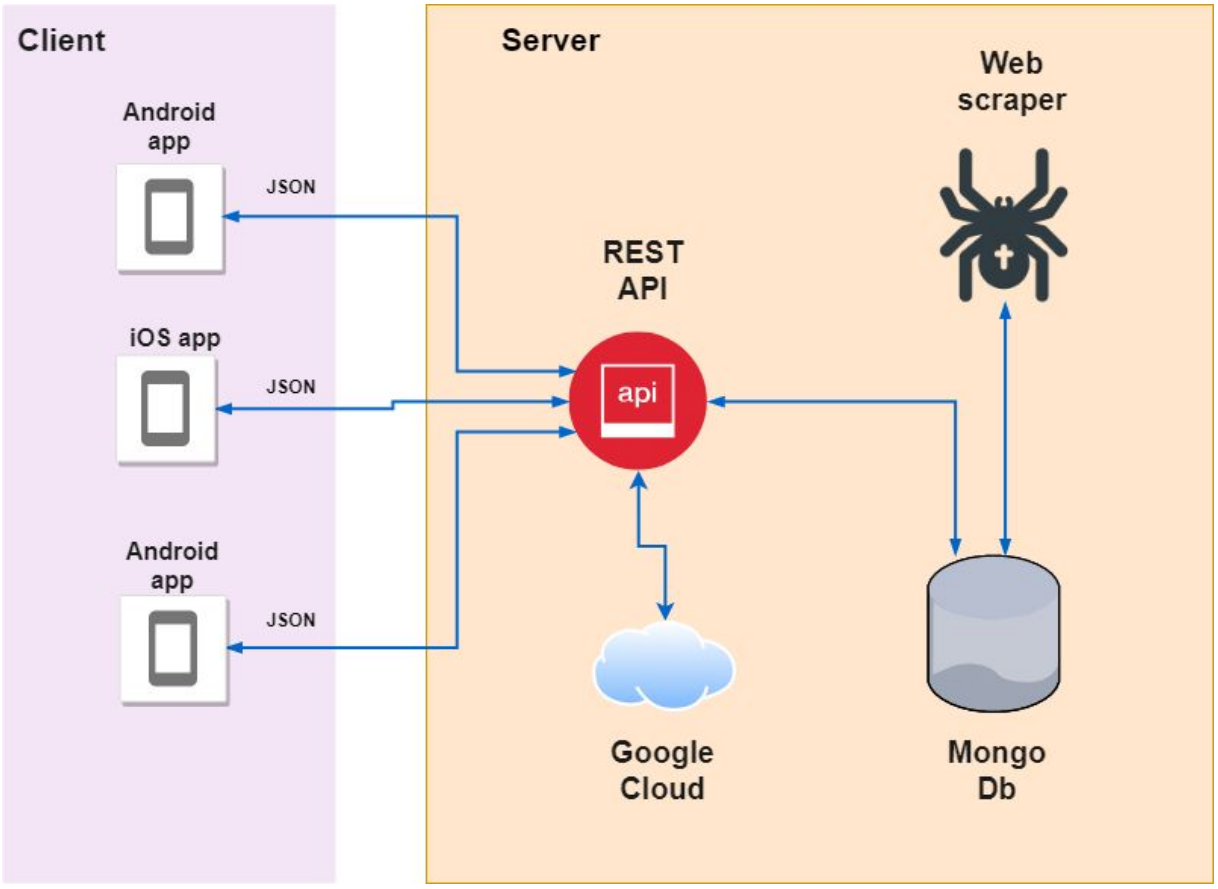
5. Aplikacja mobilna Android:

   a. ustalenie architektury aplikacji - Jetpack,
   b. wykorzystanie REST API przy użyciu biblioteki RxJava,
   c. utworzenie widoków do reprezentacji system rejestracji i logowania,
   d. utworzenie widoków do reprezentacji zarówno listy newsów jak i konkretnego newsu,
   e. utworzenie widoków do reprezentacji listy tagów, subskrybowanych przez użytkownika,
   f. pobieranie zdjęć przy użyciu korutyn - wsparcie do programowania asynchronicznego oferowane przez język Kotlin,
   g. utworzenie widoku do reprezentacji jednego newsu,
   h. utworzenie widoku do zarządzania subskrybowanymi tagami,
   i. podpięcie modułu Google Analytics do zbierania informacji o użytkownikach i subskrybowanych przez nich tagach,
   j. podpięcie modułu Crashlytics w celu zbierania informacji o crashach użytkowników w celu poprawy działania aplikacji, aby takie zdarzenia nie miały już miejsca.

6. Moduł łączący się z Google Analytics przy pomocy BigQuery:

   a. zapoznanie się ze strukturą działania konta Firebase z BigQuery,
   b. eksport danych z Analytics'a do BigQuery,
   c. utworzenie modułu do komunikacji z BigQuery,
   d. wyciągnięcie z BigQuery danych dotyczących tagów przy pomocy modułu.

11. **Rysunek oraz opis działania systemu**



Projekt został zrealizowany w architekturze klient-serwer, więc cały system został podzielony na część klienta i część serwera.

Część klienta reprezentowana jest przez aplikacje mobilne na systemy iOS oraz Android. Aplikacje te są graficznym interfejsem systemu, dzięki któremu użytkownik może korzystać z zaimplementowanych funkcjonalności.

Aplikacje oprócz zaimplementowanej własnej logiki oraz widoków korzystają z danych znajdujących się w części serwera. Komunikacja klienta z serwerem odbywa się przez Web API, które oparte jest o styl architektury REST. Klient w celu pobrania lub wysłania zasobów do bazy danych wykonuje odpowiednie zapytania HTTP do Web API. W rezultacie serwer zwraca do klienta informację o powodzeniu zapytania oraz ewentualne

zasoby w formacie JSON, które w aplikacji mapowane są na obiekty konkretnych klas modelu.

Część serwera jest bardziej rozbudowana niż część klienta. Składa się ona ze wspomnianego wcześniej Web API, które udostępnia odpowiednie punkty dostępowe, pozwalające na interakcję klienta z bazą danych z jednoczesnym przeniesieniem odpowiedzialności składowania zasobów oraz obliczeń na stronę serwera. Web API komunikuje się z bazą danych, kiedy zostanie wysłane zapytanie HTTP.

Do zarządzania danymi wykorzystane zostało MongoDb - nierelacyjna baza danych, w której dane przechowywane są w formacie JSON. Baza MongoDb istnieje w formie klastra na zdalnym serwerze, w serwisie MongoDb Cloud.

W części serwera, istnieje moduł web scrapera, który odpowiedzialny jest za gromadzenie danych poprzez odwiedzanie zadanych stron internetowych, przeszukiwanie odpowiednich węzłów oraz wpisywanie zebranych informacji do bazy danych. Web scraper powtarza cały wspomniany proces automatycznie, w odstępie czasowym 10 minut. Pobieranie newsów opiera się na podstawie tagów obserwowanych przez użytkowników. Scraper przed odwiedzeniem stron internetowych pobiera wszystkie tagi, które aktualnie są w bazie danych oraz dla każdego tagu przeszukuje newsy oraz zapisuje je w bazie danych w odpowiednim formacie.

Automatyczne działanie scrapera pozwala na bardzo szybkie pobranie danych o newsach przez klienta w dowolnym momencie. Klient nie musi oczekiwać aż scraper wykona cały proces od początku, gdyż w bazie danych zawsze znajdować się będą aktualne dane.

W części serwera znajduje się także moduł odpowiadający za komunikację z serwisem Google Cloud, w którym zapisywane są statystyki związane z subskrybowanymi przez użytkwoników tagami. Integracja z Google Cloud pozwala uzyskać najbardziej popularne w danym okresie tagi oraz przedstawić je użytkownikowi jako proponowane.

12. Specyfikacja ważniejszych algorytmów

1. Algorytm pobierania newsów i parsowania ich do postaci wykorzystywanej przez aplikację.

Wejście : struktura HTML danego serwisu, link do wyszukiwarki tagów w serwisie.

Wyjście: lista newsów pobranych z serwisu, każdy news o określonej z góry strukturze (link, tekst, zdjecie, data, autor)

Metoda:

Zbiór lokalizacji danych w postaci węzłów HTML.

```
private string htmlMessageNode = ".//div[2]/div/div[2]/div[1]/span/a/h2/span";
private string htmlUsereNode = ".//div[2]/div/div[2]/div[2]/div[2]/div/a";
private string htmlPhotoNode = ".//div[2]/div/div[1]/div/div/a/div";
private string htmlUsertLinkNode = ".//div/div/div[1]/a[1]";
private string htmlTargetLinkNode = ".//div[2]/div/div[2]/div[1]/span/a";
private string htmlDateNode = ".//div[2]/div/div[2]/div[2]/div[2]/a";
private string htlmStartingNode = "//*[@id=\"SHORTCUT_FOCUSABLE_DIV\"]/div[2]/div/div/div/div[2]/div[3]/div[1]/div[3]/div";
private string pageLink = "https://www.reddit.com/search?q=%23{0}";
```

Pobranie i parsowanie danych z wykorzystaniem zaimplementowanych metod w pętli dla wszystkich newsów na stronie (20 - 50 newsów). Wykorzystanie wielowątkowości według wzorca asynchronicznego opartego o zadnia. Na podstawie pobranych informacji tworzymy ten sam typ obiektu dla wszytkich stron internetowych.

```
public async Task getItteratorAsync(string tag)
{
    string link = String.Format(pageLink, tag);
    await getPageAsync(link);
    page = "Reddit";
    foreach (HtmlNode li in htmlPageDoc.DocumentNode.SelectNodes(htlmStartingNode))
    {
        Wrapper post = new Wrapper();

        try
        {
            post.user = getUser(li);

            post.message = getMessage(li);

            post.targetLink = getTargetLink(li);

            post.photo = getPhoto(li);

            post.date = getDate(li);

            post.page = page;

            wrapperList.Add(post);
```

Przykładowa metoda parsująca pobrane informacje do postaci akceptowalnej dla klasy News. Wykorzystanie wyrażeń regularnych, pozwalających na wybranie tylko

istotnych informacji. Obsługa błędów dla sytuacji wyjątkowych, w tym przypadku dla pola zdjęcie jest akceptowalny Null.

```
public string getPhoto(HtmlNode photoNode)
{
    try
    {
        string extractionLink;
        photoNode = photoNode.SelectSingleNode(htmlPhotoNode);
        HtmlAttribute attribute = photoNode.Attributes["style"];
        extractionLink = attribute.Value;

        Regex rx = new Regex(@"\((.*?)\)");
        MatchCollection matches = rx.Matches(extractionLink);

        photo = matches[0].Groups[0].Value.TrimStart('(').TrimEnd(')');
    }
    catch (Exception)
    {
        return null;
    }
    return photo;

}
```

Struktura klasy News, która stanowi podstawę obiektu przechowującego informacje o wydarzeniach. Jest to klasa standardowa dla każdej ze stron www. Tak reprezentowane obiekty będą przechowywane w nierelacyjnej bazie danych.

```
public class News
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string id;
    public string[] tags = new string[] { };
    public string author;
    public string text;
    public string link;
    public string photo;
    public string date;
    public string page;
```

2. Algorytm synchroniczny uruchamiający opisany powyżej pobierania stron www i parsowania ich do postaci newsów.

Wejście : pobrana lista tagów z bazy danych.

Wyjście: zapisanie listy newsów do bazy.

Metoda:

Jest to algorytm będący warstwę wyżej niż przedstawione powyżej pasowanie i pobieranie stron, zajmuje się on asynchronicznym uruchomienie funkcji odpowiedzialnych za obróbkę informacji pochodzących z serwisów i zapis do bazy danych. W celu zapewnienia synchronizacji i jednego zapisu do bazy dla newsów zebranych ze wszystkich stron, wątek główny czeka, aż wszystkie wątki poboczne skończą swoje zdanie i dopiero wtedy rozpoczyna zapis.

```
var db = UserController.ConnectToDataBase();

var allTags = FetchAllTags(db);
List<News> newsList = new List<News>();
foreach (var tag in allTags)
{
    WykopWrapper ww = new WykopWrapper();
    var tagNews = ww.getNewsListAsync(tag);

    RedditWrapper rw = new RedditWrapper();
    var tagNews2 = rw.getNewsListAsync(tag);

    TwitterWrapper tw = new TwitterWrapper();
    var tagNews3 = tw.getNewsListAsync(tag);

    Task.WaitAll(new Task[] { tagNews, tagNews2, tagNews3 });

    newsList.AddRange(tagNews.Result);
    newsList.AddRange(tagNews2.Result);
    newsList.AddRange(tagNews3.Result);

}

db.DropCollection("News");
db.CreateCollection("News");
var newsCollection = db.GetCollection<News>("News");
newsCollection.InsertManyAsync(newsList);
```

13. **Omówienie implementacji**

1. Scraper

Kod implementacji modułu scrapera wraz z komentarzem został podany w specyfikacji algorytmów, ponieważ zawiera on najważniejsze realizowane przez nas funkcje z opisem.

2. Aplikacja kliencka

Implementacja aplikacji na system iOS opiera się głównie na pobieraniu danych z serwera z pomocą Web API oraz prezentowaniu ich użytkownikowi w odpowiedniej formie. Kod aplikacji jest podzielony na kilka modułów:

- Widoki - wyświetlanie danych
- REST Service - komunikacja z Web API, wysyłanie zapytań do serwera
- Interaktory - odpowiednie akcje widoków wywołują akcje interaktora - zachodzi wówczas interakcja z modułem REST Service, który komunikuje się z serwerem oraz pobiera lub wysyła dane
- Prezentery - obrabianie danych otrzymanych z REST Service oraz przekazywanie ich do widoków, obsługa błędów zwracanych przez serwer oraz odpowiednie ich wyświetlenie w widokach

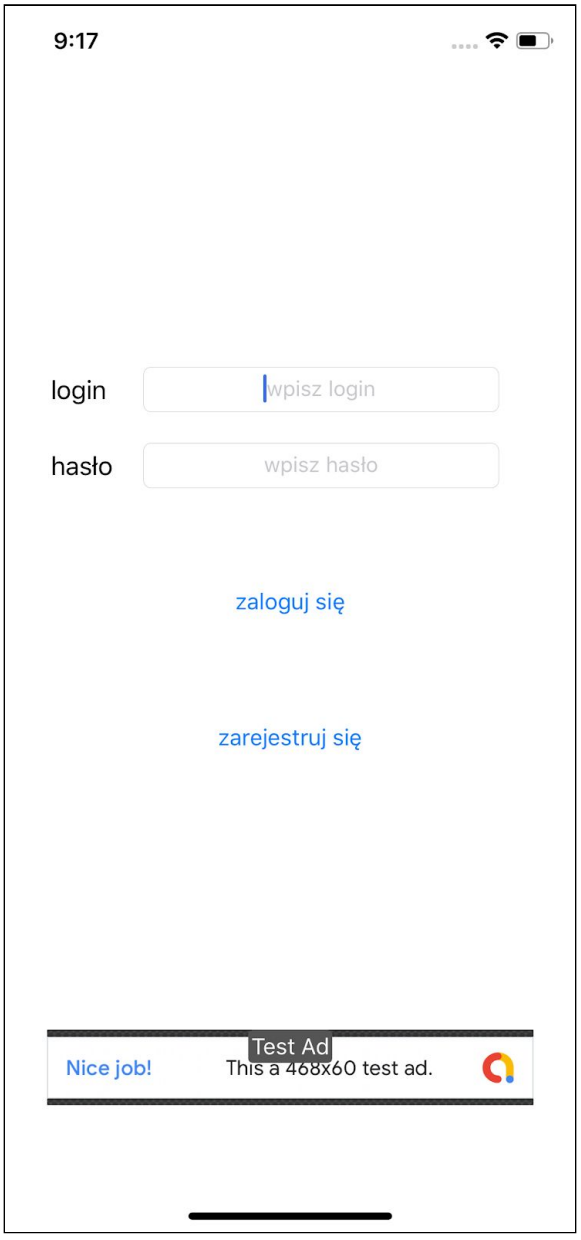
Aplikacja na system Android ma w prosty i szybki sposób pobierać dane z serwera i odpowiednio je reprezentować. W tym celu do implementacji wykorzystana została architektura Jetpack, dzięki której struktura wygląda w następujący sposób:

- Aplikacja ma jedną aktywność, która opiera się na działaniu navigation graph - pozwalającego na przechodzenie w aplikacji pomiędzy fragmentami, które są reprezentacjami widoków,
- Widoki - implementowane przy użyciu fragmentów,
- Service - część do komunikacji z Web API.

Wizualny efekt implementacji został umieszczony w kolejnym punkcie w postaci screenów.

14. **Widoki aplikacji oraz instrukcja użytkowania**

Pierwszym widokiem aplikacji jest widok logowania - z tego miejsca użytkownik może zalogować się na swoje konto w systemie, lub przejść do widoku rejestracji i utworzyć nowe konto



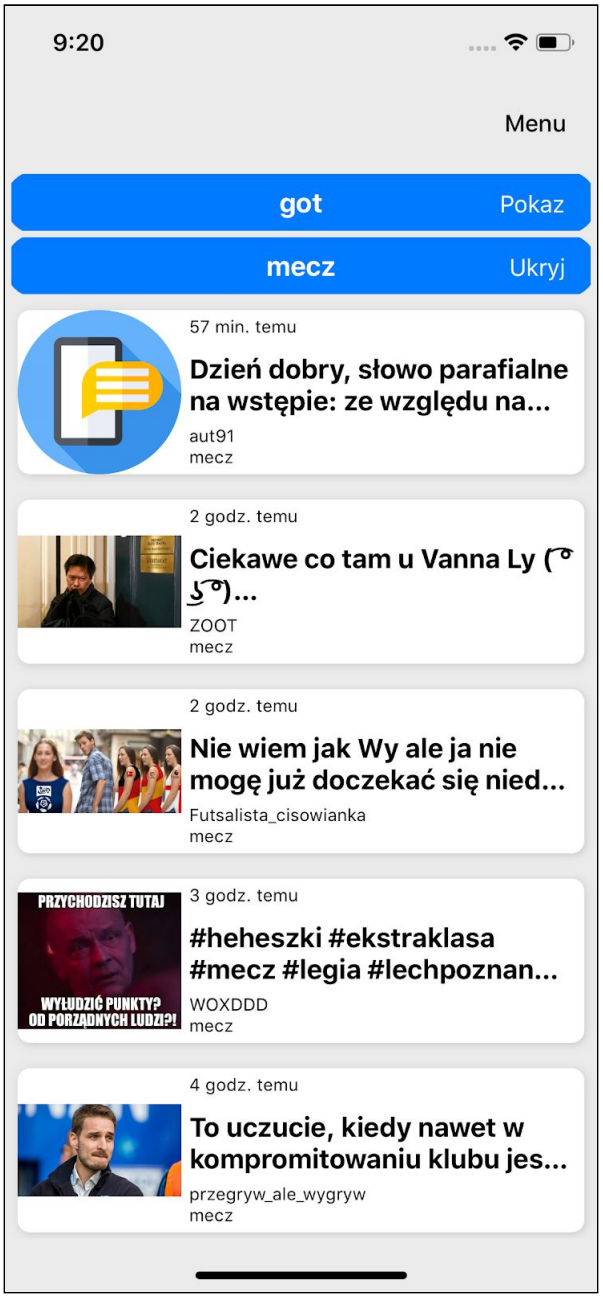
widok logowania

Po procesie logowania, aplikacja przenosi nas do widoku newsów. W widoku tym widoczne są sekcje, które reprezentują tagi subskrybowane przez użytkownika. Sekcje przedstawione są w formie czytelnej tabeli, dzięki czemu użytkownik może w prosty sposób wybrać, która sekcja (sekcje) w danym momencie go interesuje.

widok newsów

Każda z sekcji ma opcję rozwinięcia. Rozwinięcie sekcji skutkuje wyświetleniem listy newsów oznaczonych tagiem danej sekcji. News przedstawiany jest na liście jako widok z miniaturką, tytułem, opisem, czasem dodania, nazwą użytkownika, który dodał news oraz tagiem jakiego dotyczy. Jeśli dany news nie ma przyporządkowanego odpowiedniego zdjęcia, wyświetlane jest zdjęcie domyślne.

W widoku newsów dostępny jest również przycisk “Odśwież”, którego naciśnięcie skutkuje natychmiastowym uruchomieniem procesu pobierania newsów.

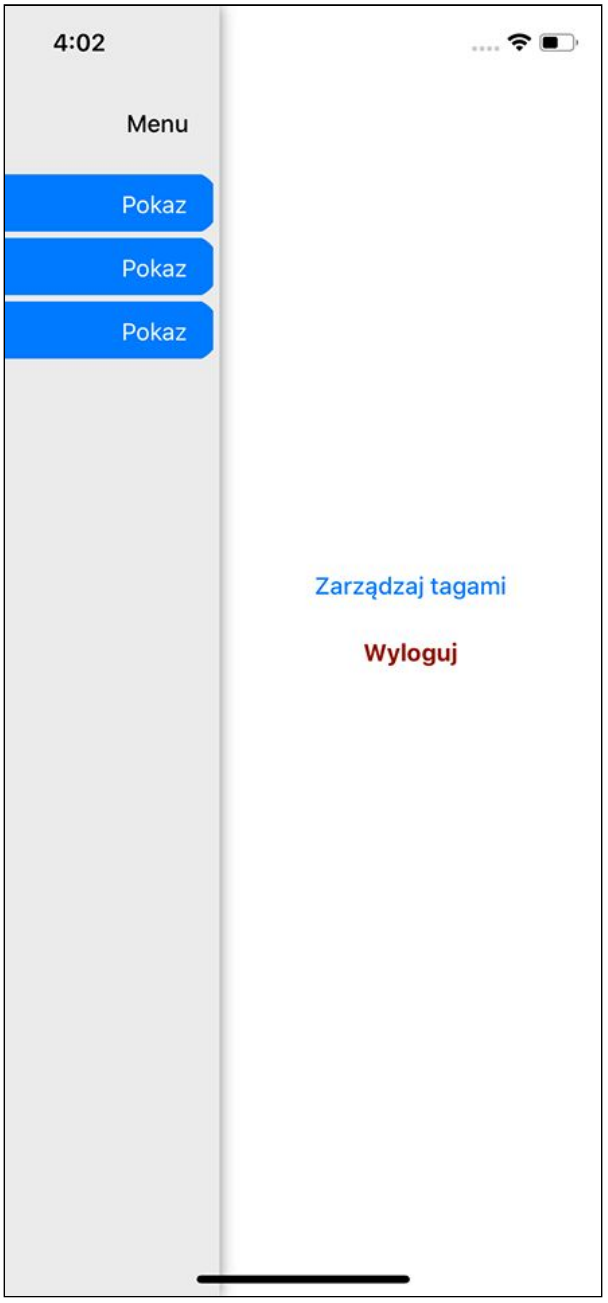


widok newsów z rozwiniętą sekcją

Użytkownik posiada możliwość wyświetlenia szczegółów interesującego go newsa. Stuknięcie w widok danego newsa, skutkuje wyświetleniem widoku szczegółów newsa. W widoku tym użytkownik może zobaczyć zdjęcie w większym formacie, pełny opis newsa oraz link do strony, z której został pobrany. Użytkownik klikając w link jest przekierowywany do danej strony w przeglądarce mobilnej.

widok szczegółów newsa

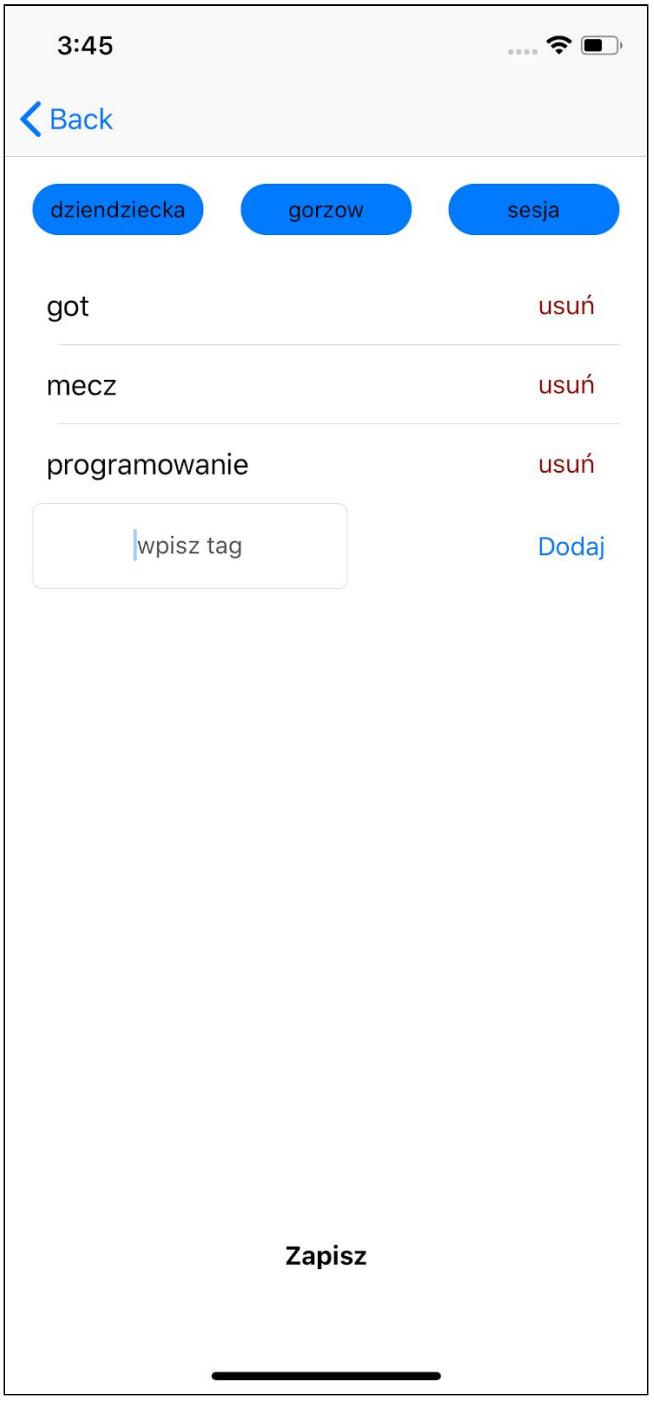
Z poziomu głównego widoku aplikacji - widoku listy newsów, użytkownik ma możliwość przejścia do widoku zarządzania subskrybowanymi tagami. W tym celu użytkownik musi stuknąć przycisk “Menu” lub przesunąć palcem z prawej krawędzi do lewej. Wysunie się wówczas boczny panel z przyciskami umożliwiającymi przejście do widoku zarządzania tagami lub wylogowanie z aplikacji.



widok menu

W widoku zarządzania subskrybowanymi tagami użytkownik ma możliwość dodawania oraz usuwania tagów, które chce obserwować. Dodatkowo widoczny jest panel, który prezentuje użytkownikowi 3 najpopularniejsze tagi w ostatnich trzech dniach. Użytkownik może wybrać dany popularny tag, co poskutkuje dodaniem go do listy obserwowanych tagów.

Po procesie dodawania oraz usuwania tagów użytkownik musi zapisać zmiany, używając przycisku “Zapisz”. Po operacji zapisu zostanie przekierowany z powrotem do widoku listy newsów.



widok zarządzania subskrybowanymi tagami

15. **Ewolucja idei aplikacji oraz napotkane problemy.**

Proces tworzenia aplikacji pozwala na konfrontację surowego projektowania jej funkcjonalności wobec możliwości danego języka programowania i umiejętności programisty. To bardzo dobry sposób na porównanie swojej wizji, która często wychodzi

poza racjonalne myślenie, z twardym światem informatycznej rzeczywistości. Nasze doświadczenia związane z implementacją NewsScrapera były zarówno twardym spotkaniem z rzeczywistością jak i ewolucja pierwotnej myśli.

Początkowo zakładaliśmy nie wychodzić poza ramę jednolitego zamkniętego systemy, pozwalajacego na komunikację miedzy serwerem a klientem bezpośrednio za pomocą TCP/IP. Gdzie jedynie serwer jest otwartą bramą na możliwości Internetu. Łatwość implementacji i wysoki poziom rozwoju technologii Web API wraz z językiem C# pozwolił nam przełamać ta myślową barierę. Dzięki czemu wykorzystaliśmy podejście REST aby lepiej i bardziej nowocześnie połączyć serwer z aplikacją klienta oraz komunikować się z bazą danych.

Innym odstępstwem od pierwotnego założenia jest ograniczenie generyczności aplikacji, gdzie nasz system miał sam znajdować odpowiednie węzły HTML, bez ręcznego określania wstępnego formatu ich rozmieszczenia na stronie. Było to bardzo ciężkie do rozwiązania, a czasami niemożliwe ze względu na:

- często całkowicie odmienny rozkład węzłów na dwóch wizualnie podobnych stronach,
- wydarzenia o podobnych charakterze posiadają różną hierarchię w budowie strony np. post w odpowiedzi, jest czym innym niż zwykły post, również dotyczy to cytowania czy udostępniania postu.
- ciężka do zrozumienia hierarchia strony,
- etykiety tych samych obiektów posiadają różne wartości losowe - Facebook.

Postaraliśmy się rozwiązać ten problem w sposób programowy, zapewniając czystą klasę w raz z generycznym konstruktorem. Wystarczy wprowadzić do niej węzły HTML podstawowych komponentów strony, aby rozpocząć generowanie newsów tak jak ze stron już przez nas dodanych.

16. **Podsumowanie i możliwe kierunki rozwoju aplikacji**

Poziom zaangażowania członków zespołu.

Członkowie zespołu wywiązywali się ze swoich zadań w określonych harmonogramem ramach czasowych. Każdy wykonywał zadania odpowiednie co do umiejętności i znajomości konkretnych technologii, dzięku czemu ograniczyliśmy liczbę błędów i problemów związanych z brakiem doświadczenia. Przez ciągłą komunikację, wzajemne raportowanie o ukończonych pracach mogliśmy stale kontrolować zaangażowanie oraz poziom realizacji konkretnych modułów.

Harmonogram zadań.

Ukończenie konkretnych zadań następowało zawsze przed określoną datą końcową. Dzięki trafnemu podziałowi zadań, ważne elementy związane z kompatybilnością modułów były realizowane w tym samym czasie. Co pozwalało nam na testowanie tworzonych rozwiązań bezpośrednio w trakcie ich implementacji.

Metodyka projektowania i implementacji.

Do projektowania wykorzystujemy metodyki zwinnej, gdzie każdy etap implementacji został obarczony datą. Każdy termin oznacza dla koniec pewnego okresu, po którym jesteśmy w stanie stwierdzić na jakim etapie budowania aplikacji się znajdujemy. Takie podejście daje nam możliwość oddania konkretnej, a w szczególności działającej funkcjonalności. Dzięki takim rozwiązaniom łatwiej jest podzielić pracę dla wszystkich osób, tak aby każdy miał określone zadania, a postępy były widoczne.

Dobór narzędzi i środowisk programistycznych.

W doborze stosowanych przez nas narzędzi i środowisk kierowaliśmy się głównie trendami wyznaczanymi przez społeczność pracującą w branży związanej z aplikacjami mobilnymi, a także serwerowymi. Stosując najnowsze i najpopularniejsze rozwiązania możemy podłapać trochę doświadczenia, które na pewno zaowocuje w przyszłości. Pozwala to także w łatwy i przyjemny sposób zdobyć najpotrzebniejszą wiedzę na temat wybranych narzędzi.

Możliwości rozwijania w przyszłości.

W przyszłości na pewno można pomyśleć o rozszerzeniu zakresu przeszukiwanych stron. W tym momencie są to trzy serwisy, tj. Wykop, Reddit, oraz Twitter.

Kolejnym aspektem, który mógłby zostać udoskonalony jest wyświetlanie po stronie klienta newsów w kontekście danych tagów, tylko z wybranych stron, przykładowo: Użytkownik chce wyświetlić tag "wybory" pobranych tylko z serwisu Wykop.

Udoskonaleniu można w przyszłości poddać również proces natychmiastowego odświeżenia newsów przez użytkownika. Po naciśnięciu przycisku "Odśwież" newsy pobierane są w czasie rzeczywistym, przez co użytkownik musi oczekiwać na zakończenie procesu. Potencjalnym rozwiązaniem tego problemu jest wprowadzenie pamięci cache w aplikacji lub bazy danych przechowanej w pamięci urządzenia mobilnego. W takim podejściu następuje pobieranie z serwera tylko niezbędnych zasobów.

Ostatnim pomysłem jest udoskonalenie przesyłania najpopularniejszych tagów, biorąc pod uwagę zarówno wiek użytkownika jak i subskrybowanych przez niego tagów.

17. **Spis literatury**

1) https://docs.microsoft.com/pl-pl/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api
2) https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/
3) https://cezarywalenciuk.pl/blog/programing/post/aysnc-await-wpf-ui
4) https://books.goalkicker.com/CSharpBook/
5) https://stackoverflow.com/questions/40281050/jwt-authentication-for-asp-net-web-api
6) https://www.raywenderlich.com/6729-android-jetpack-architecture-components-getting-started
7) https://developer.android.com/docs
8) https://swift.org/documentation/
9) https://github.com/Alamofire/Alamofire
10) https://kotlinlang.org/docs/reference/coroutines-overview.html
11) ASP.Net Core Web API - tutorial

12) Android Jetpack Architecture Components: Getting Started

13) Hello Analytics API: Python quickstart for service accounts

14) https://developer.android.com/training/constraint-layout

15) Quickstart: Using Client Libraries

16) https://github.com/ReactiveX/RxJava