



# web.scrapers

## Dokumentacja projektu

Mateusz Kaczmarek  
Szymon Kołodziejczak  
Jakub Krawczyk

## Contents

1.	Web scraper .....	2
2.	Podział prac pomiędzy członków zespołu .....	2
3.	Funkcjonalności oferowane przez aplikację .....	3
4.	Wybrane technologie .....	3
5.	Architektura rozwiązania.....	4
6.	Szablony.....	4
6.1.1.	Część URL.....	5
6.1.2.	Product .....	7
6.1.3.	Część odpowiedzialna za paginację .....	8
7.	Specyfikacja najważniejszych algorytmów .....	10
7.1.	Wykorzystanie shared preferences .....	10
7.2.	Paginacja.....	10
8.	Omówienie implementacji .....	11
8.1.	Implementacja API do obsługi pliku Shared Preferences.....	11
8.2.	Parsowanie produktów .....	13
9.	Prezentacja wyników.....	15
10.	Interesujące problemy .....	15
10.1.	Paginacja.....	15
10.2.	Struktura HTML .....	16
10.3.	Interpolacja stringu .....	17
11.	Instrukcja użytkownika.....	18
11.1.	Start aplikacji .....	19
11.2.	Główny widok aplikacji .....	20
11.3.	Aktywność pobierania .....	21
11.4.	Wyniki .....	24
12.	Testy .....	27
13.	Przyszłość aplikacji.....	27

## 1. Web scraper

Web scraping jest działaniem polegającym na ekstrakcji informacji ze strony internetowej.

[Website Parse Template](#) to szablon do parsowania stron internetowych, który jest plikiem xml, a dostęp do szczególnych elementów strony odbywa się za pośrednictwem [XPath](#).

Nasz zespół zdecydował się na nieco inne rozwiązanie: szablony [JSON](#) oraz [CSS selectors](#). Firebase [NoSQL](#) pracuje na dokumentach JSON co zwiększa wydajność naszej pracy.

Nasz projekt dotyczy parsowania księgarni internetowych. Jest on dedykowany dla osób chcąc kupić książkę przez internet nie przeglądając wszystkich księgarni sekwencyjnie.

Dlaczego wybraliśmy ten temat? Jest on związany z naszymi zainteresowaniami – technologie webowe. Implementując ten projekt sięgamy po rozwiązania, których znajomość jest wartościowa na rynku pracy oraz kształcąca. Dzięki temu, że zdecydowaliśmy się na mobilną aplikację projekt Web.scrafer mógł zostać wykorzystany do zaliczenia innego przedmiotu.

## 2. Podział prac pomiędzy członków zespołu

- Szymon Kołodziejczak
  - baza danych Firebase przechowująca szablony,
  - projekt szablonów, zarządzanie nimi, pobranie danych,
  - lokalne przechowywanie zapytań użytkownika
- Mateusz Kaczmarek
  - parsowanie stron na podstawie szablonu,
  - przetwarzanie wyników,
  - personalizacja reklam
- Jakub Krawczyk
  - wyświetlenie wyników w aplikacji,
  - widoki aktywności

### 3. Funkcjonalności oferowane przez aplikację

Aplikacja oferuje trzy główne funkcjonalności:

- Aplikacja uwzględnia potrzebę odwiedzenia kolejnych stron wyników z pojedynczego źródła oraz ich odpowiedniego sortowania.
- Jest możliwa rozszerzalność o nowe źródła danych – poprzez implementację parsera interpretującego strony poprzez wcześniej dostarczone szablony w formacie JSON.
- Jednolity sposób wyświetlania wyników z informacją o źródle danych.

### 4. Wybrane technologie

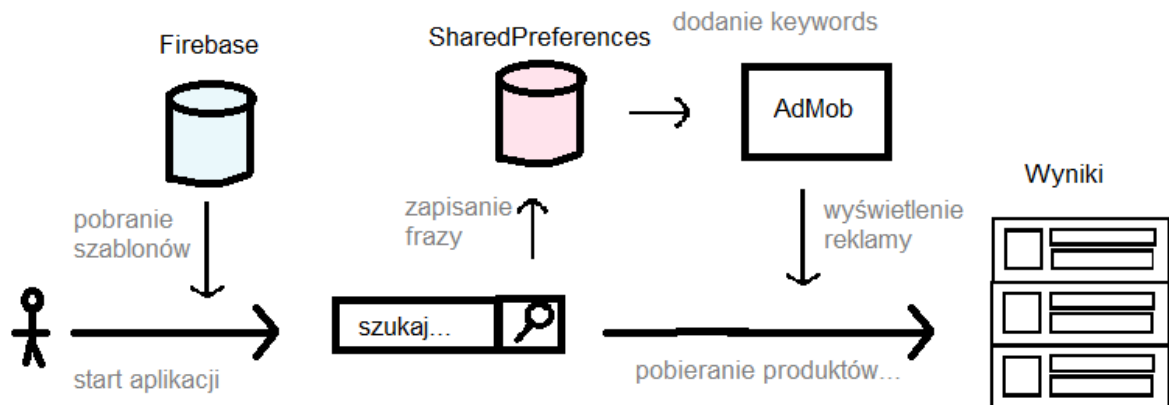
Aplikacja mobilna – zdecydowaliśmy się na napisanie aplikacji dla systemu Android. Dlaczego? Ponieważ nikt z naszej trójki nie posiada iPhone. Wybór padł zatem na środowisko Android Studio – jest to oficjalnie wspierane oprogramowanie przez Google.

Bibliotekę Jsoup wykorzystujemy do parsowania stron HTML. Jest darmowa, działa na systemie Android oraz obsługuje CSS selektory. W dodatku jest popularna, a co za tym idzie w razie problemów można łatwo znaleźć rozwiązanie problemu.

Firebase – baza danych od Google zaprojektowana z myślą o małych aplikacjach mobilnych. Android Studio ma od razu zainstalowane wsparcie jej obsługi. Jest bardzo szybka i wygodna.

AdMob – w celu monetyzacji aplikacji najlepszym i najbardziej popularnym rozwiązaniem są reklamy. AdMob to również produkt od Google, wszystkie elementy aplikacji są zatem porządnie zintegrowane bez większego wysiłku.

## 5. Architektura rozwiązania



*Zrzut ekranu 1 Rysunek poglądowy aplikacji wykonanej przez zespół projektowy*

Rysunek jest bardzo uproszczony, jednak oddaje w stu procentach idee samej aplikacji.

Użytkownik wpisując szukaną frazę zapisuje ją do SharedPreferences. Na podstawie zawartości SharedPreferences personalizowane są reklamy, które pojawiają się przed samymi wynikami.

Wszystko dzieje się w telefonie, aplikacji klienta. Nie ma tutaj wystawionego żadnego wewnętrznego API, które byłoby odpowiedzialne za pobranie danych.

## 6. Szablony

Przedstawiając funkcjonalność aplikacji w punkcie trzecim wymieniony został fakt rozszerzalności o nowe źródła danych. Web.Scraper ma zaimplementowany parser, który interpretuje strony poprzez wcześniej dostarczone szablony w formacie JSON.

Projektując go wzorowaliśmy się na 'Website Parse Template'. Odrzuciliśmy XPath na rzecz CSS selektorów. Podjęliśmy taką decyzję, ponieważ CSS selektory pozwalają nam na większą elastyczność; sama budowa wyrażeń jest mniej restrykcyjna, ścieżki są bardziej przejrzyste i intuicyjne. Kolejnym powodem jest fakt, że sam CSS jest mi znany więc w tym temacie odnalazłem się bez większego problemu. Nie ukrywam, że wybór biblioteki również poniekąd był podyktowany wsparciem selektorów.

Na końcu tego rozdziału zostanie przedstawiony przykładowy szablon zaprojektowany dla jednej z księgarni.

## 6.1. Budowa szablonu

Szablon składa się z trzech głównych części:

- URL,
- Product,
- Pagination.

Część nazwana 'URL' służy do budowania łącza do strony wynikowej, natomiast część 'product' do przeparsowania strony w celu wyekstrahowania interesujących nas informacji.

Ostatni fragment – pagination – odpowiada za funkcjonalność przeszukiwania kolejnych stron wyników. W dalszej części dokumentacji wszystkie te części zostaną dokładnie omówione wraz z przykładami.

### 6.1.1. Część URL

Część URL składa się z następujących pól:

- Pole reprezentujące protokół (protocol),
- Domenę (domain),
- Ścieżkę (path),
- Zapytanie (query),
- Wzór (pattern)

Wszystkie pola są w teorii optymalne – mogą zostać pominięte. Wszystkie mają na celu przysłużyć się do zbudowania linku strony wynikowej.

- Protokół – zazwyczaj 'http' bądź 'https'.
- Domena – składa się z dwóch części – nazwy głównej oraz końcówki – rozszerzenia. Przykładem może być 'wikipedia.org'.
- Ścieżka – w praktyce optymalna. Zazwyczaj między domeną a samym zapytaniem. Może jednak zostać użyte w dowolnym miejscu.
- Zapytanie – do tego pola 'doklejana' jest wyszukiwana fraza. Pole jest niezbędne.

Brak tutaj pola odpowiedzialnego za np. aktualnie wyświetlaną stronę. Jest to celowe działanie – ta część szablonu odpowiada jedynie za generowanie pierwszej strony wynikowej. Więcej o tym później.

Specjalne pole `pattern` jest jakby przepisem jak ma wyglądać łącze: jej zawartość to łańcuch znaków w skład którego wchodzi pozostałe pola (część z nich można pominąć). Przykładowo:

```
pattern: "protocol domain query"
```

Powyższe przykład należy interpretować następująco: link składa się z trzech pól – protokołu, domeny oraz zapytania. Warto zauważyć, że pominięty został parametr 'ścieżka' – nie zostanie on po prostu uwzględniony.

Rozumiejąc jak działa `pattern` można zauważyć, że wybrany sposób budowania łącza nie narzuca i nie ogranicza: można pominąć pole protokołu i przypisać je do domeny, a samo pole protokołu wykorzystać do jakiejś innej części łącza. Oczywiście takie działanie nie jest polecane, jednak warto o takiej możliwości wspomnieć. Pole 'ścieżka' służy właśnie w podobnym celu: jest to takie opcjonalne koło ratunkowe gdyby zabrakło nam pól. Wyjątkiem jest pole 'query' do którego programowo dodawana jest wyszukiwana fraza. Z tym polem nie należy kombinować w żaden sposób.



*Zrzut ekranu 2 Fragment szablonu przygotowany dla księgarni pwn.pl. Można zauważyć, że pole 'ścieżka' zostało pominięte. Dokładniej mówiąc - pole nie jest puste tylko nie istnieje. W polu 'query' jest zawarta kategoria przeszukiwań tak, aby wykluczyć niektóre produkty, np. ebooki czy zabawki dla dzieci.*

Szablon w akcji:

- protocol: 'https://'
- domain: 'ksiegarnia.pwn.pl'
- query: '/szukaj?fc\_category\_id=2195456&q='

Po wpisaniu: 'wyszukiwana fraza' nasze zapytanie zostanie zakodowane do postaci: 'wyszukiwana+fraza', a wygenerowane łącze przyjmie postać:

[https://ksiegarnia.pwn.pl/szukaj?fc\\_category\\_id=2195456&q=wyszukiwana+fraza](https://ksiegarnia.pwn.pl/szukaj?fc_category_id=2195456&q=wyszukiwana+fraza)

### 6.1.2. Product

Część, dzięki której możemy wyekstrahować niezbędne informacje do prezentacji produktu w naszej aplikacji. Przeszukujemy księgarnie internetowe w celu znalezienia jak najtańszej książki. Informacje, które potrzebujemy:

- tytuł książki,
- autor książki,
- cena,
- zdjęcie produktu,
- link do księgarni i produktu
- element

Każde pole jest reprezentowane w bazie danych jako para klucz – wartość. Kluczami są wyżej wymienione rzeczy a wartościami – selektory do elementów, które zawierają pożądane informacje.

Tak więc – aby uzyskać łącze do księgarni z danym produktem nie pobieramy za pomocą selektorów samego łącza a element, który go zawiera. W tym przypadku będzie to element 'a', z którego programistycznie zostaje pobrany atrybut 'href' (hypertext reference).

Podobnie ze zdjęciem – programistycznie pobierany zostanie atrybut 'src' (source) – tutaj – warto dodać, że musi zostać pobrany bezwzględny adres – mimo, że w samym pliku HTML widoczne są ścieżki względne – na szczęście biblioteka Jsoup robi takie cuda.

Cena jest dodatkowo obsługiwana: pobierane zostają jedynie wartości liczbowe – odbywa się to przy pomocy prostego wyrażenia regularnego. Potem programistycznie dodawane są napisy 'zł'. Jest to odpowiedź na fakt, iż z części stron pobierane były kwoty wraz z napisem a z innych same liczby.



Ostatnim omawianym polem w tej części jest 'Element'. Jedno z ważniejszych pól w szablonie. Zawiera CSS selektor do kontenera, który zawiera wszystkie informacje o produkcie. Tak więc tytuł, autor, cena, link, zdjęcie zawarte są w tym kontenerze, ich wartości (selektory) są podawane względem 'Element' (tego kontenera). Chcąc podać ścieżkę bezwzględną do danej informacji np. ceny należało by najpierw podać ścieżkę do elementu a potem dokleić ścieżkę ceny.

Ilość węzłów pobranych za pomocą selektora Element jest równa ilości produktów na stronie. Poniżej fragment przykładowego wpisu w bazie danych:

#### product

```
— author: "div.product-main > div.product-main-top > div.ř
— element: "div.product-containe
— link: "div.product-main > div.product-main-top > div.ř
— photoURL: "div.product-image > a > img[sr
— price: "div.product-main > div.product-main-bottom > sp
— title: "div.product-main > div.product-main-top > div.ř
```

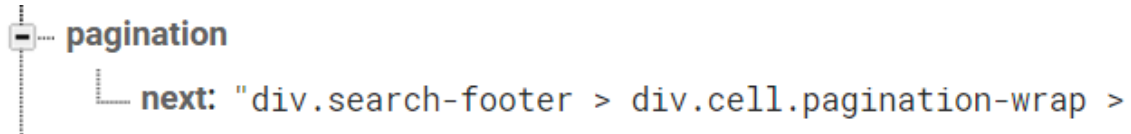
*Zrzut ekranu 3 Fragment szablonu przygotowanego dla księgarni taniaksiazka.pl. Wszystkie pola w tej części są niezbędne w celu kompletnej prezentacji produktu. Jest to jedno z założeń aplikacji, aby każdy produkt był reprezentowany w ten sam sposób. Wartości wszystkich pól są selektorami do elementów zawierających pożądaną informację*

#### 6.1.3. Część odpowiedzialna za paginację

Jak już wcześniej zostało wspomniane część URL odpowiada za budowanie tylko pierwszej strony wynikowej. Kolejne strony, jeśli istnieją są uzyskiwane przy pomocy tej części szablonu. Ta część zawiera jedynie jeden element – selektor do elementu odpowiedzialnego za przejście do następnej strony - odnośnika. Zwykle surfując w Internecie możemy go zauważyć na samym dole stron wynikowych.

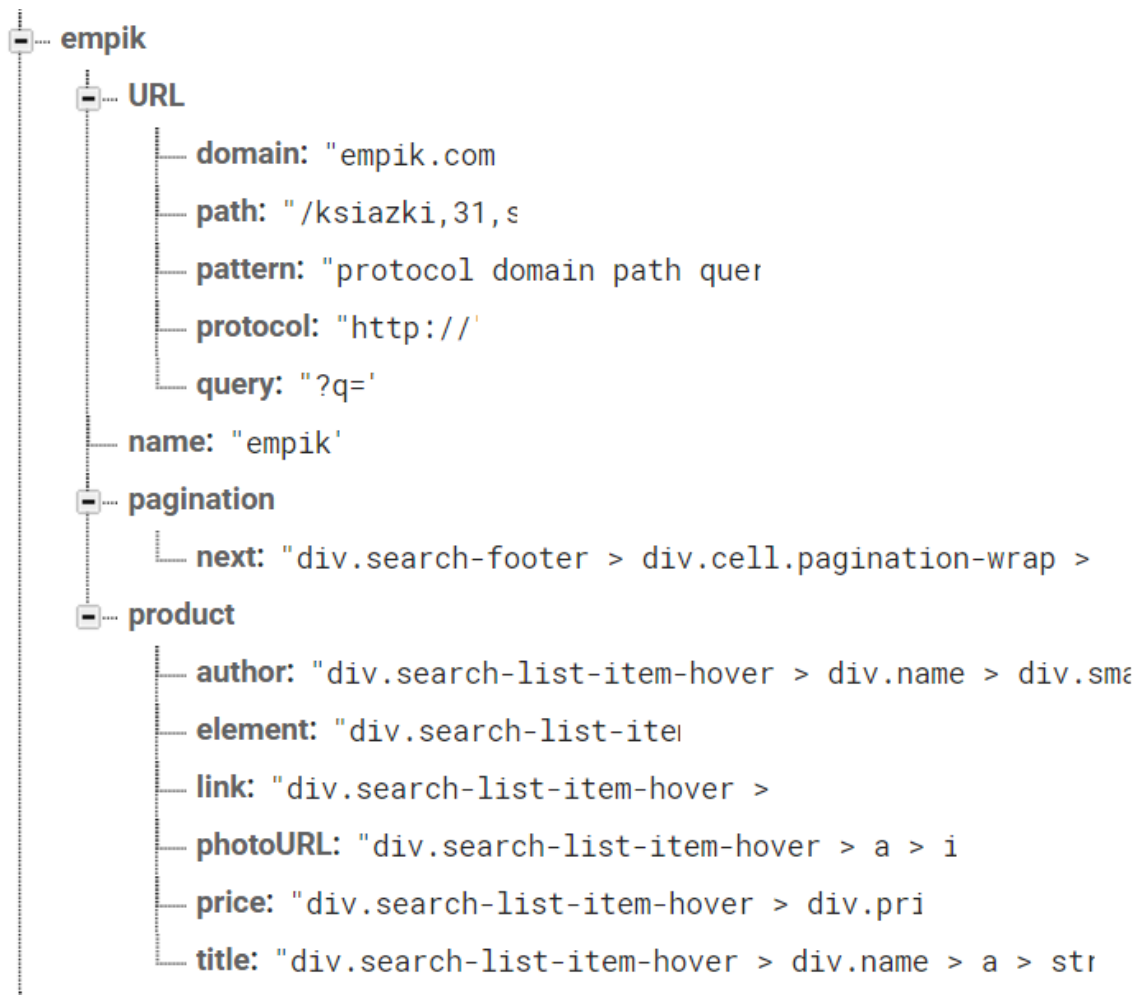
Warunkiem przejścia do następnej strony jest właśnie obecność takiego odnośnika. Ale to nie wszystko. Link do następnej strony musi być różny niż obecna lokalizacja. Ten drugi warunek został wprowadzony w późniejszym

czasie rozwoju aplikacji i sens jego wystąpienia jest opisany w rozdziale opisującym ciekawe problemy.



Zrzut ekranu 4 Fragment szablonu przygotowanego dla księgarni empik.pl. 'Next' zawiera selektor do elementu zawierającego odnośnik do następnej strony. Pierwotnie część paginacja zawierała więcej metadanych jednak na przestrzeni czasu dodatkowe informacje okazały się zbędne.

Na koniec rozdziału, zgodnie z obietnicą zostanie przedstawiony przykładowy, cały szablon:



Zrzut ekranu 5 Szablon przygotowany dla księgarni empik.pl. W powyższych tekstach został pominięty jeden element szablonu – jest to nazwa księgarni (name). Dla całej aplikacji zostały dostarczone 4 szablony.

## 7. Specyfikacja najważniejszych algorytmów

### 7.1. Wykorzystanie shared preferences

Wejście: zawartość kontrolki odpowiedzialnej za wpisanie tytułu książki.

Shared preferences pozwala na przechowywanie relatywnie małych kolekcji par klucz-wartość. W naszym przypadku wykorzystujemy tylko jeden klucz, a wartością jest tablica.

Żeby dodać do klucza wartość należy najpierw pobrać wartość, przekonwertować ją na kolekcję, dodać element, a następnie z powrotem zamienić kolekcję na wartość łańcuchową i zapisać w shared preferences:

```
wartość jako łańcuch  
-> zamiana łańcuchu w kolekcję  
-> dodanie elementu do kolekcji  
-> wartość jako łańcuch  
-> zapis.
```

Wyjście:

Do operacji na Shared Preferences zostało napisane specjalne API, które umożliwia jedynie dwie metody: dodawanie wartości i ich pobranie w postaci kolekcji. Przechowywane jest jedynie 10 szukanych fraz (więcej w implementacji).

### 7.2. Paginacja

Paginacja strony opiera się na budowaniu URL, na podstawie szablonu. Jednak ważne jest, żeby wiedzieć kiedy przestać wczytywać kolejne strony - wyjście poza indeks może skończyć się nieskończoną pętlą - niektóre strony po wyjściu poza zakres wyników wyświetlają pierwszą stronę. Są też witryny, które po przejściu do ostatniej strony wciąż mają aktywny odnośnik do następnej.

Trzeba to uwzględnić.

Warunkiem zakończenia paginacji jest wspomniany odnośnik - jednak z zastrzeżeniem, że link znajdujący się w atrybucie href jest inny niż obecny.

Pseudokod wygląda następująco:

```
nastepnaStrona = pobierzElement;  
while(nastepnaStrona != null & nastepnaStrona.href !=  
obecnaStrona.url){  
    //pobranie danych i przekazanie ich dalej...  
}
```

Wejściem jest tutaj również szukana fraza - bierze ona udział w budowaniu hiperłączy do internetowych księgarni. Wyjściem są elementy drzewa DOM, pobrane za pomocą CSS selektorów.

## 8. Omówienie implementacji

### 8.1. Implementacja API do obsługi pliku Shared Preferences

API udostępnia dwie metody: get oraz add.

```
public class QueryHistory {  
  
    private final static String tag = "queryHistory";  
    private final static Gson gson = new Gson();  
  
    private static void set(Context ctx, ArrayList<String> queryHistory) {...}  
  
    public static ArrayList<String> get(Context ctx) {...}  
  
    public static void add(Context ctx, String query) {...}  
}
```

*Zrzut ekranu 6 Obsługa przechowywania wpisanych fraz*

Zmienna 'tag' służy do identyfikacji pliku SharedPreferences - może być ich kilka. Obiekt Gson jest obiektem biblioteki o tej samej nazwie - służy ona do zamieniania Json na obiekty w Javie i odwrotnie.

Funkcja set jest prywatna i jest ona wykorzystywana przy dodawaniu wartości.

```
48 public static void add(Context ctx, String query){
49
50     ArrayList<String> queries = get(ctx);
51     if(queries.size() == 10){
52         queries.remove(0);
53     }
54     queries.add(query);
55     set(ctx, queries);
56 }
57 }
```

*Zrzut ekranu 7 Funkcja statyczna add, która zapisuje do pliku przekazaną frazę*

Funkcja add korzysta z pozostałych - żeby dodać do kolekcji frazę należy najpierw pobrać obecną wartość (linia 50).

Przechowywanych jest ostatnie 10 fraz, więc jeśli jest już ich 10 to najstarsza fraza zostaje usunięta bezpowrotnie. Po dodaniu kolekcja jest zamieniana na string i zapisywana do Shared Preferences przez funkcję set:

```
private static void set(Context ctx, ArrayList<String> queryHistory){
    SharedPreferences sharedPref = ctx.getSharedPreferences(
        ctx.getString(R.string.shared_preferenced_key), Context.MODE_PRIVATE);

    SharedPreferences.Editor editor = sharedPref.edit();

    String newArray = gson.toJson(queryHistory);

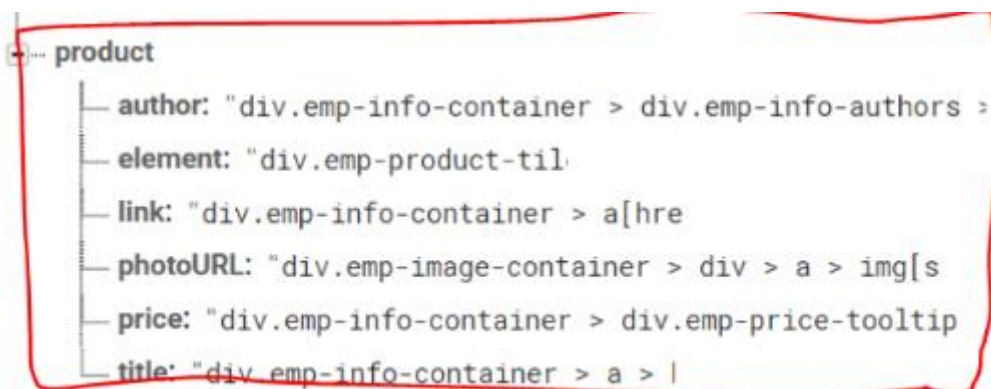
    editor.putString(tag, newArray);
    editor.commit();
}
```

*Zrzut ekranu 8 Prywatna funkcja set, która odpowiada za przypisywanie nowej wartości do klucza*

Ostatnie 3 linijki kodu w tej funkcji są “na temat”: tworzony jest string z `ArrayList<String>`, a następnie zapisywany jest jako wartość klucza ‘tag’. Commit oznacza zapisanie zmian w pliku.

## 8.2. Parsowanie produktów

Aby odnaleźć w wynikowym drzewie produkty wykorzystywane są szablony, które zawierają m. in. CSS selektory do pożądanых elementów. Poniżej zamieszczam fragment szablonu przygotowanego dla księgarni PWN:



*Zrzut ekranu 9 Fragment szablonu jednej z księgarni. Widoczny fragment dotyczy zawiera CSS selektory używane do parsowania produktów*

Funkcja `parseProducts` przyjmuje dokument HTML oraz szablon do obsługi tego dokumentu:

```

15 public class ParseHTML {
16
17     public static ArrayList<ProductView> parseProducts(Document doc, Template template){
18         ArrayList<ProductView> products = new ArrayList<>();
19
20         ArrayList<Element> productsEl = doc.select(template.product.element);
21
22
23         for(Element p : productsEl){
24
25             products.add(new ProductView(
26                 p.select(template.product.title).text(),
27                 p.select(template.product.author).text(),
28                 template.url.getBaseUrl() + p.select(template.product.link).attr("href"),
29                 p.select(template.product.price).text(),
30                 p.select(template.product.photoURL).attr("src")
31             ));
32         }
33         return products;
34     }
35 }

```

*Zrzut ekranu 10 Funkcja parseProducts z pobranego dokumentu HTML dokonuje ekstrakcji interesujących nas informacji*

W linii 20 pobierane są wszystkie produkty z drzewa DOM, przy pomocy zmiennej szablon.product.element. Funkcja doc.select(String) zwraca listę elementów, która w kolejnym kroku jest poddawana dalszej obróbce: z każdego produktu pobierane są konkretne, pożądane pola, które później są wykorzystane do prezentacji. Ze strony uzyskujemy listę już przeparsowanych produktów, z następującą strukturą:

```

public class ProductView implements Serializable{
    public String title;
    public String author;
    public String link;
    public String price;
    public String photoURL;
}

```

*Zrzut ekranu 11 Obiekt używany do prezentacji wyników. Implementowanie interfejsu jest konieczne w celu przekazywania obiektów pomiędzy aktywnościami. Mimo, że sama struktura jest podobna do tej 'szablonowej', na potrzeby widoku została stworzona osobna klasa.*

## 9. Prezentacja wyników

Wyniki w postaci pożądaných produktów są pogrupowane po źródłach. Każde ze źródeł ma generowany programistycznie kontener na produkty. W obrębie jednej księgarni książki są posortowane na podstawie ceny. Od najniższej do najwyższej.

Ustawień tych nie można zmienić, a wyświetlonych wyników nie można poddać dalszemu filtrowaniu.

Kolejność wyświetlanych źródeł jest zgodna z porządkiem liter w alfabecie.

## 10. Interesujące problemy

### 10.1. Paginacja

Aby zaprojektować uniwersalny szablon trzeba przewidzieć jak najwięcej możliwości, które mogą wystąpić podczas parsowania stron HTML. W przypadku paginacji natrafiliśmy na sytuację, która – moim zdaniem jest zwykłym niedopatrzeniem przez programistów księgarni czytam.pl. Udostępnia ona odnośnik do następnej strony, mimo, że znajdujemy się na ostatniej wynikowej stronie:



*Zrzut ekranu 12 Paginacja czytam.pl udostępnia odnośnik do następnej strony mimo iż jej nie ma - kliknięcie w odnośnik przekierowuje do strony 4*

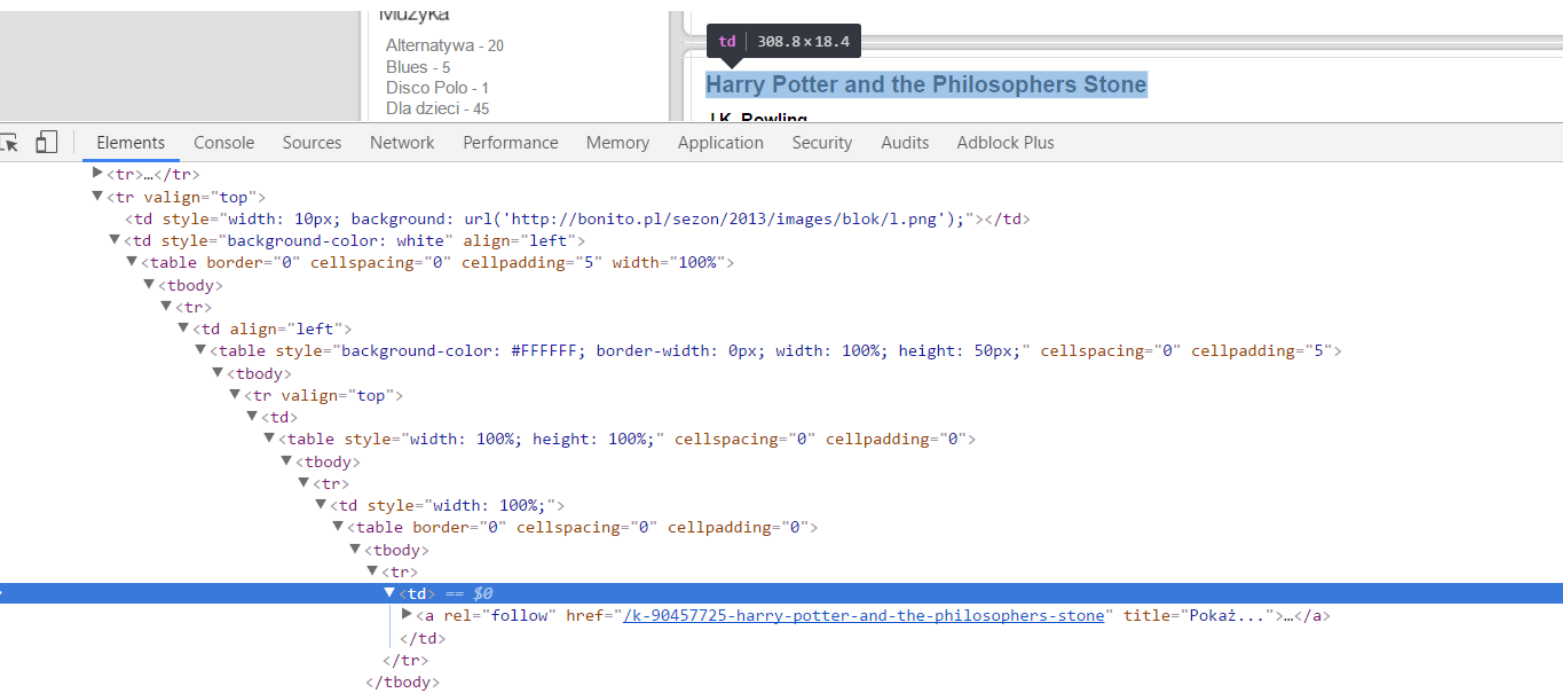
Warunkiem przerwania pobrania pierwotnie był brak odnośnika do następnej strony. Gdyby zignorować ten problem strony pobierałyby się w nieskończoność.

Rozwiązaniem tego problemu jest zwykle sprawdzenie czy odnośnik do następnej strony zawiera link inny niż aktualnie otwarta strona.



## 10.2. Struktura HTML

Księgarnia bonito.pl może pochwalić się strukturą pliku HTML godną dwóch dekad temu. Układ strony jest zaprojektowany przy użyciu tabel.



Zrzut ekranu 13 Fragment pliku HTML księgarni bonito.pl. Dzięki wcięciom widoczna jest hierarchia poszczególnych węzłów. Elementy pozbawione są atrybutów, które pozwoliłyby na ich łatwiejszą stylizację, czy samo zrozumienie zawartości pliku.

Oczywiście, dla chcącego nic trudnego i można by bez problemu dodać szablon dla tej strony. Jednak pierwszy lepszy przykład, np. selektor do tytułu książki wygląda w następujący sposób:

```
body > div:nth-child(4) > div:nth-child(1) > table:nth-
child(11) > tbody > tr:nth-child(2) > td:nth-child(2) >
table > tbody > tr > td > table > tbody > tr > td >
table > tbody > tr:nth-child(1) > td:nth-child(1) >
table > tbody > tr > td
```

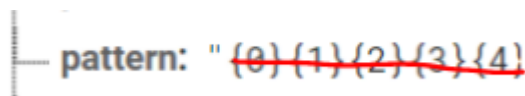
Z tego właśnie powodu ostatecznie zdecydowaliśmy się odrzucić księgarnie bonito.pl. Nie jesteśmy leniwi, jednak jakkolwiek błąd w szablonie byłby bardzo ciężki do wychwycenia.

### 10.3. Interpolacja stringu

Interpolacja stringów to w skrócie możliwość umieszczenia wartości zmiennych wewnątrz stringów bez zabawy w pilnowanie plusików, spacji czy cudzysłowu. Funkcjonalność ta jest dostępna w takich językach jak Javascript, C#, Pearl czy Python. Niestety, w Javie 1.8 nadal tego nie ma (narzekam, bo Groovy i Scala miały taką funkcjonalność na długo przed 1.8).

Narzekam też dlatego, ponieważ zależało mi, aby szablon był przejrzysty i zrozumiały dla każdego – pole URL.pattern nadawało się na idealnego kandydata do interpolacji.

Brak interpolacji – zostaje zabawa w String.format Jednak to rozwiązanie ma swoje wady. Przede wszystkim, parametry opisywalibyśmy za pomocą liczb, dlatego też tak na dobrą sprawę, moglibyśmy nie wiedzieć, co tak naprawdę w danym miejscu trzeba podstawić.



```
pattern: " {0} {1} {2} {3} {4} "
```

*Zrzut ekranu 14 Użycie String.Format w szablonie księgarni. Numery od 0 do 4 odpowiadały za sztywno zaprogramowane kolejno pola: protocol, domain, path, query i page (usunięte).*

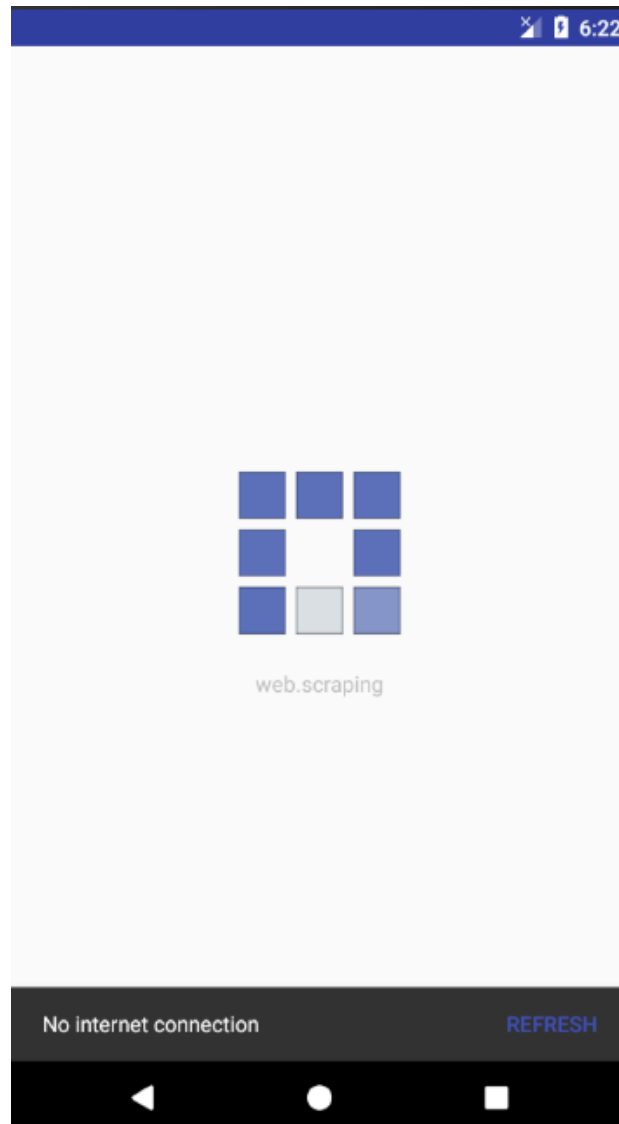
Kolejnym pomysłem, który rozwiązałby problem byłoby użycie odpowiednika funkcji eval z Javascriptu. W js eval Interpretuje i wykonuje kod zawarty w łańcuchu znaków. W Javie nie ma takiej funkcji jednak można coś podobnego uzyskać przy pomocy metod klasy ScriptEngineManager. Samo rozwiązanie jednak byłoby wyjątkowo brzydkie.

Rozwiązanie, które zostało przyjęte to wykorzystanie refleksji. Oczywiście refleksja nie rozwiązuje problemu braku interpolacji stringów, jednak w tym przypadku zdało to egzamin. Mechanizm refleksji pozwala m.in. wydobyć odpowiednie pole klasy na podstawie stringa. Bingo.



### 11.1. Start aplikacji

Widok startu to aktywność, która pojawia się po włączeniu aplikacji, jest ona wykorzystana do pobierania szablonów z bazy danych.



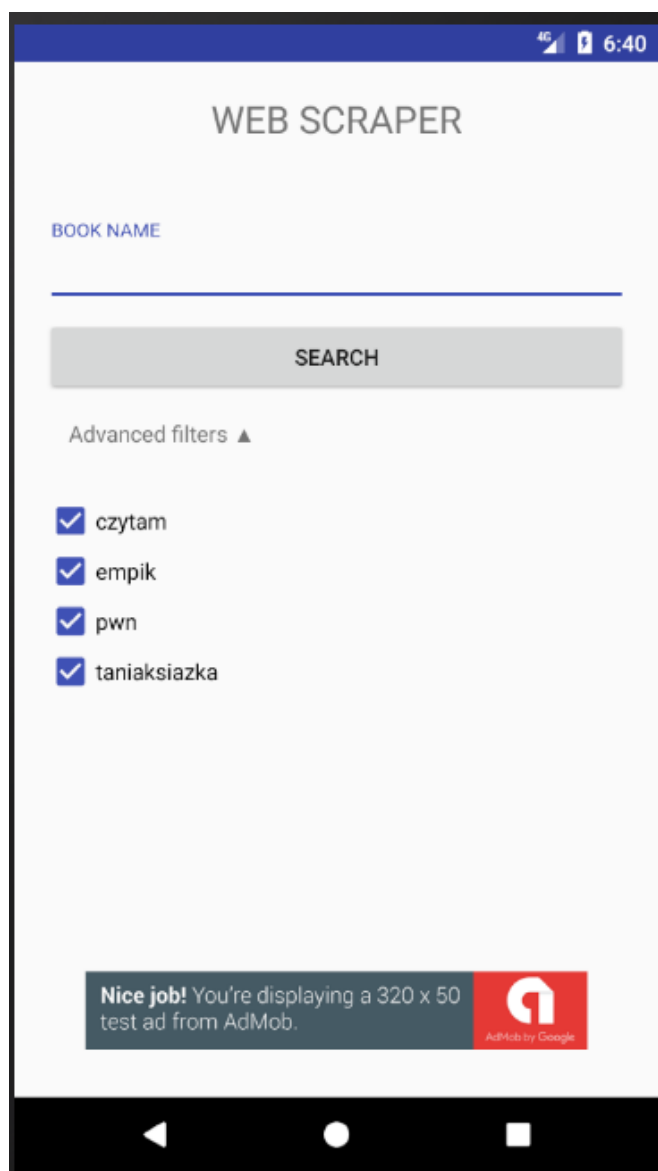
*Zrzut ekranu 17 Start aplikacji. Pobieranie szablonów jest 'uprzymilone' drobną animacją GIF.*

W przypadku braku połączenia z internetem przy pasku nawigacyjnym pojawi się o tym informacja. Kliknięcie w przycisk 'Refresh' spowoduje podjęcie kolejnej próby pobrania szablonów. Jeżeli szablony zostaną pobrane, nastąpi przejście do głównego widoku aplikacji.

## 11.2. Główny widok aplikacji

Główny widok aplikacji jest schludny i prosty: składa się z kontrolki, w którą wpisujemy szukaną frazę oraz przycisku, który odpowiada za szukanie.

Poniżej zrzut ekranu z głównego widoku po kliknięciu w napis 'Advanced filters' - wszystkie księgarnie są domyślnie brane pod uwagę:



*Zrzut ekranu 18 Główny ekran aplikacji po naciśnięciu w napis 'Advanced filters' widoczne są wszystkie obsługiwane księgarnie (ich obsługa została pobrana w ekranie startowym).*

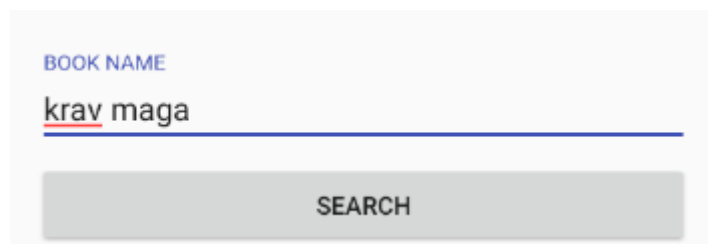
Pod przyciskiem dodatkowo znajduje się napis 'Advanced filters', który pozwala doprecyzować nasze zapytanie. Doprecyzować, konkretniej wybrać źródła z którego zostaną pobrane książki.

Każda linia - checkbox oraz nazwa księgarni - reprezentuje źródło (szablon na podstawie którego zostaną pobrane dane). Aby wykluczyć lub włączyć do zapytania którąś z księgarni wystarczy kliknąć w jej nazwę.

Na dole aktywności można zauważyć małą reklamę. Jest ona spersonalizowana pod użytkownika - aplikacja przechowuje 10 ostatnich wyszukiwanych fraz. Działa to w ten sposób, że do AdBuilder przekazujemy słowa kluczowe, które są uwzględniane podczas negocjowania reklamy.

Na powyższym obrazku jest reklama testowa, ponieważ zrzuty ekranu są wykonywane z symulatora androida. Zgodnie z regulaminem AdMob mógłbym wyświetlić nie testową reklamę z zastrzeżeniem klikania w nią - chodzi o to, żebym nie czerpał zysków poprzez klikanie w swoje reklamy. Karą jest zawieszenie konta. Reklamy to jedyny cel przechowywania fraz wyszukiwania, nie są one używane w innych częściach aplikacji.

W ramach poradnika wpiszę przykładową frazę:



*Zrzut ekranu 19 Wpisanie frazy w wyszukiwarkę. Przykładowa fraza to "krav maga".*

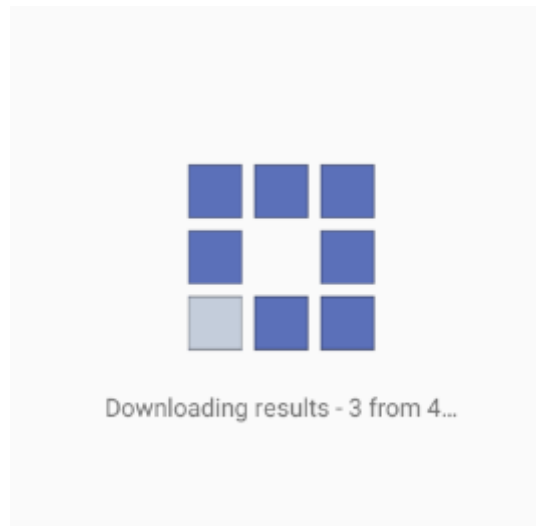
Nie zajmowałem się w ogóle 'Advanced filters' co oznacza że aplikacja przeszuka wszystkie źródła.

Po kliknięciu w przycisk 'Search' przechodzimy do widoku pobierania danych.

### 11.3. Aktywność pobierania

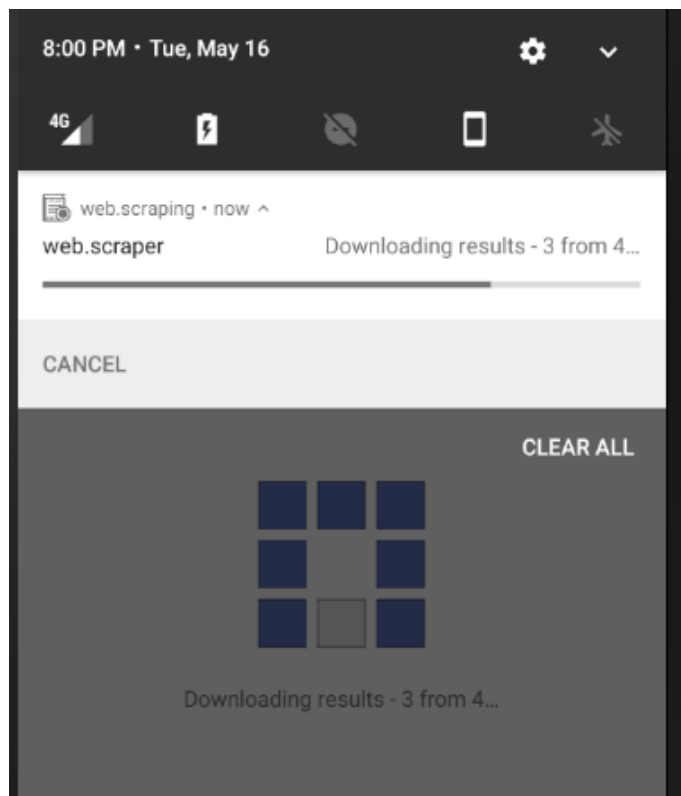
Aktywność pobierania- widok aktywny podczas pobierania danych. Pobieranie odbywa się w tle, a widok jest widoczny tylko po to, aby użytkownik wiedział co się z aplikacją dzieje.

Widok ten udostępnia progres - na poniższym zrzucie widać napis 'Downloading results - 3 from 4...' - można to przetłumaczyć jako 'Pobieranie wyników - 3 z 4'. Trzy z czterech oznacza, że aktualnie pobrane zostały dane 2 z 4 stron, a aktualnie pobierane są produkty na podstawie 3 szablonu.



*Zrzut ekranu 20 Fragment widoku aktywności pobierania. Pojawia się GIF taki sam jak w startowej aktywności. Pod nim widoczny jest aktualny progres pobierania wyników*

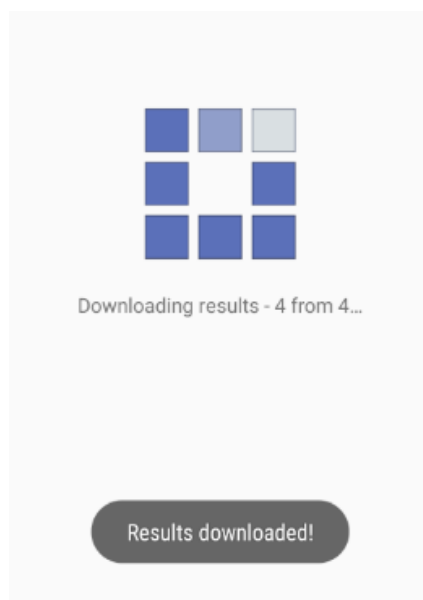
Jak wspomniałem wyżej dane pobierane są w tle. Co za tym idzie, można 'zminimalizować' aplikację (nie wiem czy ten termin występuje w aplikacjach mobilnych, mam na myśli kliknięcie w środkowy przycisk, który przechodzi do menu android). Progres jest widoczny również w powiadomieniach:



*Zrzut ekranu 21 Podgląd powiadomień w systemie android. Progres web scrapera jest dostępny w pasku powiadomień. Z tego pasku można również przerwać operację pobierania wyników*

Oczywiście może się zdarzyć sytuacja, w której będziemy chcieli przerwać pobieranie wyników - jest to możliwe przy pomocy przycisku 'Cancel' w powiadomieniach lub przycisku cofania w aktywności pobierania.

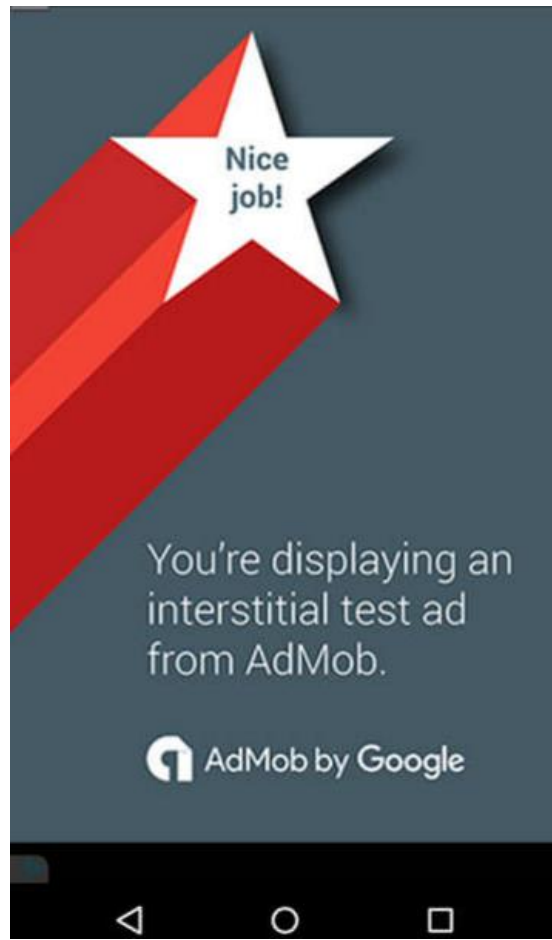
Zakończenie pobierania danych jest jasno sygnalizowane przez GUI:



*Zrzut ekranu 22 Zakończenie pobierania danych*

... ale zanim przejdziemy do prezentacji wyników czas na kolejną reklamę, tym razem pełnoekranową. Podobnie jak poprzednia, mniejsza reklama, ta również jest spersonalizowana dzięki przechowywaniu wyszukiwanych fraz.





*Zrzut ekranu 23 Reklama pełnoekranowa. Widoczna reklama jest wersją testową, ponieważ wszystkie zrzuty do instrukcji użytkownika są robione z symulatora systemu android*

#### 11.4. Wyniki

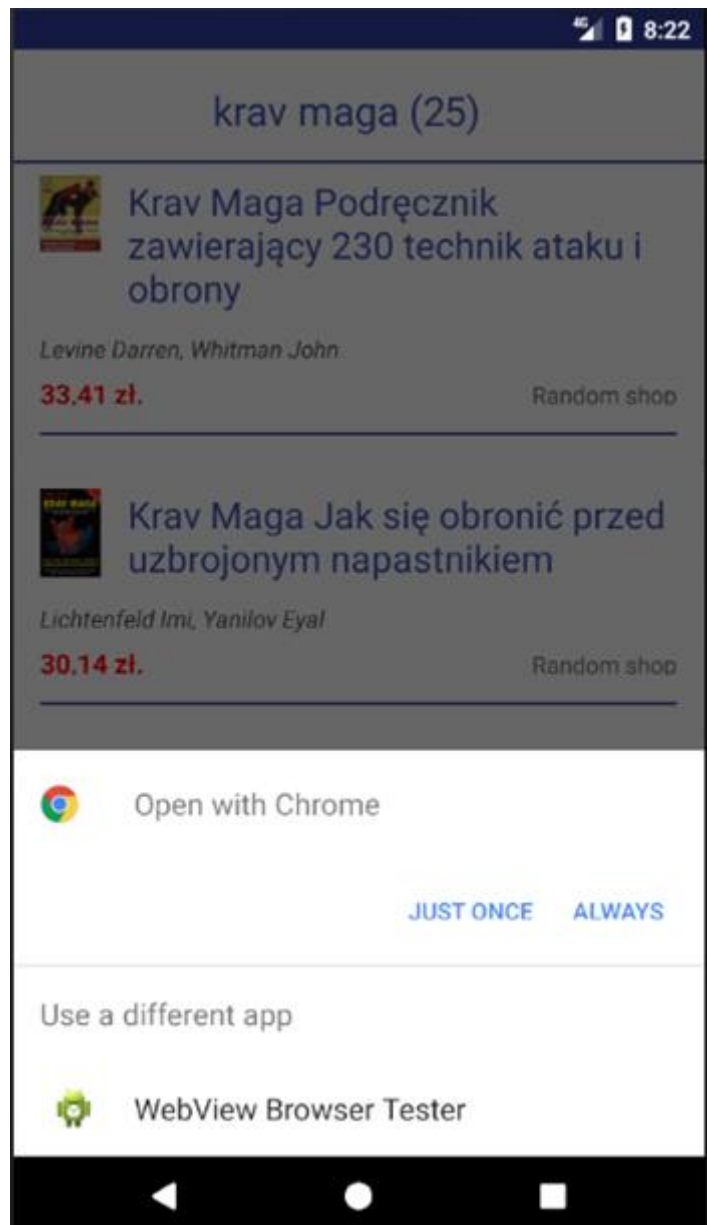
Po zamknięciu reklamy przyszedł najwyższy czas na widok zawierający pożądane wyniki.



*Zrzut ekranu 24 Fragment wyników po wpisaniu frazy 'krav maga'.  
Znaleziono 25 wyników*

Na samej górze widoczna jest wpisana fraza wraz z liczbą wyników w nawiasie.

Wyniki są grupowane po źródłach. Każdy produkt jest reprezentowany w ten sam sposób; widoczny jest tytuł, autor, miniaturka przedstawiająca produkt, cena oraz nazwa sklepu z którego dany produkt pochodzi. W obrębie sklepu produkty są posortowane na podstawie ceny rosnąco. Widok jest scrollowany. Kliknięcie w daną książkę spowoduje przejście do strony produktu:



*Zrzut ekranu 25 Wykonany po kliknięciu w konkretny produkt. Wybranie przeglądarki spowoduje otwarcie przez nią strony z produktem*

W powyższym zrzucie widzimy, że do wykonania operacji musimy wskazać aplikację, przy pomocy której wykonamy przejście do strony internetowej.

W tym miejscu się zatrzymamy. Zostały opisane wszystkie kroki, które należy wykonać aby uzyskać pożądane wyniki. Naprodukowałem kilka stron tekstu, jednak wbrew pozorom sama aplikacja jest prosta w użyciu; wystarczy wpisać konkretną frazę i czekać na wyniki... no cóż, mogłem napisać tak wcześniej.

## 12. Testy

Sama aplikacja nie posiada żadnych testów jednostkowych. Poprawność działania aplikacji opieramy na testach funkcjonalnych poprzez warstwę interfejsu i sprawdzaniu poprawności wyjścia poprzez uruchamianie aplikacji w trybie debuggowania.

## 13. Przyszłość aplikacji

Projekt można rozwijać bez naruszenia fundamentalnych założeń i podstaw implementacji. Jest to typ projektu który można rozwijać cały czas - dodając szablony czy nowe funkcjonalności. Można powiedzieć, że zostawiliśmy sobie otwartą furtkę.