POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria oprogramowania, procesów biznesowych i baz danych

Mateusz Jaszewski

Nr albumu 16032

Generyczny system do pobierania danych z portali internetowych

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, czerwiec 2018

Streszczenie

Niniejsza praca dotyczy implementacji systemu do sczytywania treści ze stron internetowych. Omówiona została w niej technika web scrapingu wraz z przykładami jej zastosowania oraz dostępnymi narzędziami. W dalszej części autor przedstawia koncepcję nowego systemu, poprzez zdefiniowanie wymagań, wykonanie projektu i implementację prototypu. Opracowane rozwiązanie stanowi system przygotowany z wykorzystaniem, po stronie serwera, takich technologii jak Java, Spring, MongoDB, PhanotmJS oraz w aplikacji klienckiej język TypeScript wraz z frameworkiem Angular. Ostatnia część pracy zawiera przegląd interfejsu zaimplementowanego prototypu, testy z wykorzystaniem przykładowej strony oraz podsumowanie stanowiące ocenę i zawierające rozważania na temat możliwego dalszego rozwoju systemu.

Słowa kluczowe

web scraping, pobieranie danych, portal internetowy, WWW, sieć, Internet, system, oprogramowanie

Spis treści

1. Wstęp

W tym rozdziale przedstawione zostały podstawowe informacje na temat celu pracy, przyjętych rozwiązań oraz oczekiwanego rezultatu. Ponadto, scharakteryzowane zostały kolejne rozdziały niniejszej pracy.

1.1. Cel pracy

Celem pracy jest opracowanie koncepcji generycznego systemu informatycznego pozwalającego na pobieranie danych z portali internetowych – realizującego technikę *web scrapingu*. Projekt oprogramowania powinien uwzględniać sczytywanie danych z witryn korzystających z mechanizmu AJAX oraz skryptów wykonywanych po stronie przeglądarki internetowej. Ponadto, praca z wykorzystaniem opracowywanego narzędzia nie powinna wymagać od użytkownika zaawansowanej wiedzy na temat działania stron internetowych.

1.2. Rozwiązanie przyjęte w pracy

Prototyp, realizujący cel pracy, został zaimplementowany jako system składający się z części działającej na serwerze oraz aplikacji uruchamianej przez przeglądarkę internetową. Do implementacji backendu wykorzystany został język Java oraz framework Spring. Ponadto, do przechowywania konfiguracji i zebranych danych wykorzystana została nierelacyjna baza danych MongoDB. Kluczowy element systemu stanowi przeglądarka PhantomJS. Posłużyła do otwierania stron internetowych i dostępu do ich struktury. Dzięki jej wykorzystaniu możliwa była obsługa dynamicznych witryn korzystających ze skryptów i doładowujących treści w trakcie działania. Część frontenodwą stanowi aplikacja zaimplementowana z wykorzystaniem języka TypeScript oraz frameworka Angular. Komunikuje się ona z częścią serwerową za pomocą API zgodnego z założeniami REST. Interfejs graficzny został przygotowany z wykorzystanie języka HTML, CSS oraz biblioteki Bootstrap.

1.3. Rezultat pracy

Oczekiwanym rezultatem pracy jest opracowanie koncepcji systemu pozwalającego na sczytywanie danych ze stron internetowych o różnych strukturach. Konfiguracja pobierania danych powinna być możliwie najprostsza, a przygotowane rozwiązanie powinno obsługiwać dynamiczne witryny internetowe.

W ramach pracy, na podstawie przeprowadzonej analizy, powstał prototyp spełniający postawione założenia. Zaimplementowany system może być wykorzystany do pobierania treści ze zróżnicowanych stron internetowych co stanowi dobrą bazę do dalszych prac.

1.4. Organizacja pracy

W drugim rozdziale pracy opisana została technika web scrapingu wraz z jej możliwymi zastosowaniami. Omówione zostały także dostępne narzędzia umożliwiające pobieranie danych z portali internetowych.

Trzeci rozdział stanowi przegląd wykorzystanych narzędzi, języków programowania oraz bibliotek. Poszczególne technologie zostały podzielona na elementy wykorzystane do implementacji backendu, frontendu oraz do zarządzania i budowania kodu źródłowego.

Czwarty rozdział omawia najpierw wymagania jakie powinien spełniać opracowywany system. Następnie przedstawiona jest jego architektura, a w ostatniej części rozdziału zawarte zostały najbardziej istotne elementy implementacji.

W piątym rozdziale zaprezentowane zostało działanie systemu na przykładowej stronie internetowej oraz interfejs graficzny, z którego podczas pracy korzysta użytkownik.

Szósty rozdział stanowi podsumowanie całej pracy. Zawarta w nim jest ocena zaimplementowanego prototypu oraz dalsze, możliwe kierunki rozwoju systemu.

2. Web scraping

Niezliczone portale i strony internetowe zawierają ogromne ilości informacji, które człowiek może odczytać przy użyciu przeglądarki. Jednak każda ze stron może mieć inną strukturę czy układ. Przetwarzanie maszynowe nieustrukturyzowanych lub tylko częściowo ustrukturyzowanych danych jest trudne, zazwyczaj programy komputerowe przystosowane są do obsługi danych w konkretnym formacie. W celu pozyskania danych z wymienionych źródeł, można posłużyć się techniką „web scrapingu" (zob. [1]). Termin ten pochodzi z języka angielskiego, „web" oznacza sieć, natomiast „scraping" – zeskrobywanie. Podczas takiego procesu z kodu stron wyodrębniane są interesujące informacje, które zapisywane są później w ustrukturyzowanej formie.

W dalszej części tego rozdziału została omówiona technika web scrapingu na przykładzie prostej strony w języku HTML, jej możliwe zastosowania oraz przykłady istniejących rozwiązań pozwalających na pozyskiwanie danych z portali internetowych.

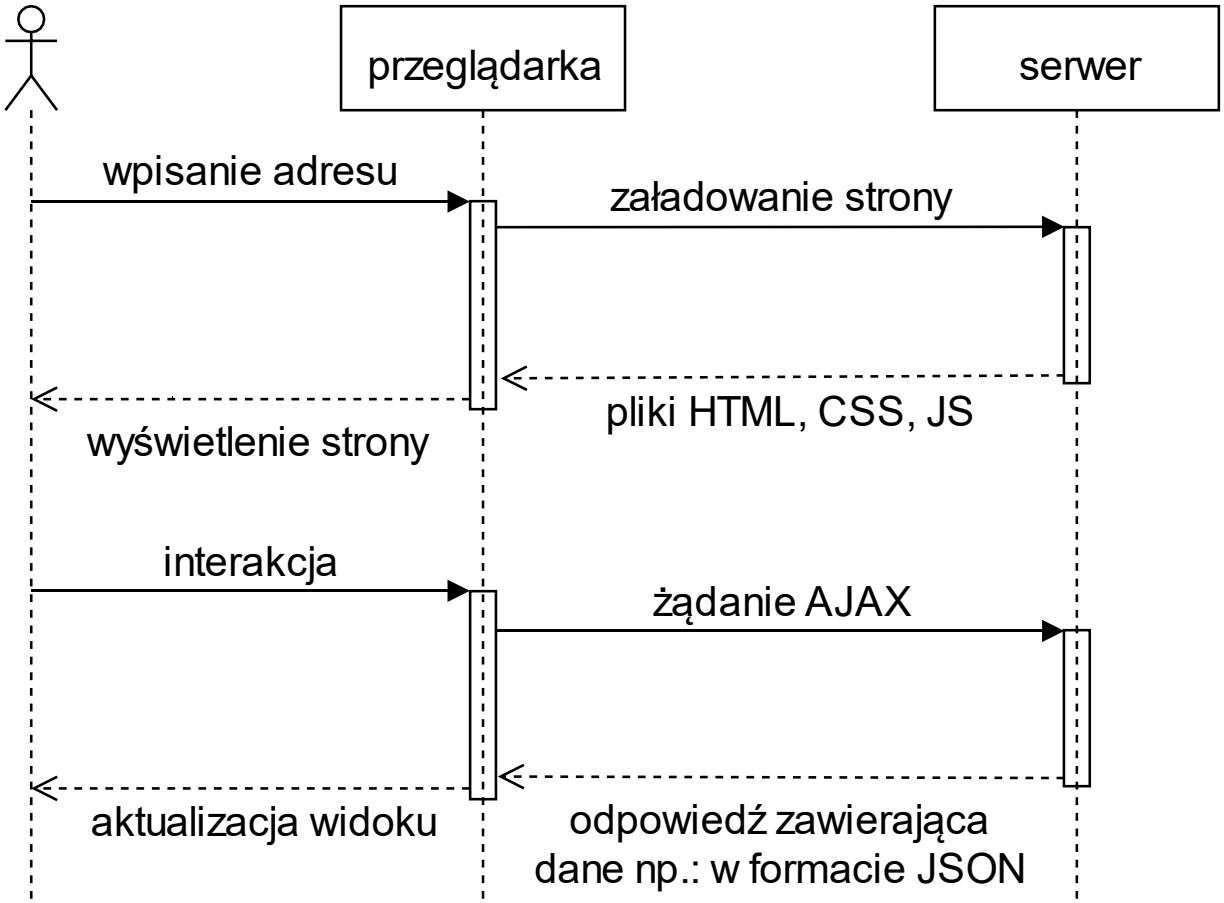
2.1. Technika Web scrapingu

Podstawą działania większości ówczesnych stron internetowych jest HTML (ang. *Hypertext Markup Language)*. Jest to język służący do opisu struktury dokumentów hipertekstowych. Powstał pod koniec lat osiemdziesiątych XX wieku, w ośrodku naukowo-badawczym CERN. Obecnie dostępna jest jego piąta wersja. Pisząc w nim, używa się zestawu znaczników, które reprezentują poszczególne elementy strony. Poniżej znajduje się lista kilku przykładowych znaczników (pełna lista dostępna jest w dokumentacji języka HTML – zob. [2]):

- akapit – znacznik *„p"*,
- odnośnik – znacznik *„a"*,
- obrazek – znacznik *„img"*,
- formularz – znacznik *„form"*,
- tabela – znacznik *„table"*.

Oczywiście oprócz samego HTML'a do budowy stron www wykorzystywany jest szereg innych technologii. Zazwyczaj do opisu sposobu wyświetlania elementów strony w przeglądarce używa się kaskadowych arkuszy styli (ang. *Cascading Style Sheets*, w skrócie *CSS*). Jednak sposób prezentacji stron podczas procesu web scrapingu nie jest istotny. Dużo większe znaczenie mają skrypty wykonywane przez przeglądarkę oraz technologia AJAX (ang. *Asynchronous JavaScript and XML*, czyli asynchroniczny JavaScript[1] i XML[2]). Często dane na stronie internetowej są doczytywane w trakcie jej przeglądania przez użytkownika. Przykład działania takiej witryny został zaprezentowany na rysunku nr 1. Na początku przesyłany jest tylko szablon strony w postaci dokumentu HTML, a dopiero później, w sposób asynchroniczny jest on wypełniany treścią przesłaną w odpowiedzi na żądanie AJAX. Zazwyczaj do tego celu wykorzystywany jest format JSON lub XML.

[1] JavaScript – skryptowy język programowania, stosowany do budowy stron internetowych.
[2] XML – ang. Extensible Markup Language – język znaczników, przeznaczony do zapisu danych w ustrukturyzowanej formie.



Rysunek 1. Schemat działania stron wykorzystujących mechanizm AJAX

Obecnie coraz większą popularność zyskują strony typu SPA (ang. *Single Page Application*), są to aplikacje internetowe, gdzie widok ładowany jest tylko raz na początku. Następnie, w zależności od akcji wykonywanych przez użytkownika, doładowywane są kolejne dane. Zazwyczaj wykorzystywane są w tym celu żądania AJAX, po wykonaniu których aktualizowany jest widok. Do budowy aplikacji SPA można wykorzystać takie frameworki[3] lub biblioteki jak Angular[4] czy React[5]. Pozwalają one na dynamiczne modyfikowanie struktury DOM[6] strony w zależności od interakcji z użytkownikiem.

Jedną z najprostszych metod pozyskiwania danych ze stron internetowych jest parsowanie plików HTML. Każdy z takich plików posiada strukturę drzewa, przykład pliku HTML znajduje się na listingu nr 1. Metoda ta ma zastosowanie tylko w przypadku statycznych stron WWW. Wystarczy przesłać odpowiednio spreparowane żądanie do serwera, a w odpowiedzi otrzymuje się kod źródłowy strony. Ponieważ operacje generowania kodu odbywają się po stronie serwera, w odpowiedzi znajdują się już wszystkie elementy, które wyświetlone zostałyby użytkownikowi. Posiadając taki plik wystarczy wyodrębnić z niego pożądane wartości znaczników lub ich atrybutów. W tym celu można posłużyć się

[3] Framework – szkielet ułatwiający tworzenie aplikacji.
[4] Angular – platforma do tworzenia aplikacji internetowych typu SPA.
[5] React – biblioteka do budowy interfejsów graficznych.
[6] DOM – ang. *Document Object Model* – czyli obiektowy model dokumentu. Przeglądarka internetowa po wczytaniu strony, przechowuje jej strukturę w formie drzewa DOM.

wyrażeniami regularnymi lub językiem zapytań *XPath*[7]. Listing nr 1 zawiera kod strony internetowej, składającej się z tabeli, w której znajdują się przykładowe produkty. Pojedynczy wiersz tabeli zawiera numer porządkowy, nazwę oraz cenę danego produktu.

Listing 1. Przykładowy kod HTML

```
<html>
  <body>
    <table>
      <tr>
        <th>lp.</th>
        <th>nazwa</th>
        <th>cena</th>
      </tr>
      <tr>
        <td>1</td>
        <td>mleko</td>
        <td>2,50 PLN</td>
      </tr>
      <tr>
        <td>2</td>
        <td>masło</td>
        <td>5,00 PLN</td>
      </tr>
      <tr>
        <td>3</td>
        <td>mąka</td>
        <td>1,50 PLN</td>
      </tr>
    </table>
  </body>
</html>
```

Przykład 1

W celu wyodrębnienia danych (nazwy oraz ceny) z kodu przedstawionego na listingu nr 1 można posłużyć się następującym wyrażeniem regularnym dla języka Java:

```
.*<td>.+</td>\s*<td>(.+)</td>\s*<td>(.+)</td>.*
```

Używając klasy *Matcher* można wyszukać wszystkie grupy pasujące do tego wyrażenia (informacje na temat działania tej klasy można znaleźć w dokumentacji języka Java – zob. [3]). Są to kolejno:

- mleko 2,50 PLN
- masło 5,00 PLN
- mąka 1,50 PLN

[7] XPath – ang. *XML Path Language* – czyli „język ścieżek XML". Pozwala na odwoływanie się do fragmentów dokumentu XML, a także dokumentu DOM.

Przykład 2

Drugim możliwym sposobem na pozyskanie interesujących danych z przykładowego kodu, przedstawionego wyżej, jest użycie języka *XPath*. W celu pobrania wszystkich nazw produktów należy wykorzystać następujące zapytanie:

```
html/body/table/tr[position()>1]/td[2]/text()
```

W rezultacie zwrócona zostanie lista elementów: „mleko", „masło", „mąka". Analogiczne zapytanie może posłużyć do pobrania cen produktów:

```
html/body/table/tr[position()>1]/td[3]/text()
```

Przedstawiona powyżej metoda nie zadziała jednak ze stronami, które dynamicznie wczytują dane. Wówczas pobrany dokument HTML nie będzie zawierał żadnej treści, a tylko szablon strony. W takim przypadku konieczne jest pobranie wszystkich skryptów umieszczonych na stronie oraz ich wykonanie. Wykonanie kodu klienckiego powoduje załadowanie dynamicznych elementów. Implementując takie rozwiązanie można wykorzystać jedną z przeglądarek stron internetowych, które działają bez użycia graficznego interfejsu[8]. Otwierając stronę w takiej przeglądarce, zostaną wykonane te same operacje co w przypadku normalnej przeglądarki. Można również zasymulować akcje użytkownika, takie jak: wypełnienie pola formularza, kliknięcie w przycisk czy przejście za pomocą hiperłącza na inną podstronę. Korzystając z API[9] takiej przeglądarki można odczytać aktualną strukturę drzewa DOM. Wówczas, proces sczytywania danych, będzie analogiczny jak w przypadku statycznej witryny internetowej.

2.2. Zastosowania Web scrapingu

Strony internetowe stanowią ogromny zbiór informacji. Zawierają dane dotyczące wszystkich dziedzin. Niestety nie wszystkie z portali internetowych udostępniają API. Dlatego, jeżeli potrzebujemy informacji z takiej witryny, możemy posłużyć się techniką web scrapingu. Poniżej zostały omówione przykładowe zastosowania dla tej techniki:

- badania naukowe – dane z witryn internetowych oraz portali stanowią doskonałe dane źródłowe praktycznie na dowolny temat,
- badania rynku – firmy zajmujące się handlem internetowym lub usługami mogą automatycznie porównywać swoją ofertę z ofertą konkurencji,
- porównywarki cen produktów – informacje o cenach różnych artykułów pobierane są automatycznie z wielu sklepów internetowych,

[8] Przeglądarki internetowe nieużywające interfejsu graficznego (ang. *headless browsers*) zostały omówione w dalszej części tej pracy, w rozdziale poświęconym przeglądarce PhantomJS.
[9] API – ang. *Application Programming Interface* – interfejs programistyczny aplikacji.

- import informacji bankowych – niektóre z banków, w ramach sprawdzania zdolności kredytowej, oferują automatyczne pobieranie historii transakcji z innych aplikacji bankowych (źródło – zob. [4]),
- pozyskiwanie danych kontaktowych – może być użyte w marketingu do pozyskiwania potencjalnych klientów lub do rozsyłania niechcianych wiadomości,
- agregatory treści – strony zawierające zbiór najciekawszych wpisów z blogów lub innych witryn internetowych na dany temat, mogą wykorzystywać web scraping do automatycznego pozyskiwania nowych treści.

Przedstawione przykłady pokazują jak szerokie i różnorodne może być zastosowanie web scrapingu. Technika ta jest użyteczna zarówno w pracy naukowej jak i w biznesie. Korzystając z pobranych treści należy również zwrócić uwagę na aspekt prawny. Wykorzystanie publicznie dostępnych danych w pracy badawczej, do przeprowadzenia pewnych analiz, nie jest w żaden sposób nielegalne. Jednak wykorzystanie sczytanych adresów email do wysyłania niechcianych wiadomości lub publikowanie treści, do których nie posiada się praw autorskich może być niezgodne z obecnie obowiązującymi przepisami prawa.

2.3. Aspekty prawne

Pobierając dane z wykorzystaniem techniki web scrapingu należy mieć na uwadze przede wszystkim zapisy zawarte w ustawie z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (źródło zob. [5]). Zgodnie z Art. 1. 1. „*Przedmiotem prawa autorskiego jest każdy przejaw działalności twórczej o indywidualnym charakterze*”, więc każda strona internetowa zawierająca „utwory” podlega ochronie. Korzystając z takich treści nie można ich publikować bez zgody autora. Łamanie przepisów zawartych w tym akcie normatywnym może wiązać się z karą grzywny lub pozbawieniem wolności do lat dwóch lub jeżeli czerpiemy korzyści materialne nawet do lat pięciu.

Nawet jeżeli zawartość witryny internetowej nie ma indywidualnego charakteru i nie jest utworem w myśl przedstawionej wcześniej ustawy, może podlegać ochronie na innych zasadach. Przykładowo bazy danych chronione są niezależnie od ustawy o prawie autorskim, regulacje dotyczące ich wykorzystania zostały zawarte w ustawie z dnia 27 lipca 2001 r. o ochronie baz danych (zob. [6]). Według tego aktu normatywnego: „*baza danych oznacza zbiór danych lub jakichkolwiek innych materiałów i elementów zgromadzonych według określonej systematyki lub metody, indywidualnie dostępnych w jakikolwiek sposób, w tym środkami elektronicznymi, wymagający istotnego, co do jakości lub ilości, nakładu inwestycyjnego w celu sporządzenia, weryfikacji lub prezentacji jego zawartości,*”. Art. 8. tej ustawy mówi, że dozwolone jest wykorzystanie bazy danych w celach dydaktycznych lub badawczych o charakterze niekomercyjnym. Jednak punkt drugi tego artykułu zabrania automatycznego pobierania danych, gdy jest sprzeczne z zasadami korzystania ze źródła: *„Nie jest dozwolone powtarzające się i systematyczne pobieranie lub wtórne wykorzystanie sprzeczne z normalnym korzystaniem i powodujące nieusprawiedliwione naruszenie słusznych interesów producenta.”*. Dlatego nawet w przypadku niekomercyjnego wykorzystania techniki web scrapingu do pozyskiwania informacji z baz danych, należy postępować zgodnie z regulaminem danego portalu internetowego, który może zabraniać automatycznego przetwarzania publikowanych treści.

Zupełnie inne przepisy regulują kwestie ochrony danych osobowych. Mimo, że na wielu portalach społecznościowych są one publicznie dostępne to wykorzystanie web scrapingu, w celu ich pozyskania, wiąże się z ich przetwarzaniem i gromadzeniem. W myśl ustawy z dnia 10 maja 2018 r.

o ochronie danych osobowych (zob. [7]) w przypadku, gdy nie posiadamy zgody osoby, której dotyczą te dane – jest to zabronione. Dlatego pozyskiwanie jakichkolwiek danych personalnych z zewnętrznych portali internetowych nie jest zgodne z przepisami polskiego prawa jak i Unie Europejskiej. Ponadto, nawet w przypadku posiadania odpowiedniej zgody na gromadzenie i przetwarzanie danych osobowych musimy spełniać szereg wymogów prawnych zawartych w przytoczonej ustawie. Nieprzestrzeganie tych zasad może wiązać się z wysokimi karami finansowymi.

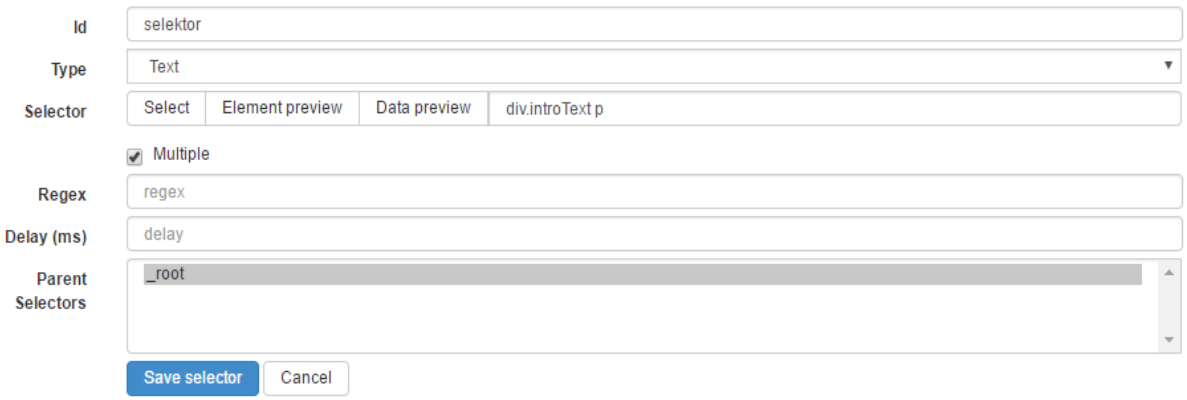
Automatyczne sczytywanie danych z portali internetowych może być niezwykle użyteczne, jednak zawsze należy pamiętać o przepisach obowiązującego prawa, w szczególności o wyżej wymienionych ustawach. Zgodnie z sentencją *„nieznajomość prawa nie jest usprawiedliwieniem”* zawsze lepiej upewnić się czy dane działania nie są sprzeczne z aktami normatywnymi lub regulaminem sczytywanej strony internetowej.

2.4. Przegląd istniejących rozwiązań

Obecnie dostępnych jest wiele różnych aplikacji pozwalających na automatyzację procesu sczytywania danych ze stron internetowych. Niestety, narzędzie posiadające rozbudowaną funkcjonalność są zazwyczaj płatne, a te darmowe posiadają liczne ograniczenia. W dalszej części tego rozdziału zostały omówione przekładowe narzędzia.

2.4.1 Web Scraper Chrome Extension

Darmowy dodatek do przeglądarki chrome (zob. [8]). Pozwala na sczytywanie danych ze stron internetowych, również tych dynamicznych, wykorzystujących JavaScript i technologię AJAX. W celu pobrania danych należy podać adres strony, a następnie utworzyć mapę strony. Proces ten polega na podaniu selektorów dla elementów, z których mają być pobierane dane. Wadą tego rozwiązania jest trudność konfiguracji. Użytkownik musi posiadać wiedzę na temat działania i budowy stron internetowych. Innym minusem tego rozwiązania jest jego działanie wyłącznie po stronie przeglądarki. Użytkownik, podczas procesu sczytywania danych, nie może zamknąć aplikacji chrome. Może to być dosyć uciążliwe w przypadku bardzo rozbudowanych stron, gdzie czas potrzebny na ich przetworzenie jest długi.



Rysunek 2. Konfiguracja sczytywania danych w aplikacji Web Scraper Chrome Extension

Pod dokonaniu konfiguracji sczytywane są dane, zgodnie z podanymi selektorami. Na rysunku nr 3 znajduje się podgląd danych pobranych ze strony z aktualnościami Polsko-Japońskiej Akademii Technik Komputerowych. Przedstawione rozwiązanie umożliwia export do pliku CSV[10].

Rysunek 3. Podgląd pobranych danych w aplikacji Web Scraper Chrome Extension

2.4.2 Dexi.io

Płatne narzędzie, działające jako aplikacja internetowa (zob. [9]). Nie wymaga żadnej instalacji od użytkownika. W celu pobrania danych z witryny internetowej należy zdefiniować nowego „robota". W tym celu można korzystać z pakietu gotowych komponentów (kliknięcie w element, iteracja po elementach strony itd.), które w wizualnym edytorze łączy się w jeden proces. Wybieranie elementów strony odbywa się z wykorzystaniem statycznego podglądu witryny.

Rysunek 4. Konfiguracja „robota" w aplikacji Dexi.io

#	Product	Description	In stock	Product image
1	Faded Short Sleeves T-shirt	Faded short sleeves t-shirt with…	299	Image (15.29 Kb, image/jpeg)
2	Blouse	Short-sleeved blouse with femini…	299	Image (17.17 Kb, image/jpeg)
3	Printed Dress	100% cotton double printed dress…	299	Image (16.26 Kb, image/jpeg)
4	Printed Dress	Printed evening dress with strai…	300	Image (30.78 Kb, image/jpeg)
5	Printed Summer Dress	Long printed dress with thin adj…	300	Image (22.65 Kb, image/jpeg)
6	Printed Summer Dress	Sleeveless knee-length chiffon d…	300	Image (21.53 Kb, image/jpeg)
7	Printed Chiffon Dress	Printed chiffon knee length dres…	300	Image (21.40 Kb, image/jpeg)

Rysunek 5. Podgląd sczytanych danych w aplikacji Dexi.io

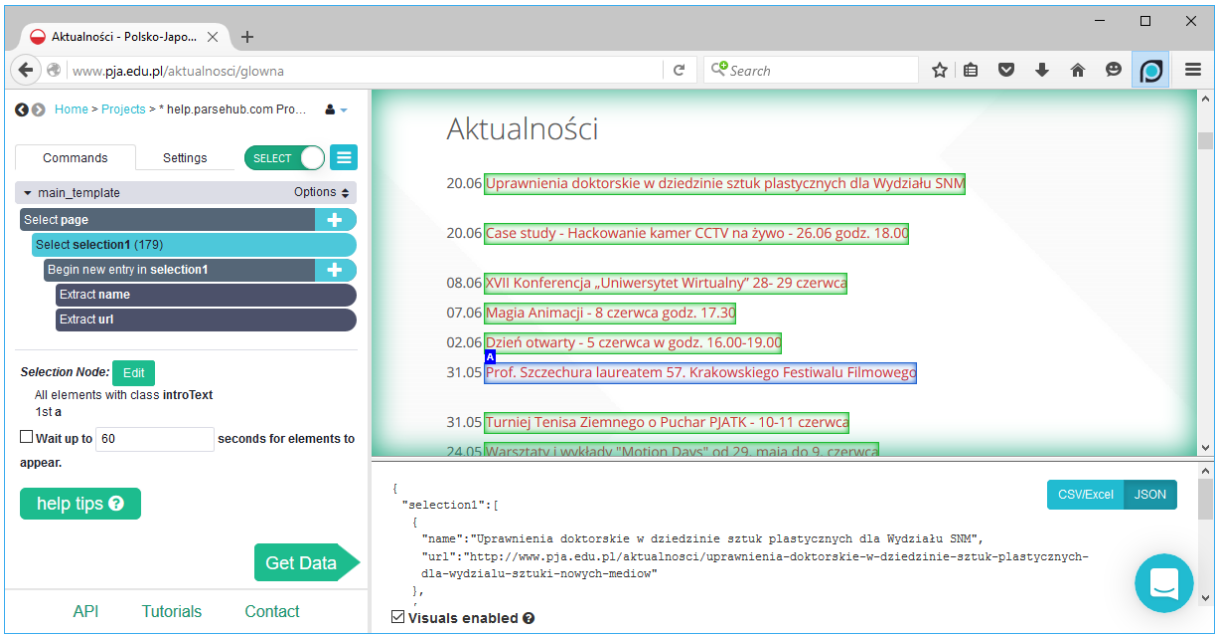
Po odpowiedniej konfiguracji można uruchomić pobieranie danych, które wykonuje się po stronie serwera. Użytkownik może zamknąć przeglądarkę, a wszystkie niezbędne operacje wykona część serwerowa aplikacji. Sczytane dane przechowywane są w aplikacji oraz można je pobrać w takich

[10] CSV (ang. *comma-separated values)* – format pliku tekstowego, gdzie poszczególne wiersze zawierają kolejne rekordy danych. Wartości poszczególnych pól, zgodnie z nazwą, są oddzielone przecinkami.

formatach jak: JSON, CSV czy XLS. Aplikacja Dexi.io ponadto udostępnia API za pomocą, którego można definiować i uruchamiać roboty oraz pobierać zebrane dane. Ciekawą funkcjonalnością jest możliwość wykorzystania serwerów proxy.

2.4.3 ParseHub

Częściowo darmowa aplikacja (zob. [10]) – darmowe konto ograniczone jest do 5 projektów, które muszą być publiczne. Ustawione są również limity na ilość pobieranych danych. W celu skorzystania z parsehub'a należy pobrać desktopową aplikację, która zawiera zintegrowaną przeglądarkę. Konfiguracja odbywa się poprzez wybieranie elementów strony, które mają zostać sczytane. Rysunek nr 6 przedstawia sposób konfiguracji na przykładzie strony PJA. Całość konfiguracji jest intuicyjna i nie wymaga od użytkownika zaawansowanej wiedzy na temat budowy stron www.

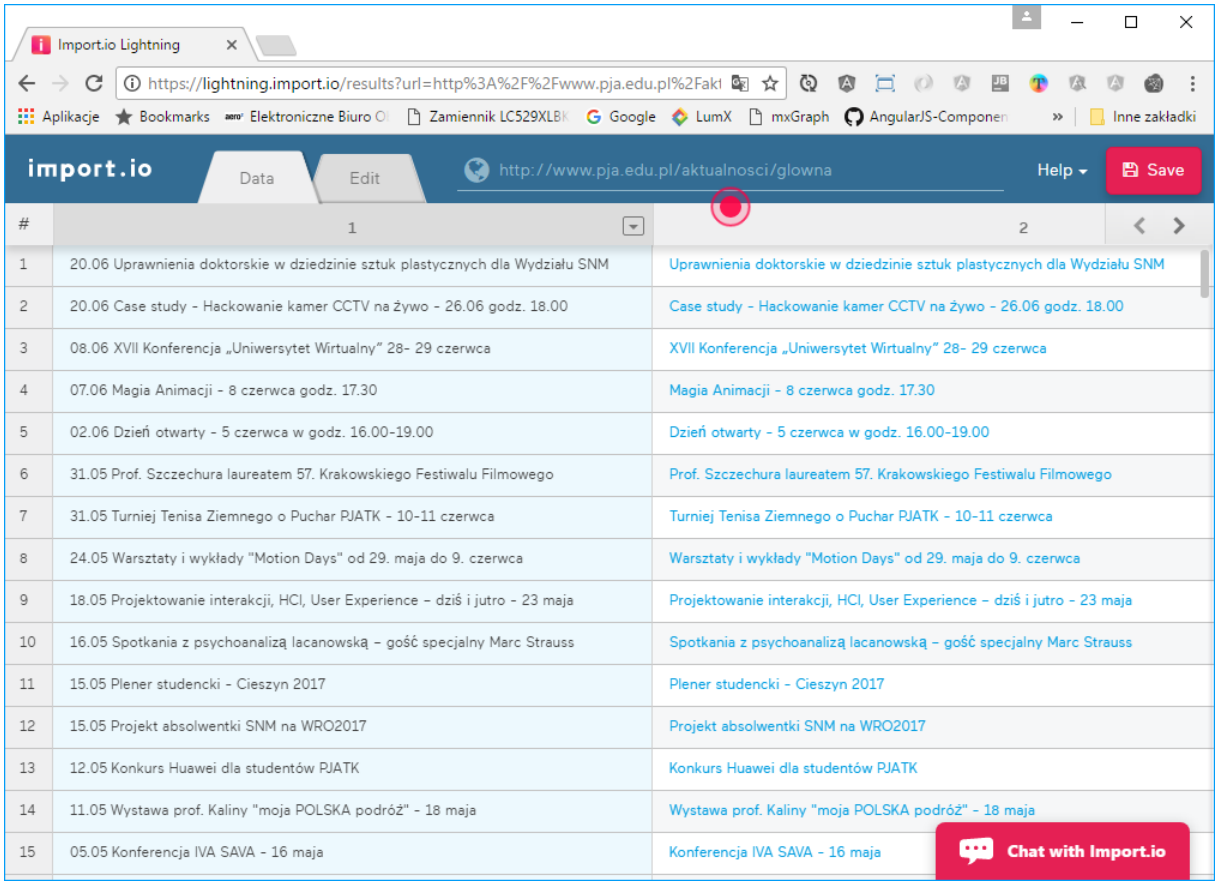


Rysunek 6. Konfiguracja sczytywania danych w aplikacji ParseHub

Po wybraniu odpowiednich elementów, w dolnej części okna wyświetlane są sczytane dane. Podobnie jak w przypadku poprzedniej aplikacji możne je wyeksportować do plików w formacie JSON lub CSV.

2.4.4 Import.io

Podobnie jak Parsehub, w aplikacji import.io (zob. [11]) darmowe konto jest ograniczone – posiada limit zapytań jakie można wykonać. Import.io nie wymaga instalacji żadnych dodatkowych programów ani pluginów. Całość pracy odbywa się poprzez stronę internetową. W celu sczytania danych należy utworzyć nowy „extractor" i podać adres url. Ciekawą funkcją jest automatyczna konfiguracja, w przykładzie została użyta strona z aktualnościami PJA i bez żadnej ręcznej ingerencji import.io poprawnie określi co powinno zostać sczytywać z tej strony.



Rysunek 7. Interfejs aplikacji Import.io

Jeżeli nie odpowiadają nam automatyczne ustawienia można je poprawić ręcznie. Wówczas wyświetlana jest strona, na której możemy zaznaczyć interesujące nas elementy. Po zapisaniu skonfigurowanego „extractora" możemy ustawić częstotliwość jego uruchomiania (np.: raz dziennie), po każdym wykonaniu otrzymamy emaila z informacją o danych jakie zostały zebrane. Wszystkie sczytane dane możemy wyeksportować do pliku CSV lub JSON. Ponadto, Import.io udostępnia API, które umożliwia dostęp do sczytanych treści. Podobnie jak w przypadku Dexi.io cały proces pobierania danych odbywa się na serwerze.

2.4.5 Wady dostępnych rozwiązań

Narzędzia przedstawione we wcześniejszej części tego rozdziału umożliwiają pobieranie danych ze stron internetowych. Pośród omówionych rozwiązań znajdują się aplikacje zarówno darmowe jaki i płatne. Wszystkie jednak posiadają pewne wady, które można byłoby poprawić w nowym systemie:

- pobieranie danych po stronie klienta - Web Scraper Chrome Extension wymaga, aby podczas sczytywania danych użytkownik miał cały czas włączoną przeglądarkę Chrome.
- konieczność instalacji dodatkowych elementów w systemie - w przypadku Web Scraper Chrome Extension jest to plugin do przeglądarki, a ParseHub wymaga instalacji aplikacji desktopowej.
- niepełny podgląd sczytywanej witryny - narzędzia będące aplikacjami internetowymi (Dexi.io oraz Import.io) wyświetlają widok strony, który jest tylko statycznym prerenderwanym podglądem wybranej witryny.

- konfiguracja scrapowania wymaga wiedzy na temat działania stron internetowych – w pluginie do przeglądarki chrome należy podać selektor CSS elementu, z którego mają zostać pozyskane dane.
- limity ilości pobieranych danych – w przypadku płatnych aplikacji, darmowe konto jest znacznie ograniczone lub aktywne tylko przez krótki czas.
- brak API – Web Scraper Chrome Extension umożliwia jedynie zapis sczytanych danych do pliku w formacie CSV.

Analiza cech przedstawionych narzędzi może być użyteczna podczas definiowania wymagań na opracowywany system. Projektowane oprogramowanie powinno łączyć w sobie zalety dostępnych rozwiązań i eliminować wady w nich występujące.

3. Przegląd wykorzystanych technologii

W tym rozdziale zostały omówione wszystkie języki programowania, biblioteki oraz narzędzie wykorzystane w procesie implementacji prototypu. Zostały one podzielone na technologie wykorzystane w części serwerowej, części klienckiej i na narzędzia wykorzystane do zarządzania kodem projektu.

3.1. Technologie wykorzystane w części serwerowej systemu

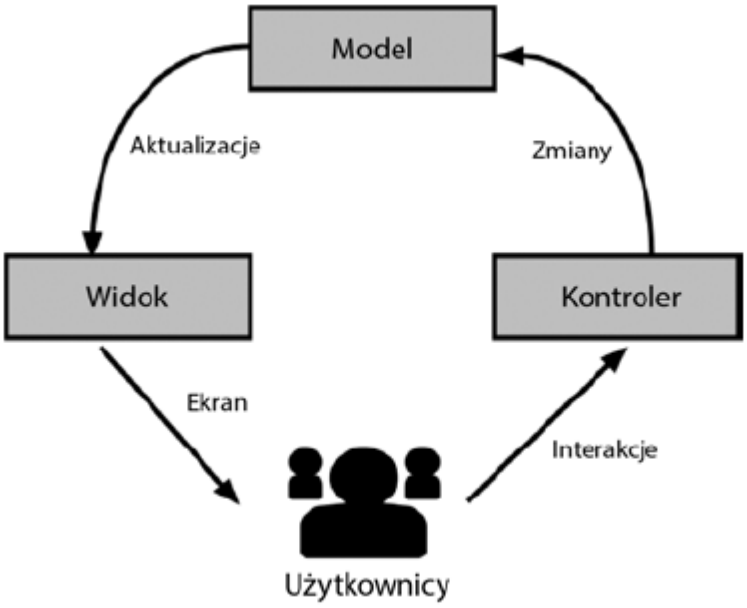
Java – obiektowy język programowania (zob. [12]). Pierwsza wersja powstała w 1996 r. w firmie Sun Microsystems, W 2010 r. firma Oracle przejęła Sun Microsystems wraz z prawami do języka Java. Obecnie dostępna jest już dziesiąta wersja tego języka. Od wersji ósmej Java posiada elementy języka funkcyjnego (zob. [13]). Programy napisane w tym języku uruchamiane są poprzez wirtualną maszynę Javy (z ang. *Java Virtual Machine* – w skrócie *JVM*). Dzięki temu język jest niezależny od środowiska. Program w nim napisany może być uruchomiony na dowolnym systemie operacyjnym, dla którego został zaimplementowany JVM. Java cechuje się statycznym typowaniem oraz mechanizmem odśmiecania pamięci. Programista nie musi manualnie alokować oraz zwalniać pamięci, wyręcza go w tym „zbieracz śmieci" (ang. *garbage collector*) stanowiący element wirtualnej maszyny. Java jest jednym z najpopularniejszych języków programowania (w rankingu popularności firmy TIOBE zajęła pierwsze miejsce – źródło zob. [14]). Przekłada się to na dostępną ogromną liczbę zewnętrznych bibliotek, a także wsparcie społeczności.

Spring – framework (zob. [15]), o otwartym kodzie źródłowym, służący do tworzenia aplikacji w języku Java. Powstał jako alternatywa dla technologii EJB (ang. Enterprise JavaBeans). W przeciwieństwie do EJB nie wymusza konkretnego modelu tworzenia oprogramowani oraz jest dużo „lżejszy" – nie wymaga dużego nakładu pracy w celu stworzenia prostej aplikacji (zob. [16]). Główną cechą Spring'a jest realizacja wzorca odwróconego sterowania (ang. *Inversion of Control* – w skrócie *IoC*) poprzez mechanizm wstrzykiwania zależności (ang. *Dependency Injection* – w skrócie *DI)*. Pisząc aplikację w Spring'u programista może definiować komponenty (klasy zarządzane przez framework), ale nie musi tworzyć ich instancji. Wbudowany kontener IoC zajmie się utworzeniem odpowiednich obiektów oraz powiązaniem ich ze sobą, czyli wstrzyknięciem zależności. Oprócz kontenera IoC framework Spring zawiera szereg modułów, dostarczających różne funkcjonalności. Poniżej zostały opisane moduły, które autor wykorzystał podczas tworzenia systemu będącego tematem tej pracy.

Spring Boot – moduł frameworka Spring (zob. [17]) upraszczający i przyśpieszający tworzenie aplikacji. W ramach odpowiednich „starterów" zawiera domyślną konfigurację innych springowych modułów. Rozwiązuje problem powielanego między projektami kodu, który służył tylko do konfiguracji poszczególnych elementów frameworka. Ponadto, dołącza do projektu wbudowany serwer aplikacji, więc projekt można dystrybuować jako archiwum jar, do którego uruchomienia wymagana będzie jedynie wirtualna maszyna Javy.

Spring MVC – moduł frameworka Spring (zob. [18]) umożliwiający tworzenie aplikacji webowych w oparciu o wzorzec Model-Widok-Kontroler (ang. *Model View Controller*). Elementy wzorca MVC zostały zaprezentowane na rysunku nr 8, są to:

- model – zawiera dane aplikacji, w przypadku Springa są to klasy POJO[11],
- kontroler – realizuje logikę związaną z przetwarzaniem akcji użytkownika,
- widok – odpowiada za generowanie, na podstawie modelu, kodu HTML, który zostanie wyświetlony użytkownikowi.



Rysunek 8. Schemat działania wzorca MVC, źródło [19]

W przypadku systemu tworzonego w ramach tej pracy nie został wykorzystany mechanizm widoków z modułu Spring MVC. Zamiast tego kontrolery zwracają dane w postaci dokumentów JSON[12] do aplikacji napisanej w frameworku Angular.

Spring Data – moduł framworka Spring (zob. [20]) odpowiedzialny za komunikację z bazą danych. Składa się z szeregu podmodułów specyficznych dla poszczególnych źródeł danych. W ramach tworzonego systemu została użyta implementacja przystosowana do pracy z bazą MongoDB. Spring data, niezależnie od typu bazy danych, dostarcza jednolitego sposobu dostępu do danych, w postaci repozytoriów. Wspiera także mapowanie obiektów na struktury bazy danych, w przypadku bazy MongoDB jest to mapowanie obiektów na dokumenty BSON[13].

Spring Security – moduł frameworka Spring (zob. [21]) dostarczający komponenty związane z zabezpieczeniem dostępu do aplikacji. Umożliwia implementację autoryzacji i uwierzytelniania użytkowników. Posiada także mechanizmy chroniące przed niektórymi atakami.

MongoDB – nierelacyjna, dokumentowa baza danych (zob. [22]) napisana w języku C++. Tworzona jest jako oprogramowanie o otwartym kodzie źródłowym. Nie posiada ściśle zdefiniowanej

[11] POJO (ang. *Plain Old Java Object*) – czyli zwykłe obiekty Javy. Termin ten stosowany jest najczęściej by podkreślić że dana klasa nie wymaga dodatkowych zależności czy implementacji interfejsów.
[12] JSON (ang. *JavaScript Object Notation*) – tekstowy format wymiany danych bazujący na podzbiorze języka JavaScript.
[13] BSON (ang. *binary JSON*) – czyli binarna wersja formatu JSON, używana głównie w bazie MongoDB.

struktury danych. Charakteryzuje ją wysoka wydajność oraz wsparcie skalowalności. Dane przechowywane są w kolekcjach w formacie BSON. Posiada API dla wielu języków programowania.

PhantomJS – przeglądarka stron internetowych (zob. [23]) nieposiadająca interfejsu graficznego (ang. *headless browser*). Zbudowana na silniku WebKit[14] z wykorzystaniem języka C++. Umożliwia wczytanie witryny internetowej oraz wykonanie wszystkich skryptów umieszczonych na niej. Początkowo zaprojektowana w celu testowania aplikacji internetowych, ale może być również wykorzystywana do generowania kodu HTML po stronie serwera, generowania plików PDF na podstawie stron www lub web scrapingu. Posiada API dla języka JavaScript umożliwiające dostęp do struktury DOM strony. W projekcie został wykorzystana sterownik GhostDriver (zob. [24]) umożliwiający komunikację programu napisanego w języku Java z przeglądarką PhantomJS.

Zuul – usługa będąca częścią otwartego stosu technologicznego firmy Netflix (zob. [25]), umożliwiającego tworzenie systemów działających w chmurze. Zuul może być wykorzystany jako brama do API (ang. *API Gateway*). Korzystając z mechanizmu filtrów pozwala na:

- zabezpieczenie poszczególnych punktów końcowych API,
- przekierowywanie żądań http na inne adresy lub do innych instancji aplikacji,
- monitoring żądań,
- modyfikację treści żądań oraz odpowiedzi.

W ramach opracowywanego systemu Zuul został wykorzystany do stworzenia pośrednika (ang. *proxy*) służącego do otwierania stron internetowych w elemencie iframe umieszczonym w aplikacji klienckiej. Dzięki przygotowaniu odpowiednich filtrów możliwe było ominięcie zabezpieczeń nie pozwalających na wstrzyknięcie własnego kodu do zewnętrznej witryny w elemencie iframe. Umożliwiło to stworzenie widoku do konfigurowania sczytywania danych bazujące na prawdziwej stronie internetowej.

Jsoup – biblioteka (zob. [26]) przeznaczona dla języka Java umożliwiająca parsowanie oraz modyfikację struktury dokumentów HTML. Została wykorzystana podczas implementacji filtrów Zuul'a. Posłużyła do modyfikacji wszystkich ścieżek (źródeł skryptów, obrazków oraz arkuszy styli) tak, by były one wczytane poprzez proxy.

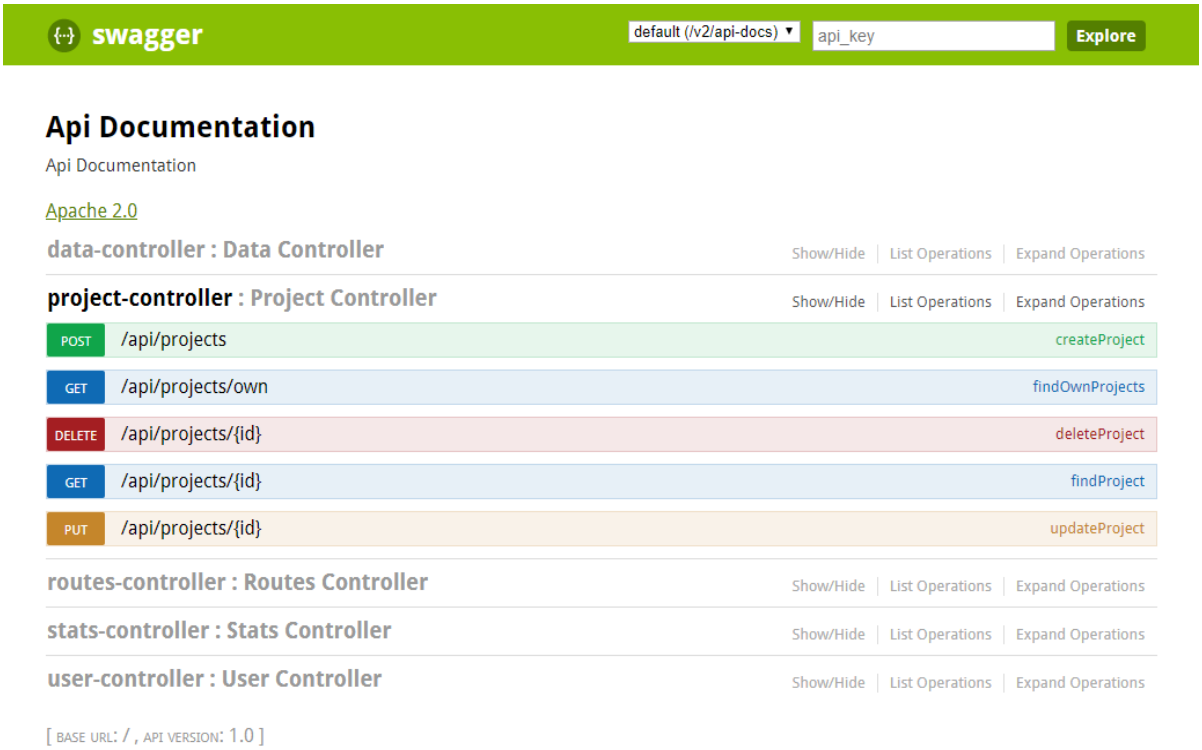
Lombok – narzędzie (zob. [27]) pozwalające na automatyczne generowanie części kodu klas dla języka Java. Dostarcza zbioru adnotacji, których użycie powoduje utworzenie, podczas budowania kodu źródłowego, dodatkowych elementów. Dzięki temu narzędziu można znacznie poprawić czytelność klas i skrócić długość plików. Lombok dostarcza między innymi następujące adnotacje:

- *@AllArgsConstructor* – generuje konstruktor zawierający jako argumenty, wszystkie pola klasy,
- *@RequiredArgsConstructor* – generuje konstruktor zawierający jako argumenty, wszystkie finalne pola klasy,
- *@NoArgsConstructor* – generuje domyślny, bezparametrowy konstruktor,

[14] WebKit – silnik przeglądarek internetowych posiadający otwarty kod źródłowy. Wykorzystany w takich przeglądarkach jak Safari, Opera oraz Chrome. Adres strony internetowej projektu: https://webkit.org/

- *@Setter* – powoduje utworzenie metody dostępowej do prywatnego pola, w przypadku umieszczenia tej adnotacji nad klasą tworzy *settery* dla wszystkich pól,
- *@Getter* – działa analogicznie jak poprzednia adnotacja, lecz powoduje utworzenie *getterów,*
- *@Builder* – dla adnotowanej klasy generuje dodatkową klasę implementującą wzorzec budowniczego.

Swagger – biblioteka (zob. [28]) generująca, na podstawie API aplikacji, stronę internetową, będącą interaktywną dokumentacją udostępnianych metod. Zawiera spis wszystkich punktów końcowych, wraz z ich parametrami oraz strukturą przyjmowanych i zwracanych danych. Pozwala w prosty sposób uruchamiać metody zawarte w API. Narzędzie to doskonale sprawdza się podczas testów manualnych, gdy nie został zaimplementowany jeszcze interfejs użytkownika. Na rysunku nr 9 znajduje się widok utworzony na podstawie klas kontrolerów zawartych w tworzonym projekcie:



Rysunek 9. Interaktywna dokumentacja API utworzona przez narzędzie *Swagger*

3.2. Technologie wykorzystane w aplikacji klienckiej

TypeScript – język programowania opracowany przez firmę Microsoft (zob. [29]). Stanowi nadzbiór języka JavaScript. Dodaje opcjonalne statyczne typowanie oraz obiektowość opartą na klasach. Formalnie każdy kod napisany w języku JavaScript jest zgodny z TypeScript'em. Programy napisane w tym języku są kompilowane bezpośrednio do JavaScript'u. Jest to również domyślny języka programowania dla frameworka Angular.

Angular – framework (zob. [30]) o otwartym kodzie źródłowym opracowany przez firmę Google. Został napisany w języku TypeScript. Wspomaga tworzenie aplikacji internetowych typu SPA. Początkowo miał być drugą wersją frameworku AngularJS, jednak ze względu na brak kompatybilności wstecznej został wydany jako oddzielny byt.

HTML – język (zob. [2]) służący do definiowania dokumentów hipertekstowych (ang. *Hypertext Markup Language*). Język HTML został dokładniej omówiony w drugim rozdziale tej pracy.

CSS – język (zob. [31]) kaskadowych arkuszy styli (and. *Cascading Style Sheets*). Służy do opisu sposobu wyświetlania stron www. Pozwala na odseparowanie od kodu HTML informacji o wyglądzie strony. Pozwala na określenie wszystkich parametrów dotyczących sposobu prezentacji treści między innymi: koloru tekstu, rozmiaru i kroju czcionek, wymiarów marginesów oraz pozycjonowania elementów.

Bootstrap – framework (zob. [32]) wspierający tworzenie widoków w aplikacjach wykorzystujących język HTML oraz CSS. Rozwijany jest przez firmę Twitter. Dołączany jest do projektu poprzez załadowanie pliku z arkuszem styli CSS oraz pliku skryptów języka JavaScript. Budowa graficznego interfejsu w Bootstrapie bazuje na siatce, która dzieli widok strony internetowej na wiersze i kolumny. Programista może nadawać odpowiednie style elementom strony by określić jaką pozycję i wielkość zajmują na tej siatce. Oprócz tego framework ten posiada wiele innych użytecznych styli, definiujących wygląd takich elementów jak formularze, przyciski, listy czy tabele. Zapewnia również wsparcie dla techniki RWD (ang. *Responsive web design*). RWD (zob. [33]) polega na projektowaniu stron www w ten sposób, by układ strony automatycznie dostosowywał się do wielkości ekranu urządzenia, na którym jest wyświetlana.

3.3. Narzędzia wykorzystane do budowania projektu i zarządzania kodem źródłowym

Git – rozproszony system kontroli wersji (zob. [34]). Udostępniony na zasadach wolnego oprogramowania. Stworzony przez Linusa Trovaldsa. Głównymi cechami wyróżniającym go są:

- rozproszoność – każdy z programistów biorący udział w projekcie posiada lokalnie kopię repozytorium, dzięki temu nawet w przypadku braku dostępu do sieci lub awarii zdalnego serwera można kontynuować pracę,
- łatwość tworzenia nowych gałęzi i ich scalania - pomaga to w pracy nad różnymi wersjami projektu,
- użycie migawek - każda rewizja jest obrazem całego projektu, a nie zbiorem różnic pomiędzy plikami,
- szybkość – Git dla dużych repozytoriów jest wielokrotnie szybszy od konkurencyjnych rozwiązań.

GitLab – oprogramowanie w postaci aplikacji internetowej (zob. [35]) umożliwiające zarządzanie repozytoriami Git oraz wspierające proces wytwarzania oprogramowania. Posiada elementy systemu zarządzania błędami oraz zadaniami, a także pozwala na tworzenie dokumentacji. Oprócz tego wspiera proces ciągłej integracji (ang. *Continuous Integration*) i ciągłego wdrażania (ang. *Continuous Deployment*). W tym celu GitLab dostarcza mechanizmu „rurociągów” (ang.

pipelines), które pozwalają na definiowanie procesu uruchamianego po wprowadzeniu zmian do repozytorium. Podczas takiego procesu kod projektu może zostać automatycznie zbudowany, mogą zostać uruchomione testy, a następnie powstałe pliki wykonywalne mogą zostać wdrożone na serwer. Podczas pracy nad prototypem systemu GitLab został wykorzystany do automatyzacji procesu budowania i wdrażania aplikacji. Każda zmiana wprowadzona w głównej gałęzi repozytorium powodowała automatyczne zbudowanie i uruchomienie nowej wersji systemu.

Maven – narzędzie (zob. [36]) automatyzujące proces budowania oprogramowania wytwarzanego na platformę Java. Służy do zarządzania cyklem życia projektu oraz zależnościami pomiędzy modułami oprogramowania i zewnętrznymi bibliotekami. Cykl budowania aplikacji składa się z następujących kroków:

1. walidacja – sprawdzenie czy konfiguracja projektu jest poprawna,
2. kompilacja – kod źródłowy projektu zostaje skompilowany,
3. testowanie – uruchamiane są testy jednostkowe,
4. pakowanie – skompilowany kod projektu pakowany jest do formatu, w którym może być dystrybuowany (dla Javy może być to JAR[15] lub WAR[16]),
5. weryfikacja – sprawdzenie poprawności przygotowanej paczki z oprogramowaniem,
6. instalacja – umieszczenie paczki z oprogramowaniem w lokalnym repozytorium,
7. wdrożenie – umieszczenie paczki w zdalnym repozytorium.

Konfiguracja projektu odbywa się poprzez tworzenie plików POM (ang. *Project Object Model*). Piki POM to dokumenty XML opisujące proces budowania aplikacji, zależności do zewnętrznych bibliotek, a także zawierające konfigurację pluginów, które rozszerzają dostępne funkcjonalności Mavena.

NPM (ang. *Node Package Manager*) – menadżer pakietów (zob. [37]) przeznaczony dla środowiska Node.js[17], można go jednak wykorzystać również do zarządzania zależnościami w aplikacjach frontendowych korzystających z języka JavaScript lub jego pochodnych. Oprogramowanie NPM pozwala na pobieranie oraz instalowanie bibliotek dostępnych w publicznych repozytoriach (istnieją również repozytoria prywatne, lecz te nie zostały wykorzystane w ramach tej pracy). Spis wszystkich zależności projektu znajduje się w pliku *package.json*, który posiada formę dokumentu JSON. Wewnątrz niego zdefiniowane są wszystkie zależności oraz informacje na temat projektu takie jak nazwa, wersja oraz rodzaj licencji.

IntelliJ IDEA – komercyjne zintegrowane środowisko programistyczne (zob. [38]) opracowane przez firmę JetBrains, przeznaczone do tworzenia oprogramowania w Javie. Posiada również wsparcie dla innych języków takich jak: HTML, CSS, JavaScript oraz TypeScript. Jest to jedno z najbardziej

[15] JAR (ang. *Java archive*) – archiwum zgodne z formatem ZIP zawierające metadane oraz skompilowane pliki klas Javy. Może zawierać aplikacje lub biblioteki.

[16] WAR (ang. *Web application archive*) – archiwum, podobne do JAR, lecz zawierające aplikację internetową napisaną w języku Java, oprócz plików klas, zawiera także pliki widoków, pliki konfiguracyjne oraz wszystkie zasoby niezbędne do działania aplikacji. Może być uruchomione na serwerze aplikacyjnym.

[17] Node.js – środowisko pozwalające na uruchomienie programów napisanych w języku JavaScript

rozbudowanych i uznanych obecnie IDE[18] dla języka Java. Posiada zbiór licznych pluginów wspomagających pracę z takimi narzędziami jak Maven, Git czy framework Spring. Autor, w ramach pracy nad projektem, wykorzystał to środowisko na zasadach darmowej licencji edukacyjnej. Każdy student w celach niekomercyjnych, może skorzystać z pełnej wersji tego oprogramowania za darmo. Dostępna jest również wersja o otwartym kodzie źródłowym (ang. *comunity edition*), jednak posiada pewne ograniczenia względem płatnej wersji.

[18] IDE (ang. *integrated development environment*) – zintegrowane środowisko programistyczne, czyli narzędzie wspomagające nie tylko pisanie kodu źródłowego ale także inne części procesu wytwarzania oprogramowania jak: testowanie czy przygotowywanie w graficznego interfejsu użytkownika.

4. Projekt oraz implementacja prototypu

W tym rozdziale został omówiony projekt oraz implementacja prototypu systemu będącego tematem tej pracy. Pierwszy podrozdział zawiera wymagania jakie powinno spełniać przygotowywane rozwiązanie. Następnie przedstawiona została architektura systemu. Kolejny rozdziała zawiera spis wszystkich narzędzi, języków programowania oraz technologii jakie zostały wykorzystane. Ostatnia część tego rozdziału jest omówieniem najbardziej istotnych elementów implementacji.

4.1. Określenie wymagań

Przejście od wizji systemu do działającego prototypu wymaga nie tylko programowania, lecz także wykonania analizy i projektu. Wymagania stanowią sformalizowany i uszczegółowiony spis cech tworzonej aplikacji. Wymagania funkcjonalne określają działania jakie są udostępniane użytkownikom. Natomiast wymagania niefunkcjonalne opisują ograniczenia przy jakich system ma działać poprawnie i jakie standardy ma spełniać.

4.1.1 Wymagania funkcjonalne

Podstawową funkcjonalnością systemu tworzonego w ramach tej pracy jest umożliwienie sczytywania danych z portali internetowych. Użytkownik korzystając z przygotowanej aplikacji powinien móc tworzyć, konfigurować i usuwać projekty[19]. Ponadto, powinien mieć dostęp do pobranych danych. Szczegółowa lista wymagań została zawarta poniżej. Natomiast rysunek nr 10 przedstawia diagram przypadków użycia[20].

Identyfikator:	WF1
Tytuł:	Logowanie
Warunki początkowe:	Użytkownik nie jest zalogowany w systemie
Scenariusz bazowy:	1. Użytkownik podaje login oraz hasło 2. Użytkownik zatwierdza wprowadzone dane 3. System wyświetla widok pulpitu użytkownika, w ramach którego widoczna jest lista projektów oraz statystyki dotyczące sczytywania danych
Scenariusz alternatywny:	3a. W przypadku podania niepoprawnych danych logowania system wyświetla komunikat informujący o tym
Warunki końcowe:	Użytkownik zostaje zalogowany do systemu

[19] Projekt – w ramach tworzonego systemu, rozumiany jest jako konfiguracja sczytywania danych dla pojedynczej witryny. Pojęcie to zostało dokładniej omówione w rozdziale poświęconym szczegółom implementacji.

[20] Diagram przypadków użycia (ang. *use case*) – przedstawia w sposób graficzny jakie akcje, użytkownik może wykonać w systemie.

Identyfikator:	WF2
Tytuł:	Wylogowywanie
Warunki początkowe:	Użytkownik jest zalogowany w systemie
Scenariusz bazowy:	1. Użytkownik wybiera opcję wylogowania 2. System wyświetla widok umożliwiający ponowne zalogowanie
Warunki końcowe:	Użytkownik zostaje wylogowany z systemu

Identyfikator:	WF3
Tytuł:	Tworzenie nowego projektu
Warunki początkowe:	Użytkownik jest zalogowany w systemie
Scenariusz bazowy:	1. Użytkownik wybiera opcję utworzenia nowego projektu 2. System tworzy pusty projekt o statusie „wersja robocza” 3. System wyświetla widok edycji projektu 4. Użytkownik podaje nazwę oraz adres strony internetowej 5. Załadowany zostaje podgląd wybranej strony 6. Użytkownika zaznacza na podglądzie strony elementy, które mają zostać sczytane oraz wybiera sposób przechodzenia pomiędzy kolejnymi podstronami 7. Użytkownik wybiera opcję aktywacji projektu
Scenariusz alternatywny 1:	5a. W przypadku błędnego wybrania elementów system informuje o tym użytkownika wyświetlając odpowiedni komunikat 7b. W przypadku błędnej konfiguracji – niewybrania żadnych pól do sczytania system nie pozwoli na aktywację projektu oraz wyświetli odpowiedni komunikat
Scenariusz alternatywny 2:	7a. Użytkownik może przerwać konfigurację projektu w dowolnym momencie, wówczas projekt pozostanie w statusie „wersja robocza”. Wznowienie konfiguracji projektu będzie możliwe po jego ponownym wybraniu z listy projektów danego użytkownika
Warunki końcowe:	W systemie zostaje utworzony projekt, przypisany do aktualnie zalogowanego użytkownika. Dla aktywowanego projektu rozpoczęty zostanie proces sczytywania danych.

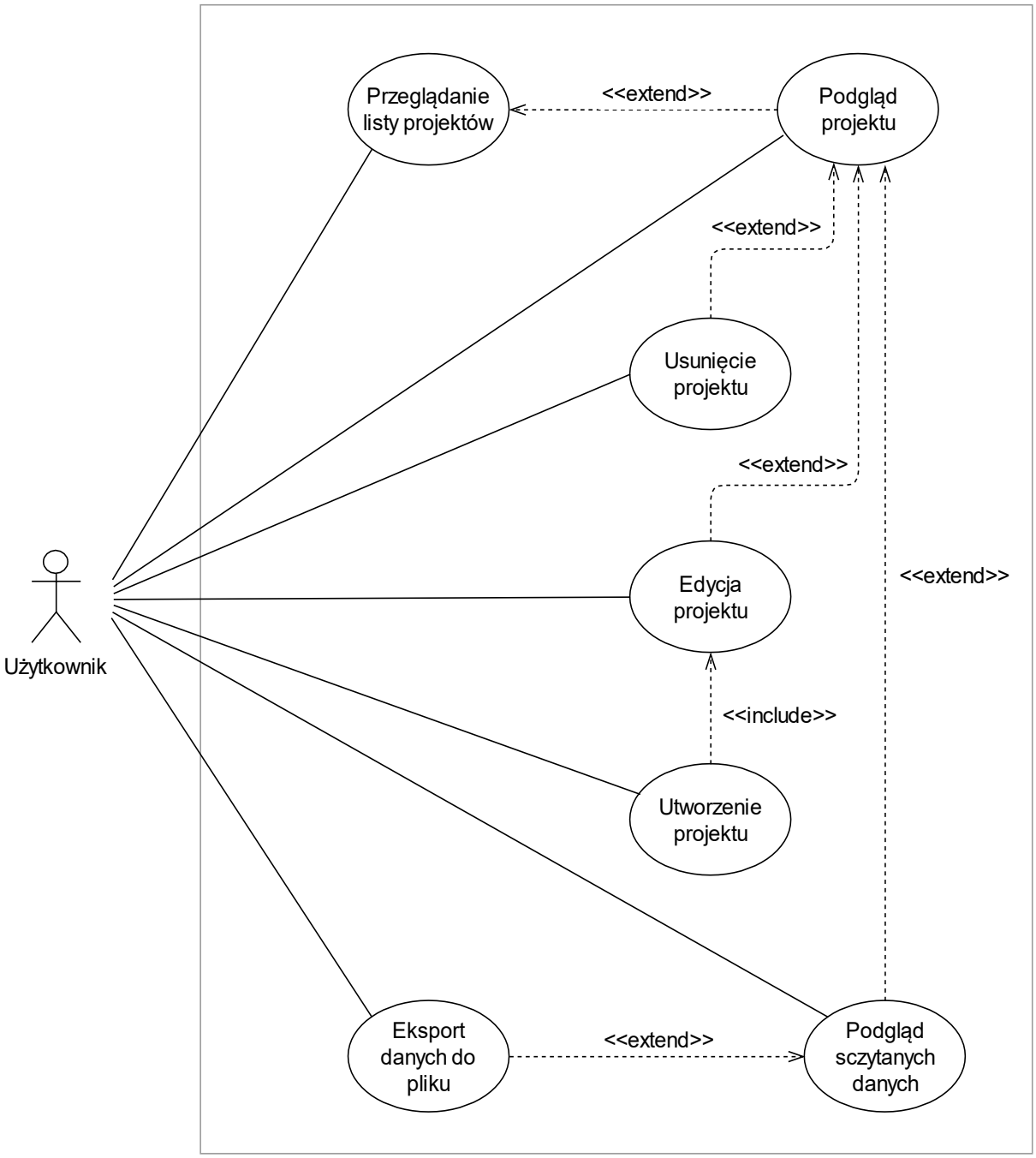
Identyfikator:	WF4
Tytuł:	Modyfikacja projektu
Warunki początkowe:	Użytkownik jest zalogowany w systemie oraz posiada projekt
Scenariusz bazowy:	1. Użytkownik wybiera projekt z listy projektów należących do niego 2. System wyświetla widok aktywnego projektu 3. Użytkownik wybiera opcję edycji projektu 4. System wyświetla widok edycji projektu oraz zmienia status projektu na wersję roboczą 5. Użytkownik modyfikuje wybrane parametry projektu 6. Użytkownik wybiera opcję aktywacji projektu
Scenariusz alternatywny:	2a. Jeżeli projekt jest w statusie „wersja robocza” system automatycznie przechodzi do punktu numer 4. 6a. W przypadku nie aktywowania projektu pozostaje on w statusie „wersja robocza” – sytuacja ta została opisana w wymaganiu WF4 w punkcie 7a.
Warunki końcowe:	System zapisuje zmiany wprowadzone w projekcie, w przypadku zmiany adresu strony lub listy sczytywanych pól dotychczas pobrane dane dla projektu są usuwane

Identyfikator:	WF5
Tytuł:	Usuwanie projektu
Warunki początkowe:	Użytkownik jest zalogowany w systemie oraz posiada projekt
Scenariusz bazowy:	1. Użytkownik wybiera projekt z listy projektów należących do niego 2. System wyświetla widok projektu 3. Użytkownik wybiera opcję usunięcia projektu 4. System wyświetla odpowiednie komunikat i przełącza widok na pulpit użytkownika
Warunki końcowe:	Z systemu zostaje usunięty projekt wraz ze wszystkimi sczytanymi dla niego danymi

Identyfikator:	WF6
Tytuł:	Podgląd sczytanych danych
Warunki początkowe:	Użytkownik jest zalogowany w systemie oraz posiada aktywny projekt
Scenariusz bazowy:	1. Użytkownik wybiera projekt z listy projektów należących do niego 2. System wyświetla widok aktywnego projektu, na którym widoczna jest lista sczytanych danych. Prezentowana jest tylko część wyników, użytkownik może przechodzić do kolejnych stron z danymi.
Warunki końcowe:	brak

Identyfikator:	WF7
Tytuł:	Eksport sczytanych danych do pliku
Warunki początkowe:	Użytkownik jest zalogowany w systemie oraz posiada aktywny projekt
Scenariusz bazowy:	1. Użytkownik wybiera projekt z listy projektów należących do niego 2. System wyświetla widok projektu 3. Użytkownik wybiera opcję eksportu sczytanych danych do pliku 4. System zapisuje plik z danymi na komputerze użytkownika
Warunki końcowe:	Zostaje pobrany plik z danymi sczytanymi w ramach wybranego projektu

Rysunek nr 10 przedstawia diagram przypadków użycia. Ze względu na czytelność zostało na nim pominięte logowanie oraz wylogowanie. We wszystkich przypadkach użycia bierze udział tylko użytkownik. W przygotowanym prototypie jest to jedyna rola w systemie. Z tego względu też spis powyższy wymagań funkcjonalnych nie zawiera informacji o aktorze.



Rysunek 10. Diagram przypadków użycia

4.1.2 Wymagania niefunkcjonalne

Poniższa lista zawiera spis wymagań niefunkcjonalnych:

- System powinien udostępniać graficzny interfejs dostępny poprzez przeglądarkę internetową. Do jego działania nie mogą być wymagane żadne dodatkowe biblioteki ani pluginy, które musiałby zainstalować użytkownik.
- Konfiguracja sczytywania danych powinna odbywać się z wykorzystaniem podglądu wybranej strony. Obsługa systemu nie może wymagać od użytkownika zaawansowanej

wiedzy na temat funkcjonowania stron internetowych, ani konieczności pisania skryptów lub zapytań.

- Użytkownik powinien mieć możliwość wyboru sposobu przechodzenia pomiędzy kolejnymi stronami.
- Proces sczytywania powinien odbywać się po stronie części serwerowej systemu. Użytkowni nie musi mieć włączonej aplikacji, by wykonywane było pobieranie danych.
- Dane z portali internetowych powinny być pobierane cyklicznie, po jednej stronie co określony czas. W przypadku pobrania danych ze wszystkich podstron, system powinien zacząć proces od początku i zapisywać tylko te rekordy, których nie ma jeszcze w bazie.
- Dane pobrane przez system powinny być przechowywane w bazie danych. Użytkownik powinien móc je podejrzeć w czytelnej formie oraz pobrać je jako pliki CSV oraz PDF.
- Opracowywane rozwiązanie musi udostępniać API zawracające pobrane dane w formacie JSON.
- System powinien pobierać dane, również ze stron korzystających ze skryptów wykonywanych po stronie przeglądarki oraz mechanizmu AJAX.

4.2. Architektura systemu

System składa się z dwóch części – części serwerowej oraz aplikacji klienckiej. Część serwerową stanowi program napisany w języku Java oraz framework'u Spring. Jako magazyn danych została wykorzystana nierelacyjna baza danych MongoDB. Ważnym elementem systemu jest również przeglądarka PhantomJS, uruchamiana jest ona w systemie jako oddzielny proces. Służy do otwierania stron internetowych bez wykorzystania trybu graficznego. Strona wczytywana jest jak w zwykłej przeglądarce, ładowane i wykonywane są wszystkie skrypty umieszczone na niej. Dzięki zastosowaniu tej technologii było możliwe pobieranie treści z dynamicznych stron wykorzystujących mechanizm AJAX.

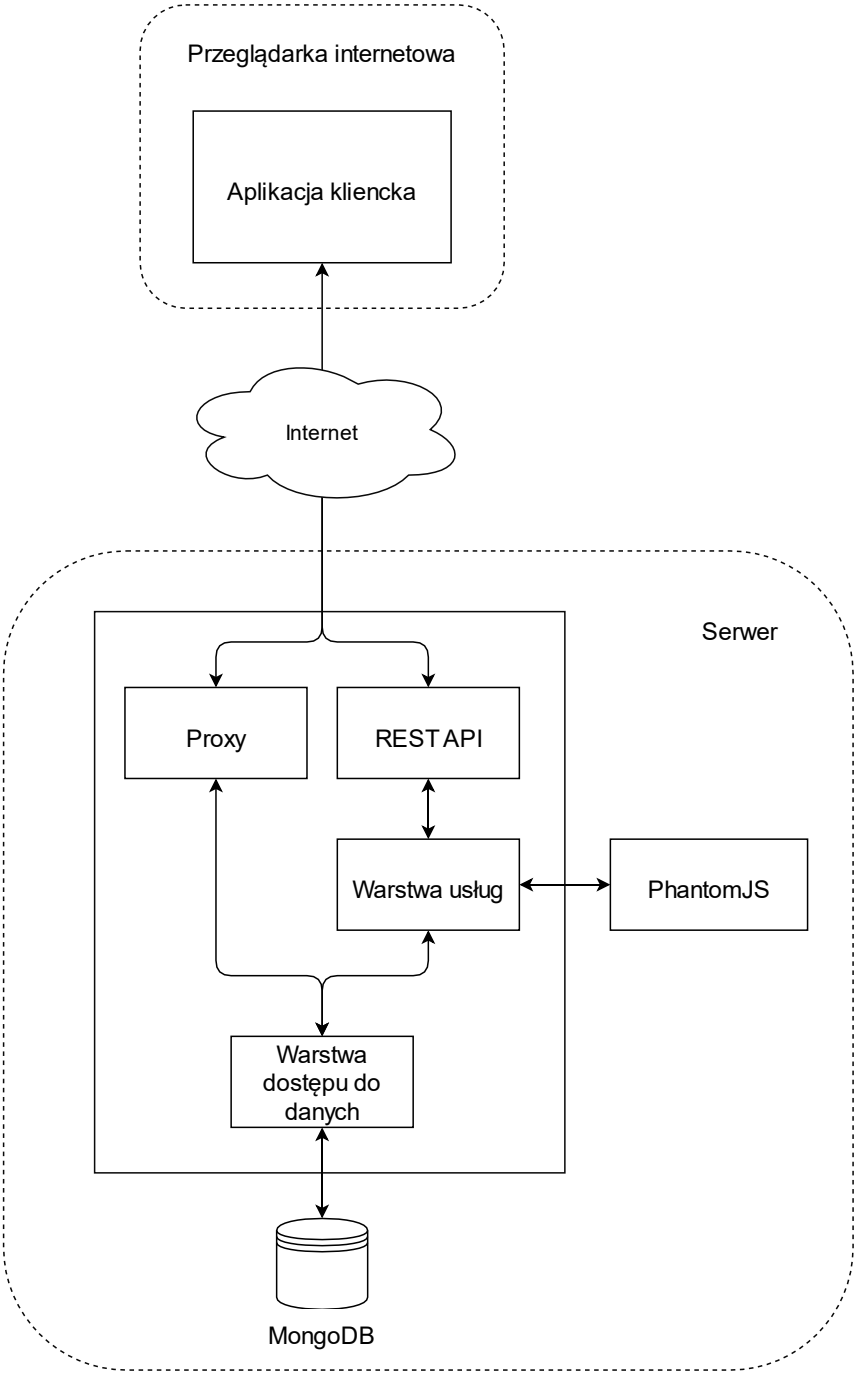
Kod serwerowy systemu został podzielony na trzy warstwy:

- API zgodne z założeniami REST, z którym komunikuje się część kliencka systemu,
- warstwę serwisową, realizującą logikę biznesową,
- warstwę dostępu do danych, odpowiadającą za komunikację z bazą danych.

Oprócz wymienionych powyżej komponentów, system posiada wbudowane proxy. Zostało one zaimplementowane przy pomocy biblioteki Zuul. Jest pośrednikiem pomiędzy aplikacją kliencką, a stroną, z której są sczytywane dane. Dzięki temu możliwe było ominięcie zabezpieczeń niepozwalających na dodanie własnych skryptów do witryny wyświetlanej wewnątrz znacznika iframe.

Część kliencką stanowi natomiast aplikacja napisana w języku TypeScript oraz framework'u Angular. Interfejs graficzny powstał natomiast przy użyciu języka HTML i CSS oraz biblioteki Bootstrap. Jest to aplikacja typu SPA, która zgodnie z wymaganiami, może być uruchomiona w przeglądarce internetowej.

Schemat prezentujący poszczególne elementy systemu oraz powiązania pomiędzy nimi znajduje się na rysunku nr 11. Szczegółowe informacje na temat poszczególnych technologii, wymienionych powyżej, znajdują się w kolejnym podrozdziale.



Rysunek 11. Architektura systemu

4.3. Najważniejsze elementy implementacji

W tym rozdziale została omówiona struktura projektu oraz najważniejsze elementy implementacji. Pierwsza część stanowi przegląd organizacji plików projektu, następnie przedstawione są najważniejsze rozwiązania techniczne wraz z realizującymi je fragmentami kodu źródłowego.

4.3.1 Struktura projektu

Kod źródłowy systemu został podzielony na dwa moduły Mavena. Pierwszy z nich o nazwie „frontend" zawiera aplikację kliencką napisaną w języku TypeScript i frameworku Angular. Drugi o nazwie „backend" jest programem działającym po stronie serwera napisanym w języku Java oraz frameworku Spring. W celu budowania aplikacji frontedowej został wykorzystany dodatkowy plugin mavena - *frontend-maven-plugin*. Pozwala on na uruchomienie menadżera pakietów Npm w celu pobrania zależności i skompilowania kodu źródłowego napisanego w języku TypeScript do języka JavaScript. Moduł backendowy skonfigurowany jest w taki sposób by skopiować pliki powstałe w module frontedowym do odpowiedniego katalogu, tak by serwowane były one potem przez wbudowany serwer aplikacji.

Listing 2. Fragment pliku *pom.xml* modułu *backend*

```
<plugin>
     <artifactId>maven-resources-plugin</artifactId>
     <version>3.0.2</version>
     <executions>
          <execution>
               <id>copy-resources</id>
               <phase>validate</phase>
               <goals><goal>copy-resources</goal></goals>
               <configuration>
                    <outputDirectory>
                         ${basedir}/target/classes/static
                    </outputDirectory>
                    <resources>
                         <resource>
                                 <directory>../frontend/dist</directory>
                             <filtering>false</filtering>
                         </resource>
                    </resources>
               </configuration>
          </execution>
     </executions>
</plugin>
```

Listing 3. Konfiguracja pluginu *fronted-maven-plugin* z pliku *pom.xml* modułu *frontend.*

```
<plugin>
	<groupId>com.github.eirslett</groupId>
	<artifactId>frontend-maven-plugin</artifactId>
	<executions>
	  <execution>
		<id>install node and npm</id>
		<goals>
			<goal>install-node-and-npm</goal>
		</goals>
		<phase>prepare-package</phase>
		<configuration>
			<nodeVersion>v8.9.0</nodeVersion>
			<npmVersion>5.5.1</npmVersion>
		</configuration>
	  </execution>
	  <execution>
		<id>npm install</id>
		<goals>
			<goal>npm</goal>
		</goals>
		<phase>prepare-package</phase>
		<configuration>
			<arguments>install</arguments>
		</configuration>
	  </execution>
	  <execution>
		<id>npm run-script build</id>
		<phase>prepare-package</phase>
		<goals>
			<goal>npm</goal>
		</goals>
		<configuration>
			<arguments>
				run-script build
			</arguments>
		</configuration>
	  </execution>
	</executions>
</plugin>
```

W wyniku zbudowania projektu (wykonania Maven'owej operacji „package") utworzony zostaje plik JAR zawierający aplikację serwerową z dołączonymi skompilowanymi plikami części

klienckiej i wbudowanym serwerem Tomcat[21]. Tak przygotowaną paczkę można uruchomić z wykorzystaniem polecenia maszyny wirtualnej javy w następujący sposób:

java -jar backend/target/easyscraper-backend-0.0.1-SNAPSHOT.jar

Ponieważ serwer aplikacyjny został dołączony do archiwum JAR, do uruchomienia aplikacji wymagane jest jedynie zainstalowane środowisko uruchomieniowe javy – JRE[22], instancja bazy danych MongoDB oraz zainstalowana przeglądarka PhantomJS.

Część serwerową systemu stanowi aplikacja napisana w języku Java. Klasy tego programu zostały pogrupowane na następujące pakiety:

- *config* – znajdują się w nim klasy konfigurujące komponenty frameworka Spring. Jest to na przykład klasa *SecurityConfig*, która zawiera ustawienia modułu *spring-security*.
- *domain* – zawiera klasy reprezentujące model danych przechowywany w bazie MongoDB oraz repozytoria, które pozwalają na dostęp do tych danych.
- *infrastructure.zuul* – wewnątrz tego pakietu znajdują się klasy implementujące filtry biblioteki *Zuul*. Pakiet ten stanowi implementację *proxy* pośredniczącego w wyświetlaniu w części klienckiej zewnętrznych stron internetowych.
- *service* – zawiera klasy tworzące warstwę usług. Znajdują się tutaj serwisy służące np.: do komunikacji z przeglądarką PhantomJS, generowania plików z pobranymi danymi oraz zapewniające dostęp do aktualnie zalogowanego użytkownika.
- *web* – w tym pakiecie znajdują się wszystkie klasy odpowiedzialne za implementację API. Są to klasy DTO[23] definiujące strukturę zwracanych danych, kontrolery realizujące poszczególne punkty końcowe oraz fasady pośredniczące pomiędzy kontrolerami, a warstwą usług i dostępu do danych.

Część kliencka systemu została przygotowana w postaci aplikacji napisanej przy użyciu języka TypeScript oraz frameworku Angular. Pliki projektu zostały pogrupowane w katalogi w następujący sposób:

- *components* – katalog zawiera wszystkie komponenty aplikacji. Komponenty stanowią podstawowy element funkcjonalny frameworka Angular. Z nich budowana jest cała aplikacja. Każdy komponent może składać się z pliku w języku TypeScript, który zawiera logikę, pliku HTML definiującego graficzny interfejs oraz pliku CSS zawierającego informacje o sposobie prezentacji komponentu.
- *model* – w tym folderze zdefiniowany jest model danych jakim posługuje się aplikacja, klasy w nim zawarte odpowiadają strukturą, klasom DTO z części serwerowej systemu.

[21] Tomcat – darmowy serwer aplikacyjny, służący do serwowania aplikacji webowych napisanych w języku Java. Do projektu został dołączony jako część pliku wynikowego JAR poprzez wykorzystanie modułu Springa – *spring-boot-strater-web*.

[22] JRE (ang. *Java Runtime Environment*) – środowisko uruchomieniowe dla programów napisanych w języku Java, składa się z maszyny wirtualnej oraz klas podstawowych bibliotek.

[23] DTO (ang. *data transfer object*) – obiekty transferujące dane.

- *pipes* – katalog zawiera jeden plik - *safe-url.pipe.ts.* Jest to implementacja Angularowej struktury *pipe*, która służy do przetwarzania danych w widoku. Przygotowany *pipe* pozwala na przypisanie zewnętrznego adresu *url* do elementu *iframe*.
- *services* – zawiera zbiór klas – usług komunikujących się z API udostępnionym przez część systemu działającą na serwerze.

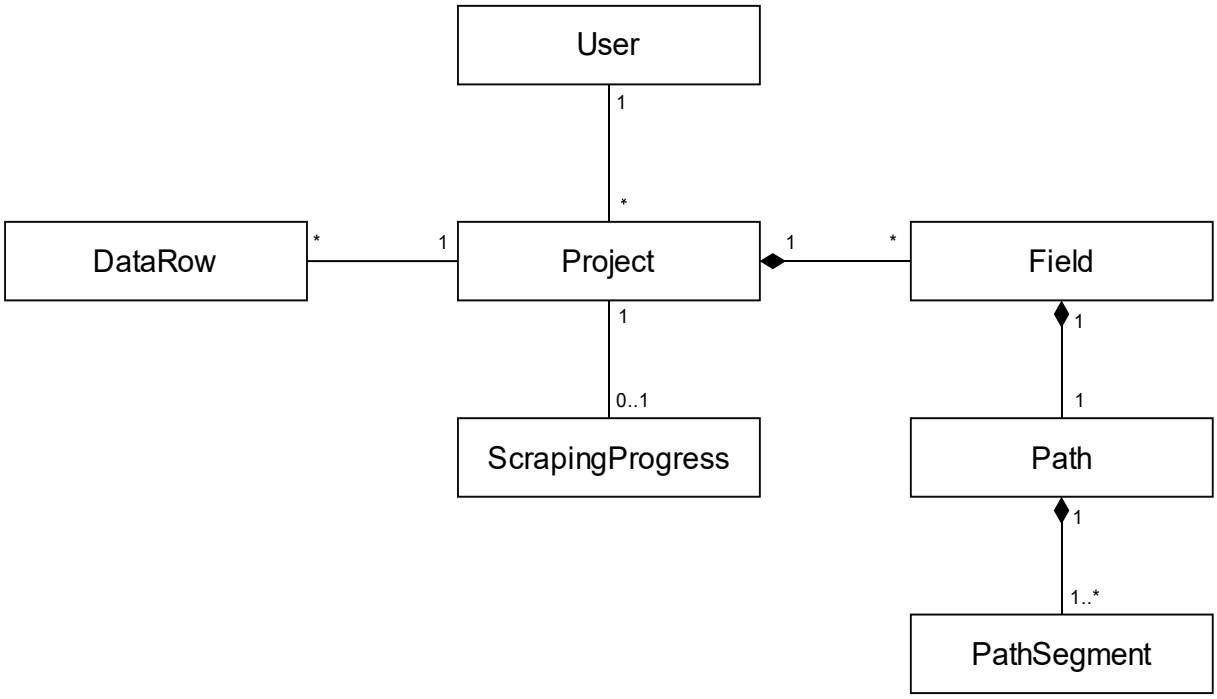
Oprócz wymienionych wyżej plików i katalogów bardzo istotny jest plik *app.mpodule.ts.* Stanowi on konfigurację głównego modułu aplikacji. Zawiera informację o importowanych komponentach i serwisach, a także powiązanie poszczególnych widoków aplikacji z ich adresami. Powiązanie to określane jest w Angularze mianem *routingu* i definiuje jaki komponent aplikacji zostanie wyświetlony po wejściu na dany adres.

4.3.2 Model danych

Do implementacji prototypu wykorzystana została baza danych MongoDB. Przechowuje ona dokumenty w postaci binarnych plików BSON, lecz komunikacja z nią odbywa się z wykorzystaniem formatu JSON. W celu odwzorowania struktury przechowywanych dokumentów zostały zaimplementowane następujące klasy języka Java:

- *User* – zawiera informacje o użytkowniku takie jak login oraz skrót hasła,
- *Project* – reprezentuje projekt. W obiektach tej klasy zapisana jest cała konfiguracja projektu, której dokonuje użytkownik, czyli: nazwa, adres strony, status projektu, lista sczytywanych pól oraz sposób przechodzenia pomiędzy podstronami.
- *DataRow* – przechowuje pojedynczy, sczytany rekord danych. Do jej implementacji wykorzystana została klasa *Map,* dzięki czemu może odwzorowywać dokumenty zawierające różne zestawy pól – baza MongoDB pozwala na przechowywanie w tej samej kolekcji dokumentów o różnych strukturach.
- *ScrapingProgress* – obiekty tej klasy zawierają informacje o aktualnie sczytywanej podstronie dla danego projektu.
- *Field* – zawiera nazwę sczytywanego pola oraz ścieżkę wskazującą na element znajdujący się na stronie, dla której został skonfigurowany projekt.
- *Path* – jest modelem dla ścieżek w formacie *XPath.* Zawiera listę składająca się z obiektów klasy *PathSegment.*
- *PathSegment* – reprezentuje pojedynczy segment ścieżki *XPath.*

Obiekty utworzone na podstawie wymienionych wyżej klas, są podczas zapisu serializowane do postaci dokumentów JSON, a w przypadku odczytu są deserializwoane z tego formatu. Mechanizm, który został wykorzystany do komunikacji z bazą danych został opisany w kolejnym rozdziale.



Rysunek 12. Diagram klas reprezentujących model danych

Ponieważ MongoDB jest bazą *NoSQL* przechowującą dokumenty, asocjacje widoczne na powyższym diagramie są reprezentowane jako pola zawierające identyfikator powiązanego obiektu. Natomiast zależności typu agregacja są zrealizowane poprzez zagnieżdżenie jednego dokumenty w innym.

4.3.3 Implementacja warstwy dostępu do danych

Warstwa dostępu do danych została zaimplementowana z wykorzystaniem biblioteki *spring-data.* Każda z kolekcji znajdujących się w bazie MongoDB posiada odpowiadające jej repozytorium. Jest to komponent Springowy oznaczony adnotacją *@Repository* i rozszerzający generyczny interfejs *MongoRepository*. Domyślnie każde z repozytoriów posiada podstawowe operacje CRUD (ang. *Create, Read, Update, Delete*) pozwalające na tworzenie, odczyt modyfikację i usuwanie dokumentów z bazy danych. Możliwe jest rozszerzenie repozytorium o dodatkowe metody. Wystarczy dodać deklarację metody o odpowiedniej sygnaturze, by Spring automatycznie wygenerował jej implementację. Przykład repozytorium rozszerzonego o dodatkowe metody znajduje się na listingu nr 4.

Listing 4. Przykład repozytorium rozszerzającego interfejs *MongoRepository*

```
public interface ProjectRepository extends MongoRepository<Project, String> {
        List<Project> findByUserId(String userId);
        List<Project> findByStatus(Status status);
}
```

Spring na podstawie nazw metod i zadeklarowanych typów utworzy odpowiednie implementacje repozytoriów. Pierwsza z metod z listingu nr 4 pozwala na wyszukiwanie projektów przypisanych do użytkownika o podanym identyfikatorze, a druga na wyszukiwanie projektów o wybranym statusie.

Oprócz przedstawionego wcześniej mechanizmu generowania metod na podstawie ich sygnatur, framework Spring pozwala na samodzielne utworzenie implementacji repozytoriów. W tym celu można posłużyć się klasą *MongoTemplate*, która udostępnia API pozwalające wyszukiwać i modyfikować dane. Przykład własnej implementacji repozytorium znajduje się na listingu nr 5. W klasie *DataRowRepositoryImpl* utworzona została metoda pozwalająca na zliczanie wierszy danych, które zostały pobrane w poszczególnych dniach. W celu wykonania tej operacji został użyty mechanizm agregacji (dokumentacja – zob. [39]) bazy MongoDB. Istotna jest również nazwa utworzonej klasy, Spring potrafi automatycznie wyszukać repozytorium o nazwie odpowiadającej utworzonej klasie i dołączyć do niego implementację dodatkowych metod.

Listing 5. Własna implementacja metody repozytorium z wykorzystaniem klasy MongoTemplate

```
@RequiredArgsConstructor
public class DataRepositoryImpl {

	private final MongoTemplate mongoTemplate;

	public List<Map> countDataRowsPerDay(List<String> projectsIds, Date from) {
		Aggregation aggregation = Aggregation.newAggregation(
			Aggregation.match(new Criteria("projectId").in(projectsIds)
				.and("creationTime").gte(from)),
			Aggregation.project()
				.and("projectId").as("projectId")
				.and("creationTime").extractDayOfMonth().as("day")
				.and("creationTime").extractMonth().as("month")
				.and("creationTime").extractYear().as("year"),
			Aggregation.group("projectId", "day", "month", "year")
				.count().as("count")
		);
		AggregationResults<Map> results = mongoTemplate.aggregate(
			aggregation, "data", Map.class
		);
		return results.getMappedResults();
	}
}
```

4.3.4 API systemu

Aplikacja webowa z częścią serwerową systemu komunikuje się poprzez API. Zostało ono zaimplementowane zgodnie z założeniami REST (ang. *Representational State Transfer*). REST jest

zbiorem praktyk, które określają sposób tworzenia usług w oparciu o protokół HTTP. Adres usługi reprezentuje zasób (ang. *resource*). Natomiast akcje możliwe do wykonania na danym zasobie definiowane są poprzez metody protokołu HTTP:

- GET – pobranie jednego lub wielu zasobów,
- POST – utworzenie nowego zasobu,
- PUT – aktualizacja istniejącego zasobu,
- PATCH – aktualizacja częściowa zasobu (np.: zmiana tylko niektórych pól),
- DELETE – usunięcie zasobu.

Adresy zasobów powinny być rzeczownikami w liczbie mnogiej, oznaczają one zbiór elementów danego typu. Każdy z elementów takiej kolekcji powinien posiadać unikalny identyfikator, podając go w kolejnej części adresu można się do niego odwołać. Możliwe jest również definiowanie pod zasobów, by to zrealizować, dodaje się do ścieżki kolejny fragment zawierający nazwę pod zasobu.

REST zleca również wykorzystanie statusów protokołu HTTP do informowania o powodzeniu lub błędach podczas wykonania danej akcji na zasobie. Tabela nr 1 zawiera spis kodów wykorzystanych w tworzonym systemie:

Tabela 1. Lista statusów HTTP wykorzystanych w systemie

Kod	**Nazwa**	**Opis**
200	OK	Ogólna informacja o powodzeniu.
201	CREATED	Pomyślnie utworzono nowy zasób.
202	ACCEPTED	Zaakceptowano przesłane dane – sukces aktualizacji zasobu.
204	NO CONTENT	Poprawnie wykonano akcje lecz nie zostaną zwrócone żadne dane (może oznaczać np. pomyślne usunięcie elementu).
400	BAD REQUEST	Przesłano żądanie w niepoprawnym formacie.
401	UNAUTHORZIED	Dostęp do zasobu wymaga autoryzacji.
403	FORBIDEN	Brak uprawnień do zasobu.
404	NOT FOUND	Zasób o podanym adresie nie istnieje.
405	METHOD NOT ALLOWED	Na wybranym zasobie nie można wykonać podanej akcji.
500	INTERNAL SERVER ERROR	Wewnętrzny błąd systemu.
504	GATEWAY TIMEOUT	Wykonanie metody trwało zbyt długo.

Kody wymienione w tabeli nr 1 stanowią tylko część wszystkich statusów dostępnych w standardzie HTTP. Ich wykorzystanie może różnić się w zależności od systemu. Przyjęte jest jednak, że statusy dzielą się na następujące grupy:

- *1xx* – kody informacyjne,
- *2xx* – kody powodzenia,
- *3xx* – kody przekierowania,
- *4xx* – kody oznaczające błędy klienta,
- *5xx* – kody oznaczające błędy serwera.

API systemu zostało zaimplementowane przy pomocy modułu *spring-boot-starter-web*. Każdy z zasobów obsługiwany jest przez oddzielny kontroler – klasę oznaczoną adnotacją *@RestController*. Poszczególne metody tej klasy odpowiadają za akcje jakie można wykonać na danym zasobie. Przykład kontrolera znajduje się na listingu nr 6. Jest to klasa odpowiedzialna za obsługę zasobu reprezentującego projekt. Udostępnia metody pozwalające na tworzenie, modyfikację, usunięcie oraz pobranie jednego projektu lub wszystkich przypisanych do aktualnie zalogowanego użytkownika.

Listing 6. Przykład kontrolera

```
@RestController
@RequestMapping("/api/projects")
@RequiredArgsConstructor
public class ProjectController {

        private final ProjectFacade projectFacade;

        @ResponseStatus(HttpStatus.CREATED)
        @PostMapping
        public Project createProject(@RequestBody Project project) {
            return projectFacade.create(project);
        }

        @GetMapping
        public List<ProjectListEntryDto> findOwnProjects() {
            return projectFacade.findOwnProjects();
        }

        @GetMapping("/{id}")
        public Project findProject(@PathVariable String id) {
            return projectFacade.findProject(id);
        }

        @ResponseStatus(HttpStatus.ACCEPTED)
        @PutMapping("/{id}")
        public Project updateProject(@PathVariable String id,
                                     @RequestBody Project project) {
            return projectFacade.updateProject(id, project);
        }

        @ResponseStatus(HttpStatus.NO_CONTENT)
```

```
        @DeleteMapping("/{id}")
        public void deleteProject(@PathVariable String id) {
            projectFacade.deleteProject(id);
        }
}
```

Metody API zwracają i przyjmują dane w formacie JSON. Klasy będące parametrami i typami zwracanymi w kontrolerach są automatycznie serializowane i deserializowane z tego formatu. Wyjątkiem są usługi pozwalające na pobranie danych w postaci plików CSV oraz PDF. Tabela nr 2 zawiera spis wszystkich zasobów wraz z akcjami jakie można na nich wykonać. Wszystkie punkty końcowe, oprócz tych dotyczących podzasobu *„data”*, wymagają wcześniejszego zalogowania. Logowanie oraz wylogowywanie odbywa się poprzez metody dostarczane przez moduł *spring-boot-starter-security*.

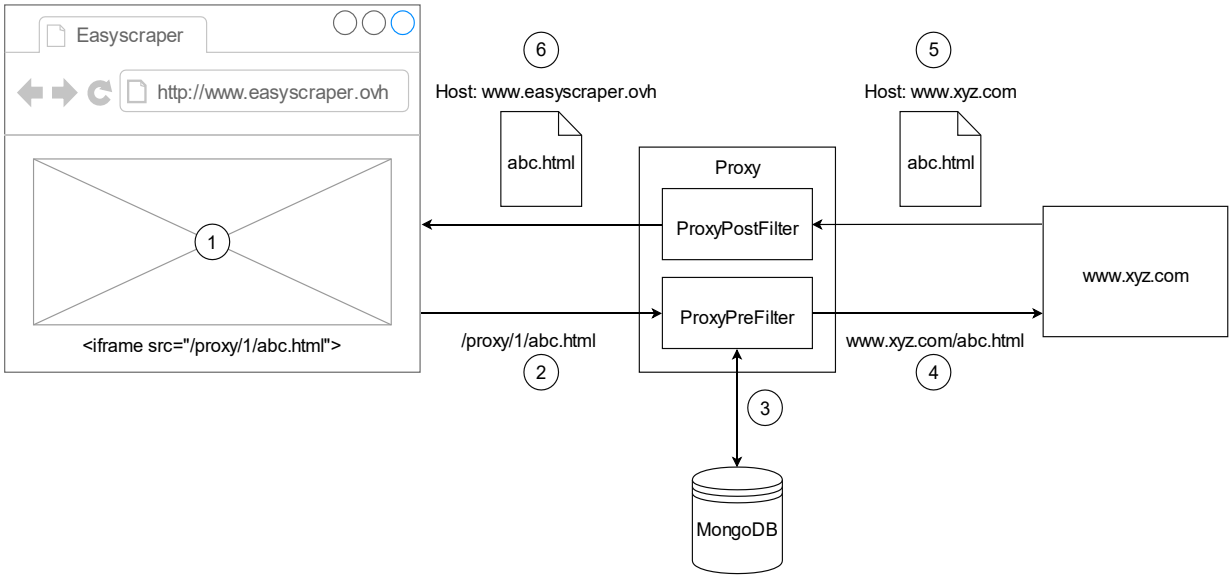
Tabela 2. Spis metod udostępnianych przez API

Metoda	**Ścieżka**	**Opis**
GET	/api/projects	Zwraca listę wszystkich projektów aktualnie zalogowanego użytkownika.
POST	/api/projects	Tworzy nowy projekt.
GET	/api/projects/{id}	Zwraca projekt o podanym identyfikatorze.
PUT	/api/projects/{id}	Aktualizuje projekt o podanym identyfikatorze.
DELETE	/api/projects/{id}	Usuwa projekt o podanym identyfikatorze.
GET	/api/projects/{projectId}/data	Zwraca dane pobrane dla projektu.
GET	/api/projects/{projectId}/data/csv	Zwraca pobrane dane w postaci pliku CSV.
GET	/api/projects/{projectId}/data/pdf	Zwraca pobrane dane w postaci pliku PDF.
GET	/api/stats/dataRowsCount	Zwraca statystyki na temat liczby pobranych rekordów danych dla projektów aktualnie zalogowanego użytkownika.
GET	/api/stats/dataRowsPerProject	Zwraca liczbę rekordów w podziale na projekty zalogowanego użytkownika.
GET	/api/stats/dataRowsCountInTime	Zwraca liczbę pobranych rekordów w podziale na poszczególne dni dla projektów zalogowanego użytkownika.

4.3.5 Mechanizm proxy

Moduł proxy był konieczny by móc wyświetlać strony internetowe wewnątrz elementu *iframe* oraz dodawać do nich własne skrypty. Przeglądarki internetowe, ze względów bezpieczeństwa, nie pozwalają na wstrzyknięcie kodu do zagnieżdżonych stron jeśli te pochodzą z innej domeny. Ponadto, część witryn internetowych posiada zabezpieczenie przed osadzaniem ich na innych stronach.

Działanie proxy opiera się na dwóch filtrach - *ProxyPreFilter* oraz *ProxyPostFilter*. Są to klasy implementujące interfejs *ZuulFilter* pochodzący z biblioteki *Zuul*. Pierwsza z tych klas odpowiada za przekierowanie żądania wysłanego na adres */proxy/{id_projektu}* do zewnętrznej witryny, której *url* zapisany jest w konfiguracji projektu. Natomiast druga przy pomocy biblioteki Jsoup modyfikuje zawartość dokumentów HTML, tak by wszystkie dodatkowe zasoby, dołączone do strony, zostały załadowane poprzez proxy. Ponadto, nadpisuje nagłówek *„X-Frame-Options”* wartością *„SAMEORIGIN”*, dzięki czemu możliwe jest osadzenie witryny wewnątrz elementu *iframe.*



Rysunek 13. Schemat działania zaimplementowanego mechanizmu proxy

Schemat zamieszczony na rysunku nr 13 obrazuje działanie proxy. Zakładając, że w powyższym przykładzie wyświetlany jest projekt o identyfikatorze równym *„1”*, który został skonfigurowany, by sczytywać dane z adresu *„www.xyz.com/abc.html”* to system zadziała w następujący sposób:

1. w aplikacji webowej zostanie wyświetlony element *iframe* ze źródłem ustawionym na *„/proxy/1/abc.html”,*
2. przeglądarka w celu wyświetlenie zawartości wyśle żądanie na wcześniej wspomniany adres, które trafi do filtru *ProxyPreFilter*,
3. za pomocą identyfikatora projektu zawartego w adresie żądania zostanie wyszukany odpowiedni dokument w bazie, z którego zostanie pobrany docelowy adres,
4. żądanie zostanie przekierowane na adres „*www.xyz.com/abc.html”,*
5. serwer hostujący witrynę zwróci odpowiedź zawierającą dokument *abc.html*, która trafi do filtru *ProxyPostFilter*,
6. następnie do przeglądarki trafi przetworzona odpowiedź pochodząca z proxy, więc z tej samej domeny.

4.3.6 Implementacja sczytywania danych z wykorzystaniem przeglądarki PhantomJS

Proces sczytywania danych uruchamiany jest cyklicznie. W bazie danych wyszukiwane są wszystkie aktywne projekty. Dla każdego z nich pobierane są dane z jednej podstrony. Czas pomiędzy

kolejnym uruchomieniem scraper'a można konfigurować za pomocą wyrażenia *cron*[24]. Dzięki zachowaniu odstępów czasu pomiędzy kolejnymi wczytaniami tej samej strony internetowej, serwer hostujący daną witrynę nie jest przeciążany. W przypadku ciągłego pobierania kolejnych treści, istnieje możliwość zablokowania danej usługi lub może być to uznane przez administrację portalu jako atak typu DoS (ang. *Denial of Servic)*[25].

Pobieranie danych ze strony rozpoczyna się od sprawdzenia w bazie, która podstrona powinna zostać załadowana. Następnie jest ona wczytana w przeglądarce PhantomJS. Kiedy cała struktura DOM jest już gotowa, za pomocą ścieżki w notacji *XPath*, wyszukiwany jest powtarzający się fragment strony. Zawiera on wszystkie elementy odpowiadające polom skonfigurowanym w projekcie. Dane odczytane z tego fragmentu zostaną zapisane w postaci jednego dokumentu w bazie danych. W celu pobrania wartości dla poszczególnych pól również wykorzystana została notacja *XPath*. Zapytanie nie jest jednak uruchamiane dla całej strony, a tylko dla wspomnianego wcześniej fragmentu. Za pomocą pętli pobierane są wartości dla wszystkich pól skonfigurowanych w projekcie. Przed zapisaniem ich w bazie danych sprawdzane jest czy nie istnieje już rekord o takich samych wartościach. Wszystkie te operacje powtarzane są dla kolejnych fragmentów strony. Po zakończeniu tego procesu wywoływana jest metoda odpowiedzialna za przejście do kolejnej strony. Kod metody realizującej sczytywanie danych znajduje się na listingu nr 7.

Listing 7. Metoda realizująca sczytywanie danych

```
private void fetchDataForProject(Project project, PhantomJSDriver driver) {
    ScrapingProgress progress =
            scrapingProgressRepository.findByProjectId(project.getId());

    if (isNull(progress)) {
        progress = new ScrapingProgress(project.getId(), project.getUri());
    }

    driver.get(progress.getLastUrl());
    String path = project.commonPathPart().toRootXPath();

    boolean foundSomething = false;
    for (WebElement container : driver.findElementsByXPath(path)) {
        foundSomething = true;
        DataRecord dataRecord = new DataRecord(project.getId());
        for (Field field : project.getFields()) {
            String xpath = field.relativePath().toXPath();
            try {
                WebElement element = container.findElement(By.xpath(xpath));
                String value = element.getText();
```

[24] Spring używa tej samej notacji do harmonogramowania zadań co, Unix'owe narzędzie *cron*.

[25] DoS (ang. *Denial of Servic)* – atak polegający na zakłóceniu pracy danej usługi bądź systemu, poprzez jej przeciążanie.

```
                dataRecord.getData().put(field.getName(), value);
            } catch (NoSuchElementException e) {
                log.debug("Cannot find element", e);
            }
        }
        if (!dataRecord.getData().isEmpty()
                && !existsInDataBase(project, dataRecord)) {
            dataRepository.save(dataRecord);
        }
    }
    gotoNextPage(project, driver, progress, foundSomething);
}
```

Po wykonaniu wcześniej opisanych operacji wywoływana jest metoda odpowiedzialna za przejście na kolejną podstronę i zapisanie jej adresu w bazie danych. Podczas konfiguracji projektu użytkownik może wybrać jedno z dostępnych ustawień dla tej metody:

- przechodzenie na kolejne podstrony poprzez zmianę parametru w adresie,
- przechodzenie na kolejne podstrony poprzez „kliknięcie” w element na stronie.

W pierwszym przypadku w adresie strony wyszukany zostaje parametr o nazwie podanej przez użytkownika. Następnie jego wartość zostaje zwiększona o „1”. W ten sposób generowany jest adres kolejnej podstrony.

Drugi sposób polega natomiast na symulacji „kliknięcia” w elementy strony (np. przycisk przechodzący na kolejną podstronę). W tym celu ponownie wykorzystana została notacja *XPath*. Za pomocą odpowiedniego zapytania wyszukiwany jest wybrany przez użytkownika przycisk i wywoływana jest akcja „kliknięcia”. Powoduje to załadowanie następnej podstrony w przeglądarce PhantomJS, a jej adres zapisywany jest do bazy danych.

W obu tych przypadkach, jeżeli nie został pobrany żaden rekord danych, jako kolejny adres ustawiany jest URL podany podczas konfiguracji projektu. W ten sposób, po sczytaniu treści ze wszystkich podstron, proces rozpoczyna się od początku. Pozwala to na pobranie nowych treści jeśli jakieś pojawiły się na stronie.

4.3.7 Komponenty aplikacji webowej

Część webową systemu stanowi aplikacja napisana przy użyciu frameworku Angular. Podstawowym elementem z jakich tworzy się interfejs użytkownika są komponenty. Składają się one z pliku HTML definiującego wygląd elementu oraz klasy w języku TypeScript odpowiadającej za logikę realizowaną w danym komponencie. Na potrzeby graficznego interfejsu użytkownika zostały przygotowane następujące komponenty:

- *dashboard* – stanowi widok, na którym prezentowane są wykresy dotyczące liczby pobranych rekordów danych dla projektów aktualnie zalogowanego użytkownika.

- *data-preview* – umożliwia podgląd sczytanych danych. Rekordy prezentowane są w postaci tabeli zawierającej wartości dla wszystkich pól projektu. Komponent posiada zaimplementowany mechanizm paginacji, więc pobierana i wyświetlana jest tylko cześć dostępnych danych.
- *login* – komponent będący formularzem do logowania. Zawiera logikę odpowiedzialną za wysłanie odpowiedniego żądania do serwera w celu uwierzytelnienia użytkownika.
- *main* – główny komponent aplikacji. Stanowi szablon, w którym osadzone są inne komponenty.
- *menu* – zawiera listę wszystkich projektów dostępnych dla aktualnie zalogowanego użytkownika. Kliknięcie w element z tej listy powoduje przełączenie widoku na wybrany projekt.
- *navabr* – komponent stanowiący górną część widoku w aplikacji. Zawiera nazwę aplikacji oraz przycisk umożliwiający wylogowanie się.
- *project* – zawiera interfejs graficzny odpowiedzialny za konfigurację projektu. Wewnątrz niego osadzone są komponenty *data-preview* oraz *web-preview*. W zależności od statusu projektu wyświetla podgląd danych lub formularz do edycji ustawień projektu.
- *web-preview* – umożliwia podgląd zewnętrznej strony internetowej. W tym celu wykorzystany został element *iframe* pozwalający na osadzenie podglądu innej witryn wewnątrz aplikacji. Komponent ten implementuje również logikę odpowiedzialną za wybieranie elementów na zagnieżdżonej stronie internetowej oraz generowanie ścieżek *xapth* wskazujących na nie.

Przełączanie poszczególnych widoków w aplikacji zostało zaimplementowane z wykorzystaniem mechanizmu *routingu* udostępnianego przez framework Angular. Interfejs użytkownika stanowią trzy główne widoki:

- widok dashboard'u,
- widok projektu – edycja ustawień lub podgląd danych, w zależności od statusu projektu,
- widok logowania.

Konfiguracja widoków zawiera informację o ścieżkach pod jakimi są dostępne oraz komponentach, które je implementują. Ustawienia *routingu* przedstawione są na listingu nr 8.

Listing 8. Konfiguracja widoków aplikacji

```
const appRoutes: Routes = [
        {path: '', component: MainComponent, children: [
            {path: '', redirectTo: '/dashboard', pathMatch: 'full'},
            {path: 'dashboard', component: DashboardComponent},
            {path: 'project/:id', component: ProjectComponent},
        ]},
        {path: 'login', component: LoginComponent}
];
```

5. Testy przygotowanego prototypu

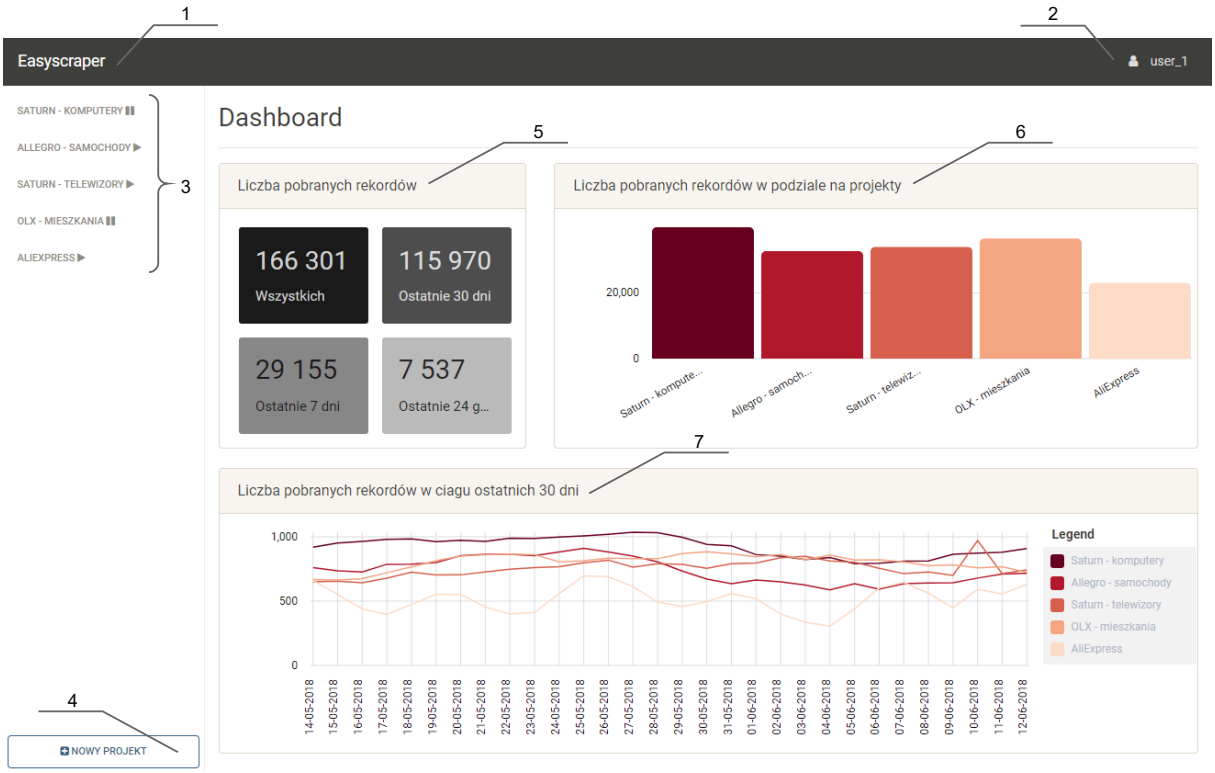
W tym rozdziale omówione zostało działanie stworzonego w ramach pracy prototypu. Pierwszy podrozdział stanowi przegląd graficznego interfejsu aplikacji, zawiera opis poszczególnych widoków prezentowanych użytkownikowi. Druga część zawiera przykład działania systemu dla prawdziwej witryny internetowej, z której pobierane są dane.

5.1. Przegląd graficznego interfejsu aplikacji

Zgodnie z wymaganiami, opracowywany system powinien być dostępny za pośrednictwem aplikacji webowej. Do korzystania z niej wymagana jest jedynie, zainstalowana przeglądarka internetowa. Użytkownik otwierając adres aplikacji, najpierw proszony jest o zalogowanie się. Formularz służący do autentykacji został zaprezentowany na rysunku nr 14.

Rysunek 14. Formularz logowania

W przypadku podania błędnej kombinacji hasła oraz loginu użytkownik informowany jest o tym stosowanym komunikatem i może spróbować ponownie się zalogować. Gdy podane dane są prawidłowe wyświetlany jest domyślny widok pulpitu użytkownika (rysunek nr 15).



Rysunek 15. Widok pulpitu użytkownika

Widok pulpitu użytkownika, widoczny na rysunku nr 15, zawiera następujące elementy:

1. Nazwa systemu – „*Easyscraper*” od słów „*easy*” czyli „*łatwy*” oraz „*scraper*” czyli „*skrobak*”. Nawiązuje do pojęcia „*web scrapingu*” i sugeruje, że praca w przedstawionym systemie powinna być prosta, nawet dla użytkowników nie posiadających zaawansowanej wiedzy na temat działania stron internetowych.
2. Nazwa aktualnie zalogowanego użytkownika – kliknięcie w nią umożliwia wylogowanie się.
3. Lista projektów użytkownika – ikona obok nazwy informuje, czy dany projekt jest w statusie aktywnym (pobierane są dane) czy w trybie edycji (pobieranie danych jest zatrzymane). Kliknięcie w wybrany element powoduje załadowanie widoku projektu.
4. Przycisk pozwalający na utworzenie nowego projektu – naciśnięcie go powoduje stworzenie nowego, pustego projektu i przejście do jego edycji.
5. Komponent prezentujący liczbę pobranych rekordów z danymi – zliczane są wszystkie rekordy pobrane dla projektów użytkownika – od początku, w ciągu ostatnich 30 dni, 7 dni oraz ostatnich 24 godzin.
6. Wykres słupkowy pokazujący liczbę pobranych rekordów w podziale na poszczególne projekty.
7. Wykres liniowy przedstawiający liczbę rekordów danych pobranych w danym dniu dla danego projektu. Zawiera informacje na temat ostatnich 30 dni.

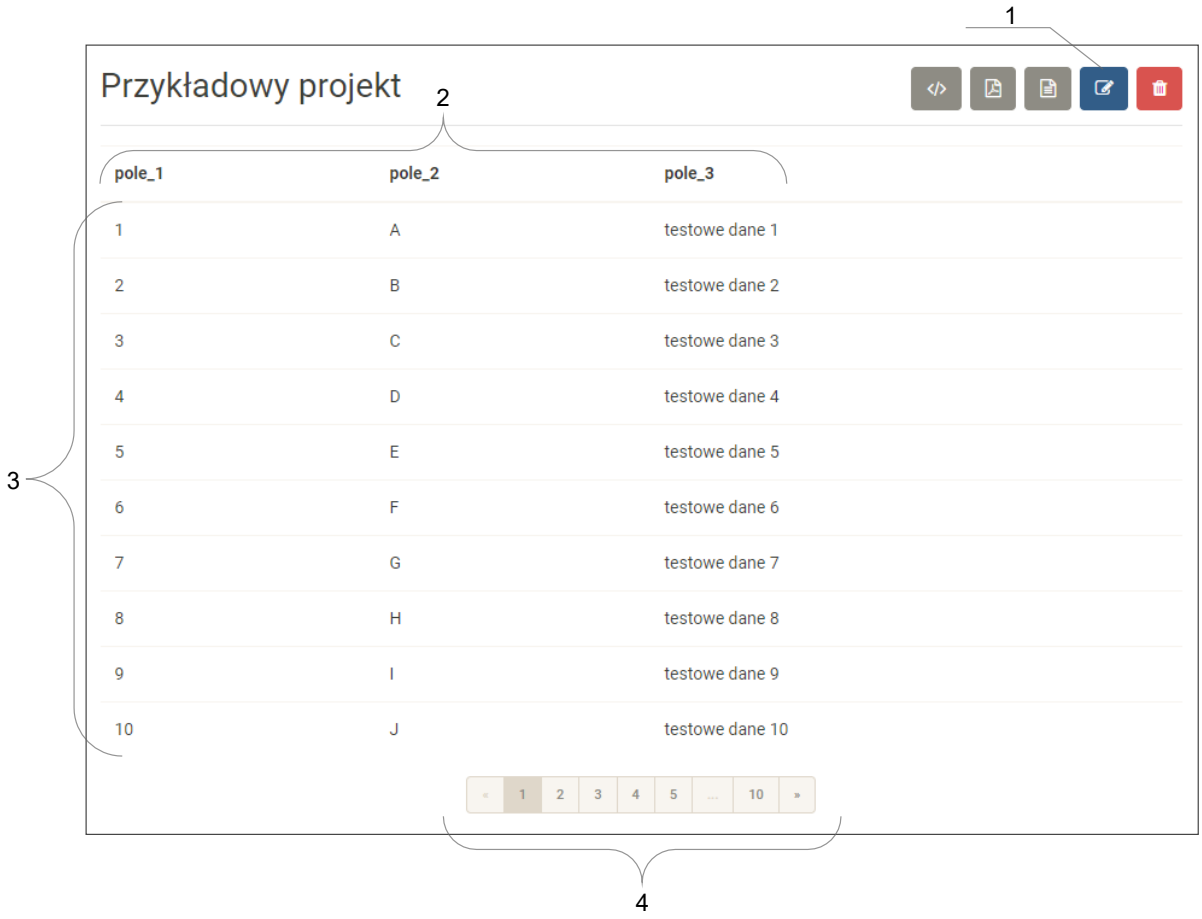
Utworzenie nowego projektu lub wybranie istniejącego powoduje przełączenie widoku. Jeżeli projekt jest w trybie edycji, w miejscu *dashboard'u* zostaje wyświetlony formularz pozwalający na zmianę ustawień oraz podgląd sczytywanej strony (rysunek nr 16). W przypadku, gdy wybrany projekt jest aktywny wyświetlona zostaje tabele zawierająca pobrane rekordy (rysunek nr 17).

Rysunek 16. Widok edycji projektu

Każda zmiana dokonana w trybie edycji zapisywana jest automatycznie do bazy danych, a użytkownik może w dowolnym momencie przełączyć widok na inny projekty. Poniżej znajduje się lista elementów interfejsu jakie dostępne są dla użytkownika, podczas konfigurowania projektu:

1. Pole tekstowe służące do zmiany nazwy tego projektu.
2. Pole tekstowe, w które należy prowadzić adres sczytywanej strony internetowej. Po jego wypełnieniu wyświetlany jest podgląd witryny.
3. Formularz umożliwiający dodawanie nowych pól. Po podaniu nazwy oraz wciśnięciu przycisku *„DODAJ”* użytkownik musi wskazać element witryny, który go interesuje. Kiedy wybierze dwa odpowiadające sobie fragmenty strony. Na ich podstawie generowana jest ścieżka *xpath*, która zostanie zapisana w konfiguracji projektu.
4. Lista skonfigurowanych pól. Wskazanie kursorem wybranego pola powoduje podświetlenie na podglądzie odpowiadających mu fragmentów strony. Natomiast za pomocą znaku *„x”* możliwe jest usunięcie wybranego pola.
5. Formularz odpowiadający za konfigurację przechodzenia pomiędzy kolejnymi podstronami. Możliwe są trzy ustawienia:
    a. brak paginacji – sczytywana jest tylko jedna strona,
    b. przechodzenie do kolejnych podstrona za pomocą symulacji wciśnięcia wybranego przycisku,
    c. modyfikacja adresu *url* poprzez zmianę wartości parametru o podanej nazwie.
6. Podgląd sczytywanej strony zaimplementowany z wykorzystaniem elementu *iframe* oraz przygotowanego *proxy*.

7. Przycisk służący do wyświetlenie informacji o API, za pomocą którego można pobrać sczytane dane. Po jego kliknięciu wyświetlane jest okno zawierające przykładowe żądanie, odpowiedź oraz opis parametrów.
8. Przycisk umożliwiający pobranie sczytanych danych w postaci pliku PDF.
9. Przycisk umożliwiający pobranie sczytanych danych w postaci pliku CSV.
10. Przycisk służący do aktywacji projekty. Po jego kliknięciu zmieniany jest status projektu i włączone zostaje sczytywanie danych.
11. Przycisk pozwalający na usunięcie projektu.



Rysunek 17. Podgląd sczytanych danych dla aktywnego projektu

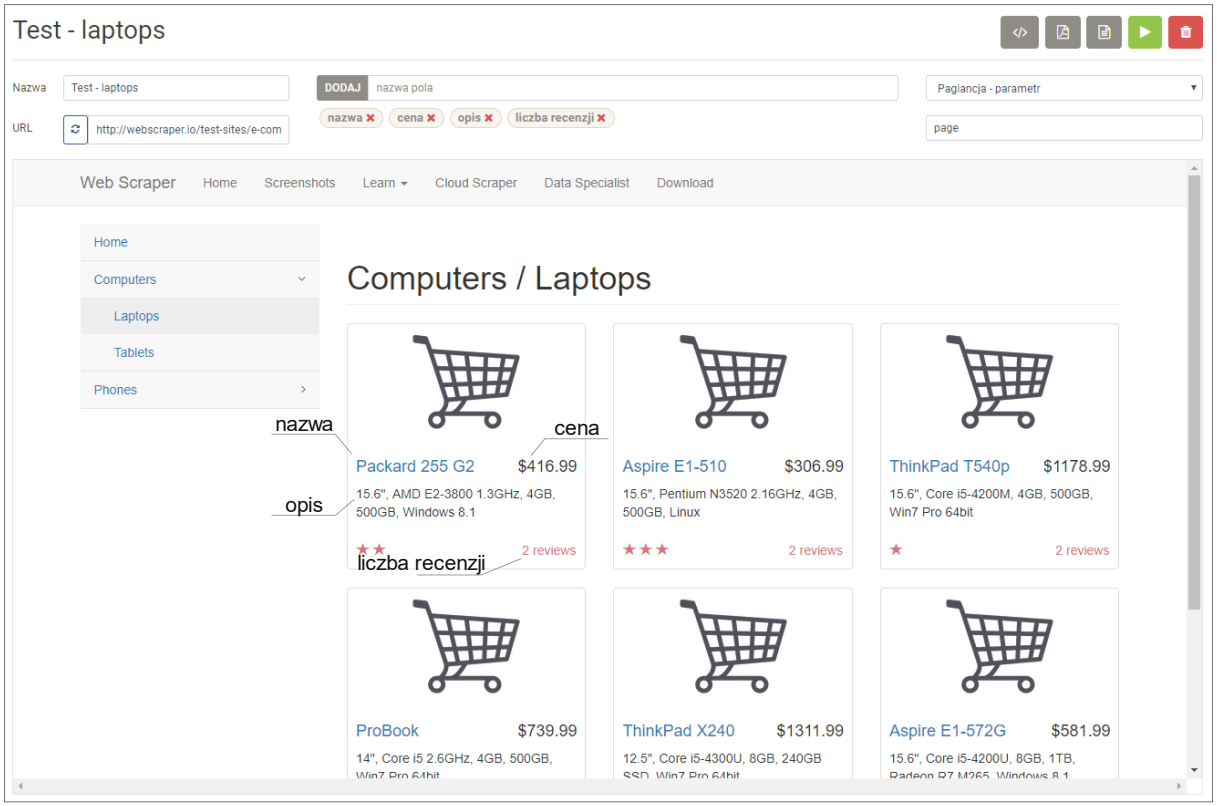
Aktywacja projektu powoduje rozpoczęcie sczytywania danych. Podgląd pobranych rekordów włącza się automatycznie po wybraniu aktywnego projektu z listy. Poniższa lista zawiera opis elementów tego widoku:

1. Zamiast przycisku aktywacji, widoczny jest przycisk przełączający projekt do trybu edycji. Pozostałe przyciski pozostają te same jak w przypadku widoku edycji.
2. Dane przedstawione są w tabeli, nazwy kolumn odpowiadają polom skonfigurowanym dla projektu.
3. Poszczególne wiersze tabeli zawierają rekordy pobrane dla wybranego projektu.
4. Widoczna jest tylko część dostępnych danych, u dołu strony znajduje się element umożliwiający wyświetlanie kolejnych stron z danymi.

5.2. Działanie systemu dla przykładowej strony internetowej

Przygotowany prototyp został przez autora przetestowany na kilku różnych stronach internetowych. Ponieważ jednak publikowanie danych pobranych metodą *web scraping'u*, może budzić wątpliwości natury prawnej. Zawarte w tym rozdziale testy przeprowadzone zostały na witrynie specjalnie do tego stworzonej[26].

Testowa strona dostępna jest pod adresem *http://webscraper.io/test-sites/e-commerce/static*. Zbudowana jest tak, by przypominać sklep internetowy. Dostępnych na niej jest kilka kategorii produktów – telefony, tablety oraz laptopy. Na pojedynczej podstronie wyświetlanych jest jedynie sześć pozycji, by wyświetlić resztę należy użyć linków do przechodzenia pomiędzy kolejnymi podstronami. Celem tego testu było sczytanie danych dotyczących laptopów. Ze strony miały zostać pobrane rekordy zawierające takie dane jak nazwa, opis, cena oraz liczba recenzji produktu. Na potrzeby tego testu został przygotowany nowy projekt o konfiguracji widocznej na rysunku 18.



Rysunek 18. Konfiguracja sczytywania danych dla przykładowej strony

Ustawienia projektu:

- URL - http://webscraper.io/test-sites/e-commerce/static/computers/laptops,
- Sczytywane pola – *nazwa*, *opis*, *cena* oraz *liczba recenzji*,
- Przechodzeni pomiędzy podstronami poprzez zmianę parametru o nazwie – *page*.

[26] http://webscraper.io/test-sites - pod tym adresem znajdują się linki do stron, stworzonych specjalnie na potrzeby nauki korzystania z oprogramowania webscraper.io.

Po zakończonej konfiguracji, projekt został aktywowany, spowodowało to rozpoczęcie sczytywania danych. Na przykładowej stronie znajdowało się 117 produktów (19 stron po 6 produktów oraz ostatnia dwudziesta strona, na której znajdowały się tylko trzy rekordy). Po dwudziestokrotnym uruchomieniu procesu sczytywania, dane ze wszystkich podstron zostały pobrane. Tabela nr 3 zawiera pierwsze dwanaście rekordów:

Tabela 3. Dane pobrane dla przykładowej strony

nazwa	**cena**	**opis**	**liczba recenzji**
Packard 255 G2	$416.99	15.6", AMD E2-3800 1.3GHz, 4GB, 500GB, Windows 8.1	2 reviews
Aspire E1-510	$306.99	15.6", Pentium N3520 2.16GHz, 4GB, 500GB, Linux	2 reviews
ThinkPad T540p	$1178.99	15.6", Core i5-4200M, 4GB, 500GB, Win7 Pro 64bit	2 reviews
ProBook	$739.99	14", Core i5 2.6GHz, 4GB, 500GB, Win7 Pro 64bit	8 reviews
ThinkPad X240	$1311.99	12.5", Core i5-4300U, 8GB, 240GB SSD, Win7 Pro 64bit	12 reviews
Aspire E1-572G	$581.99	15.6", Core i5-4200U, 8GB, 1TB, Radeon R7 M265, Windows 8.1	2 reviews
ThinkPad Yoga	$1033.99	12.5" Touch, Core i3-4010U, 4GB, 500GB + 16GB SSD Cache,	13 reviews
Pavilion	$609.99	15.6", Core i5-4200U, 6GB, 750GB, Windows 8.1	4 reviews
Inspiron 15	$745.99	Moon Silver, 15.6", Core i7-4510U, 8GB, 1TB, Radeon HD R7 M265 2GB,	12 reviews
Dell XPS 13	$1281.99	13.3" Touch, Core i5-4210U, 8GB, 128GB SSD, Windows 8.1	4 reviews
ThinkPad X230	$1244.99	12.5", Core i5 2.6GHz, 8GB, 180GB SSD, Win7 Pro 64bit	10 reviews
HP 250 G3	$520.99	15.6", Core i5-4210U, 4GB, 500GB, Windows 8.1	13 reviews

Rekordy z tabeli nr 3 zostały porównane z dwiema pierwszymi podstronami testowej witryny. Pierwsze dwanaście sczytanych produktów było identyczne jak na stronie użytej w tym teście. Oznacza to, że przeprowadzony test zakończył się sukcesem.

6. Podsumowanie

Niniejsza praca dotyczyła przygotowania koncepcji systemu służącego do sczytywania danych z portali internetowych oraz implementacji jego prototypu. Powyższe cele zostały pomyślnie zrealizowane. Zaimplementowany system realizuje wymagania postawione w tej pracy. Istnieją jednak dużo możliwości rozwoju jego funkcjonalności. W dalszej części tego rozdziału znajduje się krótka ocena wyników pracy, a także plany dotyczące dalszego rozwoju zaimplementowanego systemu.

6.1. Ocena opracowanego rozwiązania

Opracowany prototyp spełnia postawione założenia. Umożliwia pobieranie danych z dynamicznych stron internetowych – obsługa skryptów i technologii AJAX odbywa się tak jak w zwykłej przeglądarce. Osiągnięcie tego było możliwe dzięki zastosowaniu narzędzia PhantomJS. Dzięki zastosowaniu podziału na część frontendową oraz backendową, proces pobierania danych nie jest zależny od tego czy użytkownik ma włączoną aplikację kliencką, wszystkie operacje wykonywane są po stronie serwera.

Konfiguracja sczytywania danych dla nowej witryny jest prosta i sprowadza się do wskazywania przez użytkownika interesujących go treści na podglądzie wybranej strony. Osoba korzystająca z aplikacji nie musi posiadać wiedzy na temat funkcjonowania stron internetowych ani tworzyć żadnych dodatkowych skryptów czy zapytań.

Dzięki przygotowaniu REST'owego API możliwa jest integracja systemu z innymi narzędziami, może stać się np.: źródłem danych dla narzędzi analitycznych. Prototyp posiada również opcję eksportu zebranych danych do pliku. Rekordy zapisane w formacie CSV mogę być zaimportowane do wielu różnych programów.

Uruchamiane cyklicznie pobieranie danych, dla pojedynczej podstrony, powoduje że wyniki dostępne są dopiero po pewnym czasie. Mimo wszystko zastosowanie takiego rozwiązania było konieczne by nie przeciążać maszyn hostujących sczytywane witryny. Dzięki temu nie zostaną przekroczone limity żądań, a tym bardziej nie zostanie zakłócona praca serwerów.

Pomimo wymienionych powyżej zalet, przygotowany prototyp posiada dość istotną wadę - pobiera jedynie dane tekstowe. Obecnie strony internetowe składają się także z ogromnej ilości grafik czy materiałów wideo, których nie można pobrać z wykorzystaniem zaimplementowanego systemu.

Inną wadą prototypu jest zaimplementowanie jedynie prostego mechanizmu do przechodzenia pomiędzy kolejnymi podstronami z danymi. Przy jego wykorzystaniu nie można np.: sczytać różnych kategorii dostępnych jako linki w menu czy pobrać szczegółów dostępnych na innej podstronie. Ograniczenie to było jednak konieczne by w pojedynkę zaimplementować funkcjonalny prototyp.

6.2. Możliwe kierunki rozwoju

Podczas rozwoju systemu, w pierwszej kolejności warto byłoby się zająć wyeliminowaniem wymienionych wcześniej wad. Implementacja sczytywania obrazków nie powinna wymagać znacznych zmian w kodzie prototypu. Dużo trudniejsze mogła by okazać się implementacja mechanizmu służącego

do przechodzenia pomiędzy różnymi podstronami. Przede wszystkim należało by dobrze zaprojektować interfejs użytkownika, be system wciąż spełniał założenia bycia łatwym w użyciu. Mogło by być to zrealizowane np.: w postaci kreatora gdzie użytkownik z gotowych komponentów budował by proces sczytywania strony internetowej. Elementami takiego procesu mogło by być symulowanie kliknięcia w link, wypełnienie formularza, przejście do kolejnego fragmentu strony, pobranie tekstu z elementu czy zapisanie wybranego obrazka. Poniższa lista zawiera kilka innych usprawnień, które mogły by znacznie poprawić funkcjonalność systemu:

- możliwość wybierania pola, które jest identyfikatorem całego rekordu. Pozwoliło by to na aktualizację już sczytanych danych,
- dynamiczne odświeżanie widoku w przypadku pobrania nowych danych,
- eksport do innych formatów plików (np.: XLS lub XML),
- zmianę architektury systemu, w ten sposób by wydzielić proces sczytujący dane do oddzielnej aplikacji. Dzięki temu możliwe byłoby zwiększenie wydajności poprzez uruchomienie jego wielu instancji i równoległe sczytywanie wielu stron,
- zarządzanie ilością danych jakie pobierają poszczególni użytkownicy. Jest to konieczne jeżeli system miałby być użytkowany przez wiele osób. Bez takiej kontroli szybko mogło by dojść do przeciążenia systemu.

Bibliografia

[1] Hartley Brody *The Ultimate Guide To Web Scraping*, Leanpub 2017

[2] HTML 5.2 W3C Recommendation, https://www.w3.org/TR/html/ [Dostęp: 12.05.2018]

[3] Java Platform, Standard Edition 8 API Specification, https://docs.oracle.com/javase/8/docs/api/ [Dostęp: 19.05.2018]

[4] Screen scraping w bankowości, http://otwieramkonto.pl/blog/screen-scraping-bankowosc [Dostęp: 19.05.2018]

[5] Ustawa z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. 1994 nr 24 poz. 83)

[6] Ustawa z dnia 27 lipca 2001 r. o ochronie baz danych (Dz.U. 2001 nr 128 poz. 1402)

[7] Ustawa z dnia 10 maja 2018 r. o ochronie danych osobowych (Dz.U. 2018 poz. 1000)

[8] Web Scraper Chrome Extension, https://chrome.google.com/webstore/detail/web-scraper/jnhgnonknehpejjnehehllkliplmbmhn [Dostęp: 16.06.2018]

[9] Dexi.io, https://dexi.io/ [Dostęp: 06.02.2018]

[10] ParseHub, https://www.parsehub.com/ [Dostęp: 06.02.2018]

[11] Import.io, https://www.import.io/ [Dostęp: 06.02.2018]

[12] Java, https://www.oracle.com/pl/java/index.html [Dostęp: 16.06.2018]

[13] Saumont Pierre-Yves, *Java. Programowanie funkcyjne*, Helion 2017, ISBN 978-83-283-3324-6, 9788328333246

[14] TIOBE Index, https://www.tiobe.com/tiobe-index/ [Dostęp: 19.05.2018]

[15] Spring Framework, https://spring.io/ [Dostęp: 19.05.2018]

[16] Walls Craig *Spring w akcji*, Helion 2015, ISBN 978-83-283-0849-7, 9788328308497

[17] Spring Boot, https://spring.io/projects/spring-boot [Dostęp: 16.06.2018]

[18] Spring MVC, https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html [Dostęp: 16.06.2018]

[19] Warin Geoffroy *Spring MVC 4. Projektowanie zaawansowanych aplikacji WWW*, Helion 2016, ISBN 978-83-283-2347-6, 9788328323476

[20] Spring Data, https://spring.io/projects/spring-data [Dostęp: 16.06.2018]

[21] Spring Security, https://spring.io/projects/spring-security [Dostęp: 16.06.2018]

[22] MongoDB, https://www.mongodb.com/ [Dostęp: 16.06.2018]

[23] PhantomJS, http://phantomjs.org/ [Dostęp: 03.03.2018]

[24] GhostDriver, https://github.com/detro/ghostdriver [Dostęp: 03.06.2018]

[25] Zuul, https://github.com/Netflix/zuul [Dostęp: 03.06.2018]

[26] Jsoup, https://jsoup.org/ [Dostęp: 03.06.2018]

[27] Project Lombok, https://projectlombok.org/ [Dostęp: 03.06.2018]

[28] Swagger, https://swagger.io/ [Dostęp: 03.06.2018]
[29] TypeScript, https://www.typescriptlang.org/ [Dostęp: 03.06.2018]
[30] Angular, https://angular.io/ [Dostęp: 03.06.2018]
[31] Cascading Style Sheets, https://www.w3.org/Style/CSS/Overview.en.html, [Dostęp: 16.06.2018]
[32] Bootstrap, https://getbootstrap.com/ [Dostęp: 03.06.2018]
[33] Frain Ben *Responsive Web Design. Projektowanie elastycznych witryn w HTML5 i CSS3*, Helion 2016, ISBN: 978-83-283-2343-8, 9788328323438
[34] Git, https://git-scm.com/ [Dostęp: 03.06.2018]
[35] Gitlab, https://about.gitlab.com/ [Dostęp: 03.06.2018]
[36] Maven, https://maven.apache.org/ [Dostęp: 03.06.2018]
[37] NPM, https://www.npmjs.com/ [Dostęp: 17.06.2018]
[38] IntelliJ IDEA, https://www.jetbrains.com/idea/ [Dostęp: 17.06.2018]
[39] MongoDB Manual, https://docs.mongodb.com/manual/ [Dostęp: 03.06.2018]

Spis listingów

Spis rysunków

Spis tabel