



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Paweł Połczyński

Nr albumu 16054

**Elastyczny skraper stron WWW zrealizowany jako usługa
internetowa**

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, czerwiec 2018

Streszczenie

Praca dotyczy zagadnienia pozyskiwania wiedzy z nieustrukturyzowanych źródeł danych. Informacje wydobyte ze stron WWW, aby mogły być wykorzystane w aplikacjach, wymagają przetworzenia. Procesorem dla stron internetowych są *skrapery*. Skraper to narzędzie służące do przetwarzania struktur *HTML* w celu wyodrębnienia z nich interesujących informacji. Szereg dostawców na rynku udostępnia usługę utworzenia skrapera na zamówienie klienta, ale tylko pojedyncze firmy oferują możliwość samodzielnego wygenerowania skrapera za pomocą udostępnionych narzędzi. W większości przypadków narzędzia te nie są doskonałe i nie nadają się do użytku dla osób bez zaawansowanej wiedzy technicznej. Większość skrapersów nadal tworzonych jest przez programistów. Wytwarzane skrapery zazwyczaj na najniższym poziomie bazują na selektorach *XPATH* lub *CSS*. Ze względu na konstrukcję skrapery muszą być nieustannie aktualizowane, żeby nadążyć za ciągle zmieniającymi się stronami WWW. W związku z koniecznością angażowania programistów rozwiązania te są kosztowne i czasochłonne.

Z powodów wskazanych powyżej zrodziła się koncepcja zdefiniowania i zaprojektowania narzędzia, które w sposób prosty umożliwi utworzenie skrapera. Aplikacja skierowana jest do użytkowników nieposiadających wiedzy technicznej. W przystępny sposób umożliwi im stworzenie podstawowego skrapera i skorzystanie z niego. Jednocześnie udostępni ona zaawansowane opcje dla programistów. Prototyp powinien umożliwić łatwe rozszerzanie o nowe funkcjonalności przez programistów i umożliwić jego integrację z zewnętrznymi aplikacjami. Prototyp został zaimplementowany w języku *JavaScript*.

Słowa kluczowe:

Nieustrukturyzowane Źródła Danych, Web Scraping, Strony Internetowe

Spis treści

1.	WSTĘP	5
1.1.	ELASTYCZNY SKRAPER STRON WWW JAKO USŁUGA	5
1.2.	CEL PRACY	6
1.3.	REZULTATY PRACY	6
1.4.	ORGANIZACJA PRACY	7
2.	ISTNIEJĄCE ROZWIĄZANIA	8
2.1.	PLATFORMA GREPSR.COM.....	8
2.2.	PLATFORMA SCRAPEHERO.COM	9
2.3.	DATAHEN.COM.....	10
2.4.	SCRAPINGHUB.COM.....	10
2.5.	OCTOPARSE.COM.....	11
2.6.	WEBSCRAPER.IO.....	12
2.7.	PARSEHUB.COM.....	12
2.8.	DEXLIO	12
2.9.	PODSUMOWANIE ISTNIEJĄCYCH ROZWIĄZAŃ.....	13
3.	PROPOZYCJA ROZWIĄZANIA.....	14
3.1.	OPIS ARCHITEKTURY	14
3.2.	APLIKACJA INTERNETOWA.....	15
3.3.	SKRAPER.....	19
3.4.	API	21
3.4.1	User.....	21
3.4.2	Project.....	25
3.4.3	Scraper.....	31
3.4.4	Execute.....	36
4.	JĘZYKI, METODYKI, NARZĘDZIA ZASTOSOWANE W PRACY	39
4.1.	JAVASCRIPT.....	39
4.2.	TYPESCRIPT.....	39
4.3.	HTML	40
4.4.	CSS	40
4.5.	HTTP	40
4.6.	REST.....	41
4.7.	NODEJS	42
4.8.	EXPRESSJS.....	43
4.9.	POSTMAN.....	43
4.10.	WINSTON.....	44
4.11.	MORGAN.....	44
4.12.	ANGULAR 5	44
4.13.	BOOTSTRAP	45
5.	IMPLEMENTACJA	47
5.1.	KLIENT	47
5.2.	BAZA DANYCH.....	48
5.3.	KONFIGURACJA APLIKACJI.....	49
5.4.	LOGOWANIE	50
5.5.	SERWER.....	50
6.	ZASTOSOWANIE PROTOTYPU	55
6.1.	SPOSODY UŻYCIA PORTALU	55
6.2.	SKRAPER.....	68

6.3.	PRZYKŁADOWY SKRAPER ZBUDOWANY ZA POMOCĄ PROTOTYPU.....	69
7.	PODSUMOWANIE.....	76
7.1.	WADY I ZALETY ROZWIĄZANIA	76
7.2.	KIERUNKI ROZWOJU PROTOTYPU	77
7.3.	WNIOSKI KOŃCOWE	78

1. Wstęp

XXI wiek nazywany jest wiekiem informacji. Ilość bodźców i komunikatów przerosła ludzkie możliwości percepcji i analizy nawet w życiu codziennym. W celu przetwarzania danych stosowane są komputery. Pozyskiwanie wiedzy wciąż nie jest zadaniem prostym. Powstało wiele projektów mających na celu umożliwienie dostarczenia ustrukturyzowanej informacji do komputerów, takich jak na przykład *wikidata.org*. Jednak w wielu przypadkach, by uzyskać szczegółowe treści potrzebne do zasilenia aplikacji, niezbędne jest wciąż przetworzenie zasobów łatwo dostępnych dla człowieka, lecz niezrozumiałych dla maszyn. Z takiej potrzeby zrodziła się koncepcja pracy. Poniższe podrozdziały są wprowadzeniem do problemu ekstrakcji wiedzy z nieustrukturyzowanych źródeł.

1.1. Elastyczny skroper stron WWW jako usługa

Jednym z kanałów zdobywania użytecznych danych są strony internetowe. Fora, portale i aplikacje webowe zawierają informacje, które niosą wartość biznesową i mogą okazać się cenne w automatyzacji procesów, analizie rynkowej i tym podobnym.

Skroperem¹ nazywamy program komputerowy przetwarzający strony internetowe. Skrapery operują na kodzie HTML strony, wybierając z niej interesujące szczegóły. Wydobyte w ten sposób dane mogą zostać łatwo zinterpretowane przez zewnętrzne aplikacje i użyte w określonym celu. Proces umożliwia zasilanie systemów w wiedzę.

Z użytkowaniem skrapersów wiążą się kwestie prawne z obszaru ochrony własności intelektualnej. Na terenie Rzeczypospolitej Polskiej treści publikowane na stronach internetowych są chronione przez prawo o dostępie do baz danych [2]. Jest to dostosowanie polskiego prawa do regulacji Unii Europejskiej [3]. Zgodnie z ustawą z dnia 27 lipca 2001 r. o ochronie baz danych działalność taka jest niedozwolona: „Nie jest dozwolone powtarzające się i systematyczne pobieranie lub wtórne wykorzystanie sprzeczne z normalnym korzystaniem i powodujące nieusprawiedliwione naruszenie słusznych interesów producenta.” [2]. Kwestie powyższe na terenie innych krajów regulowane są na mocy praw lokalnych. Działalność taka jest legalna na przykład w Stanach Zjednoczonych Ameryki Północnej [4].

Na potrzeby pracy skorzystano z zapisu „Art. 8. 1. Wolno korzystać z istotnej, co do jakości lub ilości, części rozpowszechnionej bazy danych: 1) do własnego użytku osobistego, ale tylko z

¹ Skroper spolszczona wersja angielskiego słowa *scraper*. Czasownik *to scrap* oznacza czynność dekompozycji przedmiotów, najczęściej wykonanych z metalu, szczególnie w celu jego ponownego wykorzystania [1]. Rzeczownik *scraper* oznacza narzędzie do wykonywania tej czynności. W związku z brakiem bezpośredniego odpowiednika w języku polskim zdecydowano o użyciu w ramach pracy wersji spolszczonej słowa.

zawartości nieelektronicznej bazy danych, 2) w charakterze ilustracji, w celach dydaktycznych lub badawczych, ze wskazaniem źródła, jeżeli takie korzystanie jest uzasadnione niekomercyjnym celem, dla którego wykorzystano bazę,” [2]

Osobiste doświadczenia autora związane z optymalizacją procesów biznesowych oraz ze zrobotyzowaną automatyzacją procesów wskazują na konieczność rozwoju technologii, które są w stanie wyodrębnić wsad z nieustrukturyzowanych źródeł. Z tych doświadczeń zrodził się pomysł utworzenia systemu umożliwiającego osobie nieposiadającej ani wykształcenia, ani wiedzy technicznej, utworzenia tak zwanego skrapera WWW. System ma pozwolić na wytworzenie skrapera przy użyciu prostego interfejsu użytkownika, który będzie wykorzystany na przykład w aplikacjach klienckich.

W niniejszej pracy została przedstawiona i omówiona propozycja rozwiązania próbująca odpowiedzieć na takie właśnie zapotrzebowanie.

1.2. Cel pracy

Celem pracy jest omówienie problemów związanych ze skrapowaniem stron WWW, czyli pozyskaniem wiedzy z nieustrukturyzowanych źródeł informacji w postaci stron internetowych. Prezentacja i dyskusja zaproponowanego rozwiązania.

Dokument ma też na celu wskazanie cech, którymi powinny charakteryzować się narzędzia umożliwiające skrapowanie, by były w pełni użyteczne. W ramach analizy problemu wytworzono narzędzie umożliwiające definicję skraparów.

Praca przedstawia funkcjonalny prototyp odpowiadający na postawione w niej problemy oraz wskazuje przyszły kierunek rozwoju prezentowanego oprogramowania. Podjęta została również próba wskazania potencjalnych wad oraz zalet zaproponowanego systemu.

1.3. Rezultaty pracy

Rezultatem pracy jest prototyp aplikacji webowej będący elastycznym skraperem stron WWW. Aplikacja webowa umożliwi pobieranie informacji ze stron internetowych za pomocą wygenerowanego API lub poprzez udostępnienie pliku JSON. Skraper i API są utworzone za pomocą interfejsu użytkownika umożliwiającego definiowanie skrapera poprzez wskazanie interesujących elementów strony internetowej za pomocą wskaźnika myszki.

1.4. Organizacja pracy

W pierwszych dwóch rozdziałach pracy omówiono problem skrapowania stron WWW oraz przedstawiono dostępne na rynku narzędzia. Przeprowadzono analizę wad i zalet wybranych systemów.

W trzecim rozdziale omówiono propozycje rozwiązania problemów postawionych w pracy. Przedstawiono propozycje prototypu systemu. Pokazano jego możliwości oraz zestawiono je z istniejącymi już produktami.

Kolejny rozdział opisuje technologie, które wykorzystano podczas tworzenia prototypu.

Następny rozdział opisuje przyjęte konwencje oraz szczegóły implementacji przedstawionego prototypu.

Rozdział szósty prezentuje przykładowe użycie przygotowanego narzędzia do tworzenia skrapatorów.

W ostatnim rozdziale podsumowano pracę. Przedstawiono wady i zalety zaproponowanego rozwiązania i omówiono możliwości rozwoju aplikacji.

2. Istniejące rozwiązania

W rozdziale zostały opisane konkurencyjne rozwiązania i dostępne na rynku produkty wpisujące się w naturę prezentowanego problemu. W poniższych podrozdziałach opisano systemy, które cechują się pewną innowacyjnością i dostępnością dla użytkownika.

2.1. Platforma grepsr.com

Platforma *greprs.com* oferuje zarejestrowanym użytkownikom możliwość opisanie wymagań do zakresu pozyskiwanych danych, z których pracownicy firmy tworzą skraper [5]. Program jest uruchamiany zgodnie z zadaną częstotliwością. Rezultaty skrapowania wysyłane są do klientów za pomocą poczty elektronicznej. W bardziej zaawansowanej wersji usługi dane mogą być umieszczone na zdalnym zasobie wspierającym *FTP* lub na przykład na *Google Drive* w postaci wybranego typu pliku tekstowego.

Wymagania odnośnie zakresu skrapowanych danych można przesyłać za pomocą opisu, jak również zrzutu ekranu z zaznaczonymi elementami. Dodatkowo firma oferuje usługę skrapowania po stronie klienta za pomocą rozszerzenia do przeglądarki. Dzięki zainstalowanemu rozszerzeniu możliwe jest skrapowanie poprzez kliknięcia myszki. Rozwiązanie to nie jest darmowe i wymaga rejestracji w programie. Pobieranie większych ilości danych jest dodatkowo płatne. Wykupując najwyższy plan usługi platforma umożliwia skorzystanie z *API* w celu pobrania wyników skrapowania, jednak nie umożliwia uruchomienia skrapera z jego pomocą.

Zalety:

- Użycie nie wymaga umiejętności technicznych oraz zrozumienia struktur *HTML*;
- Skrapowanie odbywa się po stronie platformy *greprs*. Wymusza to na firmie *greprs* odpowiedzialność za zapewnienie działania skrapera oraz, że nie zostanie on zablokowany przez administratorów skrapowanej strony;
- Firma *greprs* odpowiada za jakość skrapera, a na wyższym poziomie usługi odpowiada też za jego utrzymanie i dostosowanie do zmian na stronie.

Wady:

- Brak kontroli nad wykonaniem skrapera – brak możliwości wywołania skrapera *ad hoc*;
- Możliwość zmiany skrapera tylko poprzez pośrednictwo firmy autorskiej;

- Tylko najwyższy plan taryfowy świadczenia usługi umożliwia dostęp do pozyskanych informacji za pośrednictwem API. Tańsze wersje usługi udostępniają treści wyłącznie jako pliki tekstowe.

2.2. Platforma scrapehero.com

Platforma *Scrapehero* prowadzona jest przez firmę specjalizującą się w skrapowaniu sklepów internetowych. Umożliwia ona monitorowanie dostępności oraz cen wskazanych produktów. Zarejestrowany użytkownik może zamówić skrapera pobierający dane konkretnych sklepów i konkretnych produktów [6]. Platforma skupiając się na pozyskiwaniu danych o produktach dostarcza dodatkowe usługi umożliwiające wizualizację trendów cen i dostępności artykułów, co ułatwia podejmowanie strategicznych decyzji kierownictwu i obsłudze sklepów internetowych. Dodatkową usługą jest monitorowanie cen i możliwość otrzymywania ostrzeżeń o ich zmianie. Platforma umożliwia skrapowanie *real-time* za pomocą API, które można zamówić do utworzonego skrapera. Na stronie firmy brakuje przybliżonych kosztów utworzenia skrapera i jego eksploatacji.

Zalety:

- Użycie nie wymaga umiejętności technicznych oraz zrozumienia struktur *HTML*;
- Skrapowanie odbywa się po stronie platformy *scrapehero*. Wymusza to na firmie *scrapehero* odpowiedzialność za zapewnienie działania skrapera oraz że nie zostanie on zablokowany przez administratorów skrapowanej strony;
- Firma *scrapehero* odpowiada za jakość skrapera, a na wyższym poziomie usługi odpowiada też za jego utrzymanie i dostosowanie do zmian na stronie;
- Dostępne są dodatkowe usługi umożliwiające lepsze prowadzenie sklepu *e-commerce* poprzez analizę rynku oraz ostrzeżenia o zmieniających się cenach;
- Szeroka gama typów danych wyjściowych.

Wady:

- Wysoka specjalizacja;
- Ograniczony zakres usług;
- API jako dodatkowa usługa.

2.3. datahen.com

Kanadyjska firma udostępniająca usługi skrapowania stron WWW. Oferuje możliwość zbudowania skrapera dla wybranych stron internetowych na żądanie klienta [7]. Natomiast w opcji *Enterprise* udostępnia platformę dla developerów umożliwiającą budowanie własnych skrapery. W przypadku opcji *Enterprise* zbudowanie skrapera wymaga zaawansowanej znajomości technologii internetowych. Zbudowane przez zespół deweloperski skrapery można uruchomić tylko na dedykowanej platformie firmy *datahen.com*. Firma *datahen.com* podkreśla, że jej skrapery są w stanie skrapować treści zabezpieczone system *captcha*.

Zalety:

- Firma udostępnia rozwiązania zarówno dla osób bez znajomości technologii webowych, jak i zaawansowaną wersję usługi dla zespołów deweloperskich;
- W każdym planie dostępna jest darmowa pula żądań do *API*;
- Mnogość formatów wyjściowych;
- Rozwiązanie radzi sobie z zabezpieczeniami stron internetowych typu *captcha*.

Wady:

- Brak możliwości wywołania skrapera na własnej infrastrukturze.

2.4. scrapinghub.com

Scrapinghub.com to firma świadcząca usługi skrapingowe. Została ona założona przez twórców framework-u *Scrapy*, służącego do pisania skrapery i *web-crawlerów* w języku programowania *Python*. Firma oferuje usługi doradcze, jak i budowę oraz wykonywanie skrapera na infrastrukturze klienta. Poza standardowymi usługami firma oferuje cztery narzędzia [8], dzięki którym użytkownicy mogą tworzyć własne skrapery:

- **Scrapy Cloud** – platforma umożliwiająca zbudowanie własnego skrapera/*web-crawlera* za pomocą framework-u *Scrapy*. Aplikacja umożliwia łatwe skalowanie swoich skrapery w zależności od zapotrzebowania;
- **Portia** – aplikacja webowa umożliwiająca wizualne skrapowanie stron WWW. Strona umożliwia zbudowanie i wykonanie skrapera z poziomu platformy. Budowa skrapera polega na zaznaczeniu i opisaniu elementów strony WWW;
- **Splash** – przeglądarka typu *headless*. Przeglądarka nie posiada interface-u graficznego, ale potrafi uruchomić skrypty w języku *JavaScript* w celu renderowania pełnej zawartości strony.

Przeglądarka umożliwia użycie zbudowanego skrapera i udostępnia wynik skrapowania jako *API*;

- **Crawlera** – sieć *proxy* umożliwiająca maskowanie skrapierów w celu uniknięcia blokad *IP*. *Crawlera* gwarantuje, że wielokrotne odpytywanie tej samej strony internetowej w celu pobrania jej zawartości nie skończy się zablokowaniem *IP* poprzez przekierowanie ruchu przez własne serwery. *Crawlera* automatycznie ponawia zapytania z innych *IP*, jeżeli żądanie zakończy się kodem *HTTP* mogącym oznaczać blokadę *IP*.

Zalety:

- Szeroki wachlarz oferowanych usług;
- Wygodne narzędzia dla programistów ułatwiające skrapowanie stron WWW.

Wady:

- *Portia* nie udostępnia możliwości uruchomienia / pobrania rezultatu *Crawlera* za pomocą *API*;
- Wyniki dostępne są tylko w formacie *JSON*.

2.5. octoparse.com

Firma *Octoparse.com* oferuje aplikację *Octoparse* oraz platformę *Octoparse Cloud*, które służą do tworzenia i skrapowania stron internetowych. *Octoparse* jest grubym klientem umożliwiającym generowanie skrapierów wizualnie za pomocą kliknięć na załadowanej stronie internetowej. Umożliwia zdefiniowanie serii akcji, które będą wykonane przed rozpoczęciem skrapowania, przez co umożliwia łatwe skrapowanie stron wymagających na przykład zalogowania do serwisu [9]. *Octoparse* umożliwia uruchomienie skrapera na własnej infrastrukturze i wyeksportowanie rezultatów do wybranego formatu danych. *Octoparse* udostępnia też platformę *Octoparse Cloud* umożliwiającą użycie wygenerowanych skrapierów na infrastrukturze dostarczonej przez *Octoparse.com*.

Zalety:

- Wygodne generowanie skrapierów;
- Możliwość generowania przepływów i akcji do wykonania na stronie przed skrapowaniem;
- Wynik skrapowania dostępny jest w wielu formatach.

Wady:

- Wymaga instalacji dodatkowego oprogramowania.

2.6. webscraper.io

Webscraper.io to wtyczka do przeglądarki *Chrome* umożliwiająca wizualne budowanie skrapersów. Rozszerzenie bazuje na narzędziach deweloperskich przeglądarki. Umożliwia wykonanie zdefiniowanych skrapersów na otwartych stronach. Niestety, interfejs graficzny wtyczki jest skomplikowany. Forum wtyczki jest wypełnione prośbami o pomoc w zbudowaniu działającego skrapera.

Zalety:

- Rozwiązanie jest darmowe.

Wady:

- Proces skrapowania może być uruchomiany tylko manualnie;
- Pomimo interfejsu graficznego, zbudowanie skrapera jest trudne i wymaga wiedzy na temat technologii sieciowych.

2.7. parsehub.com

Firma *parsehub.com* dostarcza grubego klienta do tworzenia skrapersów. Aplikacja jest wygodna i intuicyjna w użyciu. Umożliwia uruchomienie skrapera lokalnie lub załadowanie do konta na *parsehub*. Zdefiniowane skrapery można wywołać za pomocą *API* lub strony internetowej.

Zalety:

- Wygodny i intuicyjny interface;
- Możliwość wykonania zarówno w aplikacji jak i za pośrednictwem strony internetowej lub *API*.

Wady:

- Wymaga instalacji dodatkowego oprogramowania.

2.8. dexti.io

Firma *dexti.io* dostarcza rozwiązanie skrapujące bazując na podejściu *Robotic Process Automation*. Zamiast definiować prosty skrapers, który pobiera dane ze strony, firma proponuje zbudowanie robota. Robot zdefiniowany za pomocą aplikacji *dexti* porusza się we wskazany sposób po stronie WWW i nawiguje pomiędzy podstronami w celu pobrania wszystkich danych imitując zachowania człowieka [10].

Zalety:

- Inne podejście do skrapowania umożliwiające definiowanie zależności i nawigacji pomiędzy stronami WWW;
- Graficzne definiowanie skrapera.

Wady:

- Definiowanie robota może okazać się trudne dla niektórych użytkowników;
- W przypadku prostych skrapersów narzut na tworzenie robota jest większy niż w przypadku innych rozwiązań.

2.9. Podsumowanie istniejących rozwiązań

Rynek oferuje usługi związane ze skrapowaniem stron WWW, w większości wypadków działalność ta polega na opracowaniu skrapera na zamówienie klienta i późniejsze umożliwienie pobrania zdefiniowanych danych. Takie rozwiązanie jest dopasowane do potrzeb użytkownika nieposiadającego wiedzy z zakresu technologii. Jest ono odpowiednie także dla firm, które chcą outsource-ować taką działalność, gdyż podejście to przenosi całość odpowiedzialności na wykonawcę. Odbiorcami pozostałych dostępnych usług są eksperci potrafiący je wykorzystać. Szczególnym przykładem takiego podejścia jest *scrapinghub.com*, opisany w 2.4, który udostępnia narzędzia ułatwiające programistom tworzenie skrapersów. Pojedyncze firmy rozwijają narzędzia umożliwiające skrapowanie stron WWW za pomocą skrapersów zdefiniowanych w sposób wizualny. W przypadku zbadanych serwisów doświadczony użytkownik nie ma możliwości samodzielnej edycji utworzonego skrapera. Dostępne kreatory wizualne dzielą się na dwie kategorie:

- Dostępne za pośrednictwem grubego klienta;
- Udostępnione poprzez aplikacje webową.

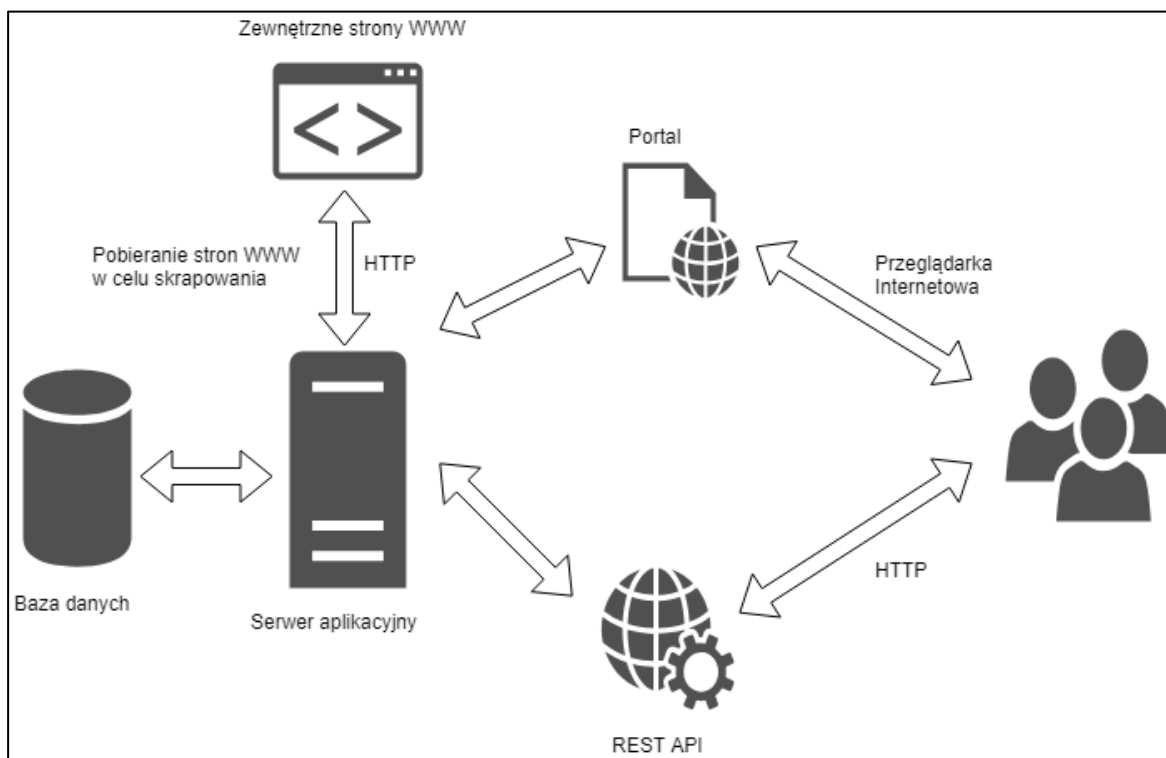
Integracja z rozwiązaniami udostępnionymi za pośrednictwem przeglądarki odbywa się poprzez wymianę plików. Platformy te udostępniają inne formy integracji tylko w przypadku wykupienia płatnych wersji usługi. Dodatkowo rzadkością jest oprogramowanie umożliwiające wykonanie skrapera *ad hoc*, a nie o zdefiniowanej częstotliwości. Pośród dostępnych rozwiązań brakuje takich, które umożliwiłyby wygodne i wizualne utworzenie skrapera oraz pozwoliłyby zaawansowanym użytkownikom na własną modyfikację skrapera oraz jego integrację.

3. Propozycja rozwiązania

W tym rozdziale została przedstawiona propozycja elastycznego skrapera stron WWW udostępnionego jako usługa internetowa. Opisana propozycja składa się z dwóch zasadniczych części: z *API* oraz aplikacji internetowej. Aplikacja internetowa jest portalem, na który użytkownik może się zalogować w celu zdefiniowania skrapera dla nowej strony lub zmiany istniejących skraparów. Drugą udostępnioną częścią rozwiązania jest bezstanowe *API*. *API* umożliwia uruchomienie skrapera oraz zarządzanie projektami i skraparami utworzonymi na platformie.

3.1. Opis architektury

Aplikacja będzie dostępna na dwa sposoby: za pośrednictwem przeglądarki internetowej oraz za pomocą *API* dostępnego poprzez *HTTP*. Udostępnione *API* będzie rozszerzać funkcjonalności dostępne w portalu. Konsumenci *API* będą mogli wykonać te same akcje co korzystający z portalu, lecz jako użytkownicy zaawansowani będą mieli dostęp do funkcjonalności nie udostępnionych w portalu. Strona WWW i *API* współdzielą jeden serwer aplikacyjny. Wszystkie żądania do zewnętrznych zasobów, takich jak na przykład skrapowane strony, wykonywane są za pośrednictwem serwera aplikacyjnego. Rysunek 1 opisuje komunikację między komponentami, użytkownikiem a podstawowymi elementami aplikacji.

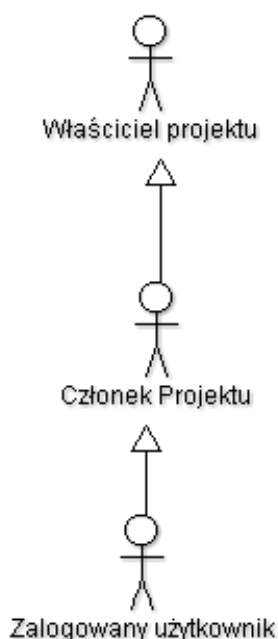


Rysunek 1. Diagram opisujący podstawową architekturę zaproponowanego rozwiązania.

3.2. Aplikacja internetowa

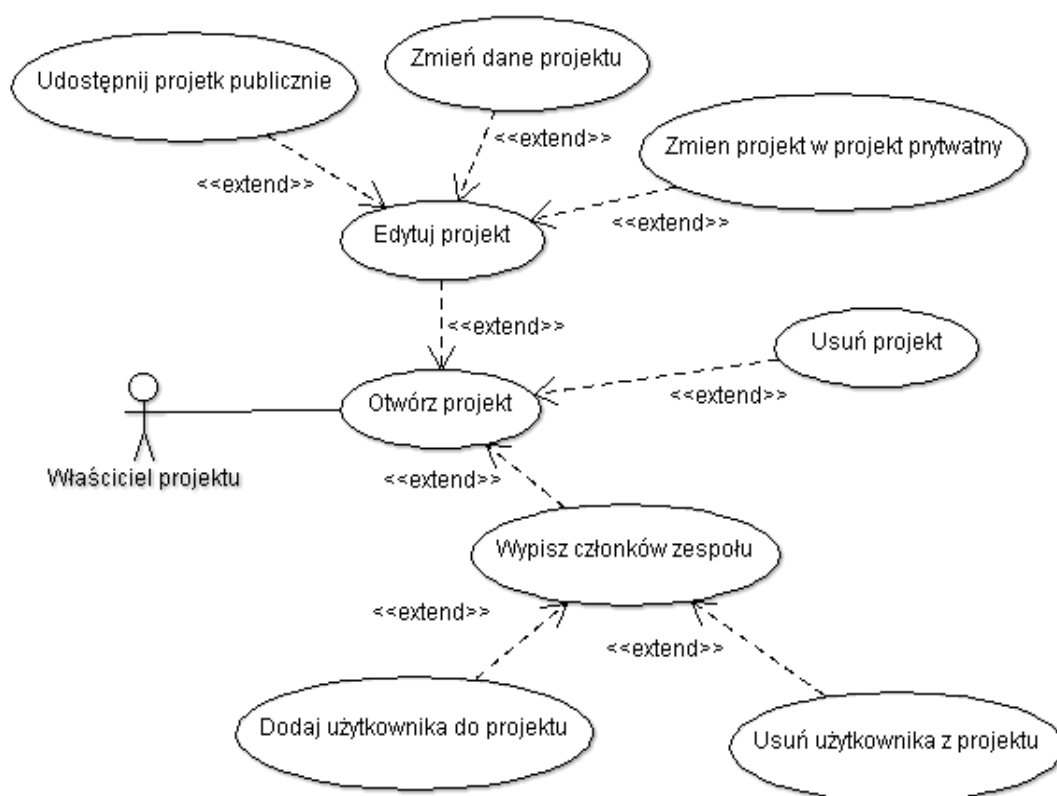
Aplikacją internetową określony został portal mający umożliwić użytkownikom stworzenie skrapera stron internetowych za pomocą prostego interface-u. Założeniem portalu jest dotarcie do szerokiej rzeszy odbiorców, zarówno analityków biznesowych, jak i programistów znających dobrze technologie webowe. Grupa odbiorców bez wiedzy technicznej będzie w stanie utworzyć skraper za pomocą wizualnego edytora. Osoby posiadające umiejętności techniczne, poza utworzeniem skrapera, będą w stanie rozwijać skraper za pomocą zaawansowanego edytora. Dodatkowo portal ma zachęcać użytkowników do dzielenia się swoimi skrapierami poprzez publiczne udostępnianie utworzonych projektów. Z analizy istniejących rozwiązań wynika, że skrapery tworzone są ekskluzywnie na potrzeby pojedynczego klienta. Zrozumiała jest potrzeba utrzymania niektórych źródeł danych w tajemnicy i faktu sprawdzania tychże. Istnieje duża część skrapierów, które mogą być udostępnione publicznie.

Zadaniem portalu będzie grupowanie skrapierów w projekty. Projekty są najwyższą jednostką organizacyjną w portalu. Projekt może być prywatny lub publiczny. Projekty publiczne będą umożliwiały podgląd i wykonanie oraz pobranie wyników skrapowania stron internetowych wszystkim zalogowanym użytkownikom portalu. Rysunek 2 przedstawia hierarchie użytkowników portalu.



Rysunek 2 Hierarchia użytkowników portalu

Właściciel projektu może zdecydować czy udostępnić go publicznie czy tylko wskazanym osobom. Odbywa się to przez dodanie ich jako członków zespołu projektowego. Właściciel projektu może usunąć projekt lub zmodyfikować jego szczegóły, jednocześnie ma możliwość wykonania wszystkich czynności na skraperach, które są dostępne także członkowie zespołu. Ponadto jest jedynym członkiem zespołu, którego nie można wykluczyć z projektu. W zaproponowanym rozwiązaniu tytuł właściciela projektu posiada tylko użytkownik, który utworzył projekt. Tytuł ten jest niezbywalny i nie można go nadać innym. Rysunek 3 przedstawia możliwe akcje dla właściciela projektu.



Rysunek 3 Diagram Use Case przedstawiający możliwości właściciela projektu

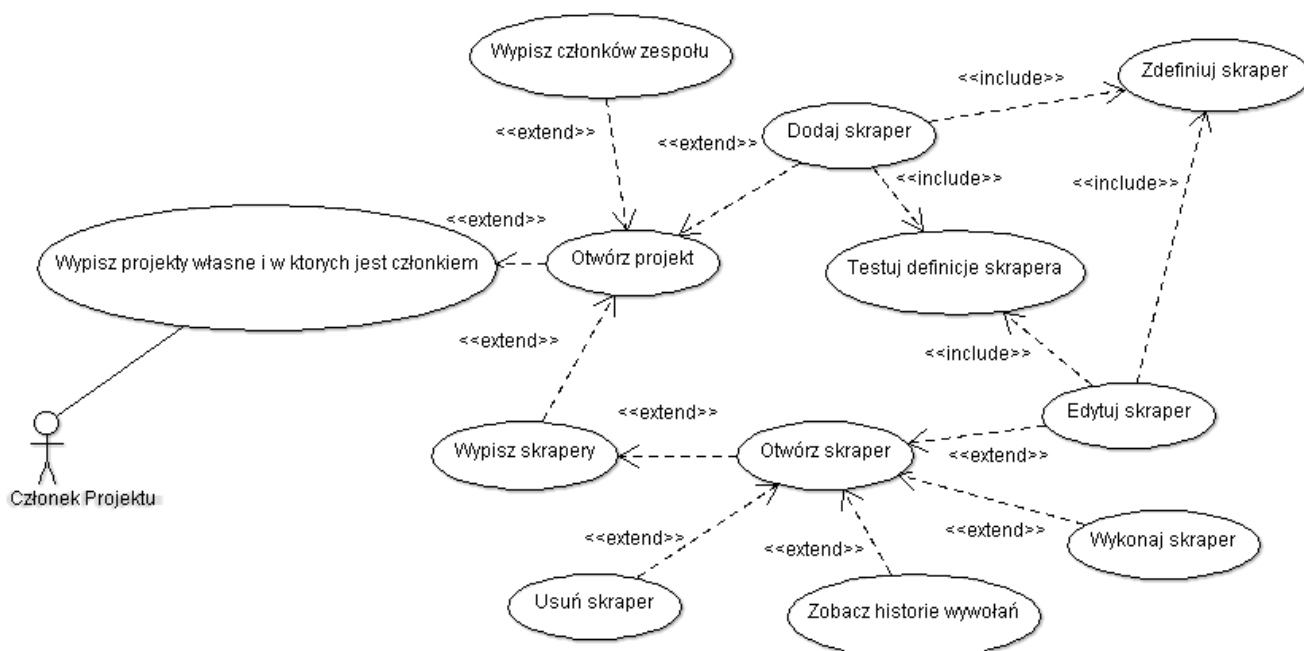
Członkowie projektu dostają dostęp do zarządzania skraperami, będącymi częścią projektu, mogą edytować skrapery, uaktualniać ich definicje i tworzyć nowe. Członek zespołu jest uprawniony do zobaczenia pełnej historii wywołania skrapera. Projekty oznaczone przez właściciela jako prywatne są widoczne tylko dla członków zespołu i właściciela.

Portal musi umożliwić utworzenie skrapera w prosty sposób. Skraper jest tworzony w sposób graficzny poprzez otwarcie wskazanej przez użytkownika strony WWW z załadowaną specjalną biblioteką *JavaScript* podświetlającą wskazany element i tworzącą selektor CSS, który będzie użyty

do skrapowania strony. Programiści będą mogli edytować ręcznie definicje skrapera poprzez modyfikacje lub dodawanie selektorów. Proces tworzenia nowego skrapera przez użytkownika będzie wyglądał następująco:

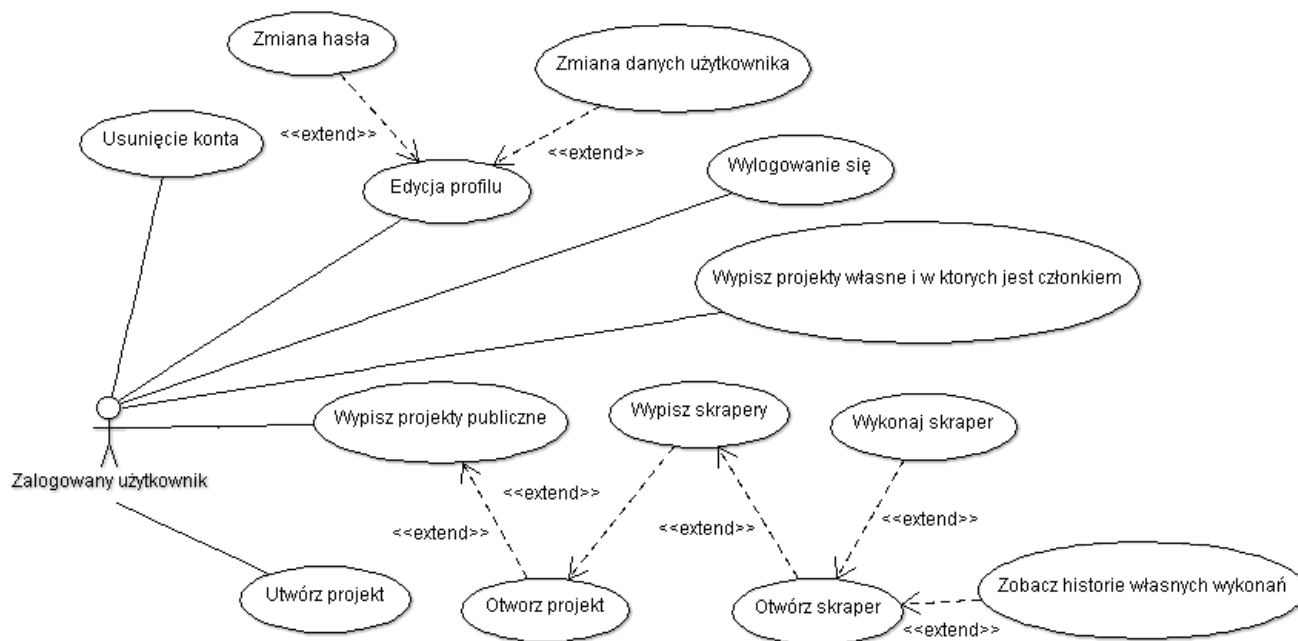
1. wybór nazwy dla skrapera oraz podanie jego opisu;
2. wskazanie adresu *URL* prowadzącego do strony internetowej, która ma być skrapowana;
3. parametryzacja *URL* za pomocą zmiennych.
4. podanie wartości zmiennych, którymi ma zostać uzupełniony *URL* na potrzeby budowania definicji.
5. otwarcie strony przez platformę, do której prowadzi uzupełniony *URL*.
6. zaznaczenie myszką elementów, które mają zostać skrapowane ze strony
7. (opcjonalnie) edycja wygenerowanych selektorów *CSS*.
8. testowanie utworzonych skraparów poprzez zmianę wartości w *URL*.
9. zapisanie skrapera w systemie.

Po utworzeniu skrapera użytkownik jest przenoszony do widoku szczegółów skrapera. W widoku szczegółów opisany jest model odpowiedzi skrapera oraz wygenerowane *API*. Uruchomienie skrapera może nastąpić z poziomu przeglądarki internetowej. Rezultat wywołania zostanie pobrany do przeglądarki w pliku *JSON*. Z poziomu przeglądarki możliwy jest też podgląd szczegółów i historycznych wywołań skrapera. Rysunek 4 przedstawia opisane możliwości użytkowników będących członkami projektu.



Rysunek 4 Diagram Use Case przedstawiający możliwości użytkownika będącego członkiem zespołu

Każdy zalogowany użytkownik ma możliwość edycji swojego profilu jak również utworzenia nowego projektu i dostępu do projektów publicznych. Osoba nie będąca właścicielem ani członkiem zespołu ma ograniczony dostęp do projektu udostępnionego publicznie. Taki użytkownik może zobaczyć wszystkie skrapery i je uruchomić. Nie ma on praw do edycji projektu i zawartych w nim skraparów. Historia wykonania skrapera jest dla niego ograniczona tylko do własnych wywołań. Wszystkie możliwości zalogowanego użytkownika przedstawione są na Rysunek 5



Rysunek 5 Diagram Use Case przedstawiający możliwości zalogowanego użytkownika

Zaproponowane podejście do tworzenia skraparów jest innowacyjne w stosunku do rozwiązań dostępnych na rynku, gdyż łączy ono i rozszerza standardowe funkcjonalności, które można znaleźć w innych produktach:

- Umożliwia wizualną generację skrapera oraz ręczną edycję dla zaawansowanych użytkowników. Jednoczesne występowanie tych dwóch opcji nie jest dostępne w żadnym ze rozpatrywanych platform;
- Udostępnia interfejs użytkownika, a także API umożliwiające wykonanie skrapera na żądanie;
- Nie wymaga instalacji dodatkowego oprogramowania, podczas gdy większość dostępnych narzędzi wymaga instalacji klientów lub rozszerzeń do przeglądarki;
- Umożliwia współpracę w ramach projektu;
- Umożliwia łatwe testowanie wytworzonego skrapera;

- Umożliwia dostęp do projektów publicznych i dzielenie się rozwiązaniami, cecha ta nie jest dostępna w innych produktach;

3.3. Skraper

Struktura definicji skrapera w portalu powinna być jak najprostsza i czytelna dla osoby posiadającej podstawową wiedzę o technologiach internetowych. Użytkownik nie powinien być zmuszony do nauki żadnych dodatkowych notacji. Jest to szczególnie ważne, gdyż portal udostępni możliwość ręcznej edycji skrapera, co umożliwi lepszą kontrolę nad zakresem skrapowanych danych oraz umożliwi tworzenie i edycję skraparów poprzez *API*. Skraper będzie zapisywany w systemie jako obiekt *JSON* zawierający selektory do poszczególnych elementów w postaci selektorów *CSS3*.

Skraper przedstawiony w Przykład 2 definiuje pobieranie danych ze strony *HTML* przedstawionej w Przykład 1. Ze strony w Przykład 1 pobierana jest wartość elementu z atrybutem *id* o wartości „*pojedynczy*” oraz wartości elementów listy „*id*” o wartości „*lista*”.

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<div>
<div id="pojedynczy">
Wartość elementu pojedynczy
</div>
</div>
<div>
<ul id="lista">
<li><div><a>jeden</a></div></li>
<li><div><a>dwa</a></div></li>
</ul>
</div>
</body>
</html>
```

Przykład 1. Przykładowy dokument HTML na potrzeby opisu struktury skrapera

Skraper dla pojedynczego elementu zapisany jest w obiekcie *JSON* jako atrybut przyjmujący wartość w postaci selektora *CSS*. W zaprezentowanym przypadku selektor zaczyna się od najwyższego elementu dokumentu, czyli *HTML*. Selektory nie muszą zaczynać się od znacznika *HTML*, ważne by wskazywały na pojedynczy element oraz były poprawnymi selektorami *CSS*. Mogą

być one oparte o atrybut *id* elementów. W przypadku chęci pobierania wartości pojawiającej się na stronie cyklicznie, tak jak np. wartość w liście z *id=lista*, konieczne jest zdefiniowanie kolekcji elementów. Dzieje się to poprzez utworzenie atrybutu przyjmującego za wartość obiekt. Obiekt ten zawiera trzy obowiązkowe atrybuty:

- **Path** – opisujący ścieżkę do rodzica elementu, wewnątrz którego powtarzają się elementy do skrapowania;
- **Common** – powtarzający się element wspólny, będący rodzicem elementów zawierających pobieraną wartość;
- **Elements** – obiekt będący kolekcją nazwa-wartość, gdzie nazwa to nazwa elementu, a wartość to selektor CSS lub obiekt definiujący zagnieżdżoną kolekcję.

```
{
  "pojedynczy": "html>body>div:eq(0)>div",
  "kolekcja": {
    "path": "html>body>div:eq(1)>ul ",
    "common": "li",
    "elements": {
      "elementKolekcji": "div>a"
    }
  }
}
```

Przykład 2. Przykład struktury skrapera

Zdefiniowane kolekcje mogą być wielokrotnie zagnieżdżane i mogą zawierać więcej niż jeden element. Definicja skrapera wpływa bezpośrednio na kształt obiektu wyjściowego. W przypadku pojedynczych elementów będą one miały tę samą nazwę, co odpowiadające elementy w skraperze. Natomiast w przypadku kolekcji pod odpowiadającą nazwą będzie znajdować się lista obiektów, które będą kolekcją klucz-wartość, gdzie klucze będą odpowiadały kluczom definiowanym w atrybucie *elements*. Wartości przyjmą odpowiednio rezultaty skrapowania albo w przypadku zagnieżdżonych kolekcji, listy odpowiadających obiektów. Przykład 3 zawiera rezultat wykonania skrapera zdefiniowanego w Przykład 2 na stronie internetowej opisanej w Przykład 1.

```

{
    „pojedynczy”: „Wartość elementu pojedynczy”,
    „kolekcja”: [{
        „elementKolekcji”: „jeden”
    }, {
        „elementKolekcji”: „dwa”
    }]
}

```

Przykład 3. Rezultat skrapowania dla skrapera podanego w Przykład 2 na stronie z Przykład 1.

3.4. API

Jednym z założeń prototypu elastycznego skrapera jest dostarczenie otwartego *API*. Celem udostępnienia *API* jest umożliwienie rozbudowy i rozwoju portalu przez społeczność oraz umożliwienie integracji deweloperom z zaproponowanym rozwiązaniem. Prosta struktura skrapera oraz otwarte *API* umożliwi społeczności rozbudowanie platformy poprzez np. utworzenie wtyczek do przeglądarek umożliwiających tworzenie skrapersów lub grubych klientów.

Udostępnione *API* będzie zaprojektowane w architekturze *REST*. Wyeksponuje ono zasoby:

- document,
- execute,
- project,
- scraper,
- user.

Większość zasobów jest dostępnych tylko dla uwierzytelnionych użytkowników. W celu uwierzytelnienia trzeba umieścić w żądaniu nagłówek *Authorization* z poprawnym uwierzytelnieniem typu *Basic*. W odpowiedzi zwrócony zostanie kod statusu odpowiedzi oraz w przypadku sukcesu (status 200) żądana reprezentacja zasobu.

Statusy odpowiedzi są spójne ze standardem *HTTP* opisanym przez *Internet Assigned Numbers Authority* [17].

3.4.1 User

Zasób *User* odpowiada za zarządzanie użytkownikami aplikacji. Udostępnia metody do tworzenia encji, modyfikacji oraz pobierania szczegółów.

- **POST /user/**

Metoda służy utworzeniu użytkownika w systemie. Treść żądania:

```
{
    „username” : „uzytkownik”,
    „email” : „uzytkownik@example.com”,
    „firstName” : „imie”,
    „lastName” : „nazwisko”,
    „password” : „tajnehaslo”
}
```

Przykład 4. Przykładowa treść żądania tworząca użytkownika.

W treści żądania należy przekazać następujące atrybuty:

- **username** – wymagany, maksymalna długość 16 znaków. Wartość wskazuje na unikatową nazwę użytkownika, która będzie wymagana przy uwierzytelnieniu;
- **email** – wymagany, maksymalna długość 60 znaków. Email;
- **firstName** - wymagany, maksymalna długość 60 znaków. Imię;
- **lastName** - wymagany, maksymalna długość 60 znaków. Nazwisko;
- **password** - wymagany, maksymalna długość 40 znaków. Hasło, które będzie używane w celu; uwierzytelniania użytkownika. Hasło nie będzie przechowywane w postaci jawnej. Przechowywana będzie tylko jego reprezentacja wygenerowana za pomocą algorytmu SHA2 wykonanego na konkatencji hasła i *UUID*-u przypisanego do użytkownika.

W Tabeli 1 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	{ „id” : „1” }	Id utworzonego użytkownika
409		Istnieje już użytkownik o zadanym <i>username</i> .
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 1. Opis dostępnych odpowiedzi usługi **POST /user/**.

- **GET /user/:id**

Usługa zwraca dane użytkownika o *ID* wskazanym w *URI*. Usługa wymaga podania uwierzytelnienia w nagłówku. W Tabeli 2 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	<pre>{ „id”: 1, „username”: „ uzytkownik „, „email”: „ uzytkownik@example.com”, „firstName”: „ imie”, „lastName”: „nazwisko”, „isActive”: 1, „isPremium”: 0, „isAdmin”: 0, „createTime”: „2017-12- 04T20:54:18.000Z”, „updateTime”: „2017-12- 04T20:54:18.000Z” }</pre>	Podstawowe dane użytkownika oraz zestaw dodatkowych flag .
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Użytkownik nie istnieje.
500		Błąd techniczny serwera.

Tabela 2. Opis dostępnych odpowiedzi usługi GET /user/:id.

- **DELETE /user/:id**

Usługa usuwa użytkownika o *ID* wskazanym w *URI*. Usługa wymaga podania uwierzytelnienia w nagłówku. W Tabeli 3 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Użytkownik został usunięty
401		Użytkownik nie został uwierzytelniony

		poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Użytkownik nie istnieje.
500		Błąd techniczny serwera.

Tabela 3. Opis dostępnych odpowiedzi usługi DELETE /user/:id.

PUT /user/:id

Usługa modyfikuje użytkownika o *ID* wskazanym w *URI*. Usługa wymaga podania uwierzytelnienia w nagłówku. Treść wiadomości jest identyczna jak w przypadku żądania utworzenia z Przykład 4. W Tabeli 4 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Użytkownik został zmodyfikowany
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Użytkownik nie istnieje.
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 4. Opis dostępnych odpowiedzi usługi PUT /user/:id.

- **PATCH /user/:id**

Usługa umożliwia zmienienie hasła użytkownika o *ID* wskazanym w *URI*. Usługa wymaga podania uwierzytelnienia w nagłówku. Przykład 5 pokazuje przykładowe wywołanie usługi.

```
{
  „password” : „nowehaslo”
}
```

Przykład 5. Przykład żądania zmieniającego PATCH /user/:id/.

W Tabeli 5 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Hasło dla użytkownika zostało zmienione
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Użytkownik nie istnieje.
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 5. Opis dostępnych odpowiedzi usługi PATCH /user/:id.

3.4.2 Project

Zasób pozwala na zarządzanie projektami użytkownika, których jest właścicielem. Właściciel projektu może edytować dane projektu i dodawać członków zespołu. W *URI* w miejscu parametru *:userId* należy podać *Id* użytkownika, w którego kontekście wykonywana jest usługa, i którego szczegóły uwierzytelnienia wymagane są przy dostępie do każdej z usług.

- **GET /user/:userId/project**

Usługa zwraca listę projektów, których użytkownik jest właścicielem lub członkiem zespołu. W Tabeli 6 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	<pre>[{ "id": 1, "title": "tytuł", "createTime": "2017-12-04T21:21:38.000Z", "ownerId": 1, "isPublic": 1, "updateTime": "2017-12-04T21:21:38.000Z" }]</pre>	Lista projektów, których użytkownik jest właścicielem lub członkiem zespołu. • id – identyfikator projektu • title – nazwa projektu • createTime - czas utworzenia projektu • updateTime – czas ostatniej modyfikacji szczegółów projektu • ownerId – identyfikator

		użytkownika będącego właścicielem projektu • isPublic – flaga wskazująca czy projekt jest udostępniony publicznie.
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
500		Błąd techniczny serwera.

Tabela 6. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/.

- GET /user/:userId/project/public

Usługa zwraca listę projektów, do których użytkownik ma dostęp z racji tego, że są udostępnione publicznie. W Tabeli 7 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	[{ "\"id\": 1, \"title\": \"tytuł\", \"createTime\": \"2017-12-04T21:21:38.000Z\", \"ownerId\": 1, \"isPublic\": 1, \"updateTime\": \"2017-12-04T21:21:38.000Z\" }]	Lista projektów publicznych • id – identyfikator projektu • title – nazwa projektu • createTime - czas utworzenia projektu • updateTime – czas ostatniej modyfikacji szczegółów projektu • ownerId – Id użytkownika będącego właścicielem projektu • isPublic – flaga wskazująca czy projekt jest udostępniony publicznie.
401		Użytkownik nie został uwierzytelniony poprawnie.

500		Błąd techniczny serwera.
-----	--	--------------------------

Tabela 7. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/public.

- **POST /user/:userId/project**

Usługa tworzy nowy projekt w systemie. Użytkownik, w którego kontekście wywołana została usługa, staje się właścicielem projektu. Przykładowa wiadomość znajduje się w Przykład 6. W treści wiadomości musi znaleźć się:

- **Title** – tytuł projektu. Wymagane pole, maksymalna długość 45 znaków;
- **Description** – opis projektu. Wymagane pole, maksymalna długość 200 znaków;
- **isPublic** – flaga wskazująca, czy projekt jest udostępniony publicznie. Wartość wymagana mogąca przyjąć jedną z dwóch wartości: 1 lub 0, gdzie wartość jeden oznacza, że projekt jest udostępniony publicznie.

```
{
  "title" : "tytuł",
  "description" : "opis",
  "isPublic" : "1"
}
```

Przykład 6. Przykładowa treść wiadomości do usługi POST /user/:userId/project.

W Tabeli 8 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	{ "id" : 1 }	Identyfikator utworzonego projektu
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 8. Opis dostępnych odpowiedzi usługi POST /user/:userId/project.

- **GET /user/:userId/project/:projectId**

Usługa zwraca szczegóły wskazanego projektu, jeśli użytkownik ma uprawnienia aby go otworzyć. W Tabeli 9 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	{ "id": 168, "title": "test_project", "createTime": "2017-12-04T21:21:38.000Z", "ownerId": 430, "isPublic": 1, "isActive": 1, "updateTime": "2017-12-04T21:21:38.000Z" }	Szczegóły projektu. Opis pól znajduje się w Tabeli 7.
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje.
500		Błąd techniczny serwera.

Tabela 9. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId.

- **DELETE /user/:userId/project/:projectId**

Usługa usuwa projekt o ID wskazanym w URI. W Tabeli 10 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Projekt został usunięty
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika

		powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje.
500		Błąd techniczny serwera.

Tabela 10. Opis dostępnych odpowiedzi usługi DELETE /user/:userId/project/:projectId.

- **PUT /user/:userId/project/:projectId**

Usługa modyfikuje projekt o *ID* wskazanym w *URI*. Treść wiadomości jest identyczna jak w przypadku żądania tworzenia użytkownika z Przykład 6. W Tabeli 11 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Projekt został zmodyfikowany
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje.
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 11. Opis dostępnych odpowiedzi usługi PUT /user/:userId/project/:projectId.

- **GET /user/:userId/project/:projectId/users**

Usługa zwraca listę użytkowników mających uprawnienia do projektu. W Tabeli 12 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	[{ "id": 430, "username": "StefanTester", "email": "testowy@tester.com", "firstName": "Stefan", "lastName": "Tester"	Lista użytkowników przypisanych do projektu.

	}}	
401		Użytkownik nie został uwierzytniony poprawnie.
403		Uwierzytnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje.
500		Błąd techniczny serwera.

Tabela 12. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/users.

- **PUT /user/:userId/project/:projectId/users/:Username**

Wskazany użytkownik poprzez uzupełnienie parametru: *Username* o nazwę użytkownika, zostaje dodany do projektu. Tylko właściciel projektu może wykonać tę akcję. W Tabela 13 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Użytkownik został dodany do projektu
401		Użytkownik nie został uwierzytniony poprawnie.
403		Uwierzytnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Użytkownik nie istnieje.
409		Użytkownik jest już członkiem projektu
500		Błąd techniczny serwera.

Tabela 13. Opis dostępnych odpowiedzi usługi PUT /user/:userId/project/:projectId/users/:Username.

- **DELETE /user/:userId/project/:projectId/users/:removeUserId**

Wskazany użytkownik, poprzez uzupełnienie parametru :*removeUserId* o *ID* użytkownika, zostaje usunięty z projektu. Tylko właściciel projektu może usuwać członków zespołu. Właściciel projektu nie może usunąć siebie z projektu. W Tabela 14 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Wskazany użytkownik został usunięty z projektu.

401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje.
409		Właściciel projektu nie może zostać usunięty z projektu
500		Błąd techniczny serwera.

Tabela 14. Opis dostępnych odpowiedzi usługi DELETE /user/:userId/project:projectId/users/:removeUserId.

3.4.3 Scraper

Zasób pozwala na zarządzanie skraperaми w projekcie :projectId. W przypadku projektów publicznych wszyscy użytkownicy mają uprawnienia do pobierania danych o skraperaх. Usługi modyfikujące skrapera dostępne są tylko dla osób uprawnionych: właściciela projektu oraz członków zespołu. W URI w miejscu parametru :userId należy podać ID użytkownika, w którego kontekście uruchamiana jest usługa, i którego szczegóły uwierzytelnia wymagane są przy dostępie do każdej z usług. W miejscu parametru :projectId należy podać ID projektu, do którego należy skrapera.

- GET /user/:userId/project:projectId/scraper/:scraperId

Usługa zwraca szczegóły skrapera wskazanego w :scraperId. W Tabeli 15 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	<pre>{ "id": 33, "status": 0, "name": "scraperName", "url": "www.url.com", "definition": {"test": "html>body>div"}", "projectId": 250, "isActive": 1, "createTime": "2017-12- 06T00:31:00.000Z",</pre>	Szczegóły skrapera wskazanego w żądaniu. • id – identyfikator skrapera • status – status skrapera • name – nazwa skrapera • url – url strony, której dotyczy skrapera • defintion – obiekt skrapera zgodny z opisem w rozdziale 3.3 • createTime - czas utworzenia skrapera • updateTime – czas ostatniej modyfikacji szczegółów skrapera

	<pre> „lastUpdate”: „2017-12-06T00:31:00.000Z”, „createdBy”: { „id”: 640, „username”: „Uzytkownik” }, „updatedBy”: { „id”: 640, „username”: „ Uzytkownik ” } </pre>	• projectId – Identyfikator projektu, do którego należy skrapier • createdBy – obiekt zawierający identyfikator i <i>username</i> użytkownika, który utworzył skrapier. • updatedby – obiekt zawierający identyfikator i <i>username</i> użytkownika, który modyfikował skrapier
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Skrapier nie istnieje
500		Błąd techniczny serwera.

Tabela 15. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/.

- **GET /user/:userId/project/:projectId/scrapper/**

Usługa zwraca listę skrapierów zdefiniowanych w projekcie. W Tabeli 16 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	<pre> [{ "id": 1, "status": 0, "name": "", "url": "", "projectId": 1 }] </pre>	Lista skrapierów przypisanych do projektu.
401		Użytkownik nie został

		uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje
500		Błąd techniczny serwera.

Tabela 16. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/.

- **POST /user/:userId/project/:projectId/scrapper/**

Usługa tworzy nowy skrapler i przypisuje go do projektu wskazanego w :*projectId*. Przykładowa treść wiadomości znajduje się w Przykład 7.

```
{
  „id” : 1,
  „name” : „skraper”,
  „url” : „www.url.pl/{param1}”,
  „definition”: “{}”,
  „description”: “opis”,
  „urlParams” : [“param1”]
}
```

Przykład 7. Przykładowa treść wiadomości do usługi POST /user/:userId/project/:projectId/scrapper/.

W Tabeli 17 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	{ „id” : 1 }	Identyfikator utworzonego skrapera
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 17. Opis dostępnych odpowiedzi usługi POST /user/:userId/project.

- **DELETE /user/:userId/project/:projectId/scrapper/:scrapperId**

Usługa usuwa skraper o *ID* wskazanym w *URI*. W Tabeli 18 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Skraper został usunięty
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Skraper nie istnieje.
500		Błąd techniczny serwera.

Tabela 18. Opis dostępnych odpowiedzi usługi DELETE /user/:userId/project/:projectId/scrapper/:scrapperId.

- **PUT /user/:userId/project/:projectId/scrapper/:scrapperId**

Usługa modyfikuje skraper o *ID* wskazanym w *URI*. Treść wiadomości jest identyczna, jak w przypadku żądania tworzenia użytkownika z Przykład 7. W Tabeli 19 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200		Skraper został zmodyfikowany
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Projekt nie istnieje.
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji.
500		Błąd techniczny serwera.

Tabela 19. Opis dostępnych odpowiedzi usługi PUT /user/:userId/project/:projectId/scrapper/:scrapperId.

- **GET /user/:userId/project/:projectId/scrapper/:scrapperId/result**

Usługa zwraca historię wywołań skrapera. Dla właściciela i członków zespołu zwracana jest pełna historia skrapera, zaś dla użytkowników tylko z dostępem publicznym, zwracana jest tylko historia ich własnych wywołań. W Tabeli 20 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	<pre>[{ "createTime": "2017-12-06T00:31:00.000Z", "timeTaken": 123, "url" : "www.example.com", "requestId": 1, "result": {}, "status": "0", "message": "" }]</pre>	Lista zawierająca historie i statusy wywołań: • createTime - czas wywołania skrapera. • timeTaken – czas skrapowania wyrażony w milisekundach. • url – URL pod którym znajduje się skrapowana strona. • requestId – identyfikator żądania. • result – obiekt reprezentujący wynik skrapowania. • Status – status wykonania skrapera. • Message – treść błędu.
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Skrapec nie istnieje.
500		Błąd techniczny serwera.

Tabela 20. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/:scrapperId/result.

- **GET /user/:userId/project/:projectId/scrapper/:scrapperId/result**

Usługa zwraca szczegóły konkretnego wywołania skrapera. W Tabeli 21 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	<pre>[{ "request": { "id": 1, "createTime": "2018-01-15T00:19:25.000Z", "urlParams": {}, "user": {</pre>	Szczegóły wykonania skrapera zawierają zarówno obiekt wywołania opisany w Przykład 8 oraz szczegóły odpowiedzi opisane w Tabeli 20.

	<pre> “username”: “example”, “id”: 1 }}, “response”: { “timeTaken”: 123, “requestId”: 1, “result”: {}, “status”: “0”, “message”: “” } }} </pre>	
401		Użytkownik nie został uwierzytelniony poprawnie.
403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Skraper nie istnieje.
500		Błąd techniczny serwera.

Tabela 21. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/:scraperId/result.

3.4.4 Execute

Endpoint służy do wykonywania zdefiniowanych skrapersów.

- **POST /execute/:scraperId**

Wykonuje skrapers o wskazanym *Id*, jeśli użytkownik ma uprawnienia do tej akcji. Umożliwia uruchomienie skrapera z określonymi parametrami. W jednym żądaniu można dokonać więcej niż jednego skrapowania przekazując listę parametrów. Dodatkowo w żądaniu można przekazać funkcję transformującą wyniki skrapowania. Funkcja przyjmuje jeden parametr i zwraca zmodyfikowany obiekt.

```
{
  "transform": "function(data) {return data;}",
  "urlParams": [{
    "example": "example"
  }]
}
```

Przykład 8. Przykładowe żądanie do wykonania skrapera.

Przykład 8 pokazuje przykładowe żądanie do skrapera. Żądanie przyjmuje następujące wartości:

- **transform** – funkcja transformująca wyniki skrapowania (opcjonalna).
- **urlParams** – lista obiektów zawierających wartości, którymi zostanie uzupełniony szablon *URL*. Podając wiele parametrów spowoduje równoległe wykonanie wielu skrapersów na wskazanych w żądaniu stronach. W odpowiedzi zostanie zwrócony jeden *requestId* i lista odpowiedzi.

W Tabeli 22 znajduje się dokładny opis dostępnych odpowiedzi usługi.

Kod odpowiedzi	Treść odpowiedzi	Opis
200	[{ "timeTaken": 123, "requestId": 1, "url": " www.example.pl " "result": {}, "status": "0", "message": "" }]	Lista zawierająca historie i statusy wywołań: • timeTaken – czas skrapowania wyrażony w milisekundach. • url – URL pod którym znajduje się skrapowana strona. • requestId – identyfikator <i>requestu</i>. • result – obiekt reprezentujący wynik skrapowania. • status – status wykonania skrapera. • Message – treść błędu.
401		Użytkownik nie został uwierzytniony poprawnie.

403		Uwierzytelnienie użytkownika powiodło się, lecz nie ma on uprawnień do tego zasobu.
404		Skraper nie istnieje.
422		Nieprawidłowa treść żądania. Treść żądania nie przeszła walidacji lub była niezgodna z definicją skrapera.
500		Błąd techniczny serwera.

Tabela 22. Opis dostępnych odpowiedzi usługi GET /user/:userId/:project/:projectId/scrapper/:scraperId/result.

4. Języki, metodyki, narzędzia zastosowane w pracy

W rozdziale zostały zaprezentowane technologie oraz narzędzia wykorzystane do wytworzenia prototypu skrapera.

4.1. JavaScript

JavaScript to język programowania zaprojektowany przez firmę *Netscape* w 1995 roku. Język używany jest przede wszystkim w tworzeniu interaktywnych stron internetowych. Został spopularyzowany wraz z rozwojem stron WWW i aplikacji webowych. *JavaScript* jest wspierany przez większość komercyjnych przeglądarek internetowych i stanowi podstawę interakcji na stronach.

Cechy charakterystyczne *JavaScript*:

- **Wieloparadygmatyczność** – język implementuje więcej niż jeden paradygmat programowania. W przypadku *JavaScript* są to paradygmaty: obiektowy, funkcyjny oraz imperatywny;
- **Dynamiczne typowanie** – typy danych w języku *JavaScript* są rozpoznawane podczas wykonywania skryptu.

W początkowych fazach życia języka powstało wiele dialektów i silników *JavaScript*, które różniły się od siebie w niektórych aspektach, szczególnie związanych z integracją ze środowiskiem uruchomieniowym przeglądarki. W celu ujednolicenia języka *JavaScript* firma *Ecma International* opublikowała standard nazywany *ECMAScript* [11]

JavaScript w zależności od środowiska rozszerzany jest poprzez odpowiednie mechanizmy:

- **Po stronie klienta w przeglądarce** – mechanizmy umożliwiające kontrolę przeglądarki, manipulacje obiektem *DOM* czy obsługę zdarzeń, takich jak na przykład zmiana wartości elementu *input*;
- **Po stronie serwera** - mechanizmy umożliwiające komunikację z bazą danych, obsługę plików oraz inne aspekty komunikacji.

4.2. TypeScript

Język *TypeScript* stworzony przez firmę *Microsoft* w 2012 roku. *TypeScript* jest nadzbiorem *JavaScript*, rozszerza język *JavaScript* o takie elementy składni, jak klasy, enumy oraz daje możliwość statycznego typowania zmiennych. Język *TypeScript* nie jest interpretowany przez przeglądarki. Z tego powodu nie jest on wykonywany, a raczej tłumaczony do języka *JavaScript* [12]. Dzięki przepisaniu framework-u *Angular* na język *TypeScript* zyskał on na popularności. Założeniem

TypeScriptu jest rozszerzenie języka *JavaScript* o statyczne typowanie, klasy i enumy w celu ułatwiania programowania dużych aplikacji z wieloma zależnościami, gdzie wsparcie dla typów oraz sprawdzanie typów podczas kompilacji ułatwia weryfikację poprawności kodu.

4.3. HTML

HTML to język hipertekstowy opisujący strukturę danych na stronie internetowej. Język *HTML* składa się z atrybutów i znaczników. *HTML* jest przetwarzany przez przeglądarkę do obiektu *DOM*, który jest interpretowany i wyświetlany w przeglądarce internetowej.

- **Znacznik (tag)** – opisuje strukturę dokumentu i zawiera treści strony;
- **Atrybuty** – opisują dodatkowe cechy znacznika. Zapisane jako para nazwa-wartość: odseparowane znakiem „=”.
- **Id** – specjalny atrybut nadający elementowi unikatowy identyfikator. Umożliwiający jednoznaczne odnalezienie elementu na stronie internetowej lub zdefiniowanie wyglądu elementu w *CSS*;
- **Style** – specjalny atrybut umożliwiający *in linowe* zdefiniowanie wyglądu elementu na stronie internetowej;
- **Class** – specjalny atrybut odnoszony do klas *CSS* opisujących wygląd elementu.

4.4. CSS

Kaskadowe arkusze stylów (z ang. *Cascading Style Sheets*). *CSS* to język stworzony w 1996 r. przez organizację *W3C* w celu odseparowania formy prezentacji od struktury *HTML*. *CSS* opisuje, jak przeglądarka ma prezentować wskazane elementy *HTML*. W arkuszu *CSS* należy zdefiniować selektory służące wskazaniu elementów, które mają podlegać danej regule oraz wartości, jakie mają przyjąć ich własności. Nazwa „kaskadowe” wynika z faktu, że kolejne reguły nawzajem się wykluczają i uzupełniają definiując finalny wygląd strony.

4.5. HTTP

Z angielskiego *Hypertext Transfer Protocol* to protokół wymiany dokumentów hipertekstowych w środowisku serwer-klient. *HTTP* to generyczny bezstanowy protokół warstwy aplikacyjnej umożliwiający wymianę danych, który dzięki swoim właściwościom znalazł szersze zastosowanie, niż tylko wymianę dokumentów *HTML*. Poprzez rozszerzenie standardowych metod oraz nagłówków znalazł on zastosowanie na przykład w integracji systemów rozproszonych. Protokół z założenia

ukrywa wszelkie szczegóły implementacji serwera, natomiast serwer nie musi znać szczegółów implementacyjnych klienta oraz celu jego żądania. Obecny standard *HTTP* opisany jest w dokumencie *RFC 7230* [14].

Protokół *HTTP* definiuje nagłówki, które są podstawą interpretacji danych zawartych w wiadomości. Nagłówki wysyłane są na początku wiadomości. Jest to kolekcja nazwa-wartość, gdzie nazwa od wartości oddzielona jest znakiem „:”, a kolejne nagłówki oddzielone są znakami *CRL*. Koniec nagłówków oznaczony jest pustą linią. Protokół *HTTP* definiuje podstawowe nagłówki, ale też nie ogranicza użytkowników przed definiowaniem własnych nagłówków.

Protokół *HTTP* wysyła w żądaniu metodę wskazującą jakiej akcji oczekuje od serwera klient:

- **GET** – metoda oczekuje odpowiedzi zawierającej reprezentację zasobu, którego dotyczy żądanie. Zgodnie ze specyfikacją protokołu metoda ta powinna być w pełni idempotentna, a żądanie nie powinno wprowadzać żadnych zmian na serwerze;
- **HEAD** – oczekuje takiej samej akcji, jak w wypadku metody *GET*, ale bez zwracania właściwej treści wiadomości. Używana jest do sprawdzania nagłówków odpowiedzi bez narzutu na transport danych;
- **POST** – żądanie wysyłane w celu utworzeniu na serwerze nowej instancji wskazanego zasobu, której szczegóły znajdują się w treści wiadomości;
- **PUT** – wskazuje, że dany zasób powinien być umieszczony pod podanym *URI*. Jeśli *URI* żądania wskazuje na istniejący już zasób, powinien on zostać nadpisany;
- **DELETE** – metoda usuwająca wskazany zasób;
- **TRACE** – metoda służąca diagnostyce komunikacji;
- **OPTIONS** – w odpowiedzi serwer powinien wskazać dozwolone metody do wykonania na wskazanym *URI*;
- **PATCH** – metoda służy zmienianiu pojedynczych własności wskazanego zasobu;
- **CONNECT** - za pomocą tej metody można połączyć się do serwera pośredniczącego w drodze do serwera docelowego, dla którego przeznaczone będą komunikaty.

4.6. REST

REST to styl architektoniczny opisujący sposób komunikacji pomiędzy systemami rozproszonymi. *REST* zawiera zestaw reguł i dobrych praktyk, z czego głównymi są:

- Jednolity interfejs:
 - Zasoby są dostępne pod unikalnym *URI*;

- Zasoby są udostępniane i modyfikowany zgodnie z ich reprezentacją;
- Samoopisowość poprzez nagłówek *media type*;
- HATEOAS – stan klienta i szczegóły żądania powinny być przesłane za pomocą nagłówków i parametrów zapytania;
- Bezstanowość:
 - Wszystkie dane potrzebne do obsługi żądania znajdują się w treści żądania;
- Możliwość cacheowania:
 - Odpowiedź serwera określa, jak powinny być cacheowane dane;
- Klient-serwer:
 - Niezależność klienta od serwera;
- Warstwowość:
 - Serwer ukrywa warstwy przed klientem;
- Code on demand:
 - Jedyna reguła opcjonalna;
 - Żądania mogą zawierać reguły/logikę, która ma zostać wykonana w ramach obsługi żądań.

4.7. NodeJS

NodeJS jest środowiskiem uruchomieniowym *JavaScript*. Stworzony w roku 2009 przez Ryana Dahla. Środowisko domyślnie działa na silniku V8, stworzonym przez firmę Google. *NodeJS* jest kompatybilny z innymi silnikami. *NodeJS* jest wieloplatformowym otwartym środowiskiem służącym do wykonywania kodu napisanego w języku *JavaScript* po stronie serwera. Głównym założeniem *NodeJS* było stworzenie środowiska umożliwiającego pisanie aplikacji webowych w jednym języku, zarówno dla *Front Endu* jak i *Back Endu*. Kolejka zdarzeń *JavaScript* w naturalny sposób nadaje się do obsługi żądań jako serwer WWW przez co *NodeJS* gwarantuje wysoką skalowalność i jest w stanie obsłużyć duże ilości żądań w jednym czasie. W *NodeJS* kolejne żądania nie tworzą nowych wątków czy procesów, zamiast tego system wywołuje zarejestrowany *callback* i żądanie trafia do kolejki zdarzeń [11].

Wraz z popularyzacją *JavaScript* dla *NodeJS* powstało bardzo wiele pakietów ułatwiających i przyspieszających rozwój oprogramowania. Większość tych bibliotek jest zarządzana przez menadżera pakietów *NPM* (*node package manager*). *NPM* powstał w roku 2010. Jest napisany w pełni

w języku *JavaScript*. Wywołuje się go za pomocą wiersza poleceń, umożliwia on pobieranie i utrzymywanie licznych zależności w aplikacjach napisanych w *NodeJS*.

4.8. ExpressJS

ExpressJS (zwany także *Express*) to framework do budowania serwerów w *NodeJS*. Jeden z podstawowych elementów *MEAN stack* - czyli stosu technologicznego *Mongo-Express-Angular-Node*. Sam *Express* jest lekkim framework-iem ułatwiającym tworzenie serwerów *HTTP*, nie zawiera on wielu funkcjonalności, ale może być łatwo rozszerzalny o dodatkowe biblioteki. To czyni go wygodnym i wszechstronnym narzędziem [13]. Podstawowe elementy *Express* to:

- **Routing** – *Router express* mapuje ścieżkę *URI* na moduł, który powinien być wykonany w żądaniu. Router umożliwia zadeklarowanie konkretnego handlera celem obsługi żądania dla danego *URI* i dla danej metody *HTTP*. Handler przyjmuje trzy obiekty na wejściu:
 - **Request** - opisujący żądanie;
 - **Response** – obiekt zawierający metody służące do ustawienia i wysłania odpowiedzi;
 - **Next** – funkcja, której wywołanie spowoduje przejście do następnego zadeklarowanego handlera pasującego do *URI* żądania. Dzięki temu mechanizmowi można na przykład rozszerzyć obiekt *request* lub obsługiwać błąd 404.
- **Middleware** – specjalny rodzaj routera, który można wykorzystać do obsługi żądań bez podawania ścieżki. Sygnatura wejścia *middlewarea*, jest ona identyczna, jak w wypadku routerów. *Middleware* jest używany na przykład w celu logowania żądań, uwierzytelniania użytkownika lub wzbogacania *requestu* o dane sesji.

4.9. Postman

Postman to narzędzie do testowania *endpoint*-ów aplikacji webowej. Umożliwia sporządzenie testów automatycznych oraz wykonywanie żądań *HTTP* z określonymi parametrami. *Postman* pozwala na zdefiniowanie *workspace-u*, w obrębie, którego można zdefiniować zmienne i stałe użyte na przykład do uzupełnienia *URL*, nagłówek lub treści żądań. W obrębie *workspace-u* można zdefiniować kolekcje. Kolekcje reprezentują grupy żądań, które mają zostać wykonane w zadeklarowanej kolejności. Do każdego z żądań można zadeklarować serie asercji umożliwiających weryfikację pożądanego zachowania *endpoint*-ów. Testy stworzone w aplikacji *postman* mogą zostać wykonane za pomocą: wtyczki do przeglądarki *Google Chrome*, samodzielnej aplikacji lub za pomocą biblioteki *newman* do *NodeJS*.

4.10. Winston

Winston to biblioteka *NodeJS* służąca do logowania. *Winston* umożliwia logowanie do wielu kanałów, takich jak na przykład baza danych, wyjście konsoli lub plik. *Winston* umożliwia logowanie komunikatów z następującymi poziomami logowania:

- error,
- warn,
- info,
- verbose,
- debug,
- silly.

Biblioteka umożliwia też zdefiniowanie własnych poziomów logowania oraz zachowania dla każdego z nich. Dodatkowo w ustawieniach domyślnych *Winston* przechwytuje wszystkie błędy na najwyższym poziomie aplikacji i loguje je.

4.11. Morgan

Morgan jest middleware-em do *ExpressJS*. Pozwala na logowanie żądań, odpowiedzi i odpowiadających im czasów. Middleware umożliwia zdefiniowanie własnych sposobów formatowania logów oraz definiowanie poziomu ich szczegółowości.

4.12. Angular 5

Angular to framework stworzony przez firmę Google w 2010. *Angular* jest framework-iem bazującym na architekturze *MVC* do tworzenia *frontend-u* stron internetowych. Od pierwszego wydania *Angular* przeszedł wiele zmian, takich jak na przykład przepisanie go na *TypeScript*.

Aplikacje napisane w *Angular* składają się z modułów. Moduły definiują komponenty i serwisy. Komponenty odpowiadają za logikę oraz za wygląd elementów za pomocą zdefiniowanych *template-ów*. Serwisy to singletony służące do dostarczania danych do komponentów. Templatey *Angular* przypominają *HTML*, ale dopiero po stronie przeglądarki są przetwarzane na poprawny *HTML*, gdzie są parsowane i renderowane. Komponenty definiują tagi i dyrektywy, które są rozwijane i wykonywane, aby utworzyć finalny wygląd strony [13].

W celu ułatwienia procesu rozwoju aplikacji w *Angular* i uniknięciu pisania dużej ilości kodu, *Angular* wyposażony jest w aplikację konsolową *angular-cli*. Aplikacja generuje strukturę folderów, pliki oraz podstawowe elementy kodu.

4.13. Bootstrap

Bootstrap jest frameworkiem CSS stworzonym przez programistów firmy Twitter. Jest to zestaw klas CSS oraz kontrolek *JavaScript* ułatwiających szybką budowę prostych stron internetowych zgodnych z *responsive design*. *Bootstrap* składa się z wielu komponentów, z których najbardziej podstawowym jest *grid* składający się z wierszy. Każdy wiersz składa się z kolumn o szerokości sumującej się do dwunastu jednostek.

Bootstrap składa się z wielu klas i komponentów, które tworzą podstawy nowoczesnych stron WWW. Przykładami takich komponentów są:

- **Scrollspy** – element nawigacji w obrębie jednej strony umożliwiający nawigację pomiędzy artykułami i sekcjami;
- **Modal** – pop-up umożliwiający pokazanie dodatkowych szczegółów lub potwierdzenie akcji;
- **Jumbotron** – kontener dla artykułów;
- **Alerts** – pasek ostrzegający lub informujący użytkownika o zdarzeniach;
- **Badges** – używane w celu przekazania użytkownikowi prostych dodatkowych informacji;
- **Cards** – karty organizujące przestrzeń kontenerów;
- **Carousel** – kontrolka wyświetlająca zdjęcia jako pokaz slajdów.

4.14. JQuery

Jquery to biblioteka rozszerzająca i ułatwiająca integrację *JavaScript* z przeglądarką. Biblioteka powstała w roku 2006 i szybko zyskała znaczną popularność. *Jquery* pozwoliło na rozwiązanie wielu problemów z niespójnościami *API* pomiędzy starszymi przeglądarkami oferując jednolite *API* dla wszystkich wspieranych przeglądarek. Na popularność *Jquery* składa się też fakt, że łatwo rozszerzyć ją o dodatkowe wtyczki. Społeczność web developerów ciepło przyjęła *Jquery* i szybko pojawiło się dużo rozszerzeń oraz komponentów. Wraz z ujednoliceniem *API* przeglądarek *Jquery* traci na popularności i stosowanie go jest coraz częściej uważane za złą praktykę [11].

Najważniejsze funkcjonalności *Jquery*:

- Selektory zgodne z CSS3 umożliwiające wygodniejsze trawersowanie *DOM*;

- Wygodna i spójna manipulacja elementami *DOM*, takimi jak atrybuty czy style;
- Rozbudowana obsługa zdarzeń na przykład ułatwione dezaktywacja handler-ów zdarzeń z elementu.

5. Implementacja

W rozdziale zostały opisane przyjęte konwencje podczas implementacji zaproponowanego prototypu, szczegóły wykonania oraz przedstawiono rezultaty pracy nad nim.

5.1. Klient

Interfejs użytkownika został wykonany we framework-u *Angular 5*, przy użyciu języka *TypeScript*. W projektowaniu interfejsu użytkownika wykorzystano bibliotekę *CSS Bootstrap 4* oraz dołączone do niej kontrolki. Aplikacja jest serwowana domyślnie na porcie 8080 za pomocą serwera *Node.JS* opisanego w rozdziale 5.5.

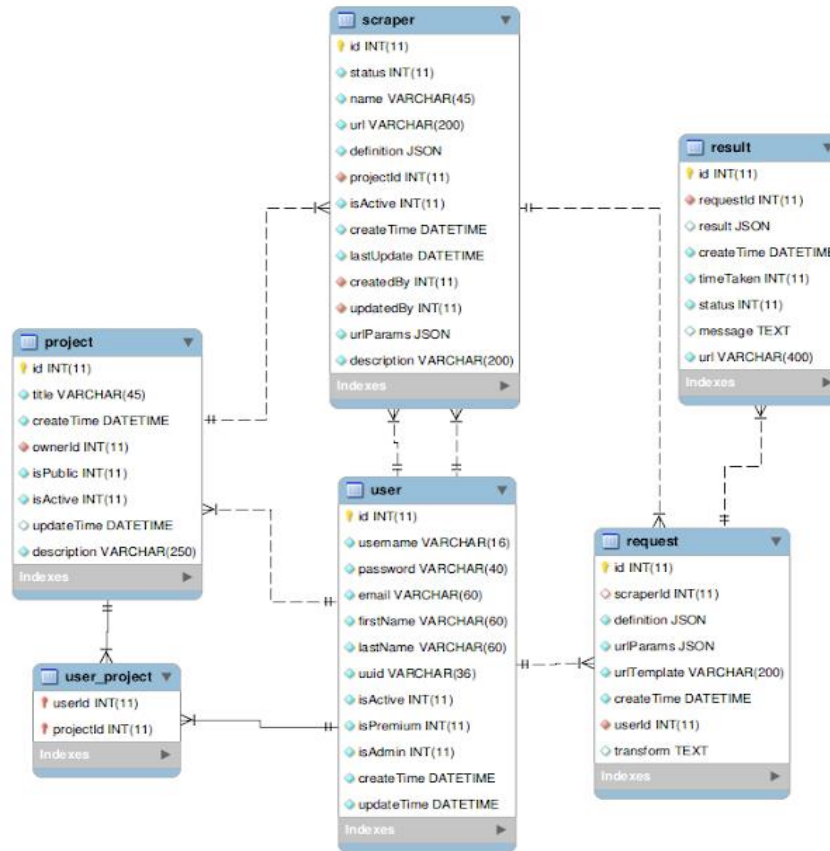
Aplikacja została podzielona na moduły:

- **User** – zawiera komponenty i serwisy odnoszące się do zarządzania użytkownikiem;
- **Project** – zawiera komponenty i serwisy odnoszące się do zarządzania projektami;
- **Scraper** – zawiera komponenty i serwisy odnoszące się do zarządzania skrapierami;
- **Cover** – zawiera komponenty opisujące landing page aplikacji;
- **Commons** – zawiera komponenty wspólne używane w obrębie aplikacji;
- **Navigation** – zawiera elementy nawigacji w portalu;
- **Session** – zawiera serwis odpowiadający za sesje użytkownika.

W rozdziale 6.1 opisano sposób użycia portalu. Jednym z najważniejszych elementów aplikacji webowej jest ekran definiowania skrapera przedstawiony na Rysunek 14. Ekran zawiera *iframe* ze stroną, na której definiowany jest skrapier. *URL* przetwarzany w *iframe* zbudowany jest następująco `<HOST>:<PORT>/api/document/<URL>`; Gdzie `<URL>` to właściwy *URL* skrapowanej strony. Przed wyświetleniem w *iframe* strona musi zostać przekierowana przez serwer by ominąć problemy związane z *Cross-Origin Resource Sharing* [16] dodatkowo strona rozszerzana jest o skrypt komunikujący się z portalem. Komunikacja odbywa się za pośrednictwem metody *postMessage* obiektu *Window*. Metoda ta jest przeznaczona do dwustronnej komunikacji z *iframe*-mami. Wstrzyknięty skrypt zajmuje się wyliczaniem selektorów oraz wszystkimi operacjami w obrębie wyznaczania jednego elementu.

5.2. Baza danych

Jako system zarządzania relacyjnymi bazami danych dla prototypu został wybrany *MySQL*.



Rysunek 6. Diagram przedstawiający strukturę bazy danych.

Rysunek 6 przedstawia strukturę bazy danych, składającą się z 6 tabel:

1. **User** – tabela przechowuje podstawowe dane użytkownika. Pole *password* zawiera *hash SHA2* hasła;
2. **Project** – tabela zawiera szczegóły projektu;
3. **User_Project** – tabela zawiera relację pomiędzy użytkownikami a projektami, do których należą;
4. **Scraper** – przechowuje definicje oraz szczegóły skrapersów;
5. **Request** – przechowuje szczegóły każdego żądania;
6. **Result** – zawiera rezultaty odpowiadających wywołań.

Jako zasadę implementacyjną przyjęto, że konto użytkownika wykorzystywane przez serwer do dostępu do bazy danych, ma uprawnienia wyłącznie do wywoływania procedur składowanych. Wszystkie operacje bazodanowe wykonywane są przez odpowiadające im procedury składowane. W

przypadku bardziej złożonych zapytań lub złączeń pomiędzy tabelami procedury korzystają z perspektyw zamiast bezpośredniego odwoływania się do tabel.

5.3. Konfiguracja aplikacji

Cała konfiguracja prototypu znajduje się w pliku *config.json* umieszczonym w głównym katalogu aplikacji serwerowej. Konfiguracja składa się z przedstawionych poniżej sekcji i parametrów:

- **Log** – opisuje konfigurację *logger-a*:
 - **Level** – minimalny logowany do pliku poziom zdarzeń. Wartości: error, warn, info, verbose, debug, silly;
 - **Path** – ścieżka, w której odkładane są logi;
 - **morganFormat** – nazwa wbudowanego formatu logów biblioteki *Morgan* służącej logowaniu żądań. Wartości: combined, common, dev, short, tiny;
- **Server** – opisuje konfigurację serwera:
 - **IP** – przyjmuje wartości 0.0.0.0 lub 127.0.0.1 w zależności od tego, czy serwer oczekiwać żądań pochodzących z *localhost* czy też z Internetu;
 - **Port** – port, na którym serwer nasłuchuje wywołań;
 - **Method** – wskazuje, czy serwer powinien działać w oparciu o *HTTP*, czy *HTTPS*;
 - **staticPath** - ścieżka, z której są serwowane statyczne zasoby.
- **Scraper** – opisuje pliki, które mają zostać wstrzyknięte do stron, które są przekierowywane w celu definicji skrapera:
 - **Injectables** – pliki lokalne wstrzykiwane do strony;
 - **JS** – lista plików *JavaScript*, które należy wstrzyknąć;
 - **CSS** – lista plików *CSS*, które należy wstrzyknąć;
 - **remoteInjectables** – pliki zdalne wstrzykiwane do strony:
 - **JS** – lista plików *JavaScript*, które należy wstrzyknąć;
 - **CSS** – lista plików *CSS*, które należy wstrzyknąć;
- **Database** – konfiguracja bazy danych:
 - **Host** – host bazy danych;
 - **User** – nazwa użytkownika - użytkownik musi posiadać uprawnienia do wykonywania procedur składowanych;

- **Password** – hasło do uwierzytelnienia w bazie danych;
- **Database** – nazwa bazy danych.

5.4. Logowanie

Wszystkie logi składowane są w katalogu wskazanym w konfiguracji opisanej w podrozdziale 5.3. Aplikacja loguje wszystkie żądania oraz ich podstawowe szczegóły do pliku *request.log*. Format logów żądań jest wskazany w konfiguracji.

Aplikacja dodatkowo loguje wszystkie ważniejsze zdarzenia do pliku *debug.log*. Minimalny poziom logowania jest konfigurowalny. Przykładowy wpis do logu znajduje się w Przykład 9. Format logu to: <data>T<czas>Z [<moduł aplikacji z którego pochodzi log>] <poziom zdarzenia>: <treść zdarzenia>

2018-01-18T15:30:00.697Z [API_PROJECT] info: Start get project with id : 424

Przykład 9. Przykładowa linia logów aplikacji.

5.5. Serwer

Aplikacja została napisana w *NodeJS*, przy zastosowaniu *ExpressJS* do zarządzania serwerem *HTTP*. Aplikacja jest podzielona na trzy warstwy:

- **API** – Definicje endpoint-ów oraz obsługa żądań. Warstwa zawiera podstawową logikę biznesową, odwołując się do modelu ustawia odpowiednie kody, odpowiedzi i treści wiadomości;
- **Model** – Warstwa waliduje poprawność i spójność danych. Przetwarza komunikaty na obiekty klas. Pośredniczy we wszystkich zapytaniach do warstwy bazy danych;
- **DB** – Warstwa przetwarza obiekty na zapytania bazodanowe i pobiera dane z bazy danych.

Udostępnione usługi zorganizowane są w taki sposób, by kod usługi zajmował się wyłącznie dostarczeniem reprezentacji żadanego zasobu. Wszystkie niezbędne szczegóły potrzebne do zrealizowania żądania oraz uwierzytelnianie i sprawdzanie, czy użytkownik ma wystarczające uprawnienia realizowane są za pośrednictwem *middleware*. Przykład 10 przedstawia przykładową definicję usługi. Zanim zostanie wykonana usługa *get*, by zwrócić reprezentację projektu, użytkownik jest najpierw uwierzytelniany za pomocą *middleware*-u *authenticate.basic* mającego na celu sprawdzić nagłówki *authenticate* metodą *basic*. Jeśli uwierzytelnienie powiedzie się, obiekt *request* zostanie uzupełniony o dane użytkownika. Następnie *projectMiddleware* pobierze szczegóły projektu i uzupełni obiekt *request* o pobrane dane. W ostatniej fazie *middleware* *authorization* sprawdzi listę

wymagań, które muszą być spełnione, żeby użytkownik miał uprawnienia do wykonania danej usługi na przykład sprawdzi, czy projekt jest publiczny lub czy żądający jest właścicielem projektu.

```
router.route('/:projectId')
  //200 OK
  //{
  //  "id": 168,
  //  "title": "test_project",
  //  "createTime": "2017-12-04T21:21:38.000Z",
  //  "ownerId": 430,
  //  "isPublic": 1,
  //  "isActive": 1,
  //  "updateTime": "2017-12-04T21:21:38.000Z"
  //}
  //401 Unauthorized
  //403 Forbidden
  //404 Not Found
  //500 Internal Server Error
  .get(authenticate.basic,
    projectMiddleware,
    authorization([utils.isHimself, utils.isPublicOrMember]),
    services.get
  )
  //200 OK
  //401 Unauthorized
  //403 Forbidden
  //404 Not Found
  //500 Internal Server Error
  .delete(authenticate.basic,
    projectMiddleware,
    authorization([utils.isHimself, utils.isOwner]),
    services.remove
  )
}
```

Przykład 10 Definicja usług *get* i *delete* w Project API.

Wszystkie serwisy zaimplementowane w aplikacji pokryte są testami automatycznymi sprawdzającymi ich działanie i poprawność. Testy napisane są w *Postman* i mogą być uruchomione za pomocą aplikacji *Postman* lub za pomocą komendy *npm test*.

Najważniejszą usługą zaimplementowaną na serwerze jest usługa wykonująca wskazany skrapler na zadanej stronie internetowej. Źródła do modułu służącego do skrapowania znajdują się w zasobie `src/scrapper-server/modules/scrapper/scrapper.js`. Moduł eksportuje pojedynczą funkcję *execute*.

```
const execute = function(definition, url) {
  let startTime = new Date().getTime();
  logger.info(`[${startTime}] Start executing scraper`);
  logger.silly(`[${startTime}] Url is : ${url} Defined is ${definition}`);
  var result = {};
  var isError = false;
  var exception = null;
  return new Promise(function(fulfill, reject) {
    JSDOM.fromURL(url, {
      runScripts: "dangerously",
      pretendToBeVisual: true,
      resources: "usable"
    }).then(dom => {
      logger.info(`[${startTime}] DOM Object built. start scrapping`);
      var result = parseDOM(dom, definition, startTime);
      dom.window.close();
      logger.info(`[${startTime}] Scraping finished, result is ${JSON.stringify(result)}`);
      let endTime = new Date().getTime();
      fulfill({ startTime: startTime, endTime : endTime, result : result,
        isError : isError, exception : exception});
    }).catch(err => {
      logger.error(`[${startTime}] error while scrapping ${err}`);
      exception = err;
      isError = true;
      let endTime = new Date().getTime();
      reject({ startTime: startTime, endTime : endTime, result : result,
        isError : isError, exception : exception});
    });
  });
}
```

Przykład 11 Fragment kodu skrapującego stronę WWW

Fragment kodu przedstawiony w Przykład 11 zawiera kod funkcji *execute*. Funkcja przyjmuje dwa argumenty:

- obiekt *JSON* zawierający definicje skrapera, który ma pobrać dane ze strony;
- *URL* strony, która ma zostać skrapowana.

Funkcja w pierwszej kolejności wyznacza czas swojego startu w celu ułatwienia korelacji logów oraz by monitorować czas działania skrapera. Funkcja natychmiast zwraca nową instancję obiektu *Promise*. Obietnica zostaje spełniona, gdy skrapowanie zostanie zakończone sukcesem w innym wypadku kończy się odrzuceniem. W funkcji przekazanej do obietnicy pobierana jest pełna treść strony, znajdujące się na niej zasoby i skrypty. Biblioteka *JSDOM* wykonuje zawarty na stronie *JavaScript* w celu wyrenderowania całości strony. Gdy obiekt *DOM* jest gotowy, wywoływana jest funkcja pomocnicza *parseDOM*. Funkcja pomocnicza przetwarza obiekt *DOM* i wybiera z niego wymagane informacje. Wynik skrapowania zostaje przetworzony i zwracany jest wraz ze spełnieniem obietnicy. Metoda *parseDOM* pośredniczy w skrapowaniu wywołując metodę *getValue* w kontekście całej strony.

```
const getValue = function getValue($, element, definition, result,
startTime) {
  Object.keys(definition).forEach((label) => {
    var scraperElement = definition[label];
    if(typeof scraperElement === 'string') {
      result[label] = getText($, element, scraperElement);
      logger.silly(`[${startTime}] processing scraper element : ${label}
path is ${scraperElement} value ${result[label]}`);
    } else {
      logger.silly(`[${startTime}] processing scraper collection : ${label}
path is ${scraperElement.path} common ${scraperElement.common}`);
      result[label] = [];
      getCollection($, element, scraperElement.path,
scraperElement.common).each(function(idx) {
        var out = getValue($, this, scraperElement.elements, {}, startTime)
        logger.silly(`[${startTime}] processing scraper collection :
${label} index ${idx}`);
        if(!hasAllPropertiesNull(out)) result[label].push(out);
      })
    }
  })
  return result;
}
```

```
const parseDOM = function(dom, definition, startTime) {  
    var $ = require("jquery")(dom.window);  
    return getValue($, null, definition, {}, startTime);  
}
```

Przykład 12 Fragment kodu przedstawiającego wykonywanie skrapownia

Metoda *getValue* przyjmuje następujące argumenty:

- Instancję biblioteki JQuery używanej w celu trawersowania obiektu *DOM*;
- Element, na którym wykonywane jest skrapowanie;
- Obecna zgłębienie definicji skrapera
- Obiekt wyniku
- Czas startu.

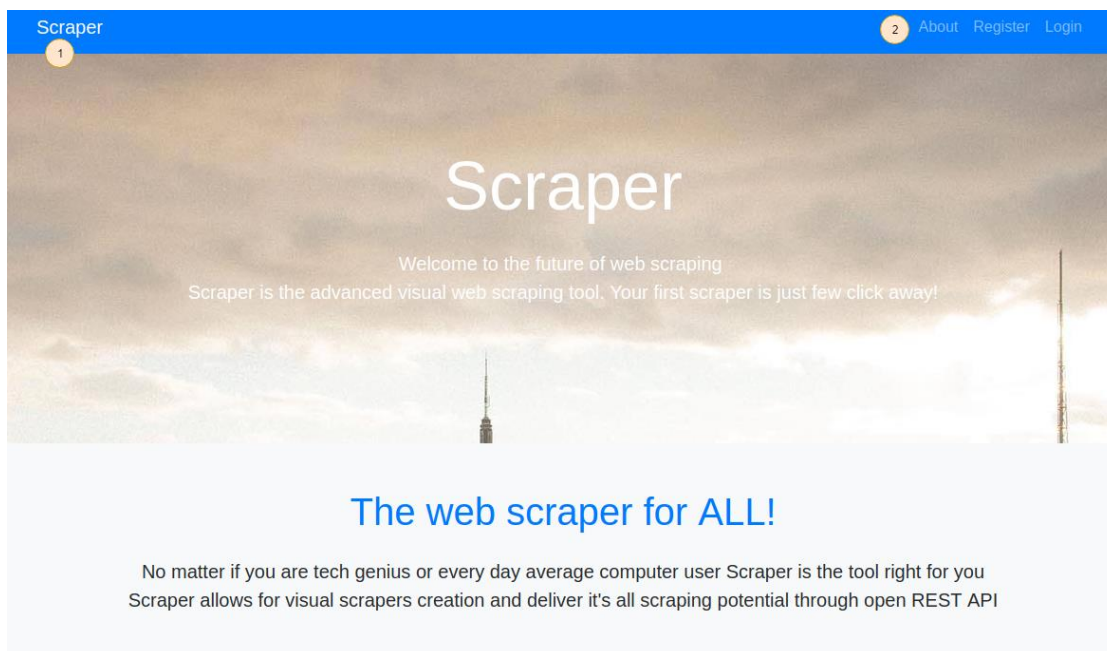
Przykład 12 prezentuje kod funkcji *getValue* i *parseDOM*. Funkcja iteruje po kluczach obiektu definicji skrapera. Dla każdego klucza sprawdzane jest czy jest on selektorem czy deklaracją kolekcji. W przypadku, gdy klucz jest selektorem wartość wybierana jest z elementu wskazanego przez selektor i przypisana do obiektu przechowującego wynik. Jeśli dla klucza wskazana była kolekcja pobierane są z obiektu wszystkie elementy spełniające selektory opisujące kolekcję. Dla każdego z wybranych elementów wykonywana jest funkcja *getValue* wraz z zagłębioną definicją skrapera. Funkcja kończy się, gdy cała definicja skrapera zostanie przetworzona, a wynik zostanie zapisany w zdefiniowanych miejscach.

6. Zastosowanie prototypu

Następne podrozdziały opisują sposoby użycia przygotowanego prototypu oraz prezentują przykładowy skrafer stworzony za pomocą portalu i rezultaty jego zastosowania.

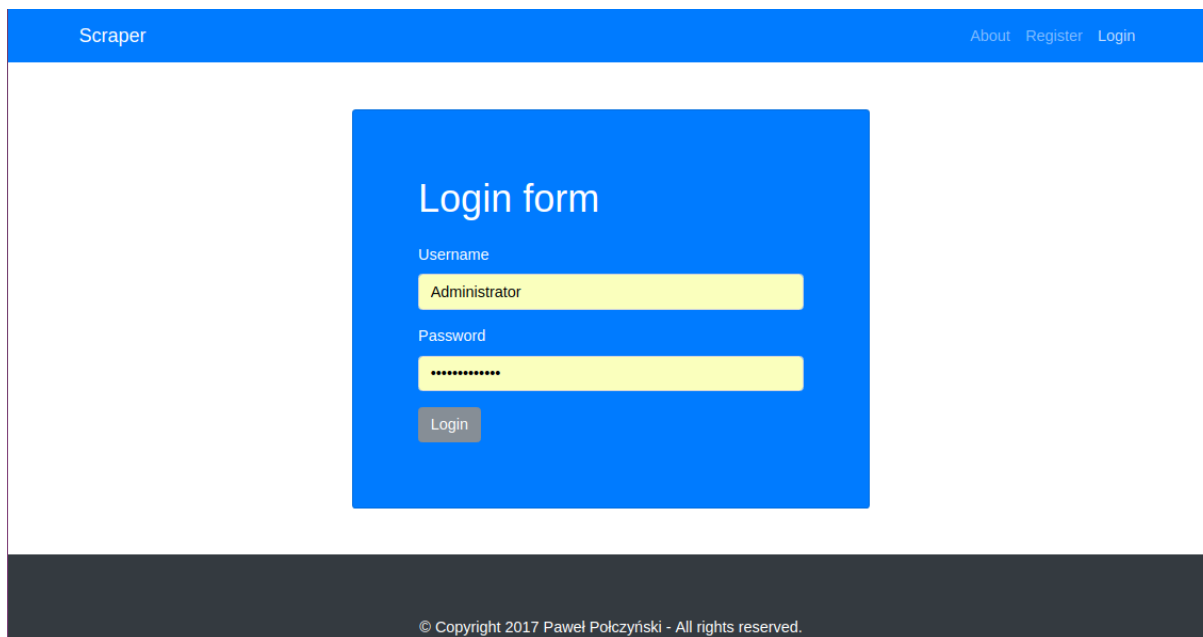
6.1. Sposoby użycia portalu

Po otwarciu aplikacji użytkownik zobaczy prosty *landing page* zawierający szczegóły biznesowe platformy na przykład informujące o ofercie i świadczonych usługach.



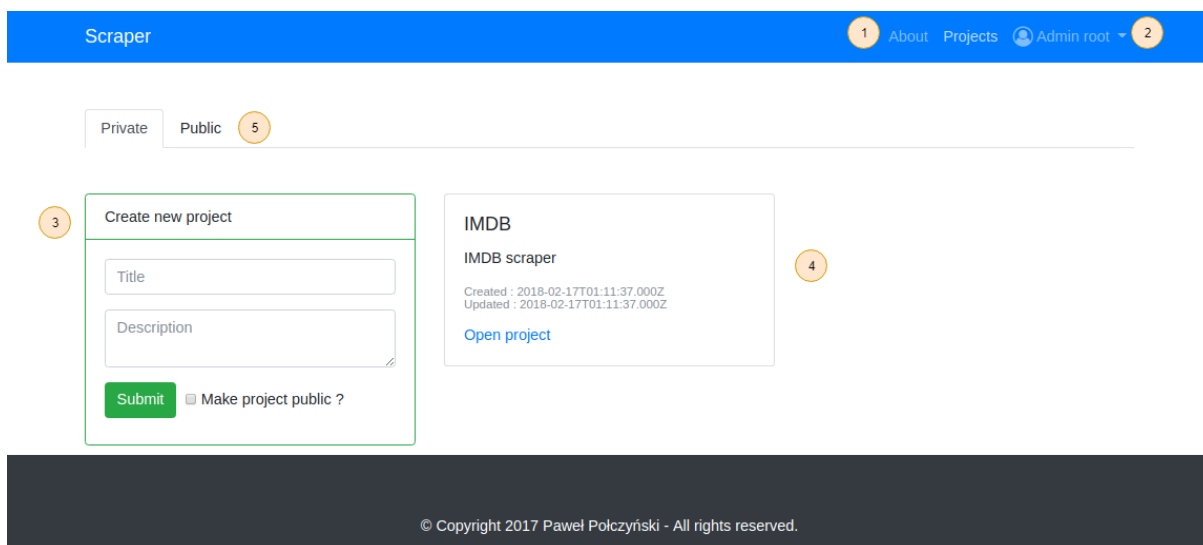
Rysunek 7. Ekran landing page aplikacji skrafer.

Na Rysunek 7 widać *landing page* przygotowanego prototypu. Nawigacja strony znajduje się na górnej belce oznaczonej na rysunku numerem 1. Znajdują się tam też linki do zakładki oznaczone numerem 2. Kliknięcie w zakładkę *About*, jak i w logo powoduje przejście do *landing page*. Link *Register* prowadzi do formularza, dzięki któremu można zarejestrować nowego użytkownika. Dla wygody w *landing page* została osadzona sekcja rejestracji. Link *Login* prowadzi do formy logowania.



Rysunek 8. Ekran logowania.

Użytkownik loguje się do aplikacji po kliknięciu w link *Login*, za pomocą formy do logowania, na której podaje swoją nazwę użytkownika oraz hasło. Forma logowania prezentowana jest na Rysunek 8.



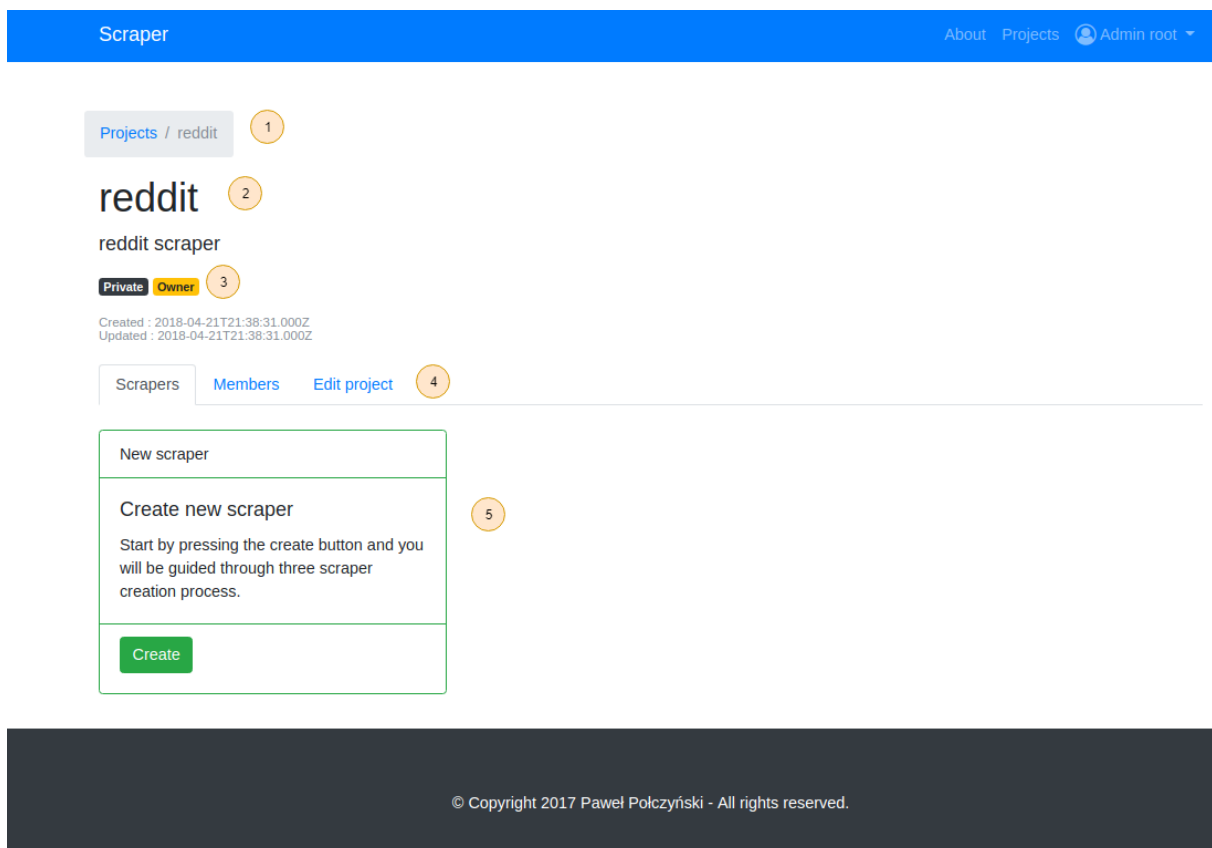
Rysunek 9. Widok projektów zalogowanego użytkownika.

Po zalogowaniu się do portalu widok nawigacji ulega zmianie i dostępne są nowe linki. Widok zalogowanego użytkownika prezentowany jest na Rysunek 9. Dodatkowo oznaczano na nim elementy widoku projektów:

1. Zmieniona nawigacja. Dostępna jest zakładka *Projects*, prezentowana na rysunku oraz możliwość wejścia w szczegóły użytkownika;

2. Szczegóły użytkownika po rozwinięciu *dropdownu*. Rozwinięcie umożliwia wylogowanie się z aplikacji oraz edycję danych;
3. Forma służąca do utworzenia nowego projektu. Aby utworzyć nowy projekt, użytkownik musi podać tytuł oraz opis nowo tworzonego projektu. Domyślnie utworzony projekt nie jest projektem publicznym, można go uczynić publicznym poprzez zaznaczenie *checkboxa*;
4. Lista zdefiniowanych już projektów zawierająca podstawowe dane o projekcie oraz link otwierający szczegóły projektu;
5. Nawigacja pomiędzy projektami publicznymi i prywatnymi. Zmieniając zakładkę użytkownik przechodzi do analogicznego widoku.

Nowoutworzone projekty trafiają na listę jako nowa karta. Aby przejść do szczegółów projektu i możliwości definiowania skrapersów należy przejść do widoku szczegółów projektu, pokazanego na Rysunek 10, klikając w link *Open project*.

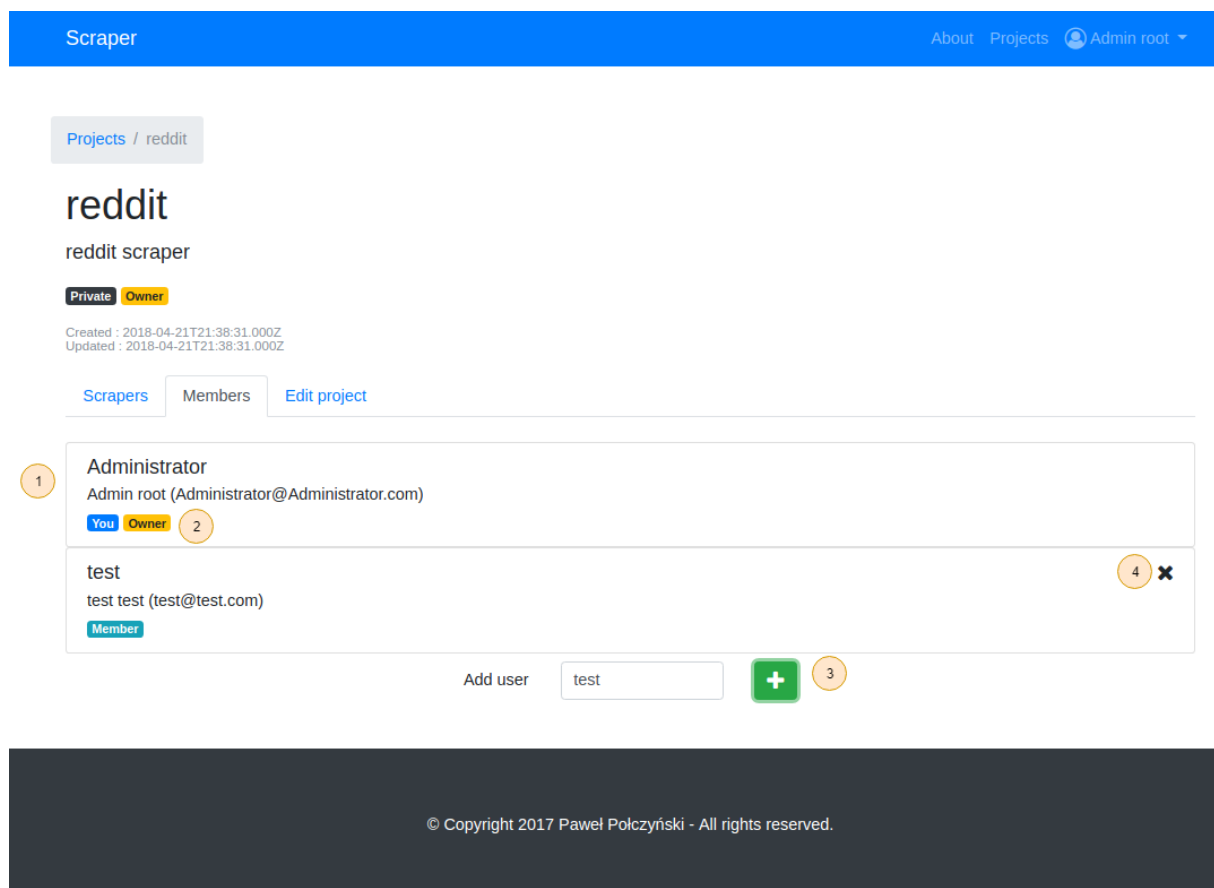


Rysunek 10. Widok szczegółów projektu.

Po wejściu na widok szczegółów projektu użytkownik widzi listę skrapersów zdefiniowanych wewnątrz projektu oraz może podjąć konkretne akcje. Elementy ekranu zostały oznaczone odpowiednimi liczbami:

1. Nawigacja typu *breadcrumb* pozwalająca na szybki powrót do widoku projektów;
1. Szczegóły projektu;
2. *Badge* wskazujący na cechy projektu i poziom dostępu zalogowanego użytkownika;
3. Zakładki umożliwiające przechodzenie pomiędzy widokami projektu:
 - a. Widok *Scrapers* wyświetla listę zdefiniowanych skrapersów w projekcie;
 - b. Widok *Members* pozwala zarządzać osobami w projekcie;
 - c. Widok *Edit project* pozwala na zmianę podstawowych szczegółów projektu;
4. Lista skrapersów oraz karta umożliwiające przejście do tworzenia nowego skrapera.

Rysunek 11 przedstawia widok zarządzania użytkownikami w projekcie. Daje on możliwość właścicielom projektów dodawania i usuwania członków zespołu.



Rysunek 11. Widok zarządzania użytkownikami w projekcie.

Istotne elementy interfejsu oznaczone zostały na Rysunek 11 za pomocą liczb:

1. Karty przedstawiające użytkowników będących członkami zespołu. Widok nie jest dostępny dla gości w projektach publicznych;
2. *Badge* prezentujące poziom uprawnień użytkownika oraz umożliwia łatwą lokalizację zalogowanej osoby na liście;
3. Element umożliwiający dodawanie członków zespołu do aplikacji. Dostępny jest on tylko dla właścicieli projektu;
4. Przycisk umożliwia wykluczenie osoby z projektu. Dostępny tylko dla właścicieli projektu. Dodatkowo nie ma możliwości usunięcia właściciela z projektu.

Analogicznie po przejściu do zakładki *Edit project* właściciel projektu będzie mógł zmienić szczegóły projektu, a także go usunąć. Powyższe funkcjonalności przedstawione są na Rysunek 12.

The screenshot shows the 'Edit project' interface for a project named 'reddit'. At the top, there is a blue navigation bar with the word 'Scraper' on the left and links for 'About', 'Projects', and 'Admin root' on the right. Below the navigation bar, a breadcrumb trail shows 'Projects / reddit'. The main heading is 'reddit', followed by 'reddit scraper'. There are two status buttons: 'Private' (black) and 'Owner' (yellow). Below these, the creation and update timestamps are displayed: 'Created : 2018-04-21T21:38:31.000Z' and 'Updated : 2018-04-21T21:38:31.000Z'. A tabbed interface has three tabs: 'Scrapers' (active), 'Members', and 'Edit project'. The 'Edit project' tab contains a form with the following elements: a title input field with 'reddit', a description input field with 'reddit scraper', and a checkbox labeled 'Is project public ?' which is currently unchecked. Below the form, there is a blue 'Update' button (labeled 2) and a red 'Delete project' button (labeled 3). The 'Delete project' button is preceded by the text 'Delete project:'. At the bottom of the page, a dark grey footer contains the copyright notice: '© Copyright 2017 Paweł Polczyński - All rights reserved.'

Rysunek 12. Widok edycji projektu.

Na rysunku zaznaczono istotne elementy interfejsu:

1. Formularz zmiany szczegółów projektu;
2. Przycisk zapisujący zmiany w projekcie;

3. Przycisk usuwający projekt. Przed usunięciem projektu zostanie zaprezentowany *modal* proszący o potwierdzenie tej akcji.

Zasadniczą częścią interfejsu graficznego jest widok tworzenia skrapera. Dostępny poprzez kliknięcie w przycisk *Create* widoczny na Rysunek 10.

Scraper

About Projects Admin root

Projects / reddit / Create scraper

1 Basic scraper details

Scraper name: reddit

Description: Reddit basic scraper

Url: 2 https://www.reddit.com/r/{subreddit}/
To add parameters to url enter them between {} like www.example.com/?query={query}

Resulting Url: 3 https://www.reddit.com/r/{subreddit}/

Value of subreddit: JavaScript 4

Next →

© Copyright 2017 Paweł Polczyński - All rights reserved.

Rysunek 13 Widok tworzenia skrapera - wprowadzanie szczegółów skrapera.

Kreator skrapera przeprowadzi użytkownika poprzez następujące kroki:

- wypełnienie podstawowych szczegółów skrapera;
- zdefiniowanie skrapera;
- testowanie.

Pierwszym krokiem w celu utworzenia nowego skrapera jest wypełnienie szczegółów za pomocą widoku pokazanego na Rysunek 13. Na rysunku liczbami zaznaczono istotne jego elementy:

1. Formatka umożliwiająca wypełnianie szczegółów skrapera;
2. *URL* do strony, która będzie skrapowana za pomocą tworzonego skrapera. Zdefiniowany tu *URL* zawiera zmienne wprowadzone poprzez umieszczenie ich pomiędzy znakami {} w ciągu znaków *URL*;
3. Wynikający *URL*. Wyrażnie zostają oznaczają zmienne, przez co użytkownik może mieć pewność, że wprowadził poprawnie sparametryzowany *URL*;

4. Sekcja wpisywania parametrów zdefiniowanych w *URL*. Podane tu wartości posłużą otwarciu odpowiedniej strony, na której definiowany będzie skrapler.

Po wypełnieniu szczegółów skrapera należy przejść dalej w celu zdefiniowania części zasadniczej skrapera. Widok definiowania skrapera widoczny jest na Rysunek 14.

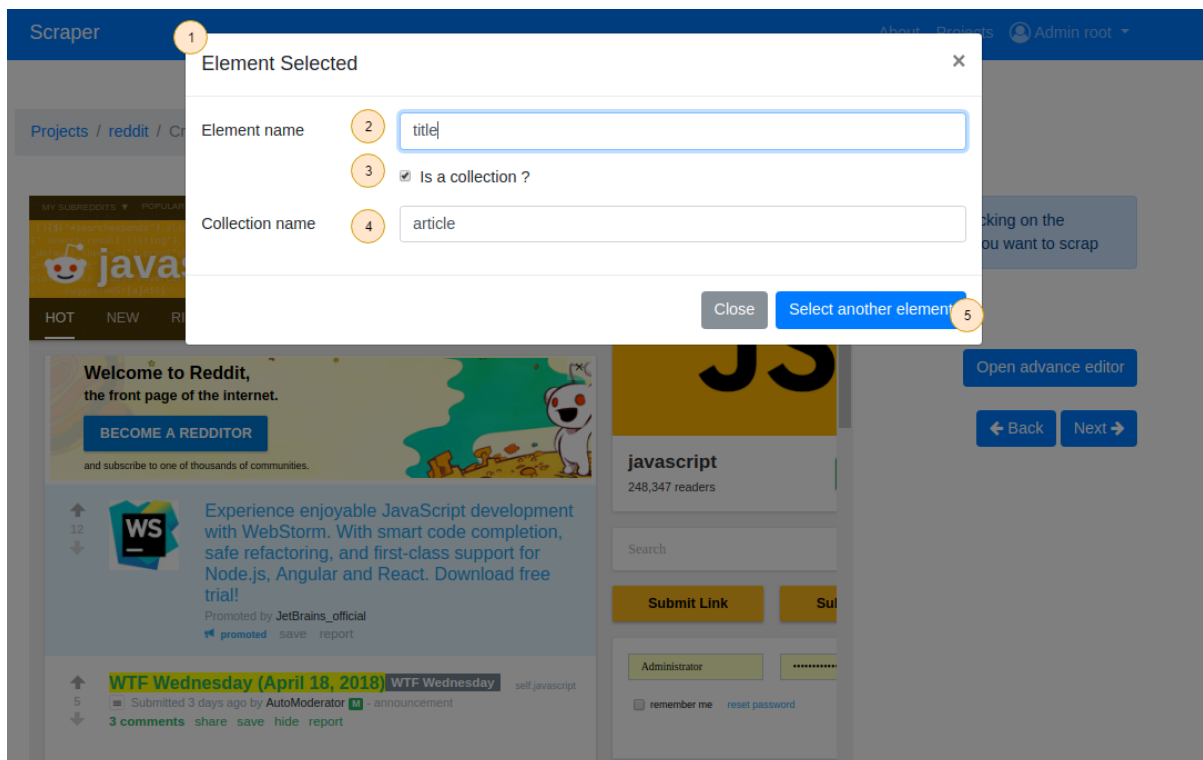


Rysunek 14. Widok definiowania skrapera.

Na Rysunek 14 zaznaczono istotne elementy interfejsu:

1. Wiadomość informująca użytkownika o dostępnych akcjach związanych z definiowaniem skrapera;
2. Wizualne odwzorowanie definicji skrapera. W tym przypadku reprezentuje skrapler dla pojedynczego elementu na stronie. Najeżdżenie na niego kursorem spowoduje podświetlenie odpowiadającego elementu na załadowanej stronie;
3. Przycisk otwierający zaawansowany edytor, poprzez który użytkownik może edytować wygenerowane selektory;
4. Załadowana strona, na której tworzona jest definicja skrapera;
5. Podświetlony element, którego kliknięcie spowoduje otwarcie *modalu* do opisanie elementu skrapowanego.

Po zaznaczeniu elementu i potwierdzeniu wyboru kliknięciem zostanie otwarty *modal* widoczny na Rysunek 15, pozwalający na opisanie wybranego fragmentu.



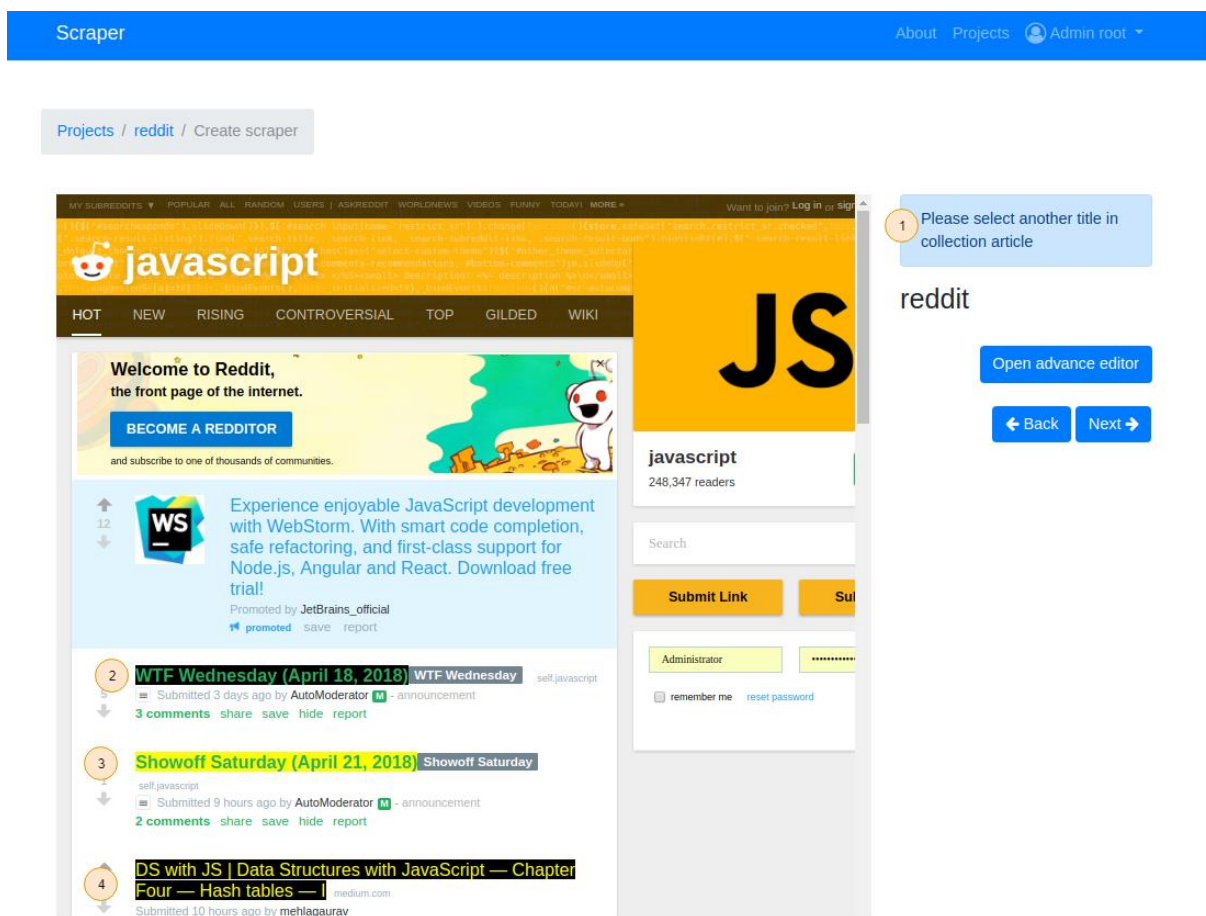
Rysunek 15. Modal potwierdzenia wyboru elementu.

Modal będzie wyglądał różnie, w zależności od kontekstu. Jeśli kliknięty element będzie znajdował się w już zdefiniowanej kolekcji, dostępna będzie lista rozwijana, umożliwiająca wybór istniejącej kolekcji. Na Rysunek 15 oznaczono ważniejsze elementy *modalu*.

1. *Modal* służący do wprowadzenia;
2. Nazwa elementu, która znajdzie odwzorowanie w obiekcie będącym wynikiem skrapowania;
3. *Checkbox* wskazujący na to, że element jest elementem nowej kolekcji;
4. Nazwa nowej kolekcji;
5. Przycisk zatwierdzający wybór. Będzie on różnił się kontekstualnie. W przypadku wyboru pojedynczego elementu lub wskazania, że element jest częścią już istniejącej kolekcji przycisk będzie zamykał *modal* i zapisywał element. W przypadku wyboru definicji nowej kolekcji użytkownik zostanie poproszony o wskazanie jeszcze jednego elementu danej kolekcji.

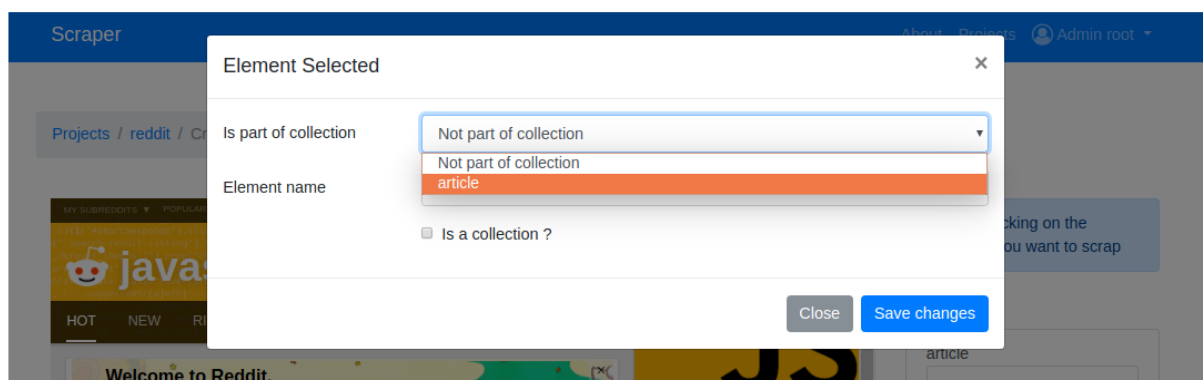
Kiedy definiowana jest kolekcja, sposób wyboru elementów na ekranie zmienia się, by ułatwić obsługującemu poprawne wybranie elementów kolekcji. Na Rysunek 16 widać, że wiadomość dla użytkownika (oznaczona na obrazku numerem 1) uległa zmianie. Zmiany widoczne są też na załadowanej stronie. Wybrany inicjalnie element podświetlony jest innym kolorem (na rysunku zaznaczony numerem 2). Gdy wybierany zostanie bratni element w kolejnej części kolekcji, zostanie on podświetlony w sposób analogiczny jak w przypadku wyboru pojedynczego elementu (numer 3 na

rysunku). Jednocześnie podświetlone zostaną wszystkie następne pasujące do definicji elementy (numer 4 na rysunku).



Rysunek 16. Ekran podczas definiowania kolekcji.

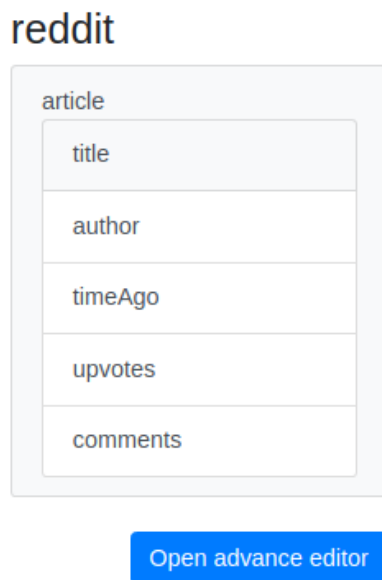
Po zdefiniowaniu kolekcji dodawanie kolejnych elementów do niej sprowadza się do wybrania pożądanej nazwy kolekcji z listy rozwijanej widocznej na Rysunek 17.



Rysunek 17. Dodawanie nowych elementów do zdefiniowanej kolekcji.

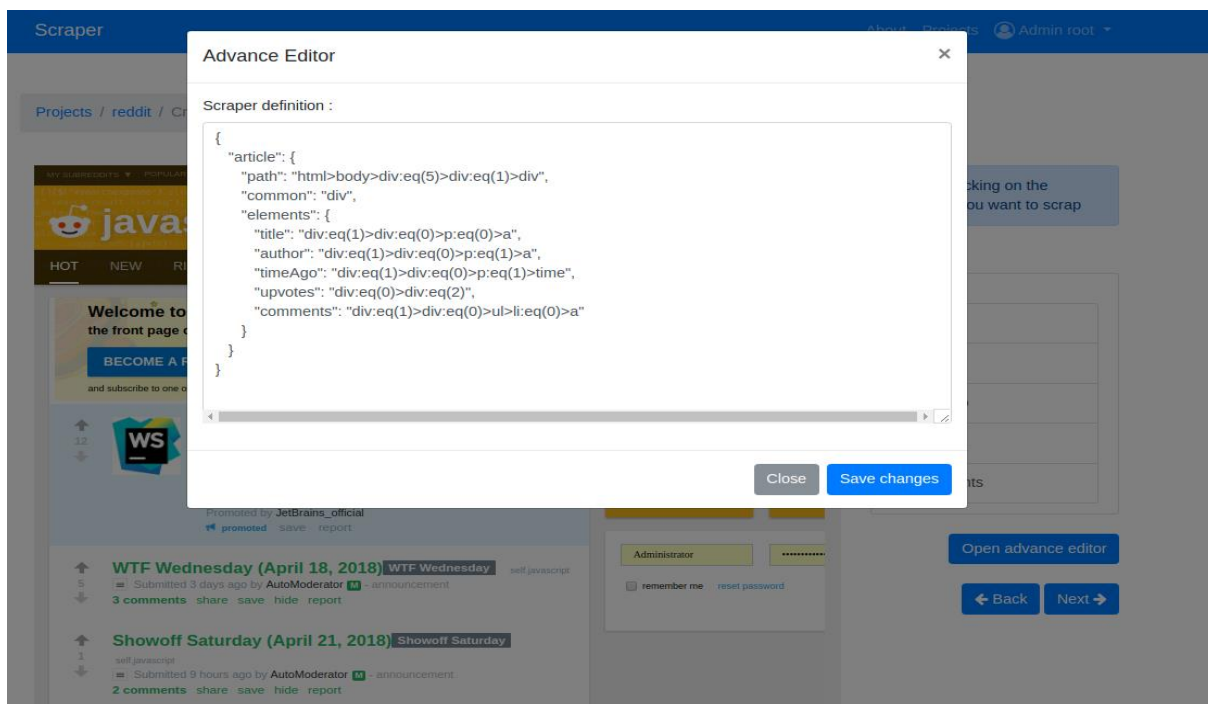
Lista rozwijana zawiera tylko nazwy kolekcji, wewnątrz których wskazany element mógłby się znajdować. W tym momencie użytkownik ma możliwość wskazania, że nowy element kolekcji będzie

nową zagnieżdżoną kolekcją. Kolekcje widoczne są w bocznym panelu w sposób przedstawiony na Rysunek 18. Elementy w kolekcji są umieszczone w elemencie nadrzędnym, co w sposób wizualny zaznacza ich hierarchię.



Rysunek 18. Reprezentacja kolekcji.

Widok definicji skrapera wyposażony jest w możliwość edycji wygenerowanych selektorów. Dostępny jest poprzez guzik oznaczony numerem 3 na Rysunek 14. Po kliknięciu w przycisk otwiera się *modal*.



Rysunek 19. Rysunek przedstawiający zaawansowany edytor selektorów.

Modal edycji skrapera przedstawiony jest na Rysunek 19 i zawiera pole edycji. Pole zawiera obiekt *JSON* zdefiniowany zgodnie ze schematem opisanym w rozdziale 3.3. Po zapisaniu zmian, reprezentacja skrapera w prawym panelu zostaje automatycznie zaktualizowana, a użytkownik może sprawdzić poprawność wprowadzonych zmian poprzez najechanie kursorem na odpowiadające elementy w panelu, co spowoduje podświetlenie ich na załadowanej stronie.

Po zdefiniowaniu skrapera użytkownik przechodzi do następnego ekranu, na którym może przetestować poprawność działania nowo utworzonego skrapera, na przykład poprzez podanie innych parametrów wejściowych.

Scraper About Projects Admin root

Projects / reddit / Create scraper

Test your scraper

Url template <https://www.reddit.com/r/subreddit>

Value of [subreddit](#) 1 worldnews

Back Test Create 2

Result 3

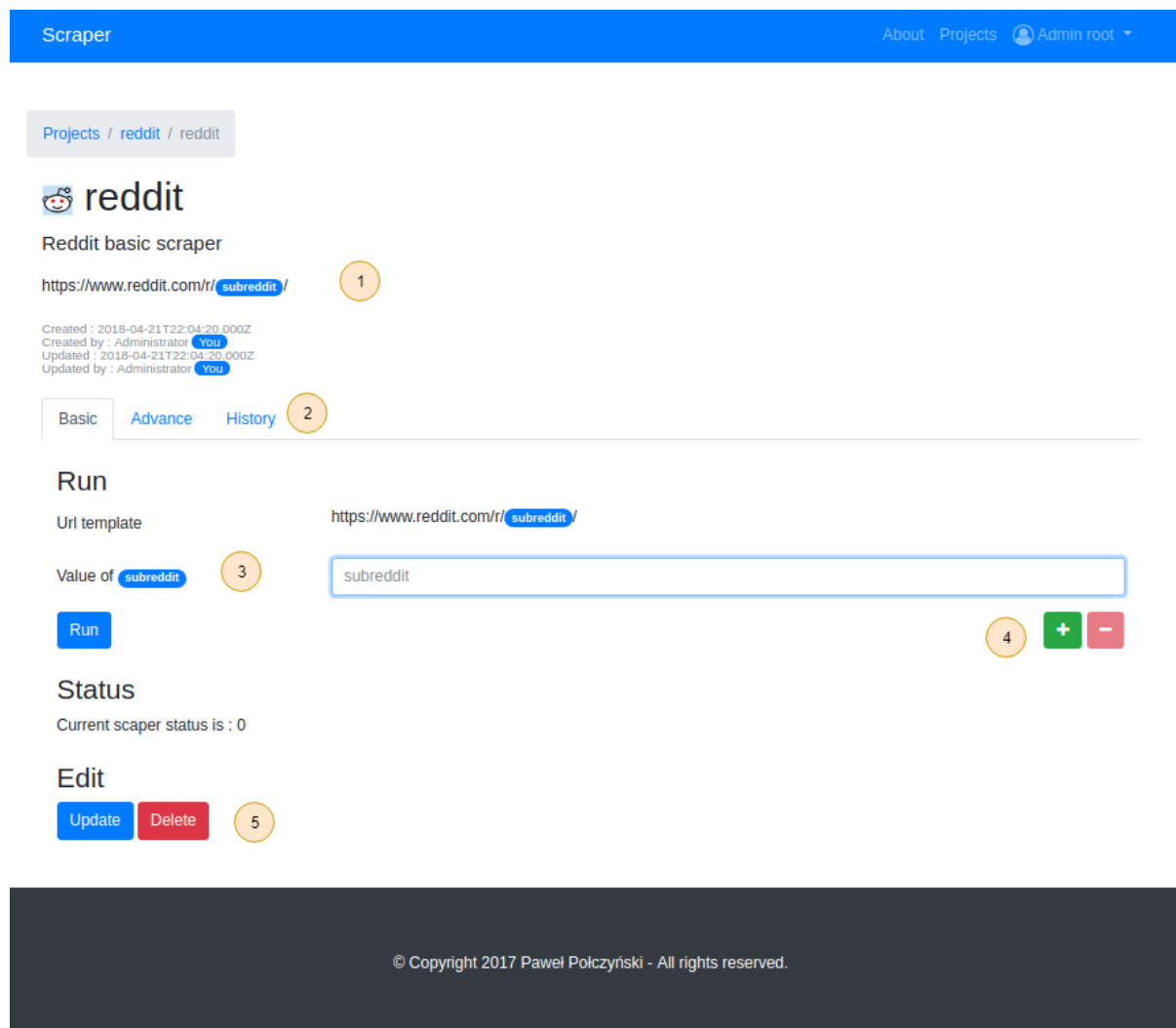
article	title	Report: Gunfire and explosions have been reported outside the home of the Saudi King in Riyadh
	author	OSINTElephant
	timeAgo	2 hours ago
	upvotes	4735
	comments	441 comments
	title	World Bank recommends that countries eliminate minimum wage, dismantle wrongful dismissal rules and contractual protections for workers
	author	xrm67
	timeAgo	5 hours ago
	upvotes	4675
comments	1042 comments	
title	Number of Beheadings in Saudi Arabia Rises by 70%	
author	middleeastnewsman	
timeAgo	8 hours ago	

Rysunek 20. Widok testowania nowego skrapera.

Na Rysunek 20 przedstawiono widok testowania skrapera. Kluczowym elementem widoku jest możliwość wprowadzenia nowych wartości podstawionych do *URL* w celu przetestowania skrapera. Wartości wprowadzone są w pole oznaczone numerem 1 na Rysunek 20. W sekcji nr 3 przedstawione są elementy skrapowania strony za pomocą stworzonej definicji po naciśnięciu przycisku *Test* w sekcji oznaczonej numerem 2. Po przetestowaniu proces tworzenia skrapera zostaje zakończony przez

naciśnięcie przycisku *Create*. W przypadku problemów ze skrapierem użytkownik może wycofać się do ekranu definiowania poprzez kliknięcie przycisku *Back*.

Po wciśnięciu *Create* utworzony skrapier jest gotowy do użycia, a użytkownik zostaje przeniesiony do widoku szczegółów skrapera zaprezentowanym na Rysunek 21.



Rysunek 21. Widok szczegółów utworzonego skrapera.

Z poziomu ekranu szczegółów widoczne są następujące elementy:

1. Podstawowe informacje o utworzonym skrapierze;
2. Nawigacja pomiędzy:
 - a. **widokiem podstawowym** - dającym możliwość uruchomienia i edycji skrapera;
 - b. **widokiem zaawansowanym** - opisującym *API*;
 - c. **widokiem historii** - zawierającym historię wywołań;
3. Panel umożliwiający podanie parametrów *URL*;

4. Przyciski +/- umożliwiają dodanie lub usunięcie dodatkowych parametrów, co umożliwia skrapowanie wielu stron naraz;
5. Przyciski umożliwiające zarządzanie skrapierem. Po kliknięciu *Update* użytkownik zostanie przeniesiony do widoku analogicznego do widoku tworzenia skrapera, który umożliwi mu zmianę szczegółów oraz definicji skrapera. Przycisk *Delete* pozwala na usunięcie skrapera.

Po kliknięciu przycisku *Run* rozpocznie się proces skrapowania a po jego zakończeniu przeglądarka rozpocznie ściąganie pliku *JSON* zawierającego wyniki.

[Projects](#) / [reddit](#) / reddit

Reddit basic scraper

<https://www.reddit.com/r/subreddit/>

Created : 2018-04-21T22:04:20.000Z
 Created by : Administrator [YOU](#)
 Updated : 2018-04-21T22:04:20.000Z
 Updated by : Administrator [YOU](#)

[Basic](#) | [Advance](#) | [History](#)

Services

POST /api/execute/464

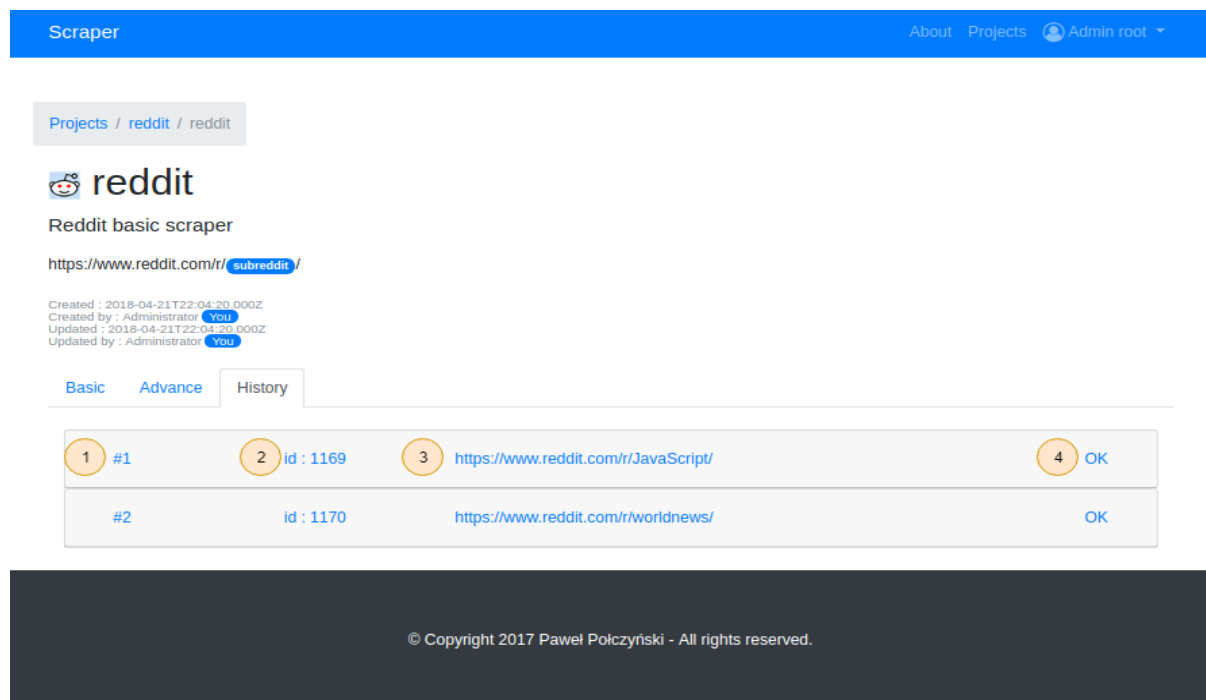
Service runs the scraper substituting variables in URL tempat with the values passed in the body of the request.

Request		Response	
Name	Description	Name	Description
Header	Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=	Status	200 Success 401 Unauthorized 403 Forbidden 404 Not Found 421 Misdirected Request 422 Unprocessable Entity 500 Internal Server Error
Body	<pre>{ "transform": "function(data) {return data;}", "urlParams": [{ "subreddit": "example subreddit" }] }</pre> urlParams : list of parameters to be substituted in url (Optional) transform : function to transform the result of scraping. Reutrn modified data.	Body	<pre>[{ "timeTaken": 123, "requestId": 1, "result": { "article": [{ "title": "example title", "author": "example author", "timeAgo": "example timeAgo", "upvotes": "example upvotes", "comments": "example comments" }] } }, "status": "0".]</pre>

Rysunek 22. Widok szczegółów skrapera przeznaczonych dla użytkowników posiadających wiedzę techniczną.

Użytkownik może zobaczyć wszystkie zdefiniowane *endpoint*-y dla danej instancji skrapera, po przejściu do zakładki widoku zaawansowanego. Widok przeznaczony jest dla programistów oraz

użytkowników posiadających wiedzę techniczną, którzy będą wykorzystać utworzony skrapler w aplikacji zewnętrznej. W kolejnych kartach opisane są szczegółowo wejścia i wyjścia dostępnych usług.



Rysunek 23. Widok historii skrapera.

Ostatnia zakładka została przedstawiona na Rysunek 23. Historia zawiera szczegóły odpowiedzi oznaczone na rysunku za pomocą liczb:

1. Numer porządkowy wywołania skrapera;
2. Identyfikator żądania, który można wykorzystać, by pobrać wszystkie szczegóły żądania i odpowiedzi;
3. Wynikowy *URL* zasobu, który był skrapowany;
4. Status odpowiedzi skrapera. Jeśli skrapowanie zakończyło się sukcesem status będzie miał wartość *OK*, jeśli podczas skrapowania wystąpiły problemy *NOT OK*.

6.2. Skrapler

Poza możliwością wykonania skrapowania za pośrednictwem strony, możliwe jest też łatwe skrapowanie za pomocą *API*. Po wygenerowaniu skrapera za pomocą platformy w zakładce

Zawansowane, widocznej na Rysunek 22, opisane są serwisy, które można wykonać za pośrednictwem dowolnego klienta umożliwiającego wysłanie żądania *HTTP*.

```
curl -X POST \
http://localhost:8080/api/execute/464 \
  H 'Authorization: Basic QWRtaW5pc3RyYXRvcjpBZG1pbmlzdHJhdG9y' \
  H 'Cache-Control: no-cache' \
  H 'Content-Type: application/x-www-form-urlencoded' \
  d
urlParams=%5B%7B%22subreddit%22%20%3A%20%22JavaScript%22%7D%5D
```

Przykład 13. Przykład wywołania skrapera za pomocą *cURL*.

W Przykład 13 prezentowana jest instrukcja służąca do wykonania skrapera za pomocą wieloplatformowej biblioteki *cURL*. Opis wszystkich interfejsów znajduje się w rozdziale 3.4.

6.3. Przykładowy skrapier zbudowany za pomocą prototypu

W celu zaprezentowania możliwości przygotowanego rozwiązania utworzono skrapier do pobierania szczegółów filmów dostępnych na stronie *imdb.com*. Po utworzeniu projektu zdefiniowany został nowy skrapier. Utworzono go dla ścieżki *https://www.imdb.com/title/:id* gdzie *:id* to unikatowy identyfikator filmu w systemie IMDB.

Na Rysunek 24 przedstawiono ekran definicji skrapera wraz ze zdefiniowanym skrapierem. Zakres skrapowanych danych o filmach oraz odpowiadające im nazwy w zbudowanym skreperze:

- Tytuł (title);
- Data premiery (date);
- Długość trwania (time);
- Kolekcja aktorów filmu (stars) z polem imię i nazwisko aktorem(star);
- Kolekcja gatunki (genres) z polem gatunek (genre);
- Reżyser (director)
- Opis filmu (decription)

Start by clicking on the elements you want to scrap

IMDB

date

time

stars

star

title

genres

genre

director

description

Open advance editor

← Back

Next →

Rysunek 24 Zrzut ekranu przedstawia zdefiniowany skraper

Zdefiniowany skraper został wykonany na stronie IMDB za pośrednictwem portalu dzięki wykorzystaniu widoku podstawowego przedstawionego na Rysunek 21. Wynikiem skrapowania jest plik JSON.

```
[
{
  "requestId": "1152",
  "result": {
    "date": "1 June 2018 (USA)\n\n",
    "time": "\n                                1h 35min\n",
    "stars": [
      {
        "star": "Logan Marshall-Green"
```

```

        {
            "star": "Richard Anastasios"
        },
        {
            "star": "Rosco Campbell"
        }
    ],
    "title": "Upgrade (2018)",
    "genres": [
        {
            "genre": "Action"
        },
        {
            "genre": "Comedy"
        },
        {
            "genre": "Horror"
        }
    ],
    "director": "Leigh Whannell",
    "description": "\n                Set in the near-future,
technology controls nearly all aspects of life. But when Grey, a self-
identified technophobe, has his world turned upside down, his only hope for
revenge is an experimental computer chip implant called Stem.\n
    },
    "timeTaken": 2597,
    "status": "0",
    "message": null,
    "url": "http://www.imdb.com/title/tt6499752"
},
{
    "requestId": "1152",
    "result": {
        "date": null,
        "time": "\n                1h 56min\n                ",
        "stars": [
            {
                "star": "Evan Peters"
            }
        ],

```

```

    {
      "star": "Ann Dowd"
    },
    {
      "star": "Barry Keoghan"
    }
  ],
  "title": "American Animals (2018)",
  "genres": [
    {
      "genre": "Crime"
    },
    {
      "genre": "Drama"
    }
  ],
  "director": "Bart Layton",
  "description": "\n          Four young men mistake their
lives for a movie and attempt one of the most audacious heists in U.S.
history.\n          "
},
  "timeTaken": 3125,
  "status": "0",
  "message": null,
  "url": "http://www.imdb.com/title/tt6212478"
}
]

```

Przykład 14 Wynik wywołania skrapera filmów IMDB

Uzyskany wynik, zaprezentowany w Przykład 14, zawiera wszystkie pożądane szczegóły filmu. Wynik skrapowania nie jest jednak pozbawiony wad. Rezultat zawiera wiele znaków białych, które czynią go nienadającym się do zastosowania w aplikacji. Zaprezentowana metoda wykorzystuje najbardziej podstawowe mechanizmy zaimplementowane w prototypie, stworzone z myślą o łatwości użycia. Przeznaczone są dla osób nieposiadających wiedzy technicznej, których prawdopodobną intencją nie będzie wykorzystanie wyniku w aplikacji. Użytkownik zaawansowany może pobrać dane za pomocą *API* i wykorzystać zaawansowane funkcjonalności takie jak transformacja wyniku.


```
[
  {
    "requestId": "1156",
    "result": {
      "date": null,
      "time": 116,
      "stars": [
        {
          "star": "Evan Peters"
        },
        {
          "star": "Ann Dowd"
        },
        {
          "star": "Barry Keoghan"
        }
      ],
      "title": "American Animals (2018)",
      "genres": [
        {
          "genre": "Crime"
        },
        {
          "genre": "Drama"
        }
      ],
      "director": "Bart Layton",
      "description": "Four young men mistake their lives for a movie
and attempt one of the most audacious heists in U.S. history."
    },
    "timeTaken": 3165,
    "status": "0",
    "message": null,
    "url": "http://www.imdb.com/title/tt6212478"
  },
  {
    "requestId": "1156",
    "result": {
      "date": "1 June 2018 (USA)",

```

```

        "time": 95,
        "stars": [
            {
                "star": "Logan Marshall-Green"
            },
            {
                "star": "Richard Anastasios"
            },
            {
                "star": "Rosco Campbell"
            }
        ],
        "title": "Upgrade (2018)",
        "genres": [
            {
                "genre": "Action"
            },
            {
                "genre": "Comedy"
            },
            {
                "genre": "Horror"
            }
        ],
        "director": "Leigh Whannell",
        "description": "Set in the near-future, technology controls
nearly all aspects of life. But when Grey, a self-identified technophobe,
has his world turned upside down, his only hope for revenge is an
experimental computer chip implant called Stem."
    },
    "timeTaken": 2561,
    "status": "0",
    "message": null,
    "url": "http://www.imdb.com/title/tt6499752"
}
]

```

Przykład 17 Przetworzony wynik skrapowania

7. Podsumowanie

Rozdział jest próbą podsumowania zaproponowanego rozwiązania, przedstawia jego wady i zalety. Zawiera również propozycje dalszego rozwoju oprogramowania.

7.1. Wady i zalety rozwiązania

Praca nad prototypem pozwoliła ujawnić niektóre jego wady. Zagadnienia wypisane poniżej są charakterystyczne dla wstępnej fazy projektu i mogą być usunięte w dalszym rozwoju.

Wady:

- Dane wyjściowe dostępne tylko w formacie *JSON*;
- Ograniczona odporność selektorów na zmiany na stronie; wynikająca z ograniczeń zaimplementowanego algorytmu wyliczania selektora;
- Nie zaimplementowano wielu funkcjonalności potrzebne do codziennego korzystania z portalu, takich jak: wyszukiwanie skrapera, sortowanie projektów i tym podobne;
- Prototyp wykonuje wszystkie żądania z jednego adresu IP co przy regularnym wykorzystaniu najprawdopodobniej skutkowałoby blokadą IP;
- Prototyp nie posiada możliwości automatycznego uruchomienie skrapera w określonych interwałach czasowych, która jest dostępna w konkurencyjnych produktach.

Zalety:

- Rozwiązanie jest łatwe w użyciu i każdy użytkownik może z niego skorzystać;
- Prototyp zawiera paletę funkcjonalności umożliwiającą efektywne stosowanie przez programistów;
- Aplikacja może być łatwo rozszerzana i rozbudowana dzięki udostępnionemu API;
- Przygotowany prototyp jest konkurencyjny dla istniejących usług i ma wysoki potencjał na dalszy jego rozwój;
- Rozwiązanie od początku celuje w budowę społeczności poprzez funkcjonalności umożliwiające udostępnianie skrapierów.

7.2. Kierunki rozwoju prototypu

Portal przygotowany na potrzeby prototypu nie jest kompletny i nie zawiera wszystkich funkcjonalności niezbędnych do wygodnego użytkowania. Rezultatem pracy jest plan udoskonalenia zaproponowanego rozwiązania oraz lista funkcjonalności, o które należy wzbogacić platformę:

- **Wersjonowanie skrapera** – Ważnym elementem zespołowej pracy nad skraperaми jest kontrola wersji. Portal powinien umożliwić wersjonowanie skrapera i możliwość powrotu do poprzednich wersji;
- **Rozbudowa funkcjonalności społecznościowych portalu**
 - możliwość dyskutowania, komentowania skraperek;
 - zaproszenia, możliwość polecania;
 - gamifikacja - promowanie aktywnych autorów skraperek;
 - system ocen skraperek.
- **Wyszukiwanie projektów i skraperek** – Dla rozwoju społeczności ważna jest możliwość wyszukiwania gotowych rozwiązań oraz inspiracji;
- **Rozwój wizualnego kreatora skraperek**
 - Automatyczne wykrywanie kolekcji;
 - Tworzenie bardziej odpornych na zmiany selektorów;
- **Wtyczka do przeglądarki** – Jednym z założeń był brak grubego klienta. Okazało się, że zdefiniowanie skrapera dla niektórych stron internetowych jest niemożliwe za pomocą zaproponowanego rozwiązania, ze względu na ich aktywną obronę przed byciem otwartym w *iframe*. Dzięki otwartemu API prototypu taka wtyczka może być rozwijana oddzielnie od platformy;
- **Rozwój API** – Udostępnienie API za pomocą SOAP;
- **Usługa generowania kodu** – Udostępnienie usługi generującej biblioteki (.dll, .jar) umożliwiające skrapowanie stron internetowych bez pośrednictwa platformy;
- **Usługa skrapowania w zadanych interwałach czasowych** - Umożliwić wykonywanie skrapera w konkretnych godzinach i wysłanie raportów do klienta;
- **Notyfikacjach o zmianach** – Rozwój skrapera by umożliwić notyfikacje o zmianach na stronach, na przykład w celu monitorowania cen.
- **Unieważnić skrapera na blokowanie IP** – Przy skrapowaniu za pośrednictwem platformy należy się spodziewać, że strony będą blokować IP, z którego pochodzi duża ilość żądań.

Blokady IP uniemożliwią działanie skreperowi. W celu uniewrażliwienia skrapera należy zastosować rozwiązanie *proxy* takie jak np. zaprezentowana w rozdziale 2.4 platforma *crawlera*;

- **Monetyzacja platformy** – Platforma powinna wprowadzić opcję premium dla użytkowników na przykład poprzez ograniczenie ilości członków projektów prywatnych lub inne ograniczenia;
- **Zwiększenie ilości formatów wyjściowych.**

Rozwiązanie, już w fazie prototypu, posiada duży potencjał do dalszego rozwoju. Prowadząc pracę w zaproponowanym kierunku można osiągnąć pełno wartościowy produkt, który może stać się liderem w swojej domenie.

7.3. Wnioski końcowe

Poprzedzające rozdziały wprowadziły czytelnika w problem skrapowania stron WWW oraz przeprowadziły dyskusję dotyczącą możliwych rozwiązań problemu. Po analizie produktów dostępnych na rynku zaprojektowano i dokonano implementacji alternatywnego podejścia. Przedstawiony pomysł elastycznego skrapera WWW wyróżnia grupa docelowa; narzędzie może być używane przez użytkownika zaawansowanego jak i nie posiadającego wiedzy technicznej. Prototyp cechuje łatwość wytwarzania skrapersów przy użyciu myszki. Jednocześnie łatwość rozszerzania oraz otwartość umożliwiają zaawansowanym użytkownikom wykorzystanie pełnego potencjału narzędzia.

Narzędzie, nawet w wersji prototypu, może konkurować z komercyjnymi produktami o podobnej tematyce.

Uważam, że zaproponowane rozwiązanie, po rozszerzeniu go o wspomniane zmiany wypracowane podczas tworzenia prototypu w pełni odpowiada na problem postawiony w temacie pracy i dostarcza elastyczny skroper stron WWW dostępny dla szerokiej grupy odbiorców.

Bibliografia

- [1] *Oxford Dictionary of English*, Oxford University Press. 2011. ISBN 10: 0199571120
- [2] Ustawa z dnia 27 lipca 2001 r. o ochronie baz danych. Dz.U. 2001 nr 128 poz. 1402
- [3] ROZPORZĄDZENIE PARLAMENTU EUROPEJSKIEGO I RADY (UE) 2016/679 z dnia 27 kwietnia 2016 r. w sprawie ochrony osób fizycznych w związku z przetwarzaniem danych osobowych i w sprawie swobodnego przepływu takich danych oraz uchylenia dyrektywy 95/46/WE (ogólne rozporządzenie o ochronie danych)
- [4] *QVC, INC. v. RESULTLY, Civil Action No. 14-6714.* - <http://www.paed.uscourts.gov/documents/opinions/16d0129p.pdf> - Dostęp 2018-05-31
- [5] *Web Scraping as a Service* - <https://www.grepsr.com/services/>. Dostęp 2018-04-28
- [6] *Convert websites into useful data* - <https://www.scrapehero.com> Dostęp 2018-04-28
- [7] *Advanced Web Crawling, Scraping & Data Extraction Services* - <http://www.datahen.com/> Dostęp 2018-04-28
- [8] *Turn websites into data* - <https://scrapinghub.com/> Dostęp 2018-04-28
- [9] *Easy Web Scraping for Anyone* - <https://www.octoparse.com/> Dostęp 2018-04-28
- [10] *Advanced Web Scraping & Automation tools* - <https://dexi.io/> Dostęp 2018-04-28
- [11] Marijn Haverbeke. *Eloquent JavaScript, 2nd Edition*, No Starch Press. 2015. ISBN-10: 1593275846
- [12] Asim Hussain, *Angular From Theory To Practice, Version 1.2.0, 2017-11-24*, Daolrevo Ltd
- [13] Simon Holmes, *Getting MEAN with Mongo, Express, Angular, and Node 1st Edition*, Manning Publications, 2015. ISBN-10: 1617292036
- [14] R. Fielding, J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, Internet Engineering Task Force ISSN: 2070-1721
- [15] *What Is REST?* - <http://www.restapitutorial.com/> - Dostęp 2018-05-03
- [16] *Cross-Origin Resource Sharing (CORS)* - <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> - Dostęp 2018-05-03
- [17] *Hypertext Transfer Protocol (HTTP) Status Code Registry* - <https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml> - Dostęp 2018-04-15

Spis przykładów

Przykład 1. Przykładowy dokument HTML na potrzeby opisu struktury skrapera.....	19
Przykład 2. Przykład struktury skrapera	20
Przykład 3. Rezultat skrapowania dla skrapera podanego w Przykład 2 na stronie z Przykład 1.....	21
Przykład 4. Przykładowa treść żądania tworząca użytkownika.	22
Przykład 5. Przykład żądania zmieniającego PATCH /user/:id/.....	24
Przykład 6. Przykładowa treść wiadomości do usługi POST /user/:userId:/project.	27
Przykład 7. Przykładowa treść wiadomości do usługi POST /user/:userId:/project/:projectId/scrapper/.	33
Przykład 8. Przykładowe żądanie do wykonania skrapera.	37
Przykład 9. Przykładowa linia logów aplikacji.	50
Przykład 10 Definicja usług <i>get</i> i <i>delete</i> w Project API.	51
Przykład 11 Fragment kodu skrapującego stronę WWW	52
Przykład 12 Fragment kodu przedstawiającego wykonywanie skrapownia	54
Przykład 13. Przykład wywołania skrapera za pomocą cURL.	69
Przykład 14 Wynik wywołania skrapera filmów IMDB.....	72
Przykład 15 Wywołanie skrapera za pomocą cURL z zaawansowanym filtrowaniem	73
Przykład 16 Funkcja użyta do transformacji wyniku	73
Przykład 17 Przetworzony wynik skrapowania	75

Spis tabel

Tabela 1. Opis dostępnych odpowiedzi usługi POST /user/	22
Tabela 2. Opis dostępnych odpowiedzi usługi GET /user/:id.	23
Tabela 3. Opis dostępnych odpowiedzi usługi DELETE /user/:id.....	24
Tabela 4. Opis dostępnych odpowiedzi usługi PUT /user/:id.	24
Tabela 5. Opis dostępnych odpowiedzi usługi PATCH /user/:id.....	25
Tabela 6. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/.....	26
Tabela 7. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/public.....	27
Tabela 8. Opis dostępnych odpowiedzi usługi POST /user/:userId/project.....	27
Tabela 9. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId.	28
Tabela 10. Opis dostępnych odpowiedzi usługi DELETE /user/:userId/project/:projectId... ..	29
Tabela 11. Opis dostępnych odpowiedzi usługi PUT /user/:userId/project/:projectId.	29
Tabela 12. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/users..	30
Tabela 13. Opis dostępnych odpowiedzi usługi PUT /user/:userId/project/:projectId/users/:Username.	30
Tabela 14. Opis dostępnych odpowiedzi usługi DELETE /user/:userId/project/:projectId/users/:removeUserId.	31
Tabela 15. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/.....	32
Tabela 16. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/.	33
Tabela 17. Opis dostępnych odpowiedzi usługi POST /user/:userId/project.....	33
Tabela 18. Opis dostępnych odpowiedzi usługi DELETE /user/:userId/project/:projectId/scrapper/:scrapperId.	34
Tabela 19. Opis dostępnych odpowiedzi usługi PUT /user/:userId/project/:projectId.	34
Tabela 20. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/:scrapperId/result.	35
Tabela 21. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/:scrapperId/result.	36
Tabela 22. Opis dostępnych odpowiedzi usługi GET /user/:userId/project/:projectId/scrapper/:scrapperId/result.	38

Spis rysunków

Rysunek 1. Diagram opisujący podstawową architekturę zaproponowanego rozwiązania.....	14
Rysunek 2 Hierarchia użytkowników portalu.....	15
Rysunek 3 Diagram Use Case przedstawiający możliwości właściciela projektu.....	16
Rysunek 4 Diagram Use Case przedstawiający możliwości użytkownika będącego członkiem zespołu.....	17
Rysunek 5 Diagram Use Case przedstawiający możliwości zalogowanego użytkownika.....	18
Rysunek 6. Diagram przedstawiający strukturę bazy danych.....	48
Rysunek 7. Ekran landing page aplikacji skrapera.....	55
Rysunek 8. Ekran logowania.....	56
Rysunek 9. Widok projektów zalogowanego użytkownika.....	56
Rysunek 10. Widok szczegółów projektu.....	57
Rysunek 11. Widok zarządzania użytkownikami w projekcie.....	58
Rysunek 12. Widok edycji projektu.....	59
Rysunek 13 Widok tworzenia skrapera - wprowadzanie szczegółów skrapera.....	60
Rysunek 14. Widok definiowania skrapera.....	61
Rysunek 15. <i>Modal</i> potwierdzenia wyboru elementu.....	62
Rysunek 16. Ekran podczas definiowania kolekcji.....	63
Rysunek 17. Dodawanie nowych elementów do zdefiniowanej kolekcji.....	63
Rysunek 18. Reprezentacja kolekcji.....	64
Rysunek 19. Rysunek przedstawiający zaawansowany edytor selektorów.....	64
Rysunek 20. Widok testowania nowego skrapera.....	65
Rysunek 21. Widok szczegółów utworzonego skrapera.....	66
Rysunek 22. Widok szczegółów skrapera przeznaczonych dla użytkowników posiadających wiedzę techniczną.....	67
Rysunek 23. Widok historii skrapera.....	68
Rysunek 24 Zrzut ekranu przedstawia zdefiniowany skrapera.....	70