POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Bartosz Pawlak
Nr albumu 19006

Pozyskiwanie danych w ramach białego wywiadu z użyciem technik informatycznych

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

miejsce, miesiąc, rok obrony

Streszczenie

Praca dotyczy istotnego problemu związanego z pozyskiwaniem danych z zasobów internetowych w ramach białego wywiadu. Dotychczasowe systemy realizujące takie zadania nie umożliwiają w swoich ramach tworzenia skomplikowanych scenariuszy. Istniejące rozwiązania web scrapingu wolno adaptują się do szybko zmieniających się wyzwań stawianych przez rozwój technologii. Stan ten powoduje konieczność stworzenia elastycznego systemu, którego elementy pozwolą na dynamiczną adaptację do zmiennego środowiska wymiany informacji przez sieć.

W związku z tym w ramach pracy wykonano prototyp systemu umożliwiającego pozyskiwanie danych ze stron internetowych. Umożliwia on tworzenie zaawansowanych scenariuszy dotyczących pozyskiwania danych za pomocą webowego interfejsu kreatora scenariuszy. Klient prototypu został zaimplementowany z wykorzystaniem technologii Angular oraz TypeScript. Stworzone scenariusze wykonują się po stronie API, które wykorzystuje bibliotekę zaprojektowaną w ramach prototypu. API oraz biblioteka zostały stworzone z wykorzystaniem technologii .NET Core oraz C#. Zadaniem prototypu jest wykonanie metod zapisanych w scenariuszu i zwrócenie ich wyniku w postaci dokumentu JSON. Zastosowane mechanizmy i abstrakcje stanowią o otwartości aplikacji na zmiany. Działanie systemu zostało przetestowane za pomocą poprawnego wykonania scenariuszy prototypowych. Aplikacja umożliwia komunikację z zasobami na poziomie żądań TCP/IP i poprzez skonteneryzowaną przeglądarkę internetową.

Słowa kluczowe: web scraping, biały wywiad, pozyskiwanie danych, OSINT

Spis treści

1. Wstęp

Biały wywiad (ang. open-source intelligence; OSINT) stał się w XXI wieku popularnym terminem. Polega on na gromadzeniu i analizie danych pochodzących z ogólnodostępnych źródeł. Spośród innych dyscyplin wywiadowczych charakteryzuje go to, iż wywiadowcy posługują się jedynie etycznymi sposobami pozyskiwania informacji ze źródeł jawnych. Źródłami tymi mogą być np. wszystkie środki masowego przekazu. Wraz z rozwojem technologii informatycznych rozpoczęło się generowanie i udostępnianie coraz większej ilości informacji. Równocześnie rozwijały się sposoby i metody ich analizy. Dane stały się walutą, którą dzisiaj możemy „zapłacić" za dostęp do serwisów lub usług. Dzięki powszechnemu dostępowi do wyszukiwarek, mediów społecznościowych, rejestrów i zbiorów danych, każdy może stać się wywiadowcom. Techniki OSINT-owe wykorzystywane są m.in. przez służby, banki, partie polityczne, przedsiębiorców lub pracodawców. Specjaliści z dziedziny cyberbezpieczeństwa również sięgają po metody wywiadowcze podczas przeprowadzania testów penetracyjnych.

W tym rozdziale przedstawiony został ogólny cel pracy jak i przyjęte rozwiązania oraz oczekiwany rezultat. Dodatkowo, omówione zostały kolejne rozdziały niniejszej pracy.

1.1. Cel pracy

Celem pracy jest rozwiązanie problemu tworzenia dedykowanych scraperów dla stron internetowych. Wynikiem rozwiązania tego problemu będzie stworzenie generycznej aplikacji webowej umożliwiającej pozyskiwanie danych ze stron internetowych i wykonywanie na nich zdefiniowanych czynności. Użytkownik aplikacji powinien mieć możliwość tworzenia scenariuszy za pomocą wizualnego kreatora. Następnie scenariusze te będą wykonywane po stronie serwera, a wynik ich działania zostanie zwrócony do klienta.

Tworzony prototyp jest odpowiedzią na zmieniające się potrzeby dziedziny web scrapingu i białego wywiadu. Celem pośrednim jest zbadanie możliwości stworzenia generycznych metod, które mogą zostać użyte w wielu różnych scenariuszach OSINT-owych. Dodatkowo projekt powinien być implementowany z myślą o możliwości przystosowania go do architektury rozproszonej.

Powstające narzędzie stanowić będzie wycinek zaawansowanego systemu służącego do białego wywiadu. W ramach pracy rozwinięty zostanie moduł do zbierania danych.

Ze względu na elastyczność i szybkość adaptacji do zmian na stronach internetowych praca z wykorzystaniem opracowanego narzędzia wymagać będzie ponadpodstawowej wiedzy o działaniu stron internetowych i językach programowania. Narzędzie skierowane jest do osób, które chciałyby przyśpieszyć i ułatwić procesy związane z tworzeniem dedykowanych web scraperów i parserów.

1.2. Rezultat pracy

Rezultatem pracy jest rozwiązanie problemu związanego z tworzeniem dedykowanych scraperów dla stron internetowych. W toku prac nad przedstawionym problemem opracowano aplikacje webowa, która umożliwia pozyskiwanie danych ze stron internetowych i wykonywanie na nich zdefiniowanych czynności. Użytkownik może stworzyć własny scenariusz OSINT-owy za pomocą wizualnego kreatora. Kreator składa się z trzech elementów: procesów, grup i metod. Wszystkie elementy scenariusza są wielokrotnego użytku.

W procesie wytwarzania prototypu aplikacji stworzono webowego klienta, serwer API oraz własną bibliotekę, która odpowiada za większość operacji. Dodatkowo biblioteka została zaimplementowana w taki sposób, iż umożliwia w przystępny sposób na dodawanie kolejnych elementów składowych scenariuszy i rozwój aplikacji. Ponadto jej budowa umożliwia adaptacje do architektury rozproszonej.

1.3. Organizacja pracy

W drugim rozdziale pracy opisane i podsumowane zostały istniejące narzędzia wspomagające biały wywiad oraz web scraping.

Trzeci rozdział stanowi omówienie koncepcji stworzenia prototypu. Poruszone są w nim także kwestie związane z motywacją stworzenia nowego systemu i wymaganiami funkcjonalnymi jakie musi spełniać.

Czwarty rozdział prezentuje wykorzystane narzędzia i technologie użyte w procesie implementacji prototypu.

W piątym rozdziale opisana została implementacja wyróżnionych elementów systemu. Dodatkowo poruszono kwestie działania aplikacji i przepływu informacji.

Szósty rozdział opisuje zastosowanie prototypu. W tym celu omówione zostało tworzenie nowych scenariuszy. Ponadto zaprezentowane zostały scenariusze prototypowe.

Ostatni rozdział stanowi podsumowanie całej pracy. Zawarto w nim wady i zalety rozwiązania jak i przyszłe możliwe kierunku rozwoju prototypu.

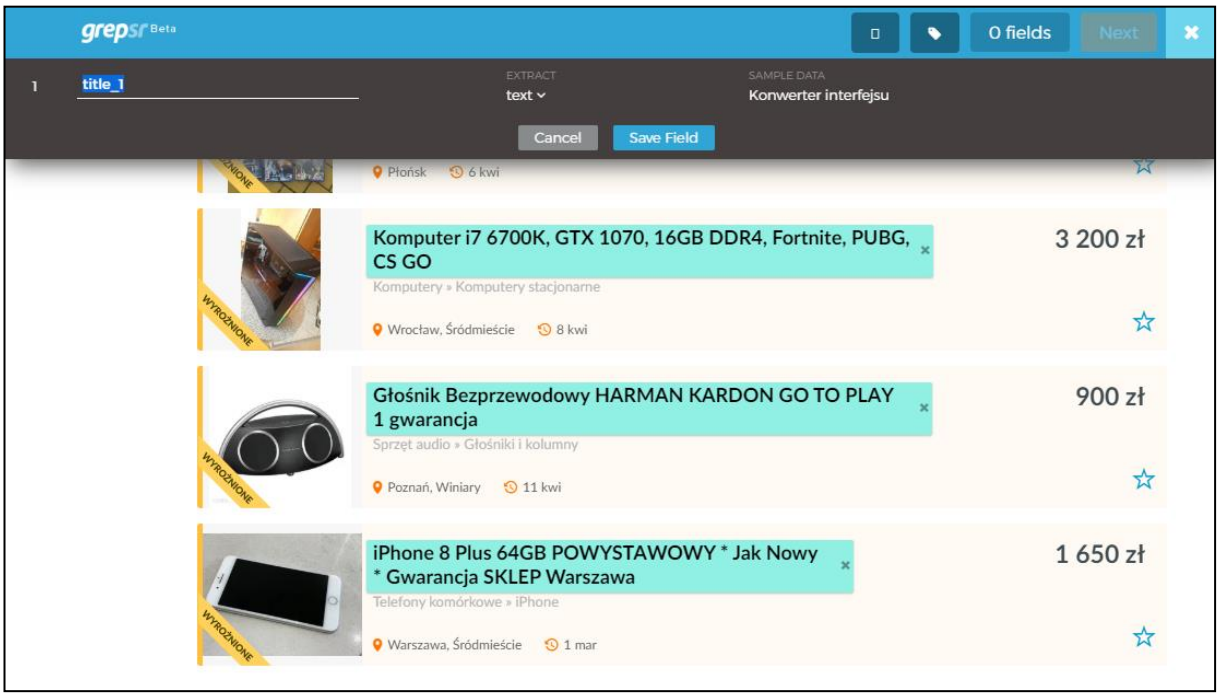
2. Istniejące rozwiązania

Narzędzia służące do pozyskiwania danych ze stron internetowych możemy podzielić na wiele rodzajów - od skryptów i wtyczek po zaawansowane systemy z modułami analitycznymi wykorzystującymi sztuczną inteligencję. Większość z nich bazuje na ekstrakcji informacji z dokumentów HTML na podstawie selektorów CSS lub XPath. Jest to jeden ze sposób w jaki można tworzyć zbiory danych. Innym sposobem pozyskiwania informacji są interfejsy API udostępnianie przez konkretne strony internetowe. Niestety dostęp do nich często jest limitowy i kontrolowany. Aplikacje OSINT-owe zasilane są przez źródła danych, a same w sobie stanowią w głównej mierze wyszukiwarki z możliwością tworzenia analiz i powiązań.

W poniższych podrozdziałach przedstawię niektóre dostępne narzędzia wspomagające biały wywiad i web scraping.

2.1. grepsr.com

Grepsr for Chrome [1] jest to komercyjna wtyczka do przeglądarki Google Chrome, która umożliwia scrapowanie stron internetowych. Przy pierwszym uruchomieniu pojawia się film instruktażowy, który krok po kroku omawia sposób użycia narzędzia. Działanie wtyczki polega na tym, że użytkownik zaznacza na stronie internetowej treści, które chce pobrać. Następnie nadaje im identyfikator lub nazwę. Dla wybranego elementu strony mamy do wyboru ekstrakcję jako tekst, wartość atrybutu lub HTML. Zdecydowanie brakuje możliwości określenia i rzutowania typu wartości - domyślnie wszystko jest łańcuchem znaków. Narzędzie posiada zaimplementowane mechanizmy związane z logowaniem użytkownika na stronie, paginacją oraz przechodzeniem w głąb stron. Po zakończonym etapie parsowania użytkownik może pobrać zescrapowane dane np. w formacie JSON lub CSV. Rysunek 1 przedstawia sposób definiowania scrapingu we wtyczce Grepsr.

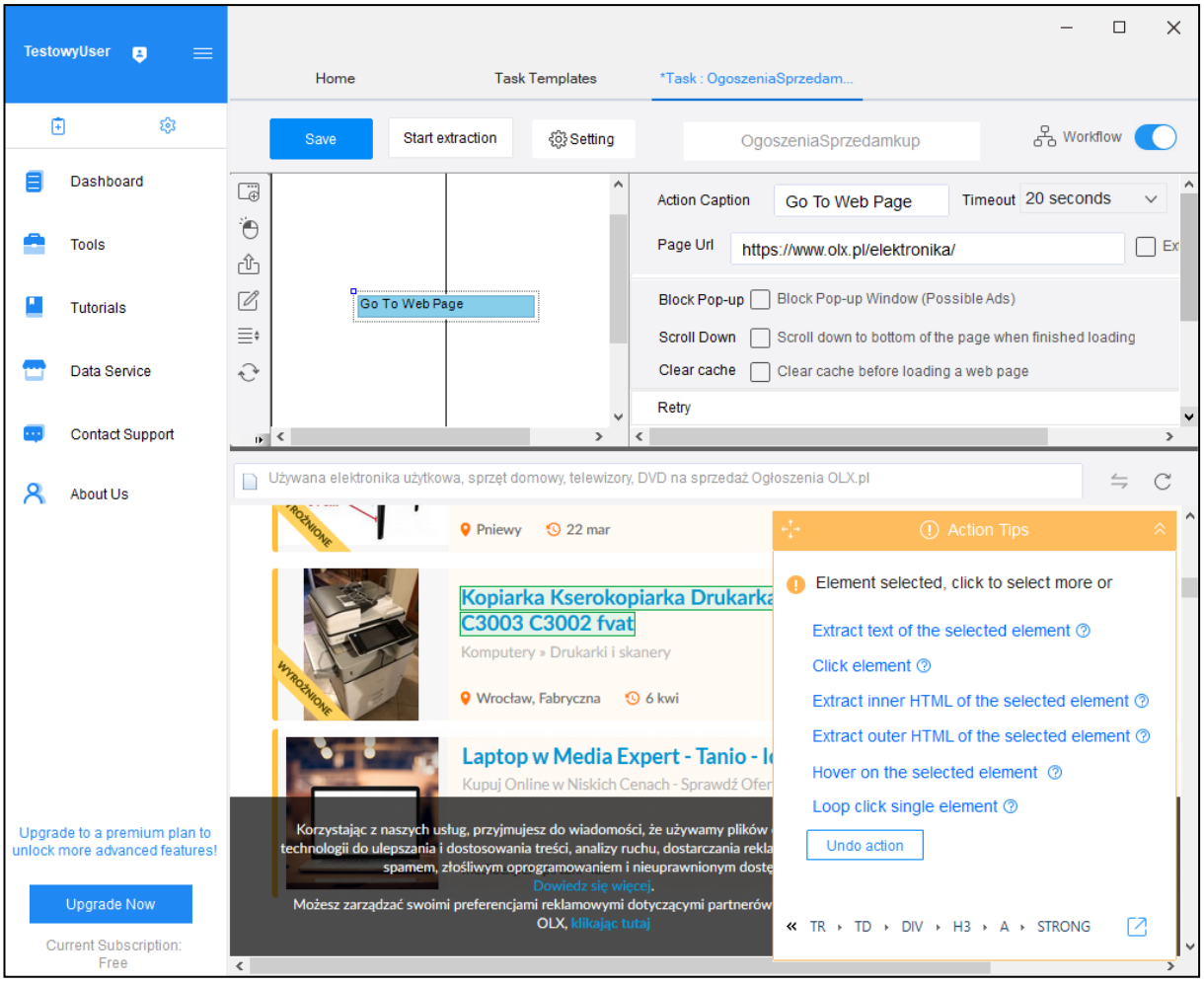


Rysunek 1. Sposób definiowania scrapingu we wtyczce Grepsr

Grepsr posiada również aplikacje webową, która umożliwia tworzenie projektów, podgląd pobranych danych, utworzenie harmonogramu - kiedy przeprowadzana ma być operacja scrapowania, oraz zdefiniowania, gdzie i czy mają być zapisywane pobrane dane (m.in. FTP, Google Drive). Darmowa wersja ograniczona jest wieloma limitami m.in. związanymi z ilością możliwych do pobrania rekordów. Wtyczka słabo radzi sobie z danymi, które są nieustrukturyzowane lub osadzone w innych kontenerach, jednakże jest bardzo łatwa w użyciu. Dodatkowo za pomocą platformy Grepsr klienci indywidualni mogą zwrócić się z prośbą o stworzenie dla nich scraperów na podstawie przedstawionych wymagań. Jedną z zalet omawianej wtyczki jest niewątpliwie możliwość uruchomienia API z pozyskanymi przez nią danymi.

2.2. octoparse.com

Octoparse [2] jest to komercyjna aplikacja desktopowa, która umożliwia scrapowanie danych. Sprzedawana jest w modelu subskrypcyjnym. Posiada zaimplementowane gotowe szablony do pozyskiwania informacji z wielu popularnych stron. Jednakże jej główną funkcją jest przede wszystkim tworzenie własnych zadań pozyskiwania danych. Narzędzie pozwala na wykonanie zadania scrapowania lokalnie lub w chmurze. Nie wymaga umiejętności programowania, ponieważ ekstrakcja danych polega na zaznaczeniu za pomocą myszki interesujących użytkownika elementów. Posiada rozbudowany interfejs aplikacji, jednakże nie jest on skomplikowany.



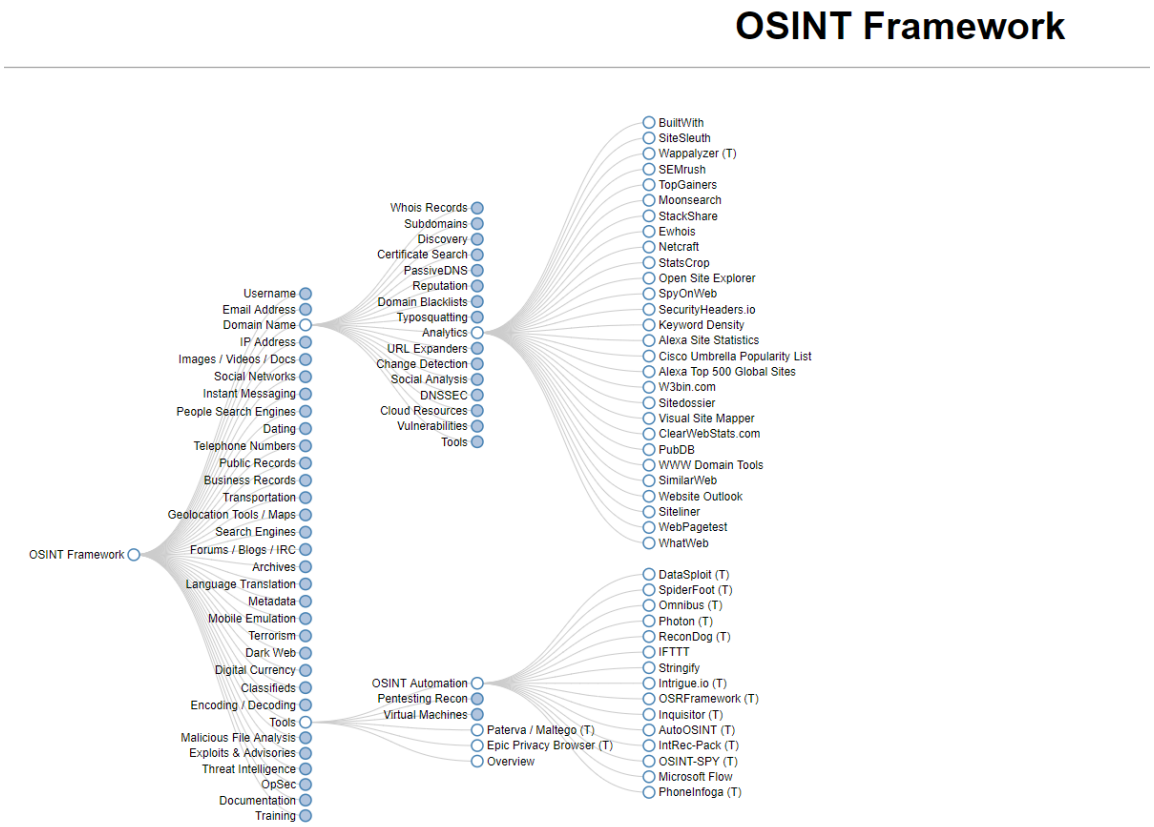
Rysunek 2. Sposób definiowania scrapingu w aplikacji Octoparse

Aplikacja ma wbudowaną przeglądarkę internetową, która jest jej nieodzownym elementem. Za jej pomocą konfigurowana jest nawigacja i wybierane są elementy do ekstrakcji. Narzędzie umożliwia blokowanie wyskakujących okien, ustawianie ciasteczek, nawigację po stronach z paginacją, logowanie, obsługę stron bazujących na AJAX i implementujących nieskończone przewijanie. Pobrane dane można wyeksportować do plików CSV lub Excel, zapisać do bazy danych lub uruchomić bazujące na nich API. Dodatkowo klienci z aktywną subskrypcją mają do dyspozycji wiele usług w chmurze oraz automatyczną rotację adresu IP, aby unikać blokad.

Octoparse jest to rozbudowana platforma i narzędzie, jednakże w głównej mierze nastawione na klientów subskrypcyjnych. Świadczyć o tym może możliwość złożenia zapytań biznesowych w celu pozyskiwania informacji z wybranych kategorii stron internetowych oraz wyróżnieni klienci na stronie głównej aplikacji.

2.3. osintframework.com

OSINT Framework [3] jest to zbiór źródeł i narzędzi wykorzystywanych w ramach białego wywiadu. Źródła prezentowane są w graficznej formie drzewa oraz pogrupowane są w tematyczne kategorie. Zbiór ten udostępniony jest w publicznym repozytorium na GitHubie i utrzymywany jest przez społeczność. OSINT Framework zawiera odnośniki do darmowych źródeł, bądź takich, które umożliwiają na częściowo darmową interakcję. Są to m.in. wyszukiwarki użytkowników, adresów email, domen czy adresów IP. Jak i również repozytoria ze skryptami lub narzędziami, które umożliwiają na działania wywiadowcze w danych dziedzinach. Przykładowo narzędzia do emulacji lub inżynierii wstecznej.

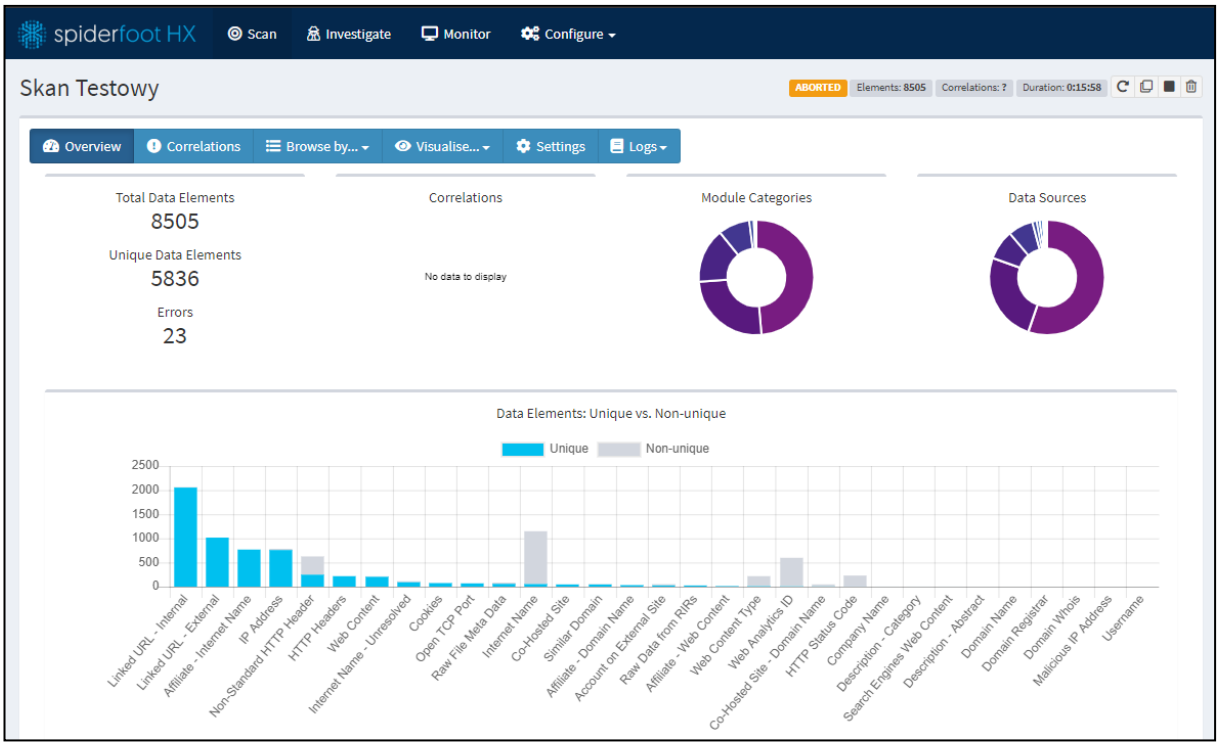


Rysunek 3. Interfejs aplikacji webowej OSINT Framework

OSINT Framework jest miejscem do którego odsyłany jest każdy, kto chce poznać techniki, sposoby i źródła pozyskiwania informacji stosowane w ramach białego wywiadu. Rysunek 3 przedstawia interfejs aplikacji webowej OSINT Framework.

2.4. spiderfoot.net

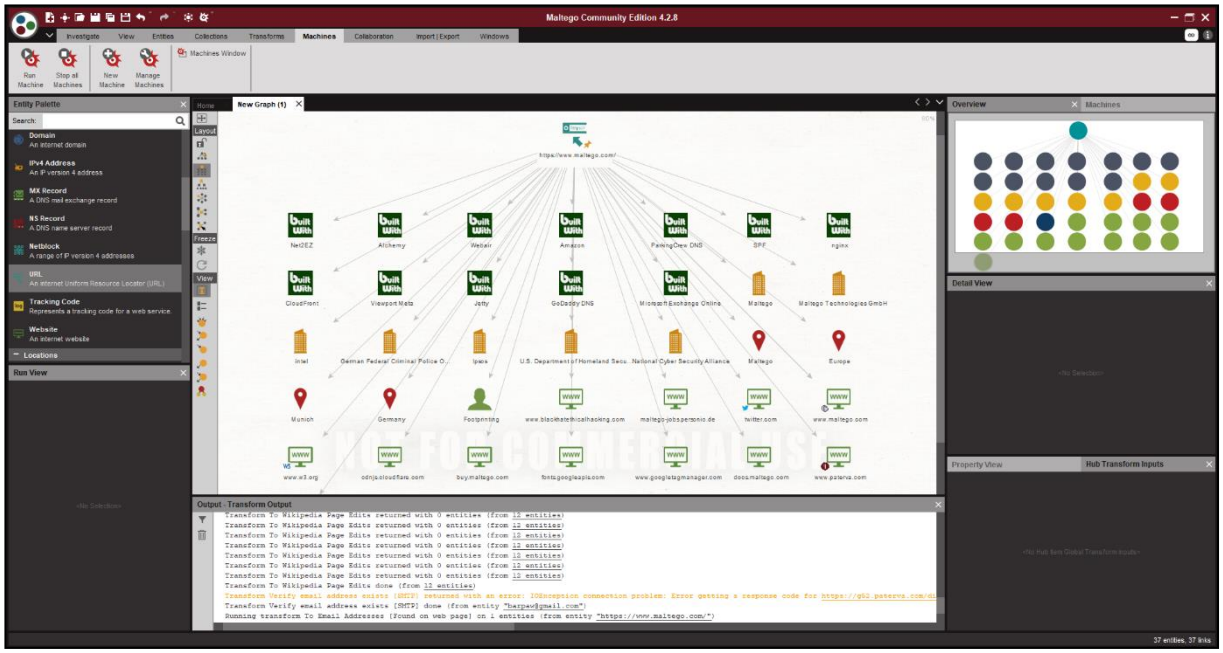
SpiderFoot [4] jest to aplikacja webowa służąca do rekonesansu, która ma zaimplementowane ponad sto otwartych źródeł danych. Bazowe encje po których można dokonać wyszukiwania to adresy email, numery telefonów, adresy IP, domeny internetowe i wiele więcej. Możliwe jest uruchomienie aplikacji lokalnie lub skorzystanie z wersji sieciowej działającej po stronie wytwórcy, która nosi nazwę SpiderFoot HX. Źródła danych zaimplementowane są w postaci modułów. Każdy moduł posiada dodatkowo swoje ustawienia, które można modyfikować. Wiele z zaimplementowanych modułów (źródła zewnętrzne) wymaga podania klucza API w związku z tym istnieje mechanizm ich importowania i eksportowania. Użytkownik ma możliwość personalizacji profilu skanów w którym dokonuje się aktywacji lub dezaktywacji modułów, które będą użyte podczas wyszukiwania. Po każdym skanie dostępny jest raport, który prezentuje w pogrupowanej formie tekstowej jak i wizualnej znalezione powiązane encje z parametrem naszego zapytania. Aplikacja SpiderFoot HX bazuje na modelu subskrypcyjnym, darmowe konto umożliwia przeprowadzenie tylko trzech skanów miesięcznie z jednym parametrem. Rysunek 4 przedstawia przykładowy raport po wykonaniu skanu w aplikacji SpiderFoot HX.



Rysunek 4. Raport po wykonaniu skanu w aplikacji SpiderFoot HX

2.5. maltego.com

Maltego [5] jest to narzędzie używane do białego wywiadu stworzone przez firmę Paterva. Umożliwia ono wykonywanie operacji data miningowych w formie desktopowej aplikacji analitycznej wykorzystującej grafy do przedstawiania relacji pomiędzy encjami. Na encjach (wierzchołkach grafu) można wykonywać transformacje, które rozszerzają graf o kolejne obiekty i powiązania. Do dyspozycji użytkownika dostępne są zaimplementowane gotowe transformacje bazujące na wielu popularnych źródłach. Istnieje również możliwość pisania własnych transformacji. Maltego dystrybuowany jest w czterech różnych wersjach (Maltego CE, Maltego Classic, Maltego XL, Maltego CaseFile). Wersje różnią się między sobą licencjami oraz skalą funkcjonalności aplikacji. Przykładowo pojedyncza transformacja w darmowej niekomercyjnej wersji systemu zwrócić może jedynie 12 encji, a w wersji komercyjnej 64 tysiące encji. Grafy tworzone w komercyjnej wersji Maltego mogą mieć maksymalnie milion encji. Aplikacja umożliwia min. scalanie danych, export analiz do wielu formatów oraz stosowanie różnych układów graficznych grafu. Rysunek 5 przedstawia interfejs programu Maltego.



Rysunek 5. Interfejs programu Maltego

2.6. Podsumowanie istniejących rozwiązań

Istniejące rozwiązania do pozyskiwania danych ze stron internetowych oferują podstawowe funkcjonalności w tym zakresie. Ich zaletą jest to, że są łatwe w obsłudze, jednakże nie zawsze potrafią dokonać ekstrakcji danych ze skomplikowanych strukturalnie dokumentów HTML. Dodatkowo są nastawione na prostotę działania dlatego nie oferują obsługi kompleksowych scenariuszy. Problemem przy istniejących rozwiązaniach jest również fakt, iż niektóre z nich skupiają się jedynie na ekstrakcji informacji i nie umożliwiają dopasowania lub rzutowania typu.

Mechanizmy oznaczania danych do pobrania w rozwiązaniach rynkowych są łatwe w obsłudze dla użytkownika, jednakże operują na przeglądarce po stronie klienta co jest mało efektywne. Oczywiście niweluje to problemy związane z pozyskiwaniem danych doczytywanych asynchronicznie ale znacząco obniża wydajność. Najbardziej optymalnym rozwiązaniem byłoby stosowanie silników przeglądarek tylko w przypadku, gdy nie ma innego wyjścia i bazowanie na komunikacji opartej o same zapytania HTTP.

Oprócz narzędzi do scrapingu, istnieją aplikacje dedykowane białemu wywiadowi. Implementują one mechanizmy analityczne lub takie, które sprzyjają wyszukiwaniu informacji. Jednakże najczęściej operują one na źródłach zewnętrznych (API) do których dostęp jest kontrolowany, limitowany lub płatny. Brakuje aplikacji, która byłaby pośrednikiem między wykonywaniem skomplikowanych operacji związanych z wydobyciem danych oraz niosła ze sobą możliwość ich wyszukiwania i analizy. Dodatkowo takie narzędzie musiałoby charakteryzować się prostotą w dodawaniu nowych źródeł.

Istniejące rozwiązania nie są na tyle elastyczne aby móc wykorzystywać je do zaawansowanych scenariuszy OSINT-owych. Brakuje generycznego narzędzia, które pozwoliło by na tworzenie kompleksowych przepisów na pozyskiwanie danych ze źródeł jawnych. Inną wadą istniejących technologii jest to, iż większość z nich jest płatna. Należy również pamiętać o kwestiach bezpieczeństwa. Niektóre narzędzia informują wprost, że przechowują ciasteczka sesyjne, aby skonfigurowany przepis na scrapowanie danej strony działał poprawnie. Na uwadze trzeba mieć też fakt, że narzędzia rynkowe mogą przechowywać informacje o tym co leży w kręgu zainteresowań konkretnego użytkownika lub podmiotu, który licencję zakupił.

3. Koncepcja rozwiązania

W tym rozdziale zostanie przedstawiona propozycja generycznej aplikacji webowej, która umożliwia pozyskiwanie danych ze stron internetowych oraz wykonywanie na nich zdefiniowanych czynności. Dodatkowo poruszone zostaną kwestie związane z motywacją nowego rozwiązania i przedstawienie ogólnych wymagań funkcjonalnych aplikacji.

3.1. Propozycja nowego rozwiązania

Proponowane rozwiązanie stanowi moduł zbierania danych dla koncepcyjnego większego systemu służącego do wykonywania zadań związanych z białym wywiadem. Moduł ten składa się z kreatora scenariuszy, który umożliwia tworzenie generycznego zestawu instrukcji. Instrukcjami tymi są m.in. metody, które przyjmują parametry i zwracają wartość. Zaimplementowane metody można wykorzystywać wielokrotnie w różnych scenariuszach. Uruchomienie danego scenariusza generuje dokument JSON, w którym zawarty jest cały przebieg wykonywanych instrukcji.

Rozwiązanie składa się z trzech elementów:

- Webowego klienta;
- Własnej biblioteki;
- Webowego API.

Najważniejsze operacje systemu zostały zaimplementowane w bibliotece. To w niej zawarte są mechanizmy kontroli i wykonywania procesów zdefiniowanych w scenariuszach. Została ona zaprojektowana w taki sposób, aby łatwo można było dodawać nowe elementy składowe scenariuszy. Biblioteka umożliwia również otrzymanie informacji oraz metadanych na temat zaimplementowanych metod. Klient aplikacji nie ma szczegółowej wiedzy na temat składowych elementów scenariuszy. Korzysta on z informacji, które udostępnia biblioteka. Webowe API jest interfejsem komunikacyjnym, który na podstawie zadanych żądań wywołuje konkretne operacje biblioteki jak i zdefiniowane operacje bazodanowe.

3.2. Motywacja nowego rozwiązania

Motywacją nowego rozwiązania jest przede wszystkim brak kompleksowego narzędzia, które umożliwiałoby wykonywanie złożonych scenariuszy OSINT-owych. Większość z istniejących tego typu narzędzi ma pewne ograniczenia związane ze sposobem i szybkością działania. Przeważnie programiści tworzą dedykowane scrapery dla każdego kompleksowego rozwiązania. Wiąże się to później z problemami dotyczącymi utrzymania jakości kodu oraz posiadaniem kilku praktycznie takich samych projektów. Proponowane rozwiązanie umożliwia tworzenie dedykowanych scenariuszy pozyskiwania danych, bez tworzenia podobnej lub powtarzalnej warstwy kodu po stronie programistycznej.

Istniejące aplikacje mają ograniczone funkcjonalności i nie każda z nich pozwala na implementacje własnych pomysłów. Przeciwieństwem tego jest tworzone narzędzie, które wykracza poza ramy zwykłego scrapera i parsersa stron internetowych. Tworzony system umożliwia również wykonywanie czynności na stronach internetowych. Dodatkowo dzięki swojej generyczności pozwala zaimplementować metody, które w ramach scenariusza wykonywałyby komendy w terminalu linuxa lub uruchomiały skrypty.

3.3. Wymagania funkcjonalne

Poniżej przedstawiono ogólne wymagania funkcjonalne, które opisują funkcje możliwe do wykonania w aplikacji.

1. Użytkownik ma możliwość dodawania i wykonywania nowych scenariuszy.
2. Użytkownik ma możliwość edycji i usuwania scenariuszy.
3. Użytkownik ma możliwość wyświetlenia stu ostatnich zapisanych scenariuszy.
4. Użytkownik ma możliwość wyświetlenia stu ostatnich wykonanych scenariuszy.
5. Użytkownik ma możliwość wyświetlenia stu ostatnich zwróconych dokumentów.
6. Podgląd scenariusza umożliwia sprawdzenie jego struktury w formacie JSON, uogólnionego drzewa oraz encji i zmiennych.
7. Podgląd dokumentu umożliwia sprawdzenie jego struktury w formacie JSON.
8. Kreator scenariusza wykorzystuje mechanizm drag and drop.
9. Użytkownik ma możliwość sprawdzenia w oknie modalnym pomocy, informacje o konkretnym elemencie składowym scenariusza.
10. Widok dodawania metod pozwala na przeszukiwanie ich na podstawie nazwy.
11. Użytkownik ma możliwość zdefiniowania nazwy grupy w oknie modalnym dodawania grup.
12. Użytkownik ma możliwość wyboru pomiędzy różnymi typami procesów w oknie modalnym dodawania procesów.
13. Procesy w kreatorze scenariusza powinny wyróżniać się poprzez przypisane im unikalne kolory.
14. Użytkownik ma możliwość dodania podstawowych informacji dotyczących tworzonego scenariusza.
15. Kreator scenariusza powinien dostarczać funkcjonalności tworzenia scenariuszy operujących na żądaniach HTTP.
16. Kreator scenariusza powinien dostarczać funkcjonalności tworzenia scenariuszy operujących na przeglądarce internetowej.
17. Kreator scenariusza powinien udostępniać metody umożliwiające parsowanie dokumentów HTML.
18. Kreator scenariusza powinien udostępniać metody umożliwiające zapis informacji do bazy danych.
19. Kreator scenariusza powinien udostępniać metody umożliwiające przypisywanie adresów URL oraz łańcuchów znaków do scenariusza.
20. Kreator scenariusza powinien udostępniać statystyki związane z ilością wystąpień danych składowych przepisu.

4. Technologie i narzędzia użyte w pracy

W tym rozdziale przedstawione zostaną technologie i narzędzia użyte do przygotowania prototypu aplikacji.

4.1. .NET Core

.NET Core jest to otwarta platforma programowania ogólnego przeznaczenia. Umożliwia ona tworzenie aplikacji w technologii .NET Core dla systemów operacyjnych Windows, macOS i Linux. Posiada wsparcie dla procesorów x64, x86, ARM32 i ARM64. Pierwsza wersja platformy opublikowana została w 2016 roku przez Microsoft. Pozwala ona na pisanie aplikacji w językach C#, F# oraz Visual Basic. .NET Core jest otwartą i multiplatformową implementacją .Net Framework. Udostępnia interfejsy dla tworzenia aplikacji działających w chmurze, internetu rzeczy, aplikacji desktopowych WPF oraz uczenia maszynowego [6].

Właściwości platformy .NET Core:

- Wsparcie dla wielu systemów operacyjnych (Windows, macOS i Linux).
- Kod źródłowy platformy jest dostępny publicznie. Każdy może kontrybuować własne zmiany do projektu. Projektem zarządza .NET Foundation, czyli niezależna innowacyjna organizacja.
- Implementuje nowoczesne paradygmaty programowania takie jakie programowanie asynchroniczne lub zarządzanie zasobami dla kontenerów.
- Dzięki wbudowanym funkcjom związanym z zarządzaniem zasobami pozwala na wysoką wydajność.
- Platforma działa spójnie na wspieranych systemach operacyjnych. Umożliwia uruchamianie kodu na różnych systemach operacyjnych (kod zachowuje się tak samo na każdym systemie operacyjnym).
- Za pomocą udostępnionych narzędzi linii poleceń można w przyjemny sposób rozwijać aplikacje lokalnie lub wykorzystać je do budowy rozwiązań wykorzystywanych w procesie ciągłej integracji (*Continuous Integration*).
- Umożliwia elastyczne wdrażanie np. z wykorzystaniem kontenerów.

Możliwości wykorzystania platformy .NET Core:

- Aplikacje w chmurze (ASP.NET Core);
- Aplikacje IoT (System.Device.GPIO);
- Aplikacje mobilne (Xamarin);
- Aplikacje desktopowe Windows (WPF i Windows Forms);
- Uczenie maszynowe (ML.NET).

Platforma .NET Core [7] składa się z:

- Środowiska .NET Core;
- Środowiska ASP .NET Core;
- .NET Core SDK i kompilatora (Roslyn oraz F#);
- Polecenia dotnet.

4.2. TypeScript

TypeScript [8] jest to otwartoźródłowy oraz wieloplatformowy język programowania stworzony i rozwijany przez Microsoft. Stanowi on nadzbiór języka JavaScript. Umożliwia on statyczne typowanie oraz programowanie zorientowane obiektowo. TypeScript kompiluje się bezpośrednio do języka JavaScript. Dodatkowo każdy program napisany w technologii JavaScript jest również poprawnym programem TypeScript.

Język ten używany jest do tworzenia aplikacji klienckich jak również nie ma przeciwskazań, aby używać go tworząc oprogramowanie działające po stronie serwera. Transkompilacji kodu można dokonać na kilka sposobów. Za pomocą wbudowanego narzędzia lub np. za pomocą technologii babel. Kompilator TypeScript napisany jest w technologii TypeScript co oznacza, że został skompilowany do postaci JavaScript. Większość popularnych zintegrowanych środowisk programistycznych posiada wsparcie dla TypeScript.

Główną zaletą jaką wnosi TypeScript jest wczesne wykrywanie błędów w kodzie i możliwość definiowania typów. Znacząco podnosi to wydajność procesu wytwarzania oprogramowania i ułatwia analizę kodu wytworzonego przez inne osoby. Dodatkowo język ten posiada wsparcie dla funkcjonalności zdefiniowanych w standardzie ECMAScript 2015. Co oznacza, że można w nim m.in. definiować klasy i moduły oraz używać funkcji strzałkowych.

Niektóre funkcjonalności języka TypeScript:

- Adnotacje typów;
- Inferencja typów;
- Interfejsy;
- Typ wyliczeniowy Enum;
- Typy generyczne;
- Tuple;
- Async/await;
- Przestrzenie nazw.

4.3. Angular

Angular [9] jest to otwartoźródłowy framework do tworzenia webowych aplikacji klienckich typu SPA (*Single Page Application*). Napisany został w języku TypeScript. Implementuje swoje fundamentalne i opcjonalne funkcjonalności jako zbiór bibliotek, które programiści importują do tworzonych aplikacji. Za rozwój oraz wsparcie technologii Angular odpowiada Google.

Wydanie pierwszej wersji Angular JS miało miejsce w 2009 roku. Zadaniem biblioteki było wdrożenie wzorca MVC (*Model-View-Controller*) do klienckich aplikacji webowych oraz

umożliwienie tworzenie aplikacji SPA. Framework bardzo szybko zyskał popularność. W 2016 roku wydana została kolejna wersja nazwana Angular 2. Niestety porzucała ona kompatybilność wsteczną, co początkowo zraziło wielu programistów i częściowo wymusiło na zespołach projektowych przepisywanie projektów. Od tego momentu kolejne wersje technologii Angular oznaczane są numerem wersji oraz porzucony został suffix „JS".

Najważniejszym elementem technologii Angular są moduły. Odpowiadają one za grupowanie powiązanych komponentów, modeli, serwisów czy dyrektyw. Moduły mogą, choć nie muszą posiadać trasowania. Każdy projekt stworzony w technologii Angular posiada moduł główny. Dobrą praktyką jest dzielenie funkcjonalności aplikacji na większą liczbę modułów. Komponenty są jednymi z czołowych elementów projektów opartych o Angular. Składają się z części odpowiedzialnej za logikę oraz interfejs użytkownika. Dyrektywy natomiast dają możliwość przypisania pewnych zachowań do elementów HTML. Serwisy niosą funkcjonalność komunikacji z serwisami zewnętrznymi. Mogą wspomagać też komunikację pomiędzy komponentami, gdzie nie zachodzi relacja rodzic-dziecko.

Niektóre cechy technologii Angular:

- Możliwość tworzenia własnych atrybutów i znaczników HTML;
- Bindowanie;
- Oddzielenie logiki aplikacji od interfejsu użytkownika;
- Podział na moduły;
- Określony cykl życia komponentów;
- Śledzenie zmian;
- Łatwe tworzenie formularzy i ich walidacja;
- Angular CLI (narzędzie linii poleceń, umożliwiające generowanie szkieletu aplikacji).

4.4. Clarity Design

Clarity Design [10] jest to otwartoźródłowy projekt, który specyfikuje i implementuje zasady designu opracowanego w swoich ramach. Za rozwój i utrzymanie frameworka odpowiada VMware.

Projekt został podzielony na cztery biblioteki:

- clr/icons – biblioteka ikon;
- clr/ui – statyczne style do budowania komponentów HTML;
- clr/angular – komponenty Angular;
- clr/core – web komponenty, działające z każdym JavaScriptowym frameworkiem.

W skład Clarity Design wchodzi wiele gotowych do wykorzystania komponentów. Dodatkowo projekt posiada bogatą dokumentację wraz ze szczegółowo opisanymi przykładami użycia. Implementacja komponentów oferowanych przez ten framework znacznie przyśpiesza proces prac programistycznych. Innym popularnym frameworkiem wykorzystywanym w parze z technologią Angular jest Material Design. Clarity oferuje znacznie więcej funkcjonalności jeżeli chodzi o komponenty związane z prezentacją danych.

Przykładowe komponenty udostępniane przez Clarity Design:

- Okna modalne;
- Kreatory;
- Widok drzew;
- Rozbudowane listy z danymi;
- Elementy formularzy.

4.5. Selenoid

Selenoid [11] jest to otwartoźródłowa implementacja huba Selenium, która wykorzystuje kontenery Dockera do uruchamiania w nich przeglądarek i wykonywania testów. Narzędzie rozwijane jest przez firmę Aerokube. Głównym przeznaczeniem Selenoid jest możliwość uruchamiania równolegle wielu testów automatycznych w przeglądarkach. Rozwiązanie to jest wydajniejsze niż stosowanie domyślnego huba Selenium Grid. Dodatkowo Selenoid pozwala na nagrywanie wykonywanych testów oraz daje możliwość podłączenia się do nich w czasie rzeczywistym przez VNC. Według zaleceń twórcy technologii Selenoid optymalnym jest, aby każdy kontener z przeglądarką miał dostęp do przynajmniej jednego rdzenia procesora i jednego gigabajta pamięci. Dodatkowo technologia ta umożliwia dostęp do pobranych przez przeglądarkę plików w kontenerze i wykorzystanie schowka kopiuj-wklej.

Informacje o technologii Selenoid:

- Zużywa 10 razy mniej pamięci niż bazujący na Javie serwer Selenium (pod tym samym obciążeniem);
- Plik binarny zajmuje zaledwie 6 megabajtów i nie wymaga zewnętrznych zależności (nie wymaga Javy);
- Posiada interfejs API (np. za jego pomocą można pobierać nagrane filmy z testów);
- Umożliwia wysyłanie logów do scentralizowanego serwera logów;
- Stanowi w pełni izolowane i odtwarzalne środowisko.

4.6. MongoDB

MongoDB [12] jest to otwartoźródłowy, nierelacyjny i wieloplatformowy system zarządzania bazą danych. Cechuje go wydajność oraz skalowalność. Dane przechowywane są w postaci zbliżonej do dokumentów JSON i gromadzi się je w ramach kolekcji. Kolekcje są odpowiednikiem tabel w relacyjnych bazach danych. Powszechnym zastosowaniem tej bazy jest wykorzystanie jej jako magazynu danych dla usług sieciowych. Wersje MongoDB dzielą się na darmową Community oraz płatną Enterprise. Dokumenty przechowywane w bazie danych mają postać par klucz-wartość, podobnie jak obiekty JavaScript.

Główne funkcjonalności bazy danych MongoDB:

- Zapytania ad-hoc;
- Indeksowanie;
- Replikacja;

- Load balancing;
- Agregacje;
- Wykonywanie JavaScript po stronie serwera.

4.7. Docker

Docker [13] jest to otwartoźródłowe i wieloplatformowe oprogramowanie służące do automatyzacji procesu wdrażania aplikacji za pomocą kontenerów. Kontenery są czymś w rodzaju lekkich maszyn wirtualnych, które funkcjonują we własnym izolowanym kontekście. Umożliwiają one programiście stworzenie minimalnego, lecz w pełni wystarczającego środowiska do uruchomienia danej aplikacji. Dzięki temu zachodzi pewność, że dany kontener będzie zachowywał się tak samo na różnych systemach operacyjnych. Dodatkowo podejście to powoduje, że system hostujący dany kontener nie musi posiadać zainstalowanych zależności potrzebnych do uruchomienia danego oprogramowania. Wynika to z tego, iż wszystko co jest potrzebne do działania aplikacji znajduję się już w kontenerze. Konteneryzacje można porównać do tworzeniem przenośnych artefaktów binarnych.

W pewien sposób kontenery przypominają maszyny wirtualne, lecz różnią się od nich kilkoma rzeczami. Każda maszyna wirtualna potrzebuje osobnego systemu operacyjnego oraz wszystkich zależności potrzebnych do uruchomienia danej aplikacji. Wiąże się to z dużym zapotrzebowaniem na zasoby i może generować problemy. Natomiast kontenery działają w obrębie jądra hostującego systemu operacyjnego i zawierają jedynie zależności tworzące środowisko potrzebne do uruchomienia danego oprogramowania. To znacząco zmniejsza alokacje zasobów. Kontenery izolują się od systemu operacyjnego i od siebie, ale można zdefiniować komunikację pomiędzy nimi.

Podstawowe elementy Dockera [14]:

- Docker Engine – aplikacja typu klient-serwer;
- Deamon Dockera – usługa obsługująca żądania API, która jest pośrednikiem zarządzania obrazami, kontenerami, wolumenami i ustawieniami sieci;
- Klient Dockera – jest podstawowym sposobem interakcji użytkownika z Dockerem jako pośrednik komunikacji z usługą;
- Rejestr Dockera – jest miejscem, gdzie przechowywane są obrazy;
- Obiekty Dockera – obrazy, kontenery, usługi.

4.8. Visual Studio Code

Visual Studio Code [15] jest to otwartoźródłowy i wieloplatformowy edytor kodu autorstwa Microsoftu. Umożliwia on pracę z wieloma językami programowania takimi jak:

- Java;
- JavaScript;
- Go;
- Python;
- C++;
- C#;

- TypeScript.

Aplikacja bazuje na rozwiązaniu Electron [16] co oznacza, że Visual Studio Code jest tworzone z wykorzystaniem technologii aplikacji webowych, które zostały opakowane jako aplikacja desktopowa. Visual Studio Code jest agnostyczny względem kodu. Innymi słowy edytor nie faworyzuje swoimi możliwościami danego języka programowania. Zamiast tego umożliwia konfigurację środowiska pracy za pomocą wielu rozbudowanych rozszerzeń, które rozwijane są przez społeczność. Dzięki nim ten prosty edytor można skonfigurować do postaci zintegrowanego środowiska programistycznego wysokiej klasy.

Za pomocą rozszerzeń możemy usprawnić funkcjonalności edytora m.in. o:

- Obsługę języków programowania;
- Motywy;
- Statyczną analizę kodu;
- Generowanie kodu;
- Formatowanie kodu;
- Podpowiedzi.

4.9. Git

Git [17] jest to otwartoźródłowy i rozproszony system kontroli wersji. Został stworzony przez Linusa Torvaldsa przy okazji prac nad jądrem Linux. Narzędzie to pozwala zapisywać zmiany jakie zostały dokonane np. w danym folderze lub kodzie. Dodatkowo umożliwia powrót do poprzednio zapisanych wersji. Zmiany możemy przetrzymywać w repozytorium lokalnym lub zdalnym. Technologia ta głównie wykorzystywana jest przez zespoły programistyczne, ponieważ umożliwia szybkie i łatwe dzielenie się zmianami w kodzie oraz porównywanie co, kiedy i przez kogo zostało zmienione.

Podstawowe komendy związane z narzędziem Git [18]:

- Git clone – umożliwia klonowanie projektu z repozytorium zdalnego do repozytorium lokalnego;
- Git fetch – pobiera zmiany z repozytorium zdalnego;
- Git merge – scala dwie gałęzie w jedną;
- Git pull – git fetch + git merge;
- Git add – dodaje zmienione pliki do indeksu;
- Git commit – dodaje zmiany z indeksu do repozytorium;
- Git push – wypycha zmiany do repozytorium zdalnego;
- Git checkout – umożliwia przełączanie się pomiędzy gałęziami.

5. Implementacja

W tym rozdziale przedstawione zostaną szczegóły związane z implementacją prototypu aplikacji realizującej cele przedmiotowej pracy. Rozdział ten omawia poszczególne elementy składowe stworzonego systemu wraz z ich technicznym opisem. Na te elementy składają się: biblioteka, API oraz klient.

5.1. Biblioteka

Zaprojektowana biblioteka jest rdzeniem całej aplikacji. Została napisana w języku C# przy użyciu technologii .NET Core. Implementuje ona między innymi modele scenariusza i dokumentu, które są podstawowymi elementami komunikacji stworzonego systemu. Definiuje ona również mechanizmy przepływu informacji pomiędzy poszczególnymi jej komponentami.

5.1.1 Model scenariusza

Scenariusz jest to opis tego co i gdzie ma się wykonać. Za „co" rozumiemy metody oraz ich parametry, a „gdzie" oznacza proces. Procesy są to grupy metod, które mogą być wykonywane w architekturze rozproszonej. Tabele 1, 2, 3, 4, 5, 6 prezentują poszczególne elementy składające się na model scenariusza wraz z opisami ich pól. W stworzonym prototypie scenariusz reprezentowany jest przez dokument JSON [19].

Hierarchia elementów scenariusza zaprezentowana w postaci drzewa:

- Scenariusz
    - Lista Procesów
        - Lista Grup
            - Lista Metod

Tabela 1. Model obiektu Scenario – składowa modelu scenariusza

Typ	Nazwa	Opis
Guid	Id	Unikalny identyfikator scenariusza
string	Codename	Unikalna nazwa „kodowa" scenariusza
string	Domain	Domena z którą powiązany jest kontekst działania scenariusza
string	Description	Opis tego co dany scenariusz wykonuje
string	Icon	URL do ikony powiązanej ze scenariuszem
string	Image	URL do obrazu powiązanego ze scenariuszem
DateTimeOffset	CreatedAt	Data dodania scenariusza
DateTimeOffset	UpdatedAt	Data ostatniej aktualizacji scenariusza
Lista Scenario Processes	ScenarioProcesses	Lista procesów scenariusza

Tabela 2. Model obiektu ScenarioProcess – składowa modelu scenariusza

Typ	Nazwa	Opis
int	Stage	Indeks procesu, który jest tożsamy z pozycją obiektu *ScenarioProcess* na liście
string	Category	Kategoria procesu
string	Type	Typ procesu
Guid	ProcessId	Unikalny identyfikator procesu
Lista Scenario Group	Groups	Lista grup scenariusza

Tabela 3. Model obiektu ScenarioGroup – składowa modelu scenariusza

Typ	Nazwa	Opis
string	GroupCategory	Kategoria grupy
string	GroupName	Nazwa grupy
Lista Scenario Method	Methods	Lista metod scenariusza
int	Order	Indeks grupy, który jest tożsamy z pozycją obiektu *ScenarioGroup* na liście

Tabela 4. Model obiektu ScenarioMethod – składowa modelu scenariusza

Typ	Nazwa	Opis
string	Method	Nazwa metody
Lista Parameter Dto	Parameters	Lista parametrów metody, które są zdefiniowane przed uruchomieniem scenariusza
Lista Values From Context Dto	ValuesFromContext	Lista wartości, które również stanowią parametry dla metody, jednakże pochodzą z dotychczasowych wyliczeń uruchomionego scenariusza
string	Context	Kontekst jest to unikalny identyfikator w obrębie grupy metod, który wskazuje na wartość wyniku wykonania metody
string	ContextType	Typ kontekstu
int	Order	Indeks metody, który jest tożsamy z pozycją obiektu *ScenarioMethod* na liście
int	Group	Indeks grupy do której należy metoda
string	GroupCategory	Kategoria grupy
string	GroupName	Nazwa grupy
string	EntityType	Typ encji

Tabela 5. Model obiektu ParameterDto – składowa modelu scenariusza

Typ	Nazwa	Opis
string	ParameterName	Nazwa parametru
object	Parameter	Wartość parametru
string	ParameterType	Typ parametru

Typ	Nazwa	Opis
bool	IsOptional	Czy parametr jest opcjonalny
bool	IsVariable	Czy parametr ten może stanowić zmienną

Tabela 6. Model obiektu ValuesFromContextDto – składowa modelu scenariusza

Typ	Nazwa	Opis
string	GroupCategory	Kategoria grupy
string	GroupName	Nazwa grupy
Lista Scenario Method	Methods	Lista metod scenariusza
int	Order	Indeks grupy, który jest tożsamy z pozycją obiektu *ScenarioGroup* na liście

Listing 1 przedstawia przykładowy minimalny scenariusz, który zawiera tylko jedną metodę. Metoda ta nosi nazwę się *„TargetUrlStrict”* i przyjmuje tylko jeden parametr *„Url”*, który jest typu string. Dostęp do wartości zwracanej przez wykonanie metody uzyskać można za pomocą klucza kontekstu *„Url”*. Zadaniem tej metody jest powiązanie odnośnika URL ze scenariuszem. Przesłaniem takiego zachowania jest możliwość korzystania z tej wartości przez mogące występować, kolejne metody przepisu. W strukturze omawianego dokumentu można zauważyć pewną denormalizację i przechowywanie wartości, które można wyliczyć np. indeks obiektu znajdującego się na liście. Jednakże jest to działanie celowe, które rozwiązuje wiele potencjalnych problemów. Scenariusz ten został przygotowany za pomocą webowego kreatora, który szczegółowo przedstawiony jest w podrozdziale 5.3.1.

Listing 1. Dokument JSON z minimalnym poprawnym scenariuszem

```
{
  "id": "bbba53fa-cb8e-441c-b46c-62d2f70cbf88",
  "codename": "Example",
  "domain": "example.com",
  "description": "To jest przykład struktury scenariusza.",
  "icon": "https://example.org/icon.png",
  "image": "https://example.org/image.png",
  "createdAt": "2020-05-06T14:34:59.4417614+02:00",
  "updatedAt": "2020-05-06T15:17:54.1422417+02:00",
  "scenarioProcesses": [
    {
      "stage": 0,
      "category": "ScenarioSettings",
      "type": "ScenarioSettings",
      "processId": "7c19c00e-d2b4-49bb-82c9-fa9c519ac439",
      "groups": [
        {
          "groupCategory": "Default",
          "groupName": "Default",
          "order": 0,
          "methods": [
            {
              "method": "TargetUrlStrict",
              "parameters": [
                {
                  "parameterName": "Url",
                  "parameter": "https://example.com",
                  "parameterType": "string",
```

```
                    "isVariable": false,
                    "isOptional": false
                  }
                ],
                "context": "Url",
                "contextType": "Url",
                "valuesFromContext": [],
                "order": 0,
                "group": 0,
                "groupCategory": "Default",
                "groupName": "Default",
                "entityType": "Url"
              }
            ]
          }
        ]
      }
  ]
}
```

5.1.2 Model dokumentu

Dokument jest to rodzaj raportu, który dostarcza wszystkich kluczowych informacji dotyczących wykonanego scenariusza. Celem jaki towarzyszył przy tworzeniu jego modelu było to, aby w szybki i prosty sposób ocenić czy dany scenariusz wykonał się poprawnie. Dodatkowo zaimplementowany został w taki sposób, by umożliwić wychwycenie informacji o tym co, kiedy i gdzie nie zadziałało poprawnie. Dzięki swojej strukturze może on pełnić funkcję źródła informacji dla potencjalnego walidatora, który ponawiałby niepoprawnie wykonane scenariusze. Tabele 7, 8, 9, 10 prezentują poszczególne elementy składające się na model dokumentu wraz z opisami ich pól. Dokument tak jak scenariusz reprezentowany jest przez dokument JSON.

Tabela 7. Model obiektu Document - składowa modelu dokumentu

Typ	Nazwa	Opis
Guid	Id	Unikalny identyfikator dokumentu
Guid	ScenarioId	Identyfikator scenariusza, którego wynikiem wykonania jest ten dokument
string	ScenarioName	Wartość pochodząca z nazwy kodowej scenariusza
int	CurrentStage	Identyfikator aktualnie przetwarzanego procesu
string	CurrentProcess	Nazwa aktualnie przetwarzanego procesu
string	CurrentMethod	Nazwa aktualnie przetwarzanej metody
string	CurrentType	Nazwa aktualnie przetwarzanego typu metody
DateTimeOffset	CreatedAt	Data utworzenia dokumentu
DateTimeOffset	UpdatedAt	Data ostatniej aktualizacji dokumentu
TimeSpan	Duration	Czas wykonywania scenariusza
int	Errors	Ilość błędów, która wystąpiła podczas wykonywania scenariusza
bool	IsFinished	Czy wykonywanie scenariusza się zakończyło
Lista Process	Stages	Lista procesów scenariusza

Tabela 8. Model obiektu Process – składowa modelu dokumentu

Typ	Nazwa	Opis
int	Stage	Indeks procesu, który jest tożsamy z pozycją obiektu *Process* na liście
string	ProcessName	Nazwa procesu
string	Type	Typ procesu
Słownik <string, ValueDto>	MethodsValues	Słownik, którego kluczem jest nazwa kontekstu scenariusza, a wartość zawiera wynik wykonania metody wraz z dodatkowymi informacjami o metodzie
Metdadata	Metdadata	Obiekt zawierający metadane związane z przebiegiem danego procesu

Tabela 9. Model obiektu ValueDto – składowa modelu dokumentu

Typ	Nazwa	Opis
string	Method	Nazwa metody
object	Value	Wynik wykonania metody
string	ValueType	Typ wyniku wykonania metody
int	Group	Indeks grupy do której należała metoda
string	GroupCategory	Kategoria grupy
string	GroupName	Nazwa grupy
int	Order	Kolejność metody w grupie

Tabela 10. Model obiektu Metdadata – składowa modelu dokumentu

Typ	Nazwa	Opis
DateTimeOffset	ProcessStart	Data rozpoczęcia wykonywania procesu
DateTimeOffset	ProcessEnd	Data zakończenia wykonywania procesu
TimeSpan	Duration	Czas trwania przebiegu procesu
Guid	ProcessId	Unikalny identyfikator procesu
Guid	MicroServiceId	Unikalny identyfikator potencjalnego mikroserwisu, który odpowiadał za przetwarzanie procesu
Lista string	Exceptions	Lista wyjątków
Słownik <string, string>	ExceptionsUserFriendlyInfo	Lista wyjątków przyjaznych dla klientów

Listing 2 przedstawia wynik działania przetworzenia przez prototyp scenariusza z Listingu 1. W porównaniu do modelu scenariusza, model dokumentu posiada znacznie mniej zagnieżdżeń. Schemat dokumentu jest przygotowany na działanie w architekturze rozproszonej. Zachowuje on stan każdego wykonanego procesu. Teoretycznie dzięki takiej budowie istnieje możliwość ponownego uruchamiania niepoprawnie wykonanych scenariuszy z zachowaniem pewnego stanu. W takim przypadku mogłoby to się odbywać od momentu procesu, którego wykonanie nie udało się. Daje to duże możliwości rozwoju prototypu.

Listing 2. Przykładowy wynik poprawnie wykonanego scenariusza - dokument

```
{
  "id": "4cefd260-4ca9-4af5-8217-2d1233ada739",
  "scenarioId": "8d3fa0dd-4e59-4f52-8a2b-253f9054e8b6",
  "scenarioName": "Example",
  "currentStage": 0,
  "currentProcess": "ScenarioSettings",
  "currentMethod": "TargetUrlStrict",
  "currentType": "ScenarioSettings",
  "createdAt": "2020-05-06T16:16:01.1563771+02:00",
  "updatedAt": "2020-05-06T16:16:01.2084884+02:00",
  "duration": "00:00:00.0000029",
  "errors": 0,
  "isFinished": true,
  "stages": [
    {
      "stage": 0,
      "processName": "ScenarioSettings",
      "type": "ScenarioSettings",
      "methodsValues": {
        "url": {
          "method": "TargetUrlStrict",
          "value": "https://example.com",
          "valueType": "string",
          "group": 0,
          "groupCategory": "Default",
          "groupName": "Default",
          "order": 0
        }
      },
      "metdadata": {
        "processStart": "2020-05-06T16:16:01.1743635+02:00",
        "processEnd": "2020-05-06T16:16:01.2084672+02:00",
        "duration": "00:00:00.0341037",
        "processId": "3c304a7a-843d-4a1c-95c8-888b9f1518fd",
        "microServiceId": "a36795b8-e42b-45f0-acb3-c333c0923c9b",
        "exceptions": [],
        "exceptionsUserFriendlyInfo": {}
      }
    }
  ]
}
```

5.1.3 Budowa biblioteki

Biblioteka została stworzona jako aplikacja konsolowa .NET Core. Do jej implementacji użyto języka C#. W bibliotece zaimplementowane są wszystkie procesy, grupy i metody, które mogą zostać wywołane i dodane do scenariuszy. Dodatkowo biblioteka potrafi zbudować i wystawić bazę wiedzy na temat swoich funkcjonalności. Przykładowo informacje dla klienta o tym jakie metody można dodać z poziomu interfejsu użytkownika do scenariusza pochodzą z biblioteki. To samo tyczy się metadanych wystawianych dla sekcji pomocy w kliencie. Dzieję się tak dzięki zastosowanym w kodzie abstrakcjom i refleksji. Przyjęte reguły pozwalają w dość prosty sposób rozbudowywać bibliotekę, a tym samym aplikację o nowe metody, które są następnie widoczne z poziomu kreatora scenariuszy.

Z powodu problemów związanych z przechowywaniem w bazie danych MongoDB typów obiektów został zaimplementowany dwukierunkowy słownik mapujący typy na łańcuchy znaków. Dlatego w scenariuszu i dokumencie typy są łańcuchami znaków.

W tabeli 11 przedstawiona została struktura stworzonej biblioteki za pomocą omówienia jej przestrzeni nazw.

Tabela 11. Przedstawienie struktury biblioteki za pomocą przestrzeni nazw i opisów

Przestrzeń nazw w projekcie biblioteki	Opis co się w niej znajduje
Biblioteka.Common	Wszelkiego rodzaju klasy pomocnicze i rozszerzenia np. metoda do kompresji łańcuchów znaków lub dwukierunkowy słownik wykorzystywany do mapowania typów
Biblioteka.Dictionaries	Statyczne klasy reprezentujące słowniki m.in. dla procesów, grup, metod; wykorzystywane podczas definiowania relacji lub nowych elementów scenariusza
Biblioteka.Dto	Klasy służące do tworzenia obiektów, które odpowiadają jedynie za transfer danych
Biblioteka.Model	Klasy definiujące modele
Biblioteka.Modules.Abstractions	Klasy abstrakcyjne i interfejsy wykorzystywane do tworzenia nowych składowych elementów scenariuszy
Biblioteka.Modules.Group	Klasy definiujące grupy możliwe do wykorzystania w scenariuszach
Biblioteka.Modules.Method	Klasy definiujące metody możliwe do wykorzystania w scenariuszach
Biblioteka.Modules.MethodGroups	Klasy umożliwiające podział grup na typy; służy to możliwości definiowania relacji np. przypisania grupy tylko dla konkretnego procesu
Biblioteka.Modules.Process	Klasy definiujące procesy możliwe do wykorzystania w scenariuszach
Biblioteka.Modules.ProcessGroups	Klasy umożliwiające podział procesów na typy; służy to możliwości definiowania relacji np. przypisania metod tylko dla konkretnego procesu
Biblioteka	Klasy odpowiadające za rdzeń biblioteki definiujące przepływ i mapowanie informacji oraz relacji

Lista zależności stworzonej biblioteki:

- HtmlAgilityPack;
- Serilog;
- Serilog.Exceptions;
- Serilog.Sinks.Console;
- Jering.Javascript.NodeJS;
- Selenium.WebDriver;
- Selenium.WebDriver.ChromeDriver;
- DotNetSeleniumExtras.WaitHelpers;
- MongoDB.Driver;
- Newtonsoft.Json.

W tabeli 12 omówione zostały publiczne metody, które udostępnia biblioteka. Główny konstruktor klasy biblioteki przyjmuje dwa parametry – scenariusz i dokument. Metody: *ProcessStage* i *ProcessAll* operują na parametrach udostępnionych przez ten konstruktor klasy.

Tabela 12. Publiczne metody udostępniane przez zaimplementowaną bibliotekę

Zwracany typ	Nazwa	Opis
Document	ProcessStage	Metoda ta na podstawie dokumentu wykonuje następny w kolejności zdefiniowany w scenariuszu proces; dzięki takiemu podejściu, możliwe jest rozproszenie wykonywania procesów np. z wykorzystaniem brokera wiadomości
Document	ProcessAll	Metoda ta rekurencyjnie wykonuje funkcję *ProcessStage*, co umożliwia przetworzenie wszystkich procesów scenariusza w architekturze monolitycznej
string (json)	GetMethods	Informacje o zaimplementowanych metodach dla klienta
string (json)	GetMethodsMetadata	Metadane metod dla klienta
string (json)	GetProcesses	Informacje o zaimplementowanych procesach dla klienta
string (json)	GetProcessesMetadata	Metadane procesów dla klienta
string (json)	GetGroups	Informacje o zaimplementowanych grupach dla klienta
string (json)	GetGroupsMetadata	Metadane grup dla klienta

5.1.4 Przepływ informacji

Wykonywanie scenariuszy polega na przetwarzaniu ich kolejnych procesów przez bibliotekę. Załóżmy, że chcemy wykonać scenariusz składający się z kilku procesów, grup i metod. W takim przypadku algorytm w przybliżeniu wygląda następująco:

1. Na wejściu przyjmij scenariusz oraz dokument. Podczas tworzenia nowego dokumentu automatycznie uzupełniane są podstawowe informacje, które można pozyskać na podstawie scenariusza. Indeks aktualnie przetwarzanego procesu w dokumencie = 0.
2. Do listy procesów w dokumencie dodaj proces oraz uzupełnij go informacjami pochodzącymi ze scenariusza, którego indeks procesu równa się indeksowi aktualnie przetwarzanego procesu dokumentu. Do procesu w dokumencie dodawane są: indeks procesu, nazwa procesu, typ procesu, nowy obiekt metadanych.
3. W aktualnie przetwarzanym procesie w dokumencie zaktualizuj obiekt metadanych o dane: data rozpoczęcia wykonywania procesu, identyfikator procesu i usługi.
4. Na podstawie scenariusza i dokumentu parsuj aktualnie przetwarzany proces i uzyskaj zdefiniowane w nim metody do wykonania. Zwróć posortowaną listę obiektów *ScenarioMethod* dla aktualnego procesu.
5. Iteruj po liście metod, każdą z nich wywołaj za pomocą refleksji, a ich wyniki dodaj do dokumentu.
6. W aktualnie przetwarzanym procesie w dokumencie zaktualizuj obiekt metadanych o dane: data zakończenia wykonywania procesu, długość trwania procesu.
7. Jeżeli w scenariuszu istnieje kolejny proces, to zaktualizuj w dokumencie: indeks aktualnie przetwarzanego procesu w dokumencie (inkrementacja o 1), aktualnie przetwarzany proces oraz jego typ, następnie wykonaj algorytm od początku. Jeżeli w scenariuszu nie istnieje kolejny proces to zaktualizuj flagę *IsFinished* na *True* i zakończ algorytm.

Listing 3 prezentuje implementacje przykładowej prostej metody, której zadaniem jest przypisać podany w parametrze adres URL do dokumentu. Przygotowane reguły wymagają, aby klasa implementująca nową metodę nazywała się tak samo jak ma nazywać się metoda. Klasa ta powinna dziedziczyć po generycznej klasie abstrakcyjnej *„GenericMethod”* dla której parametr generyczny odpowiada typowi zwracanemu przez metodę. Klasa ta powinna również implementować interfejs *IName*. Interfejs ten wymusza implementację właściwości *Name*, która przyjmuję wartość nazwy klasy, czyli tym samym metody. Abstrakcyjna klasa *„GenericMethod”* implementuje interfejs *„IMethod”*, który jest interfejsem markującym. Oznacza to, że służy tylko do tego, aby odnieść się do implementujących go klas za pomocą refleksji. Klasa *GenericMethod* implementuje również metody pozwalające w łatwy sposób klasom dziedziczącym otrzymywać parametry oraz wartości z kontekstu dokumentu. Dodatkowo posiada ona metody abstrakcyjne, które są nadpisywane przez klasy dziedziczące. Tymi metodami są: *MethodImplementation* oraz *MethodMetadata*. Metoda *MethodImplementation* implementuje i wykonuje funkcję lokalną, która nazywa się tak samo jak klasa. Metoda *MethodMetadata* tworzy obiekt *ClientMethods*, który zawiera informację o metodzie. Informacje te mogą zostać udostępnione dla klienta. Obie te metody wywoływane są przez refleksję.

Listing 3. Implementacja przykładowej metody – składowej scenariusza

```
1.  public class TargetUrlStrict: GenericMethod < string > , IName {
2.
3.   public string Name => this.GetType().Name;
4.   public TargetUrlStrict() {}
5.   public TargetUrlStrict(object[] parameters): base(parameters) {}
6.
7.   public override string MethodImplementation(object[] parameters) {
8.    string url = HandleInnerParameters("Url");
9.
10.   string TargetUrlStrict(string url) {
11.    return url;
12.   }
13.
14.   return TargetUrlStrict(url);
15.  }
16.
17.  public override ClientMethods MethodMetadata() {
18.
19.   string methodName = Name;
20.   string methodDescription = "Metoda przypisuje adres URL żądania.";
21.   string returnContext = Relations.GetReturnContextsTypesByMethodName(Name);
22.
23.   ClientMethods clientMethods = new ClientMethods();
24.
25.   clientMethods.ScenarioMethod = new ScenarioMethod() {
26.    Method = methodName,
27.     Parameters = new List < ParameterDto > () {
28.      new ParameterDto() {
29.       Parameter = null,
30.        ParameterName = "Url",
31.        ParameterType = Types.GetStringByType(typeof(string)),
32.      },
33.     },
34.     ValuesFromContext = new List < ValuesFromContextDto > (),
35.     Context = returnContext,
36.     ContextType = returnContext,
37.     EntityType = EntityTypes.None
38.   };
39.
40.   return clientMethods;
41.
42.  }
```

```
43. }
```

Listing 4 przedstawia przykład implementacji refleksji, która używana jest do zbudowania listy informacji o zaimplementowanych metodach dla klienta. Metoda działa następująco:

1. Utwórz listę, która będzie przechowywać informację o metodach dla klienta.
2. Przeszukaj klasy, które implementują interfejs *IMethods*.
3. Stwórz instancję tych klas.
4. Iteruj po utworzonych instancjach i wyszukaj w nich metody *„MethodMetadata”*, którą następnie wywołaj, a wynik jej działania dodaj do listy.
5. Zwróć listę przechowującą informacje o metodach dla klienta.

Listing 4. Przykład implementacji refleksji służącej do wydobycia informacji o metodach

```
1.  public List < ClientMethods > GetClientMethods() {
2.
3.   List < ClientMethods > clientMethods = new List < ClientMethods > ();
4.
5.   var instances = from t in Assembly.GetExecutingAssembly().GetTypes()
6.   where t.GetInterfaces().Contains(typeof(IMethod)) && t.GetConstructor(Type.EmptyTyp
    es) != null && t.IsGenericTypeDefinition == false
7.   select Activator.CreateInstance(t);
8.
9.   foreach(var instance in instances) {
10.
11.   Type classType = instance.GetType();
12.   MethodInfo theMethod = classType.GetMethod("MethodMetadata");
13.   var retVal = theMethod.Invoke(instance, null) as ClientMethods;
14.   clientMethods.Add(retVal);
15.
16.  }
17.
18.  return clientMethods;
19. }
```

5.2. API

Klient aplikacji komunikuje się z częścią serwerową za pomocą API. To w tym elemencie stworzonego systemu wywoływane są metody przedstawionej w rozdziale 5.1 biblioteki oraz operacje bazodanowe. W poniższych podrozdziałach omówiona zostanie architektura API oraz punkty dostępu.

5.2.1 Architektura

Część serwerowa prototypu została zaimplementowana jako aplikacja webowa ASP.NET Core przy użyciu języka programowania C#. Składa się ona z czterech projektów z których każdy odpowiada za oddzielną warstwę systemu. W tabeli 13 zaprezentowane zostały poszczególne elementy architektury oraz ich opis. Przyjęty podział można sklasyfikować jako *clean architecture* lub *onion architecture* [20]. Polega on m.in. na tym, że odseparowujemy model biznesowy od świata zewnętrznego. Niesie to ze sobą większe możliwości testowania kodu oraz poszerza jego niezależność względem wykorzystanych bibliotek lub bazy danych. Architektura ta często prezentowana jest za pomocą rysunku koła w którym wpisane są kolejne koła w zależności od ilości warstw. Każde z kół oznacza poszczególną warstwę. To koło, które jest wpisane w samych środku nie ma żadnych zależności, a każdy poziom warstwy wyżej zależny jest od warstw poniższych.

Tabela 13. Warstwowa architektura API

Warstwa	Opis
Core	Warstwa w której zdefiniowane są encje oraz interfejsy do implementacji repozytoriów (warstwa nie posiada zależności)
Application	W warstwie aplikacji zaimplementowane są elementy CQRS czyli komendy i zdarzenia oraz ich handlery
Infrastructure	W warstwie infrastruktury zdefiniowane są klasy umożliwiające mapowanie encji na dokumenty MongoDB, handlery dla zapytań pobierających dane z bazy danych oraz implementacja wzorca repozytorium do obsługi operacji na bazie danych
Api	W warstwie API zdefiniowane są punkty dostępu

Serwer API został zaimplementowany z udziałem biblioteki *Convey* [21]. Biblioteka ta wspomaga tworzenie mikroserwisów, ponieważ zawiera zdefiniowane abstrakcje i wzorce, które ułatwiają prace, i rozwiązują wiele problemów architektury rozproszonej. Jednakże jest ona na tyle uniwersalna, że również można ją wykorzystać przy tworzeniu monolitycznego serwisu. Przykładem użytej w prototypie abstrakcji zaimplementowanej przez tą bibliotekę jest wzorzec CQRS (*Command Query Responsibility Segregation)* [22]. W dużym uproszczeniu polega on na tym, że oddzielamy zapis od odczytu. Zamiast jednego modelu do wszystkiego, używamy osobne modele do pobierania danych oraz ich zapisywania. Komendy odpowiadają za zapis, a zapytania za odczyt. Na listingu 5 i 6 przedstawiona została implementacja przykładowej komendy oraz jej handlera. Odpowiada ona za zapis scenariusza do bazy danych. Komenda posiada atrybut „*Contract*" oraz implementuje interfejs markujący *ICommand*. Klasa obsługująca wykonanie komendy implementuje interfejs *ICommandHandler*, tym samym wymaga zdefiniowania ciała metody *HandleAsync*. Wywołanie komendy powoduje wyzwolenie jej handlera.

Listing 5. Implementacja komendy odpowiadającej za zapis scenariusza

```
1.  [Contract]
2.  public class CreateScenario: ICommand {
3.   public Guid Id { get; }
4.   public string Codename { get; }
5.   public string Domain { get; }
6.   public string Description { get; }
7.   public string Icon { get; }
8.   public string Image { get; }
9.   public DateTimeOffset CreatedAt { get; }
10.  public DateTimeOffset UpdatedAt { get; }
11.  public IList < ScenarioProcess > ScenarioProcesses { get; }
12.
13.  public CreateScenario(Guid id, string codename, string domain, string description,
    string icon, string image,
14.   DateTimeOffset createdAt, DateTimeOffset updatedAt, IList < ScenarioProcess > scen
    arioProcesses) {
15.   Id = id == Guid.Empty ? Guid.NewGuid() : id;
16.   Codename = codename;
17.   Domain = domain;
18.   Description = description;
19.   Icon = icon;
20.   Image = image;
21.   CreatedAt = createdAt;
22.   UpdatedAt = updatedAt;
23.   ScenarioProcesses = scenarioProcesses;
24.  }
25. }
```

Listing 6. Implementacja handlera komendy odpowiadającej za zapis scenariusza

```
1.  internal class CreateScenarioHandler: ICommandHandler < CreateScenario > {
2.   private readonly IScenariosRepository _repository;
3.   private readonly ILogger < CreateScenarioHandler > _logger;
4.
5.   public CreateScenarioHandler(IScenariosRepository repository, ILogger < CreateScena
    rioHandler > logger) {
6.    _repository = repository;
7.    _logger = logger;
8.   }
9.
10.  public async Task HandleAsync(CreateScenario command) {
11.   var scenario = new Scenario() {
12.    Id = command.Id,
13.     Codename = command.Codename,
14.     Description = command.Description,
15.     Domain = command.Domain,
16.     Icon = command.Icon,
17.     Image = command.Image,
18.     CreatedAt = DateTimeOffset.Now,
19.     UpdatedAt = DateTimeOffset.Now,
20.     ScenarioProcesses = command.ScenarioProcesses
21.   };
22.
23.   await _repository.AddAsync(scenario);
24.  }
25. }
```

5.2.2 Punkty dostępu (Endpointy)

API udostępnia kilkanaście punktów dostępu, które są możliwe do odpytania. W tabeli 14 przedstawione zostały zaimplementowane endpointy API wraz z ich opisem oraz wyzwalającą je metodą HTTP. Dzięki zastosowaniu abstrakcji z biblioteki Convey.WebApi [23] możliwe jest definiowanie kontrolerów punktów dostępu zintegrowanych z CQRS. Na listingu 7 przedstawiono przykładową deklarację endpointu, który odpowiada za utworzenie (zapis) nowego scenariusza. W odpowiedzi zwracany jest identyfikator utworzonego elementu. Wysłanie żądania pod dany endopoint wyzwala komendę lub zapytanie z nim związane. W deklaracji punktu dostępu oprócz podania elementu, który ma zostać wyzwolony, definiujemy również typ obiektu przez niego zwracanego. Przedstawiono to na listingu 8, gdzie umieszczono przykład definicji punktu dostępu odpowiadającego za odczyt scenariuszy.

Tabela 14. Endpointy API

Metoda HTTP	**Endpoint**	**Opis**
GET	/	Endpoint zwraca nazwę serwisu; może być wykorzystywany, żeby sprawdzić czy serwer API jest uruchomiony i czy odpowiada na żądania
GET	/test-scenarios	Endpoint zwraca 100 ostatnich rekordów testowych scenariuszy
GET	/test-documents	Endpoint zwraca 100 ostatnich rekordów testowych dokumentów
GET	/test-scenario/{ScenarioId}	Endpoint zwraca testowy scenariusz na

Metoda HTTP	Endpoint	Opis
		podstawie jego identyfikatora
GET	/testdocument/{ScenarioIdInDocument}	Endpoint zwraca testowy dokument na podstawie identyfikatora scenariusza, którego wywołanie utworzyło dokument
GET	/scenarios	Endpoint zwraca 100 ostatnich rekordów zapisanych scenariuszy
GET	/scenario/{ScenarioId}	Endpoint zwraca scenariusz na podstawie identyfikatora
PUT	/scenario/{ScenarioId}	Endpoint aktualizuje scenariusz na podstawie identyfikatora
DELETE	/scenario/{id}	Endpoint usuwa scenariusz na podstawie identyfikatora
GET	/processes	Endpoint zwraca informacje o zaimplementowanych w bibliotece procesach
GET	/processesmetadata	Endpoint zwraca metadane procesów zaimplementowanych w bibliotece
GET	/groups	Endpoint zwraca informacje o zaimplementowanych w bibliotece grupach
GET	/groupsmetadata	Endpoint zwraca metadane grup zaimplementowanych w bibliotece
GET	/methods	Endpoint zwraca informacje o zaimplementowanych w bibliotece metodach
GET	/methodsmetadata	Endpoint zwraca metadane metod zaimplementowanych w bibliotece
POST	/create-scenario	Endpoint dodaje nowy scenariusz do bazy danych
POST	/start-scenario-test	Endpoint uruchamia proces testowania scenariusza, czyli jego przetworzenia przez prototyp w celu uzyskania raportu (dokumentu)

Listing 7. Przykład deklaracji endpointu odpowiadającego za zapis scenariusza

```
1.  .UseDispatcherEndpoints(endpoints => endpoints
2.    .Post < CreateScenario > ("create-scenario",
3.   afterDispatch: (cmd, ctx) => {
4.    return ctx.Response.Created($ "{cmd.Id}");
5.   })"))
```

Listing 8. Przykład deklaracji endpointu odpowiadającego za odczyt scenariuszy

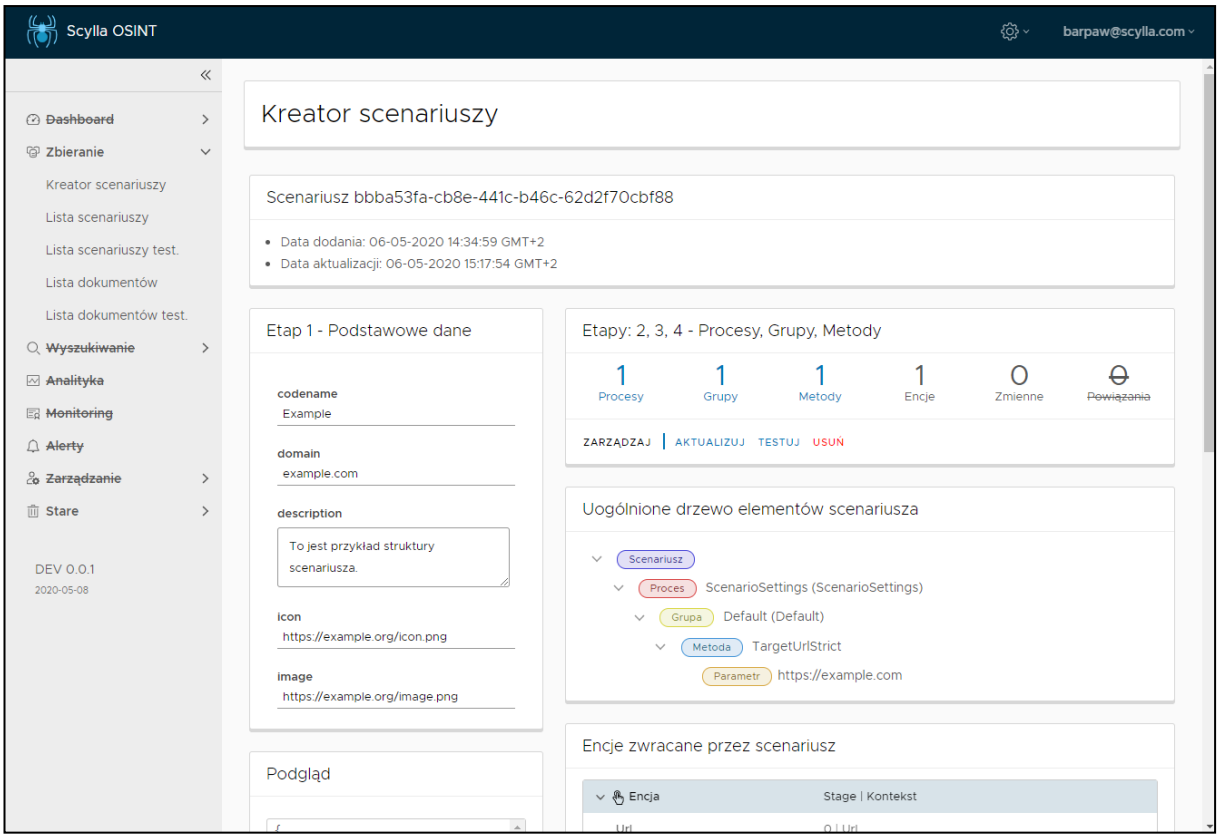
```
1.  .UseDispatcherEndpoints(endpoints => endpoints
2.    .Get < GetScenarios, List < Scenario >> ("scenarios")
```

5.3. Klient

Klient prototypu został zaimplementowany w technologii Angular z wykorzystaniem języka TypeScript. Stworzone komponenty bazują na stylach i funkcjonalnościach opartych o biblioteki Clarity Design oraz Angular CDK. W poniższych podrozdziałach omówione zostaną techniczne aspekty klienta prototypu.

5.3.1 Kreator scenariuszy

Kreator scenariuszy jest głównym komponentem klienta stworzonego prototypu. Umożliwia on tworzenie scenariuszy za pomocą graficznego interfejsu. Celem jaki towarzyszył podczas jego implementacji było to, aby tworzenie scenariuszy było proste i intuicyjne. W tym celu zamiast tradycyjnego formularza zastosowano mechanizmy okien modalnych oraz funkcjonalność drag and drop. Na rysunku 6 przedstawiono interfejs kreatora scenariuszy. Składa się on z kilku komponentów, które zostały omówione w tabeli 15.



Rysunek 6. Interfejs kreatora scenariuszy

Tabela 15. Komponenty wchodzące w skład kreatora scenariuszy

Nazwa komponentu	Opis
modal-document	Modal, który umożliwia podgląd dokumentu w formacie JSON, koloruje składnie oraz pozwala na zwijanie i rozwijanie zagnieżdżeń
modal-group	Modal, który odpowiada za dodawanie grup do formularza
modal-group-metadata	Modal, w którym pokazywane są dedykowane metadane dla wybranej grupy
modal-mesage-box	Modal, który umożliwia wyświetlanie zdefiniowanej informacji
modal-method	Modal, który odpowiada za dodawanie metod do formularza

Nazwa komponentu	Opis
modal-method-metadata	Modal, w którym pokazywane są dedykowane metadane dla wybranej metody
modal-process	Modal, który odpowiada za dodawanie procesów do formularza
modal-process-metadata	Modal, w którym pokazywane są dedykowane metadane dla wybranego procesu
modal-scenario	Modal, który umożliwia podgląd scenariusza jako JSON lub w postaci uogólnionego drzewa; dodatkowo z poziomu tego komponentu można zobaczyć encje oraz zmienne zdefiniowane w scenariuszu
scenario-creator	Komponent główny kreatora scenariuszy; jest rodzicem wszystkich wymienionych w tej tabeli komponentów
scenario-entities	Komponent umożliwiający podgląd encji zdefiniowanych w scenariuszu
scenario-tree	Komponent umożliwiający podgląd scenariusza w postaci uogólnionego drzewa
scenario-variables	Komponent umożliwiający podgląd zmiennych zdefiniowanych w scenariuszu

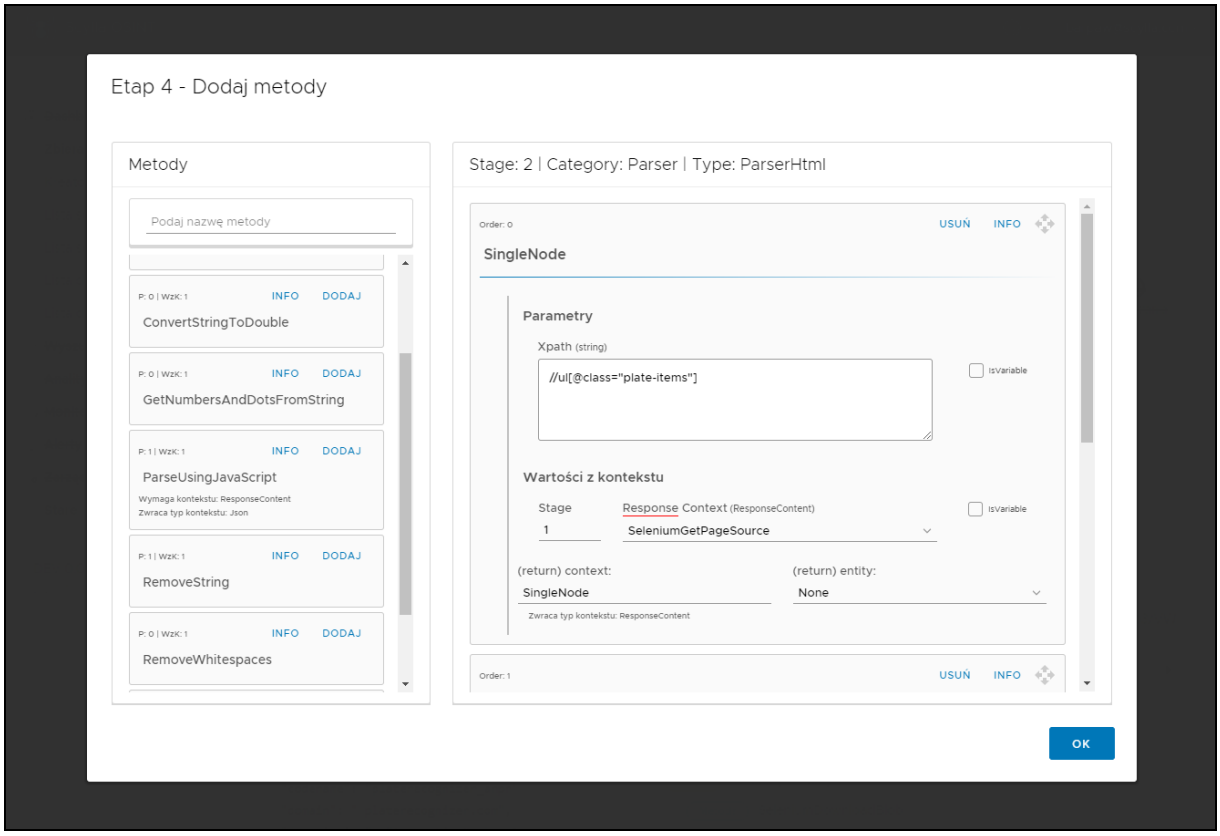
Tworzenie scenariuszy składa się z kilku etapów:

- Etap pierwszy polega na wypełnieniu podstawowych danych związanych z tworzonym bytem tj. nazwa, domena, opis itp.
- Etap drugi polega na wybraniu procesów.
- Etap trzeci polega na wybraniu grup oraz możliwości ich nazwania.
- Etap czwarty jest to najbardziej rozwinięty element kreatora. Umożliwia on dodawanie oraz wyszukiwanie metod. W zależności od wybranej metody musimy uzupełnić jej parametry lub wartości z kontekstu. Mechanizmy, które zostały przygotowane w tym celu są nastawione na intuicyjność i szybkość tworzenia oraz edycji. Na rysunku 7 zostało przedstawione okno modalne służące do dodawania metod.

Przykładowe zaimplementowane mechanizmy, które ułatwiają pracę z kreatorem:

- Drag and drop – elementy wchodzące w skład scenariuszy możemy dodawać za pomocą przeciągania ich między listami. Dodatkowo możliwe jest przenoszenie elementów w obrębie docelowej listy kreatora.
- Podpowiadanie pól związanych z wartościami z kontekstu – na podstawie elementów znajdujących się w tworzonym scenariuszu, wyliczane są możliwe do wybrania wartości pól.
- Wyszukiwanie metod – funkcjonalność umożliwia wyszukanie metody po ciągu znaków znajdujących się w jej nazwie.
- Moduły „Info” – są to komponenty, które umożliwiają wyświetlanie metadanych poszczególnych elementów. Zawierają szczegółowe informacje o wybranym bycie.
- Komponent „Uogólnione drzewo scenariusza” – pozwala na prezentacje tworzonego scenariusza w postaci drzewa.
- Po kliknięciu na element drzewa automatycznie otwiera się okno modalne związane z wybranym bytem. Dodatkowo następuje automatyczne przewijane do wybranego elementu na liście, który ponadto jest podświetlony na czerwono. Pozwala to na komfortową edycje scenariuszy.

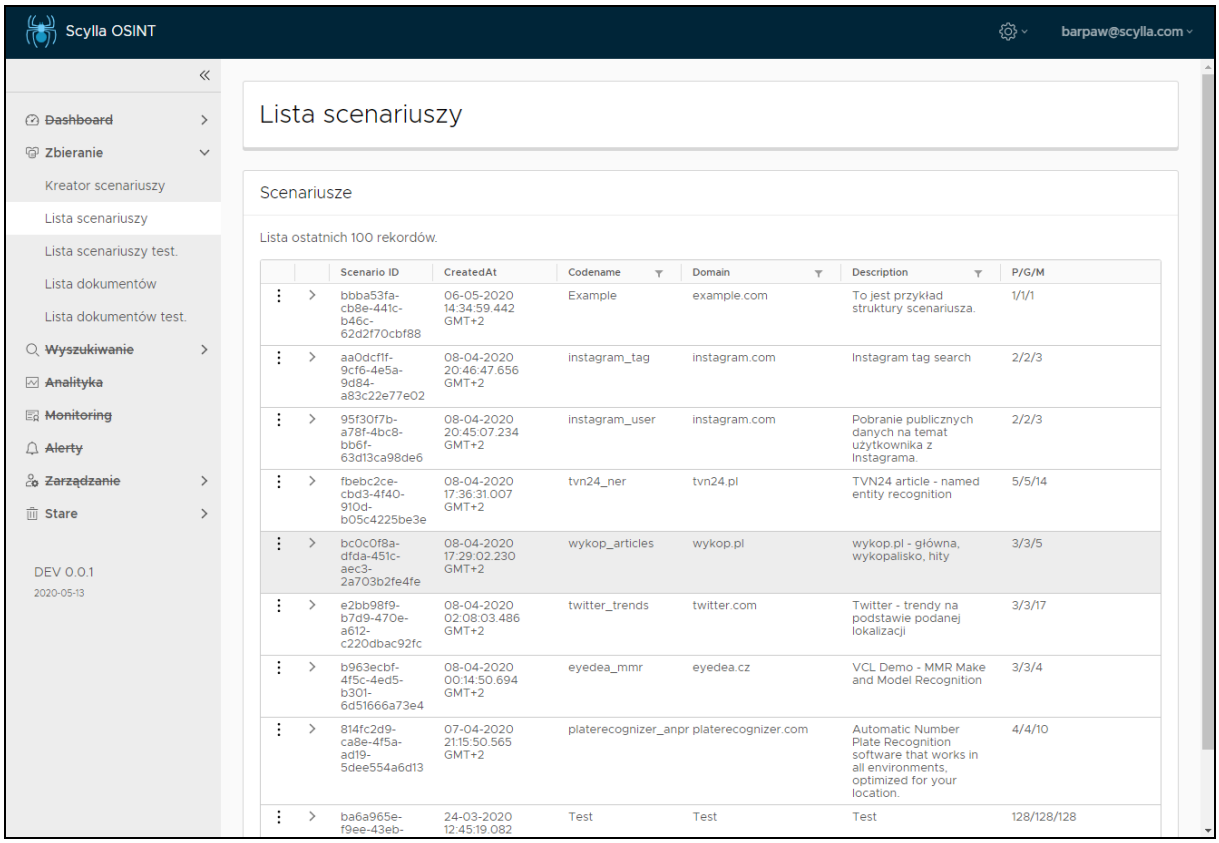
- Klient na podstawie metadanych zdefiniowanych w bibliotece pozwala na dynamiczne tworzenie poprawnych typów elementów formularza. Biblioteka udostępnia informację jakiego typu powinien być element formularza w kliencie. Klient natomiast na podstawie tej informacji dynamicznie wyświetla ten element np. pole dla tekstu lub liczb.
- Rozwojowe komponenty: encje i zmienne, które mogą w przyszłości rozszerzyć funkcjonalność prototypu.
- Komponent „Podgląd JSON" – umożliwia weryfikowanie i sprawdzanie jak docelowo będzie wyglądał scenariusz w postaci obiektu JavaScript.



Rysunek 7. Interfejs okna modalnego służącego do dodawania metod do scenariusza

5.3.2 Listy scenariuszy i dokumentów

Listy scenariuszy i dokumentów są to komponenty odpowiadające za prezentacje danych w postaci tabelarycznej. Pozwalają one na sortowanie, filtrowanie oraz paginacje informacji. Z poziomu listy scenariuszy możemy przejść w tryb podglądu lub edycji danego scenariusza. W widoku podglądu scenariusza prezentowane są komponenty związane z prezentacją: scenariusza w postaci JSON, uogólnionego drzewa, encji i zmiennych. Natomiast z poziomu listy dokumentów możemy podejrzeć strukturę JSON dokumentu oraz przejść do podglądu scenariusza, którego uruchomienie wygenerowało ten dokument. Na rysunku 8 przedstawiono interfejs listy scenariuszy.

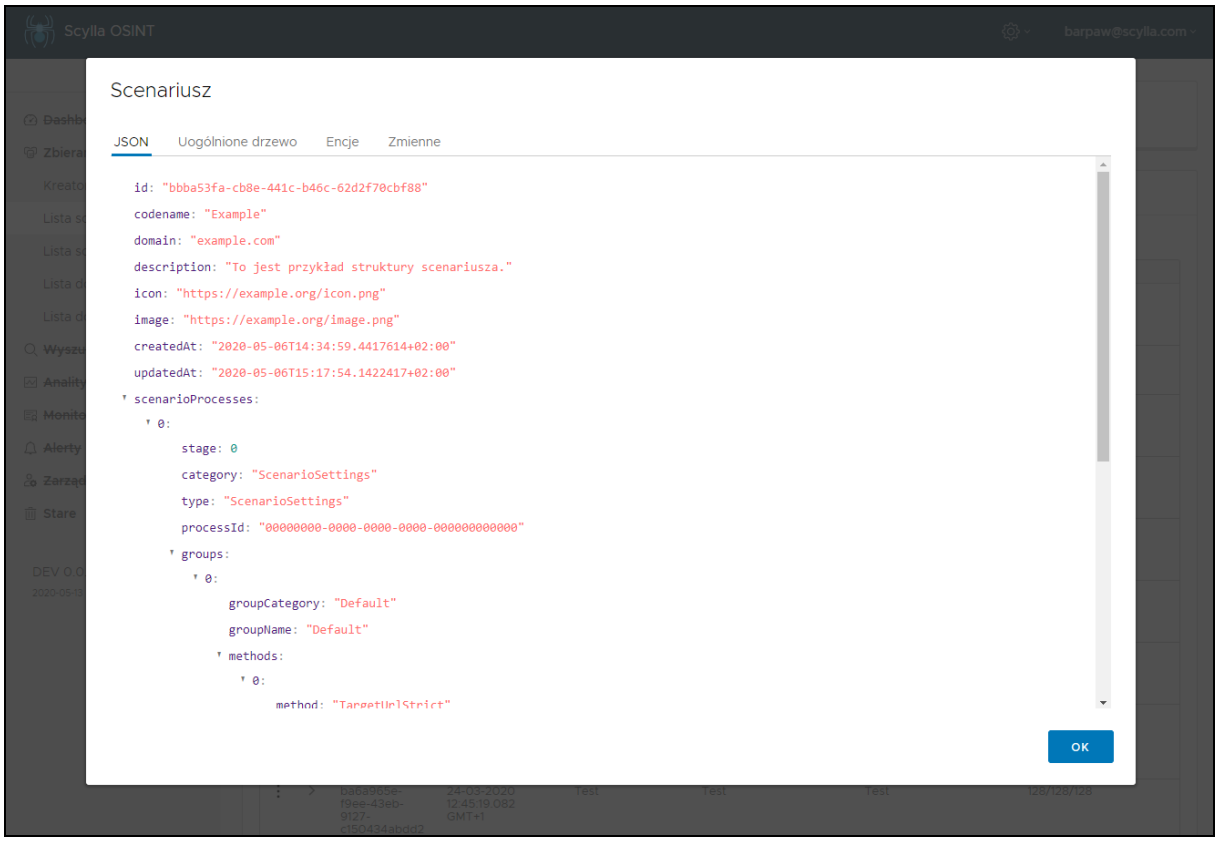


Rysunek 8. Interfejs listy scenariuszy

W tabeli 16 przedstawione zostały komponenty wykorzystane do budowy list scenariuszy i dokumentów. Można zauważyć, że niektóre wymienione w niej komponenty wchodzą również w skład kreatora scenariuszy. Powodem tego jest wydzielenie komponentów, które mogą mieć zastosowanie w różnych miejscach aplikacji. Innymi słowy zamiast powtarzać ten sam kod w kilku miejscach – tworzymy komponent. Zastosowanie takiego podejścia umożliwia utrzymanie czystości kodu i łatwą edycję. Rysunek 9 prezentuje interfejs podglądu scenariusza wybranego z poziomu listy.

Tabela 16. Komponenty wchodzące w skład list scenariuszy i dokumentów

Nazwa komponentu	**Opis**
list-document-test	Komponent główny listy dokumentów testowych
list-scenario	Komponent główny listy scenariuszy
list-scenario-test	Komponent główny listy scenariuszy testowych
modal-scenario	Modal, który umożliwia podgląd scenariusza jako JSON lub w postaci uogólnionego drzewa; dodatkowo z poziomu tego komponentu można zobaczyć encje oraz zmienne zdefiniowane w scenariuszu
scenario-entities	Komponent umożliwiający podgląd encji zdefiniowanych w scenariuszu
scenario-tree	Komponent umożliwiający podgląd scenariusza w postaci uogólnionego drzewa
scenario-variables	Komponent umożliwiający podgląd zmiennych zdefiniowanych w scenariuszu



Rysunek 9. Interfejs poglądu scenariusza z poziomu listy scenariuszy

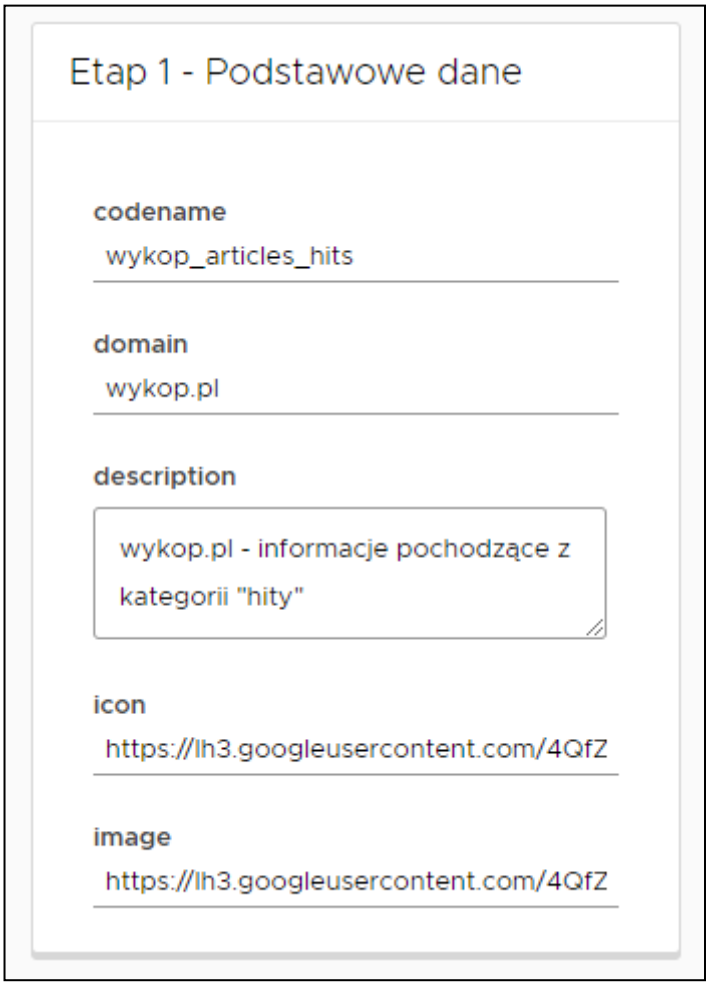
6. Zastosowanie prototypu

W tym rozdziale omówione zostanie tworzenie nowego scenariusza oraz przedstawione zostaną przygotowane scenariusze prototypowe.

6.1. Stworzenie nowego scenariusza

Ten podrozdział objaśnia przebieg procesu tworzenia nowego scenariusza i porusza kwestie związane z jego kreatorem. Tworzony scenariusz będzie miał na celu wyłuskanie informacji ze strony internetowej https://wykop.pl z kategorii „hity”. Struktura przykładowych uzyskanych informacji przedstawiona jest na listingu 8. Wykop jest to strona internetowa oferująca funkcjonalność zbliżoną do agregatora linków tworzonego przez społeczność.

Tworzenie nowego scenariusza należy rozpocząć od przejścia do zasobu kreatora scenariuszy. Pierwszym krokiem jaki należy zrobić jest wypełnienie podstawowych danych na temat tworzonego bytu. Na rysunku 10 przedstawiony został wypełniony formularz dotyczący pierwszego etapu w kreatorze.



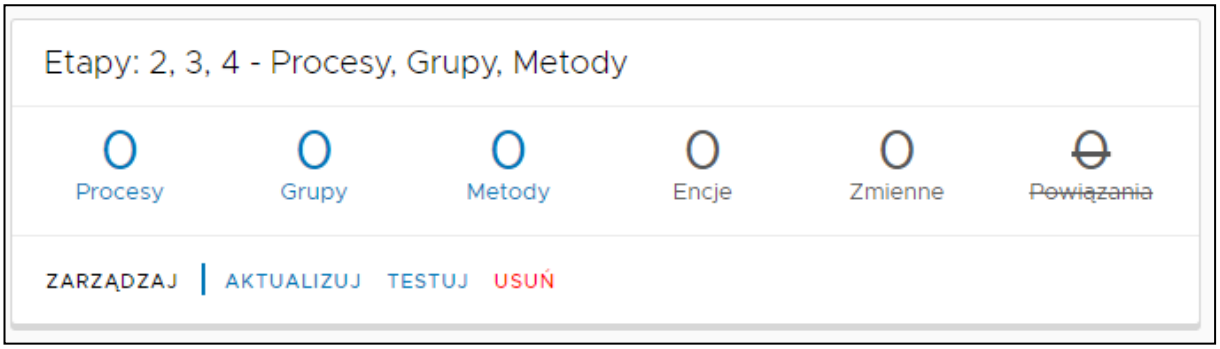
Rysunek 10. Wypełniony formularz dotyczący pierwszego etapu w kreatorze scenariuszy

Omówienie poszczególnych pól etapu pierwszego:

- Codename – unikalna nazwa kodowa scenariusza;
- Domain – domena z którą najsilniej powiązany jest scenariusz;
- Description – opis scenariusza;
- Icon – link do ikony skojarzonej ze scenariuszem;
- Image – link do obrazu skojarzonego ze scenariuszem.

Kolejnym krokiem jest kliknięcie w przycisk „zarządzaj", który otwiera okno modalne. Możemy w nim wybrać procesy dla naszego scenariusza. Na rysunku 11 przedstawiony został panel zawierający statystyki i akcje związane z kreatorem scenariuszy.

W przypadku standardowego tworzenia przepisów - etapy związane z dodawaniem procesów, grup i metod wykonuje się zgodnie z wymienioną kolejnością w kilku iteracjach. Jednakże styl i kolejność nie są narzucone przez system. W celach prezentacyjnych omówione zostaną one z podziałem na grupy bez iteracji.



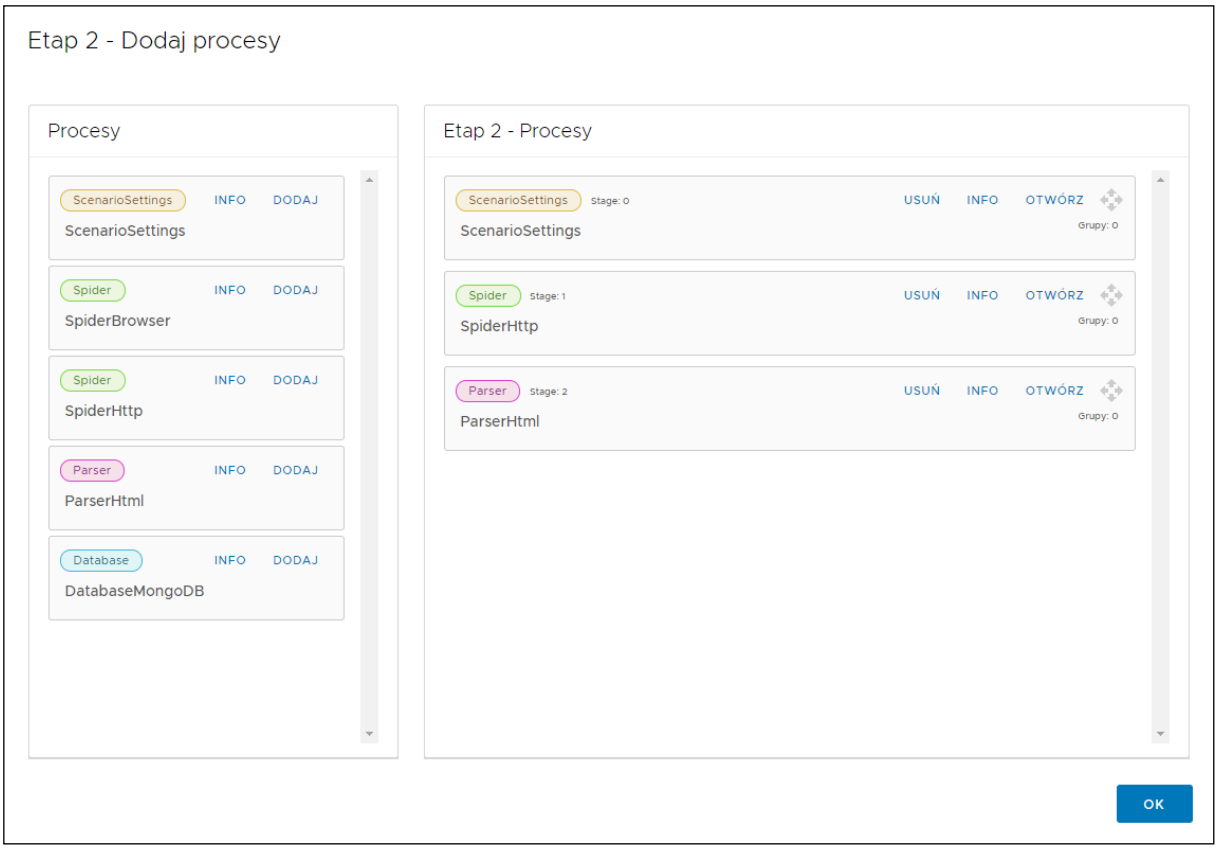
Rysunek 11. Panel statystyk i akcji dotyczących kreatora scenariuszy

Przykładowe procesy jakie zostały zdefiniowane podczas tworzenia prototypu to:

- *ScenarioSettings* – jest to proces, którego metody umożliwiają dodanie do scenariusza pewnych stałych. Stałe to informacje o których wiemy jeszcze przed wykonaniem innych procesów. Przykładowo mogą to być adresy URL lub łańcuchy znaków oznaczające np. login. Proces ten umożliwia przypisanie takich bytów do scenariusza, aby następnie mogły być użyte jako „wartości z kontekstu" z poziomu innych metod.
- *SpiderBrowser* – jest to proces, który skupia w swoim obrębie metody operujące na technologii Selenium. Innymi słowy metody tego procesu odpowiadają za kontrole operacji wykonywanych w skonteneryzowanej przeglądarce internetowej.
- *SpiderHttp* – jest to proces, którego metody umożliwiają operacje związane z żądaniami HTTP.
- *ParserHtml* – jest to proces, który skupia w swoim obrębie metody odpowiadające za parsowanie dokumentów HTML.
- *DatabaseMongoDB* – skupia metody, który pozwalają np. na dodanie dokumentu JSON do wybranej bazy danych i kolekcji.

Tworzony scenariusz będzie wymagał zdefiniowania następujących procesów: *ScenarioSettings*, *SpiderHttp* i *ParserHtml*. W tym celu w otwartym oknie modalnym należy przeciągnąć wybrane elementy z listy procesów i umieścić je na liście docelowej kreatora. Można

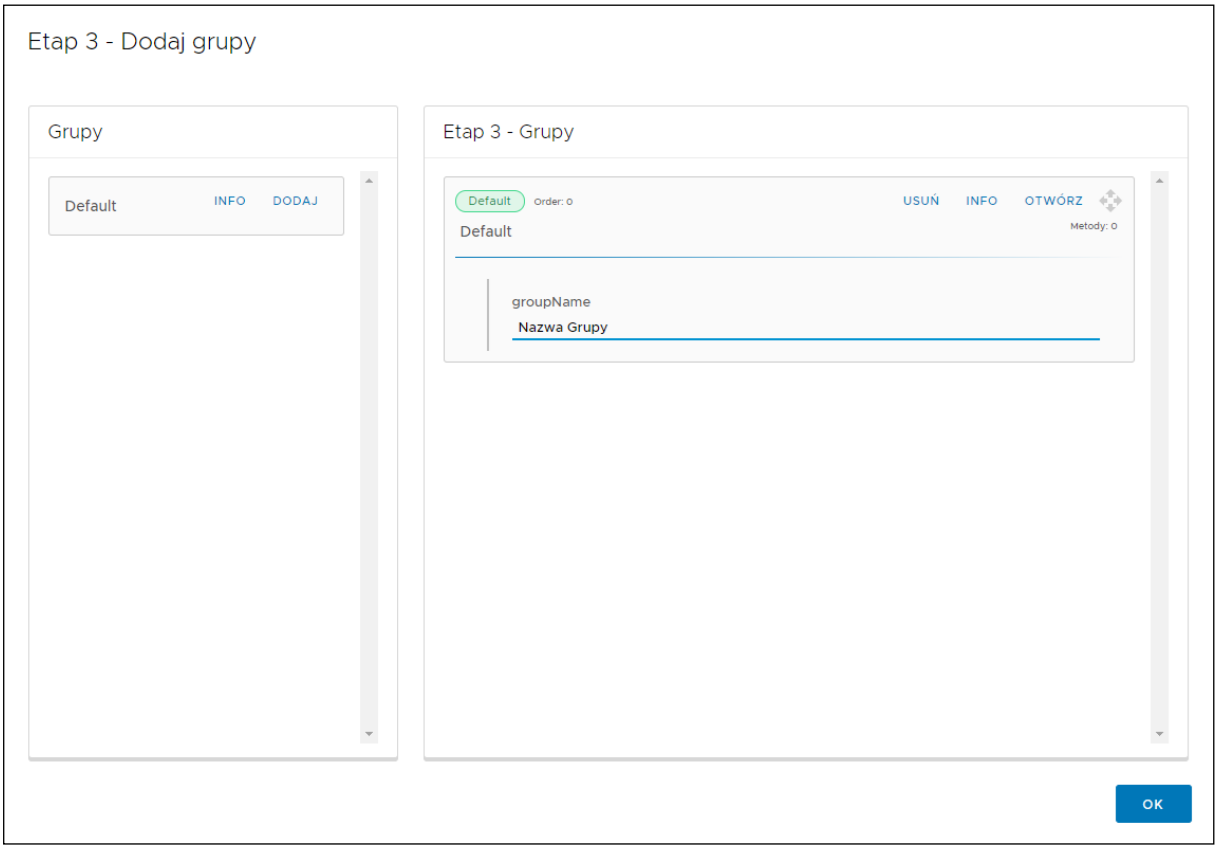
również wykonać tę samą operacje naciskając przycisk „dodaj”. Jeżeli się pomylimy możemy w każdej chwili usunąć niechciany proces. Jeśli zachodzi potrzeba zmiany kolejności elementów na liście to możemy tego dokonać za pomocą kliknięcia i przetrzymania ikony strzałek, a następnie przesunięcia elementu w obrębie listy. Ponadto użytkownik może skorzystać z funkcji „Info”, która otwiera okno z metadanymi dotyczącymi wybranego procesu. Na rysunku 12 przedstawiono okno modalne odpowiadające za dodawanie procesów do scenariusza.



Rysunek 12. Okno modalne umożliwiające dodawanie procesów

Z poziomu listy wybranych procesów istnieje możliwość otwarcia okna modalnego odpowiadającego za dodawanie grup. Grupy są elementem scenariusza, który umożliwia grupowanie metod. Dzięki temu w obrębie procesu możemy zdefiniować kilka indywidualnie nazwanych grup z różnymi metodami. Niesie to za sobą funkcjonalność separacji i wpływa pozytywnie na poruszanie się w obrębie kreatora. W przypadku okna modalnego dodawania grup, również zastosowano mechanizm drag and drop.

Dla docelowo tworzonego scenariusza zastosowano pojedyncze standardowe grupy *„Default”*. Nazwa grupy nie ma większego znaczenia dla przebiegu procesu wykonywania scenariusza. Niesie ona jedynie możliwość nazwania elementów w celach porządkowych i organizacyjnych. Na rysunku 13 przedstawiono okno modalne umożliwiające dodawanie grup.



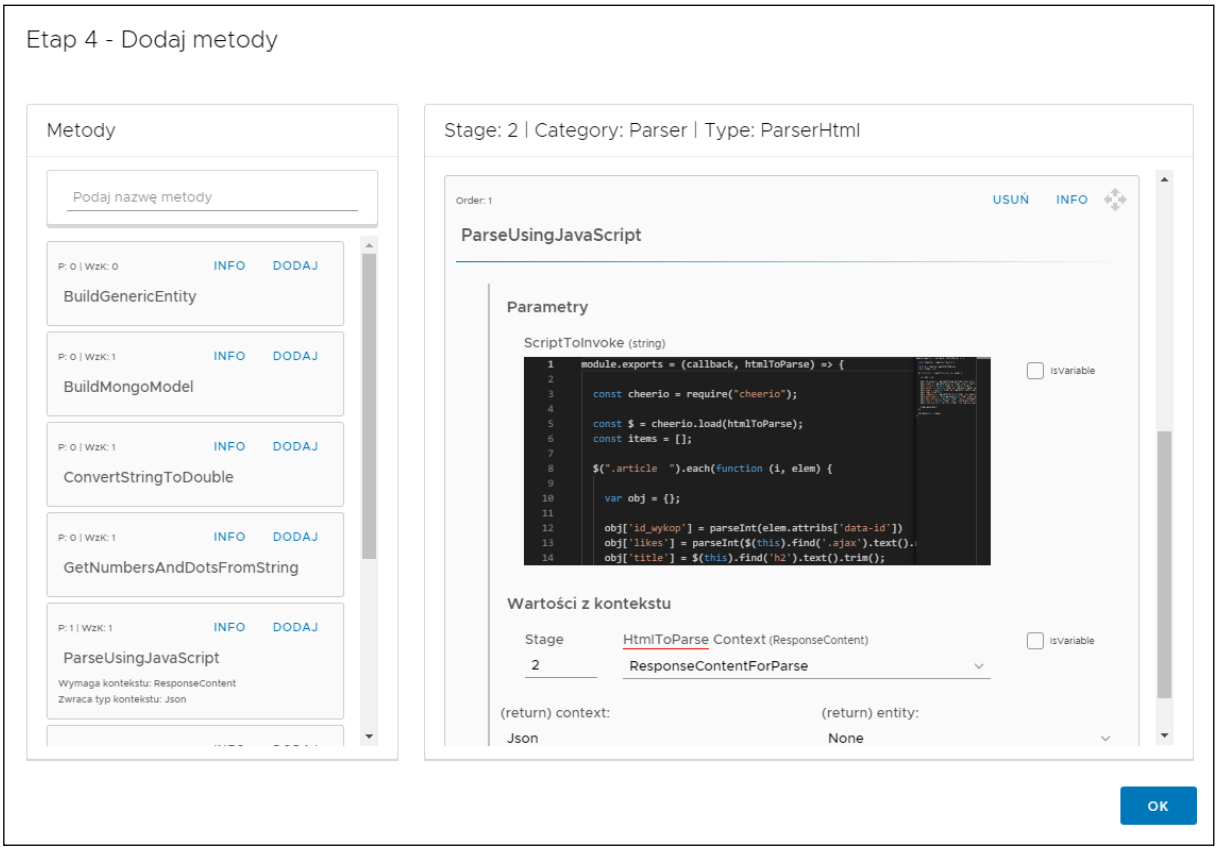
Rysunek 13. Okno modalne umożliwiające dodawanie grup

Najbardziej zaawansowaną składową scenariuszy są metody. Za ich pomocą definiujemy przebieg tego co ma się wykonać. Okno modalne dodawania metod posiada najwięcej funkcjonalności spośród dotychczas omówionych okien. Oprócz standardowych mechanizmów umożliwia również wyszukiwanie metod po nazwie. Metody dodatkowo posiadają zaimplementowane elementy rozwojowe – zmienne i encje. Wybrane parametry lub wartości z kontekstu można oznaczyć jako zmienne. Niesie to w przyszłości możliwość traktowania scenariusza jako pewnego szablonu lub funkcji. Encje pozwalają na oznaczenie typu jaki jest zwracany przez metodę.

W ramach tworzonego przykładowego scenariusza wykorzystano następujące metody:

- *TargetUrlStrict* – umożliwia przypisanie adresu URL do dokumentu.
- *MakeHttpRequest* – umożliwia wykonanie żądania HTTP do zdefiniowanego zasobu (istnieje możliwość wybrania opcji czy chcemy uzyskać odpowiedź skompresowaną czy jawny tekst dokumentu HTML w odpowiedzi). Metoda ta zwraca obiekt, który zawiera informację o żądaniu, odpowiedzi i czasie wykonania.
- *GetContentFromResponse* – umożliwia wyselekcjonowanie jedynie samej odpowiedzi z żądania HTTP.
- *SingleNode* – umożliwia wyselekcjonowanie elementu z dokumentu HTML na podstawie podanej ścieżki selektora XPath.
- *ParseUsingJavaScript* – umożliwia zdefiniowanie skryptu w języku JavaScript, który sparsuje wyselekcjonowany dokument HTML i zwróci obiekt JSON z uzyskanymi informacjami.

Opisane powyżej metody są pewnego rodzaju podstawowym algorytmem, który wykonywany jest w procesie pozyskiwana danych ze stron internetowych. Dzięki stworzonemu prototypowi możliwe jest szybkie i łatwe tworzenie takich rozwiązań. Na rysunku 14 przedstawione zostało okno modalne umożliwiające dodawanie metod.



Rysunek 14. Okno modalne umożliwiające dodawanie metod

Struktura stron internetowych ma to do siebie, że często ulega zmianie. Stanowi to duży problem dla rozwiązań pozyskujących z nich dane. Z tego powodu prototyp aplikacji implementuje metodę, która umożliwia parsowanie dokumentów HTML za pomocą skryptu JavaScript. Zdefiniowanie generycznych metod obsługujących wszystkie przypadki umożliwiające pozyskiwanie danych jest problematyczne. Zarządzanie potencjalną dużą ilością metod w scenariuszu w przypadku zmiany struktury dokumentu HTML na danej stronie internetowej byłoby uciążliwe. Wymaganie znajomości języka programowania znacznie podnosi próg dostępności i zrozumienia obsługi prototypu aplikacji. Rekompensowane jest to poprzez możliwość łatwej adaptacji do zmian w strukturach stron internetowych. Zarządzanie kilkunastoma linijkami kodu jest prostsze i wydajniejsze od zarządzania kilkunastoma metodami w scenariuszu. Listing 7 przedstawia skrypt odpowiedzialny za pozyskanie danych w ramach omawianego przykładowego scenariusza.

Listing 7. Skrypt parsujący dokument HTML w przykładowym scenariuszu

```
1.  module.exports = (callback, htmlToParse) => {
2.
3.    const cheerio = require("cheerio");
4.
5.    const $ = cheerio.load(htmlToParse);
6.    const items = [];
7.
```

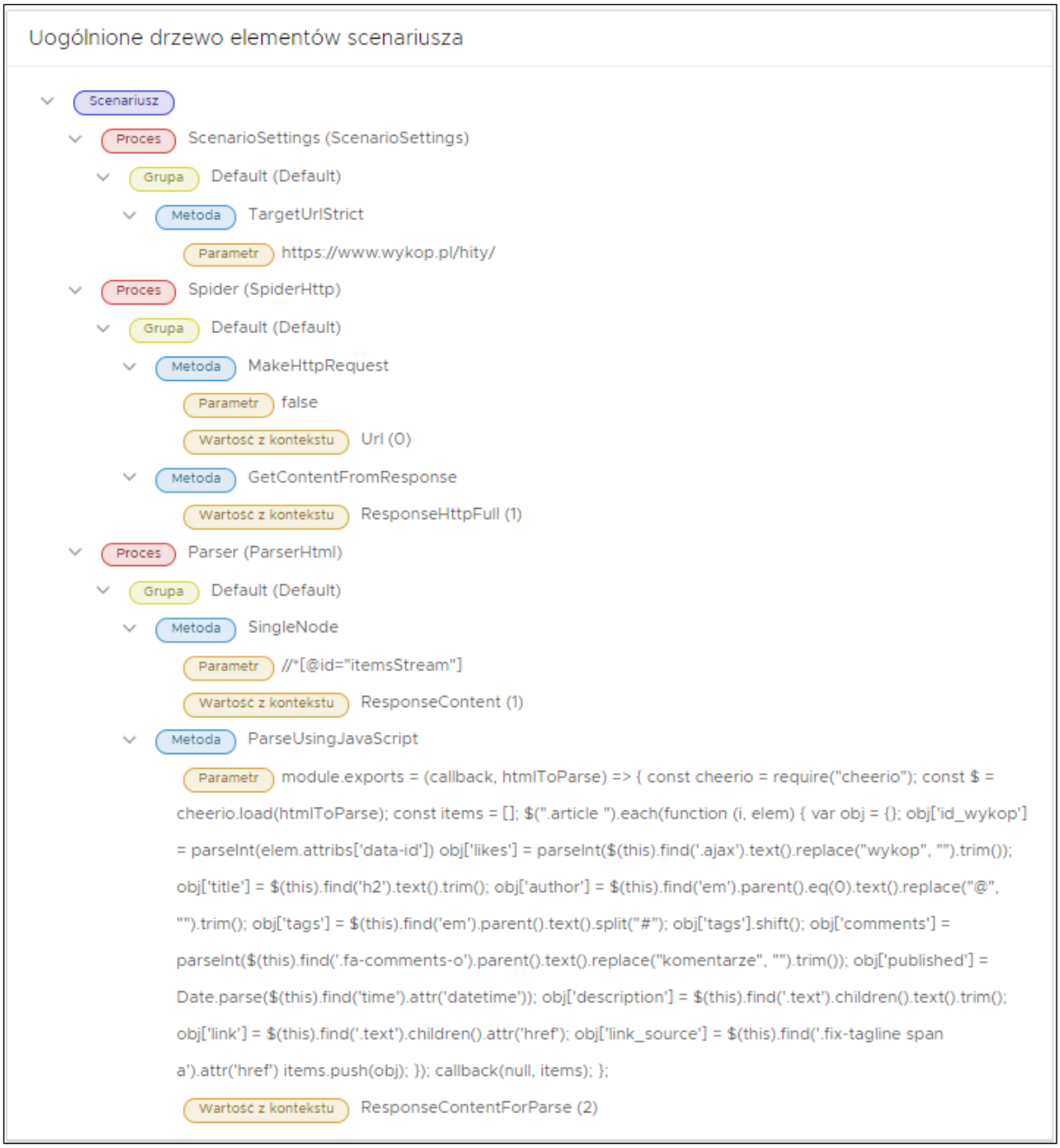
```
8.    $(".article  ").each(function (i, elem) {
9.
10.     var obj = {};
11.
12.     obj['id_wykop'] = parseInt(elem.attribs['data-id'])
13.     obj['likes'] = parseInt($(this).find('.diggbox).text().replace("wykop", "").trim
    ());
14.     obj['title'] = $(this).find('h2').text().trim();
15.     obj['author'] = $(this).find('em').parent().eq(0).text().replace("@", "").trim()
    ;
16.     obj['tags'] = $(this).find('em').parent().text().split("#");
17.     obj['tags'].shift();
18.     obj['comments'] = parseInt($(this).find('.fa-comments-
    o').parent().text().replace("komentarze", "").trim());
19.     obj['published'] = Date.parse($(this).find('time').attr('datetime'));
20.     obj['description'] = $(this).find('.text').children().text().trim();
21.     obj['link'] = $(this).find('.text').children().attr('href');
22.     obj['link_source'] = $(this).find('.fix-tagline span a').attr('href')
23.
24.     items.push(obj);
25.   });
26.
27.   callback(null, items);
28. };
```

Skrypt przedstawiony na listingu 7 działa z pomocą biblioteki cheerio [24]. Biblioteka ta umożliwia parsowanie dokumentów HTML z wykorzystaniem syntaktyki zbliżonej do składni bardzo popularnej niegdyś biblioteki jQuery. Technologia stojąca za implementacją omawianej metody umożliwia wykorzystywanie wszystkich dobrodziejstw środowiska JavaScriptowego. Omawiany skrypt ma za zadanie przetworzyć dokument HTML i zwrócić listę wyłuskanych obiektów. Przykładowy uzyskany obiekt został przedstawiony na listingu 8.

Listing 8. Przykładowy obiekt zwracany przez skrypt

```
1.  {
2.    "id_wykop": 5495771,
3.    "likes": 5474,
4.    "title": "Opłata reprograficzna. \"Smartfony nawet o 300 zł droższe\"",
5.    "author": "orangeduke",
6.    "tags": [
7.      "polska",
8.      "smartfon",
9.      "podatki",
10.     "zaiks",
11.     "oplaty",
12.     "prawaautorskie"
13.   ],
14.   "comments": 303,
15.   "published": 1589216461000,
16.   "description": "ZAiKS to prywatna firma, która w oparciu o przepisy prawne (…),
17.   "link": "https://www.wykop.pl/link/5495771/oplata-reprograficzna-smartfony-nawet-
    o-300-zl-drozsze/",
18.   "link_source": "https://www.money.pl/gospodarka/oplata-reprograficzna-smartfony-
    nawet-300-zl-drozsze-6508290766223489v.html"
19. }
```

Na rysunku 15 przedstawiony został omawiany przykładowy scenariusz w postaci uogólnionego drzewa. Zaimplementowane mechanizmy pozwalają na to, żeby z poziomu tego widoku również przejść do edycji wybranego elementu. Aby to zrobić wystarczy kliknąć daną etykietę.



Rysunek 15. Stworzony scenariusz w postaci uogólnionego drzewa scenariusza

Uruchomienie przetwarzania scenariusza odbywa się poprzez naciśnięcie przycisku „Testuj”, który znajduje się w panelu statystyk i akcji. Panel ten został przedstawiony na rysunku 11. Z poziomu tego panelu możemy również zapisać, zaktualizować lub usunąć scenariusz. Wykonanie operacji testowania powoduje, że scenariusz jest procesowany po stronie API z pomocą stworzonej biblioteki. Efektem wykonania scenariusza jest dokument JSON, którego struktura została przedstawiona na listingu 2. Na rysunku 16 przedstawiony został wynik przetworzenia testowego scenariusza.

Rysunek 16. Wynik przetworzenia testowego scenariusza

6.2. Omówienie prototypowych scenariuszy

W ramach prac nad prototypem aplikacji powstało kilka scenariuszy. Zostaną one przedstawione w tym podrozdziale w celu przybliżenia funkcjonalności i możliwości stworzonej aplikacji. W tabeli 17 zawarto informacje dotyczące prototypowych scenariuszy.

Tabela 17. Zestawienie prototypowych scenariuszy

Nazwa kodowa	Powiązana domena	Czy używa przeglądarki	Ilość zdefiniowanych procesów, grup i metod	Krótki opis scenariusza
platerecognizer_anpr	platerecognizer.com	Tak	4/4/10	Rozpoznawanie numerów tablic rejestracyjnych na podstawie zdjęcia.
eyedea_mmr	eyedea.cz	Nie	3/3/4	Rozpoznawanie numerów tablic rejestracyjnych, koloru, marki oraz modelu pojazdu na podstawie zdjęcia.

Nazwa kodowa	Powiązana domena	Czy używa przeglądarki	Ilość zdefiniowanych procesów, grup i metod	Krótki opis scenariusza
twitter_trends	twitter.com	Tak	3/3/17	Pozyskanie trendów z Twittera z uwzględnieniem kraju pochodzenia.
tvn24_ner	tvn24.pl	Tak	5/5/14	Wyłuskanie nazwanych encji z danego artykułu znajdującego się na portalu informacyjnym Tvn24.
instagram_user	instagram.com	Nie	2/2/3	Pobranie informacji z profilu użytkownika Instagram na podstawie nazwy konta.
instagram_tag	instagram.com	Nie	2/2/3	Pobranie informacji powiązanych z danym tagiem z portalu Instagram.
wykop_articles	wykop.pl	Nie	3/3/5	Pobranie artykułów opublikowanych na danej stronie w serwisie Wykop.

6.2.1 Scenariusz platerecognizer_anpr

Scenariusz o nazwie kodowej „platerecognizer_anpr" umożliwia rozpoznawanie numerów tablic rejestracyjnych pojazdów znajdujących się na danym zdjęciu. W tym celu wykorzystywane jest testowe API https://platerecognizer.com. Strona ta udostępnia formularz poprzez który można wgrywać zdjęcia. Algorytm po stronie aplikacji ma za zadanie znaleźć potencjalne numery rejestracyjne pojazdów znajdujących się na przesłanym zdjęciu, a następnie wyświetlić je na stronie internetowej. Omawiany scenariusz automatyzuje ten proces i zwraca obiekt JSON przedstawiony na listingu 9.

Uogólniony algorytm wykonania tego scenariusza wygląda następująco:

1. Zdefiniuj adres URL do zasobu ze zdjęciem pojazdu.
2. Otwórz przeglądarkę w kontenerze, przejdź do zasobu i pobierz zdjęcie.
3. Przejdź do strony https://platerecognizer.com umożliwiającej testowanie API aplikacji.
4. Zasymuluj wybranie pobranego zdjęcia i prześlij je do procesowania przez aplikacje.
5. Pobierz dokument HTML pochodzący z odpowiedzi serwera.
6. Sparsuj go i zwróć informacje o potencjalnie znalezionych numerach rejestracyjnych pojazdów w postaci JSON.
7. Zapisz zwrócony dokument w bazie danych.

Listing 9. Przykładowy obiekt zwracany po wykonaniu scenariusza platerecognizer_anpr

```
{
  "licensePlate": [
    {
      "value": "ga0348w",
      "confidence": "89.90%"
    }
  ],
  "candidates": [
    {
      "value": "ga0348w",
      "confidence": "89.90%"
    },
    {
      "value": "gao348w",
      "confidence": "89.80%"
    },
    {
      "value": "ga034bw",
      "confidence": "77.70%"
    },
    {
      "value": "gao34bw",
      "confidence": "77.60%"
    }
  ]
}
```

6.2.2 Scenariusz eyedea_mmr

Scenariusz o nazwie kodowej „eyedea_mmr" umożliwia uzyskanie szczegółowych informacji na temat pojazdu znajdującego się na danym zdjęciu. Oprócz numerów znajdujących się na tablicy rejestracyjnej algorytm rozpoznaje kolor, model i markę pojazdu. Scenariusz wykorzystuje testowe API serwisu https://eyedea.cz i nie wymaga uruchamiania przeglądarki. Na listingu 10 został przedstawiony zwracany przez omawianą funkcję obiekt.

Uogólniony algorytm wykonania tego scenariusza wygląda następująco:

1. Zdefiniuj adres URL do zasobu ze zdjęciem pojazdu, który następnie będzie podstawiony jako parametr zapytania do API https://eyedea.cz.
2. Wykonaj zapytanie.
3. Z odpowiedzi API uzyskaj obiekt JSON.
4. Zapisz zwrócony obiekt do bazy danych.

Listing 10. Przykładowy obiekt zwracany po wykonaniu scenariusza eyedea_mmr

```
{
  "status": "success",
  "photos": [
    {
      "width": 1728,
      "height": 1296,
      "module_version": 33,
      "tags": [
```

```
        {
          "top_left": {
            "x": 562.281982421875,
            "y": 718.0570678710938
          },
          "bottom_right": {
            "x": 959.5261840820312,
            "y": 846.1248779296875
          },
          "confidence": 54.060852,
          "category": "CAR",
          "category_score": 0.9999996,
          "make": "BMW",
          "make_score": 0.9999979,
          "model": "3",
          "model_score": 0.9999977,
          "color": "BLACK",
          "color_score": 0.99945027,
          "generation": "F30 (2011,2015)",
          "generation_score": 0.9999978,
          "variation": "",
          "variation_score": "NaN",
          "lp_text_content": "RLE28661",
          "lp_text_score": 0.99894184,
          "ilpc": "PL",
          "view": "frontal",
          "auto_detection": 1
        }
      ]
    }
  ]
}
```


6.2.3 Scenariusz twitter_trends

Scenariusz o nazwie kodowej „twitter_trends" jest jednym z najbardziej rozbudowanych scenariuszy prototypowych. Umożliwia on uzyskanie aktualnych trendów, czyli popularnych hashtagów lub zwrotów używanych w danym kraju na Twitterze. Zaawansowanie tego przepisu objawia się tym, iż wykonuje on takie operacje jak logowanie i nawigacja po elementach strony internetowej. Przykładowy obiekt zwracany po wykonaniu scenariusza został przedstawiony na listingu 11.

Uogólniony algorytm wykonania tego scenariusza wygląda następująco:

1. Należy zdefiniować login i hasło konta na portalu https://twitter.com oraz nazwę kraju lub województwa. Dane te będą wykorzystywane do logowania oraz zmiany ustawień w profilu użytkownika.
2. Uruchom przeglądarkę w kontenerze i przejdź do strony logowania w serwisie Twitter.
3. Zaloguj się podając zdefiniowane dane.
4. Przejdź do zasobu umożliwiającego zmianę kraju dla pokazywanych trendów.
5. Jako kraj, którego trendy chcemy widzieć ustaw miejsce zdefiniowane w pierwszym kroku.
6. Przejdź do zasobu z trendami i pobierz dokument HTML.
7. Sparsuj dokument HTML i zwróć obiekt JSON.

Listing 11. Przykładowy obiekt zwracany po wykonaniu scenariusza twitter_trends

```
[
  {
    "lp": 1,
    "trend": "#pizgaczhell",
    "tweets": "3 869"
  },
  {
    "lp": 2,
    "trend": "#ThisIsUs1D2020",
    "tweets": "1 524"
  },
  {
    "lp": 3,
    "trend": "#wtylewizji",
    "tweets": "-"
  },
  {
    "lp": 4,
    "trend": "#6ix9ineisoverparty",
    "tweets": "-"
  },
  {
    "lp": 5,
    "trend": "#RainOnMe",
    "tweets": "47,4 tys."
  },
  {...}
]
```

6.2.4 Scenariusz tvn24_ner

Innym przykładem dość skomplikowanego przepisu jest scenariusz o nazwie kodowej „tvn24_ner". Umożliwia on pozyskanie nazw własnych znajdujących się w artykułach publikowanych w serwisie informacyjnym https://tvn24.pl. W tym celu używane jest testowe API https://ws.clarin-pl.eu, które umożliwia rozpoznawanie nazw własnych i wyrażeń temporalnych znajdujących się w tekście. Przykładowy obiekt zwracany po wykonaniu omawianego scenariusza został przedstawiony na listingu 12.

Uogólniony algorytm wykonania tego scenariusza wygląda następująco:

1. Zdefiniuj adres URL do artykułu ze strony https://tvn24.pl.
2. Wyślij żądanie do tego zasobu.
3. Otrzymaną odpowiedź sparsuj, aby uzyskać samą treść artykułu.
4. Otwórz przeglądarkę w kontenerze i przejdź do zasobu z testowym API umożliwiającym pozyskiwanie nazwanych encji z tekstu.
5. Na otwartej stronie internetowej wklej do formularza tekst artykułu, a następnie wywołaj akcję przetwarzania.
6. Po przetworzeniu artykułu odpowiedź w postaci HTML sparsuj i zwróć obiekt.

Listing 12. Przykładowy obiekt zwracany po wykonaniu scenariusza tvn24_ner

```
{
  "nam_loc": [
    "Polski",
    "Polska",
    "Polski",
    "Europie",
    "Lombardii",
    "Polsce",
    "Europie",
    "Polska"
  ],
  "nam_liv": [
    "Maria Miko",
    "Luciana Matarese",
    "Maria Miko",
    "Giuseppe Conte",
    "Giuseppe Conte",
    "Lucia Azzolina",
    "Luciana Matarese",
    "Maria Miko"
  ],
  "nam_org": [
    "Ministerstwa Edukacji"
  ],
  "nam_adj": [],
  "nam_eve": [
    "Wenecji Euganejskiej"
  ],
  "nam_fac": [],
  "nam_num": [],
  "nam_oth": [
    "internet",
    "euro"
  ]
}
```

6.2.5 Scenariusze instagram_user i instagram_tag

Pozostałe scenariusze prototypowe oznaczone są nazwami kodowymi „instagram_user” oraz „instagram_tag”. Wykorzystują one prostą sztuczkę związaną z pozyskiwaniem danych w postaci JSON z portalu https://instagram.com. Polega ona na tym, że na końcu standardowego zapytania do tego serwisu dodajemy ciąg znaków „?__a=1”. Zabieg ten sprawia, że odpowiedzi serwera prezentowane są w postaci obiektu JavaScriptowego. Przepis oznaczony nazwą kodową „instagram_user” polega na tym, iż podajemy nazwę użytkownika portalu https://instagram.com, a w zamian dostajemy informacje profilowe w postaci JSON. To samo tyczy się scenariusza z nazwą kodową „instagram_tag”. W tym przypadku podajemy interesującą nas nazwę tagu, a w odpowiedzi uzyskamy informacje o nim w postaci JSON.

7. Podsumowanie

Rozdział ten stanowi próbę podsumowania zaproponowanego rozwiązania. Przedstawia jego wady oraz zalety. Poruszone zostały również kwestie związane z kierunkiem rozwoju prototypu i wnioski końcowe.

7.1. Wady i zalety rozwiązania

Prace nad stworzonym prototypem pozwoliły na ujawnienie niektórych z jego wad i zalet. Zostały one przedstawione poniżej.

Wady:

- Prototyp został zaprojektowany jako usługa w architekturze monolitycznej.
- Jeżeli brakuje jakiejś metody potrzebnej do wykonania w scenariuszu, to trzeba ją zaimplementować.
- W procesie tworzenia prototypu skupiono się na jego funkcjonalnościach i wszechstronności. Może to rzucać cień na wydajność systemu.
- Nie jest to kompletny system OSINT-owy. W ramach prac powstał moduł pozyskiwania danych.
- Do obsługi systemu potrzebna jest wiedza na temat znaczników i struktur dotyczących dokumentów HTML.
- W zależności od wybranych metod scenariusza może być potrzebna wiedza na temat działania różnych technologii np. Selenium lub język JavaScript.

Zalety:

- Mimo, że usługa została stworzona jako monolit, to przystosowana jest do tego, aby w przyszłości móc ją rozproszyć.
- Raz zaimplementowana metoda, może być używana wiele razy przez różne scenariusze.
- Zaprojektowane abstrakcje umożliwiające szybkie i spójne dodawanie nowych metod do biblioteki.
- Prototyp jest otwarty na rozwój. Zastosowane kluczowe mechanizmy nie wiążą ściśle poszczególnych elementów systemu.
- Opracowanie generycznych mechanizmów i modeli związanych z pozyskiwaniem danych ze stron internetowych.
- Interfejs klienta aplikacji został zaprojektowany w taki sposób, aby był jak najbardziej intuicyjny dla użytkownika.
- Okna modalne zawierające informacje stanowiące pomoc dla użytkownika, znacząco mogą ułatwić pracę z systemem.

7.2. Kierunki rozwoju prototypu

Prace badawcze i implementacyjne nad prototypem aplikacji pozwoliły ujawnić pewne problemy, i wyzwania stojące przez pozyskiwaniem danych ze stron internetowych. Dodatkowo stworzenie prototypowego systemu umożliwiło dostrzeżenie różnych możliwości oraz kierunków rozwoju. Nabyta wiedza i doświadczenie sugeruje następujące kierunku rozwoju:

- Stworzenia bytu „Fabuła" – który w hierarchii byłby ponad scenariuszami. W jego ramach można by definiować m.in. akcje oraz przepływy informacji pomiędzy różnymi scenariuszami.
- Zmiana interfejsu użytkownika. Zamiast list z mechanizmami drag and drop można zastosować interfejs z obszarem projektowym taki jak występuje np. w oprogramowaniu „*SQL Server Integration Services*".
- Rozproszenie usług prototypu. Zastosowanie kolejki zdarzeń.
- Obsługa błędów w scenariuszach np. ponowne wykonywanie nieudanego procesu.
- Implementacja walidacji parametrów metod.
- Stworzenie modułów: wyszukiwania, harmonogramowania, monitoringu, alarmów, zarządzania oraz kreatora dashboardów.
- Implementacja kolejnych generycznych metod scenariusza m.in. takich, które umożliwiałyby wykonywanie kodu lub sterowanie innymi aplikacjami.
- Dodanie obsługi proxy.
- Zabezpieczenie platformy i pozyskanych danych przed cyberprzestępcami.

7.3. Wnioski końcowe

W procesie tworzenia prototypu udało się wypracować i zbadać mechanizmy oraz modele pozwalające na generyczne podejście do tematu pozyskiwania danych ze stron internetowych. Dzięki zdobytej wiedzy i doświadczeniu możliwa jest kontynuacja prac nad systemem wspomagającym biały wywiad. Utworzony moduł zawierający kreator scenariuszy może konkurować z oprogramowaniem dostępnym na rynku. Zastosowane mechanizmy abstrakcji po stronie biblioteki jak i mechanizmy związane z interfejsem użytkownika spełniły swoje zadanie, i częściowo będą mogły być wykorzystywane w kolejnych wersjach systemu.

Uważam, że stworzone narzędzie po dodaniu omówionych brakujących funkcjonalności będzie w pełni odpowiadało na problem postawiony w temacie pracy i dostarczy system umożliwiający pozyskiwanie danych w ramach białego wywiadu.

Bibliografia

[1]. Grepsr.com, https://grepsr.com/
[Dostęp: 31.12.2019]

[2]. Octoparse.com, https://octoparse.com/
[Dostęp: 31.12.2019]

[3]. Osintframework.com, https://osintframework.com/
[Dostęp: 31.12.2019]

[4]. Spiderfoot.net, https://spiderfoot.net/
[Dostęp: 31.12.2019]

[5]. Maltego.com, https://maltego.com/
[Dostęp: 31.12.2019]

[6]. Microsoft.com, https://docs.microsoft.com/pl-pl/dotnet/core/introduction
[Dostęp: 18.04.2020]

[7]. Microsoft.com, https://docs.microsoft.com/pl-pl/dotnet/core/about
[Dostęp: 18.04.2020]

[8]. N. Rozentals., Język TypeScript. Tajniki kodu. Wydanie II, Helion 2017, ISBN 978-83-283-3641-4, 9788328336414

[9]. Y.Fain., A. Moiseev., Angular. Programowanie z użyciem języka TypeScript. Wydanie II, Helion 2019, ISBN 978-83-283-5666-5, 9788328356665

[10]. Clarity.design, https://clarity.design/documentation/get-started
[Dostęp: 19.04.2020]

[11]. Github.com , https://github.com/aerokube/selenoid
[Dostęp: 19.04.2020]

[12]. J. Dickey., Nowoczesne aplikacje internetowe. MongoDB, Express, AngularJS, Node.js, Helion 2016, ISBN 978-83-283-1758-1, 9788328317581

[13]. Docker.com, https://www.docker.com/resources/what-container
[Dostęp: 19.04.2020]

[14]. Docker.com, https://docs.docker.com/get-started/overview/
[Dostęp: 19.04.2020]

[15]. Visualstudio.com, https://code.visualstudio.com/docs
[Dostęp: 22.04.2020]

[16]. Electronjs.org, https://www.electronjs.org/
[Dostęp: 22.04.2020]

[17]. Git-scm.com, https://git-scm.com/docs/git
[Dostęp: 22.04.2020]

[18]. Gitlab.com, https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html
[Dostęp: 22.04.2020]

[19]. Json.org, https://www.json.org/json-en.html
[Dostęp: 28.04.2020]

[20]. Cleancoder.com,
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
[Dostęp: 28.04.2020]

[21]. Github.io, https://convey-stack.github.io/
[Dostęp: 01.05.2020]

[22]. Martinfowler.com, https://martinfowler.com/bliki/CQRS.html
[Dostęp: 01.05.2020]

[23]. Github.io, https://convey-stack.github.io/documentation/Web-API/
[Dostęp: 04.05.2020]

[24]. Github.com, https://github.com/cheeriojs/cheerio
[Dostęp: 06.05.2020]

Spis listingów

Spis rysunków

Spis tabel