**Wstęp**

Aplikacja webowa *WebsiteMonitor*, będąca tematem tej pracy, jest rozwiązaniem, które zapewnia nieustanne i w pełni automatyczne monitorowanie zmian pojawiających się na stronach internetowych, całych lub na wybranych ich fragmentach. Aplikacja będzie informować o nich w wybranym przez użytkownika czasie, np. raz w tygodniu, raz dziennie lub w czasie rzeczywistym - gdy tylko nowe treści się pojawią. Informacja będzie przekazywana za pomocą różnych środków komunikacji: mail, powiadomienia w przeglądarce lub za pomocą dynamicznie wyświetlanych komunikatów w aplikacji, a historia zmian będzie prezentowana w czytelny sposób, wraz z możliwością wygenerowania pliku pdf.

Zakres pracy obejmuje parsowanie oraz porównywanie plików tworzących strony internetowe, a także prezentację wykrytych zmian. Działanie aplikacji polega przede wszystkim na cyklicznym pobieraniu pliku *html* z danej strony internetowej, jednak nie tak, jak robi to polecenie *curl* lub *wget* w systemie operacyjnym *Linux*, ponieważ obecnie dużo stron ładuje dodatkowe treści do plików *html* dynamicznie, za pomocą skryptów napisanych w języku *javascript*. Oznacza to, że nie wszystkie treści są umieszczane w pliku *html* od razu, część z nich może być na przykład pobierana z innych źródeł lub na bieżąco wyliczana. Serwer aplikacji będzie więc imitował działająca przeglądarkę, powodując rozpoczęcie działania zawartych na stronie skryptów, a następnie pobierze już gotowy plik *html* z użytymi w nim innymi plikami jak zdjęcia, czy pliki stylujące *css*. Następnie wyświetli użytkownikowi pobraną stronę oraz doda do niego odpowiedni skrypt umożliwiający łatwe, na przykład za pomocą podświetlania, wybranie konkretnych jej elementów do obserwacji. W wybranym przez użytkownika czasie i za pomocą preferowanych przez niego środków poinformuje go o zmianie, a także umożliwi mu łatwe przejrzenie i udostępnienie innym historii zmian na stronie.

W sieci jest ponad 1,7 mld\* stron internetowych. Nie przypominają one już stron z początkowych lat XXI wieku, są bardziej złożone, nie składają się już z samych pików *html*, ale także wielu skryptów, czy bibliotek pobieranych za pomocą *CDN (content delivery network)*. Sprawia to, że strony są bardziej dynamiczne. Treści wyświetlane na nich są często pobierane z różnych źródeł, co oznacza, że zmieniają się bardzo często. Śledzenie tych zmian lub pojawiania się na stronach konkretnych treści, wymaga częstego ich odwiedzania oraz odświeżania okna przeglądarki wraz z przeładowaniem zapamiętanych przez nią plików w pamięci podręcznej. Jest to zadanie żmudne, przy którym niezbędna jest spostrzegawczość i dobra pamięć, aby dostrzec występujące zmiany.

Biorąc pod uwagę dynamikę stron internetowych oraz ich ogromną liczbę, można założyć, że zwykły użytkownik Internetu dość często spotyka się z potrzebą uzyskania natychmiastowego komunikatu o zmianie na stronie. Może to nastąpić na przykład w sytuacjach :

* konkurencji z innymi w wyścigu o uzyskanie najlepszych biletów na wydarzenie kulturalne, które właśnie zostały wystawione do sprzedaży
* uzyskania produktu, który właśnie został przeceniony
* niezwłocznego uzyskania informacji o wyniku egzaminu albo rozstrzygnięciu przetargu
* nowego wpisu na ulubionym blogu, czy nowym artykule na ulubionej stronie popularnonaukowej

Zadania, które aplikacja może pomóc realizować nie ograniczają się jedynie do zadań prywatnych, znajdzie ona również zastosowanie w zadaniach biznesowych, na przykład :

* zmiany na stronie konkurencji, która właśnie wypuściła nowy produkt
* nowe zmiany prawne na stronie rządowej, mające wpływ na działanie biznesu
* nowe oferty pracy
* właściciel przedsiębiorstwa może śledzić fora internetowe, w oczekiwaniu aż nazwa jego firmy zostanie wspomniana i dzięki temu natychmiast na taki wpis odpowiedzieć

Również właściciele strony internetowych, którzy korzystaliby z *WebsiteMonitor* mogliby odnieść korzyści, związane głównie z poprawą bezpieczeństwa ich stron. Fakt, że ich strona jest nieustannie monitorowana oraz, że w przypadku wystąpienia na niej jakiekolwiek zmiany zostaną natychmiast o tym fakcie poinformowani jest tutaj kluczowy. Może pomóc to, we wczesnym wykryciu ataku na stronę, przede wszystkim w atakach biorących na cel nie serwer, lecz klienta, pobierającego kod strony internetowej. Po takim ataku dostaje on stronę z niewidocznymi pod względem wyglądu strony zmianami, zawartymi często tylko w kodzie strony. Na przykład przy ataku *XSS* (*cross-site scripting*), polegającym na wstrzyknięciu złośliwego kodu JavaScript do plików pobieranych przez odwiedzającego stronę lub przy ataku *DNS-spoofing*, w którym zaatakowany serwer DNS będzie przekierowywał klientów na stronę atakującego, pomimo wprowadzenia w przeglądarce poprawnej nazwy domeny. Każdy tego typu ataku, będzie łatwiejszy do wykrycia, jeśli właściciel strony dostanie natychmiast powiadomienie o zmianie na stronie, której sam nie wykonał. Będzie to dla niego wtedy jasny sygnał, że na stronie dzieje się coś niepokojącego i należy się temu przyjrzeć.

Także w przypadkach stron, w których treści mogą być dodawane przez użytkowników (np. fora internetowe), administrator może takie treści na bieżąco moderować np. sprawdzać czy nie łamią one żadnego punktu regulaminu lub na nie niemal natychmiast odpowiadać, co w obecnych czasach może być dużą korzyścią dla popularności strony lub biznesu, który ona reprezentuje.

Celem pracy jest zaprojektowanie i implementacja aplikacji webowej umożliwiającej śledzenie zachodzenia jakichkolwiek zmian na stronie internetowej lub pojawiania się na niej konkretnych słów kluczowych. Aplikacja ta powinna mieć następujące funkcjonalności:

* rejestracja wraz z utworzeniem konta, które będzie umożliwiać dostęp do wszystkich funkcji aplikacji
* cykliczne i w pełni automatyczne monitorowanie zmian zachodzących na podanej przez użytkownika stronie internetowej, w określonym przedziale oraz interwale czasowym
* monitorowanie całej strony lub jedynie wybranych jej fragmentów
* wraz z utworzonym monitorowaniem powstanie tzw. ‘pokoju obserwacji’, z którym klient nawiąże komunikację za pomocą protokołu *Websocket*, co pozwoli mu na obserwowanie wyników przeprowadzonych skanów strony w czasie rzeczywistym
* możliwość udostępnienia „pokoju obserwacji” innym, nawet niezarejestrowanym użytkownikom za pomocą długiego, trudnego do zgadnięcia linku. Użytkownicy nie będący właścicielami pokoju będą mogli przeglądać wyniki przeprowadzonego monitorowania strony, ale nie będą mogli zmieniać żadnych ustawień dotyczących monitoringu
* natychmiastowe powiadomienie użytkownika o zmianie na stronie drogą mailową lub notyfikacją w przeglądarce internetowej, zarówno mobilnej, jak i używanej na komputerach stacjonarnych
* wyeksportowanie raportu o zmianach w postaci pliku pdf
* automatyczne generowanie wykresów zmian wybranych wartości liczbowych na stronie
* udostępnienie API dla aplikacji zewnętrznych

W Internecie można znaleźć wiele firm oferujących aplikacje webowe do monitorowania zmian pojawiających się na stronach internetowych. Żadnej firmie nie udało się jak dotąd zająć pozycji lidera na rynku, w różnych rankingach pojedyncze aplikacje zajmują różne pozycje. Do najbardziej popularnych zaliczyć można na pewno:

<https://changetower.com>  
<https://www.wachete.com/>  
<https://visualping.io/>

Poniższa tabela przedstawia porównanie funkcjonalności oferowanych przez najpopularniejsze aplikacje webowe do śledzenia zmian na stronach internetowych oraz aplikacji *WebsiteMonitor*.

|  |  |
| --- | --- |
| **FUNKCJONALNOŚCI** | **APLIKACJE** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *ChangeTower* | *Wachete* | *Visualping* | ***WebsiteMonitor*** |
| monitorowanie zmian na całej stronie |  |  |  |  |
| monitorowanie zmian tylko wybranego elementu na stronie |  |  |  |  |
| wysyłanie powiadomień drogą mailową |  |  |  |  |
| przeprowadzanie skanów z różnych lokalizacji na świecie |  |  |  |  |
| intergracja ze *Slackiem* |  |  |  |  |
| intergracja z *MS Teams, Discord* |  |  |  |  |
| intergracja z *Zapier* |  |  |  |  |
| natychmiastowe, niewymagające odświeżania powiadomienia w przeglądarce |  |  |  |  |
| automatyczne generowanie wykresów zmian wybranych wartości liczbowych na stronie |  |  |  |  |
| możliwości łatwego podzielenia się wynikami utworzonego monitorowania strony z innymi niezarejestrowanymi osobami |  |  |  |  |
| wygenerowania historii zmian strony w postaci pliku pdf |  |  |  |  |
| możliwość ustawienia parametru zakończenia i rozpoczęcia monitorowania strony |  |  |  |  |
| udostępnienie dla aplikacji zewnętrznych API |  |  |  |  |
| możliwość zaprogramowania przez klienta serii akcji, które powinny zostać wykonane przed pobranie kodu strony, są to np.: wciśnięcie wybranego przycisku, wybranie opcji z wyświetlonej listy opcji, usunięcie elementu ze strony, odświeżenie strony, dodanie ciasteczek, wpisanie tekstu w pole, a także odczekanie podanej liczby sekund |  |  |  |  |

Wszystkie trzy powyższe rozwiązania oferują monitorowanie zmian na stronie internetowej lub jej części, wraz z łatwą integrację z różnymi popularnymi w obecnych czasach internetowymi komunikatorami, co stanowi ich niewątpliwą zaletę. Dodatkowo oferowana przez nie możliwość wykonania zautomatyzowanych akcji przed rozpoczęciem pobierania kodu strony, na przykład: ‘w pole wpisz wartość’ lub ‘wciśnij przycisk’, daje możliwość automatycznego zalogowania się na monitorowaną stronę, a tym samym, skanowanie stron dostępnych dopiero po zalogowaniu, takich jak konto bankowe, czy skrzynka pocztowa. Podawanie swoich haseł podmiotom trzecim może wzbudzać jednak uzasadnione obawy użytkowników. Dlatego też, ta funkcjonalność nie będzie dostępna w aplikacji *WebsiteMonitor*.

Niewątpliwą wadą powyższych rozwiązań jest brak możliwości łatwego podzielenia się wynikami utworzonego monitorowania strony z innymi niezarejestrowanymi osobami, tak, aby mogły one przejrzeć historie zmian na śledzonej stronie, oferuje to tylko *Wachate*, brakuje jej jednak powiadomień w czasie rzeczywistym, strona nie używa notyfikacji przeglądarkowych, protokołu Websockets, ani nie odświeża automatycznie strony po przeprowadzonym skanu strony. Żadna strona nie ma możliwości wygenerowania historii skanów w postaci pliku pdf, a automatyczne stworzenia wykresów zmiany wskazanych danych liczbowych na stronie jest dostępne tylko w aplikacji *Wachate*. Przy ustawieniach monitoringu strony w aplikacjach *Visualping*  oraz *Wachete* brakuje parametru zakończenia przeprowadzanych skanów strony, który nie zmuszał by użytkownika do ręcznego usunięcia ustawionego monitoringu. Z całej trójki tylko *Wachete* udostępnia API dla programistów, którzy chcieliby sami obrobić dane za pomocą własnych skryptów. Dodając do tego fakt, iż wszystkie powyższe usługi w wersjach rozszerzonych są płatne, można dojść do wniosku, iż na rynku wciąż nie ma narzędzia zaspokającego potrzeby wszystkich klientów.

Aplikacja webowa *WebsiteMonitor*, która jest tematem tej pracy, wszystkie wymienione wyżej braki wypełni i będzie sprawdzać się wszędzie tam, gdzie łatwość udostępniania wyników monitoringów, ich łatwa i czytelna prezentacja, powiadomienia w czasie rzeczywistym oraz wygodne API będą kluczowe.

Aplikacja będzie składać się z REST APInapisanego w języku *Python* z użyciem framworku *Flask,* korzystającego z bibliotek *pywebcopy* do ściągania stron*, …* do porównywania zmian treści na nich oraz *flaskmysql\_db* do łączenia z bazą danych *MySQL*, w której będą zapisywane dane. Wyświetlana aplikacja webowa będzie stworzona przy pomocy javascriptowej biblioteki *React*, a dostęp do niej będzie umożliwiał serwer *nginx,* pełniący rolę reverse proxy. Całość zostanie skonteneryzowana za pomocą technologii Docker.

Praca składa się z czterech rozdziałów. W pierwszym rozdziale omawiane są szczegółowo założenia funkcjonalne aplikacji oraz zagadnienia informatyczne, które były wykorzystywane przy jej stworzeniu. W drugim rozdziale przedstawiony jest projekt systemu oraz sposób kooperacji jego głównych komponentów. Trzeci rozdział opisuje technologie użyte do implementacji systemu, a także przedstawia ścieżki dostępu do serwera API.

W czwartym rozdziale znajduje się instrukcja, w postaci listy kroków, pełnego wdrożenia systemu. Został również zaprezentowany wygląd poprawnie działającej aplikacji. Końcowy rozdział jest krótkim podsumowaniem pracy.

**Analiza problemu**

W niniejszym rozdziale szczegółowo omówione zostały założenia funkcjonalne aplikacji. Następnie przedstawiono analizę zagadnień informatycznych związanych z aplikacją \textit{WebsiteMonitor}, a w tym problemów, które należało rozwiązać podczas jej projektowania oraz późniejszej implementacji.

**0. Założenia funkcjonalne**

**Rejestracja oraz logowanie**

Podczas pierwszego wizyty w aplikacji użytkownik musi założyć nowe konto, umożliwi mu ono dostęp do wszystkich funkcjonalności. Po wypełnieniu formularza rejestracyjnego, automatycznie zostanie on zalogowany. Przy każdorazowym logowaniu użytkownik otrzyma, ważny przez 24 godziny klucz autoryzacyjny, które zapewni brak konieczności ponownego logowania, aż do czasu upłynięcia jego ważności, chyba, że użytkownik sam wyloguje się wcześniej lub usunie klucz z pamięci podręcznej przeglądarki internetowej.

**Monitorowanie zmian zachodzących na stronie internetowej oraz występowaniu na niej słów kluczowych**

Użytkownik może dla każdej dostępnej publicznie strony internetowej, czyli takiej, do której dostęp nie wymaga logowania, utworzyć cykliczne i w pełni automatyczne monitorowanie zmian na niej zachodzących. Podczas tworzenia monitoringu, konieczne jest podanie takich parametrów jak:

data i godzina rozpoczęcia oraz zakończenia monitoringu

interwał czasowy w jakim strona ma być pobrana i porównana z jej poprzednią wersją, przy czym minimalna długość interwału wynosi 1 minutę

adres mailowy, na który będą wysyłane powiadomienia o wykrytych zmianach

słowa kluczowe, o których wystąpieniu na stronie, chce zostać poinformowany

rodzaj analizowanych zmian na stronie. Jeden monitoring, może przeprowadzić więcej niż jeden rodzaj analizy, dostępne są jej dwa rodzaje: porównanie hashy wszystkich plików tworzących stronę porównanie tekstów występujących na stronie

**Monitorowanie wybranego fragmentu strony internetowej**

Możliwe jest nie tylko monitorowanie zmian w treściach tekstów występujących na całej stronie, ale także zawężenie monitorowania do wybranego elementu strony, również funkcjonalność obserwacji pojawiania się słów kluczowych, rozszerzona jest o taką możliwość. Podczas tworzenia nowego monitorowania, użytkownikowi zostaje wyświetlona podanego przez niego strona,

gdy wskaże on za pomocą kursora myszki jej element, to zostanie on otoczony ramką, po dwukrotnym kliknięciu na niego ramka zmieni kolor i na stałe zostanie wyświetlona przy danym elemencie, oznacza to że element został oznaczony jako ‘do obserwacji’. Ponowne dwukrotne kliknięcie w zaznaczony element odznaczy go. Możliwe jest wybranie tylko jednego elementu strony do obserwacji. Zaznaczenie drugie spowoduje odznaczenie pierwszego.

**Obserwacja na żywo pojawiających się wyników monitoringów**

Po utworzeniu nowego monitoringu, użytkownik automatycznie zostaje przeniesiony do tzw. „pokoju obserwacji”. W pokoju tym, bez potrzeby odświeżania strony, pojawiają się automatycznie wyniki przeprowadzanych porównań poprzednich wersji obserwowanej strony. Dla każdego pokoju zostaje wygenerowany długi, trudny do zgadnięcia link. Każda osoba, nawet niezarejestrowana, za jego pomocą może dostać się do pokoju i obserwować w nim napływające wyniki monitoringu. Z poziomu pokoju można również zmieniać parametry monitoringu, może zrobić to jednak jedynie osoba, która go utworzyła.

**\subsection{Udostępnianie stworzonego monitoringu innym osobom }**

Dla każdego pokoju zostaje wygenerowany długi, trudny do zgadnięcia link. Każda osoba, nawet niezarejestrowana, za jego pomocą może dostać się do pokoju i obserwować w nim napływające wyniki monitoringu.

**\subsection{Natychmiastowe powiadomienia}**

Podczas tworzenia monitoringu użytkownik, może podać adres e-mail, na który będą wysyłane informacje, o przeprowadzonym skanie strony. Domyślenie to pole uzupełnione jest adresem użytym do zalogowania, jednak użytkownik może wprowadzić też inny adres. Do informowania o wykonaniu nowego skanowania wykorzystywane są również powiadomienia push w przeglądarce internetowej.

**\subsection{Raport pdf}**

W "pokoju obserwacji", pojawiające się wyniki przeprowadzonych skanowań monitorowanej strony, mogą zostać pobrane przez użytkownika w postaci pliku pdf.

**\subsection{API dla aplikacji zewnętrznych}**

Serwer API zapewniający funkcjonalności monitorowania stron internetowych w aplikacji \textit{WebsiteMonitor} jest dostępny również dla innych aplikacji zewnętrznych, tak aby mogły z niego korzystać aplikację mobilne, czy przygotowane przez niezależnych programistów ich własne skrypty.

**1. Pobieranie treści dynamicznych stron internetowych**

Obecnie strony internetowe, nie przypominają już tych z przed kilkunastu lat - statycznych, opierających się na danych z jednego źródła, plików html i css. Aktualnie, dość często spotyka się serwery zwracające klientowi, po wpisaniu przez niego adresu strony w przeglądarkę internetową, plik html, który nie zawiera jeszcze wszystkich udostępnianych przez stronę treści. Dodawane one są do pliku html, w sposób dynamiczny, już po stronie klienta podczas renderowania strony przez przeglądarkę. Odpowiedzialne są za to, skrypty napisane w języku javascript, mogą one na przykład obliczać pewne wartości, zwracać aktualną datę i godzinę, a nawet wykonywać zapytania do innych serwerów.

W aplikacji *WebsiteMonitor*  to serwer API będzie odpowiedzialny za pobieranie treści ze stron internetowych, do późniejszej ich obróbki i analizy. Nie będzie on jednak pobierał ich bezpośrednio za pomocą zapytań http o pliki tworzące stronę, w tym plik html, ponieważ nie uzyskał on by treści, które są do niej dodawane dynamicznie. Serwer API pobiera więc stronę za pomocą *Headless Chrome*, czyli przeglądarki internetowej, nie posiadającej interfejsu graficznego, której można używać bezpośrednio z terminala. Przeglądarka ta, spowoduje wykonanie skryptów pobieranej strony i dopiero wtedy przekaże ją serwerowi API.

**2. Wybór elementu do monitorowania**

**1. Bezpieczeństwo i sposoby zabezpieczania stron internetowych**

**1.1 SOP – Same origin policy**

SOP czyli Same-origin policy, jest jednym z podstawowych mechanizmów zabezpieczających stron internetowe stosowanym przez współczesne przeglądarki. Polega on na blokowaniu przez przeglądarkę komunikacji między stronami pochodzącymi z różnych źródeł. „Nie mogą one ściągać, osadzać ani odpytywać swoich wzajemnych elementów."\cite{SOP} Pochodzenie z różnych źródeł, oznacza tutaj posiadanie przez strony różnego \textit{orginu}, definiowanego jako trójka: protokół, host oraz port.

**1.2 X-Frame-Options**

X-Frame-Options to nagłówek odpowiedzi HTTP, dodawany przez serwery, wskazujący na to, czy zwracana strona może być wyświetlona na stronie w znacznikach \textit{html}: \textit{frame}, \textit{iframe}, \textit{embed} lub \textit{object}. Nagłówek może przyjąć jedną z 2 wartości: ‘DENY’ – oznaczający, że nie strona nie może być wyświetlona w tych znacznikach na żadnej stronie, nawet na niej samej lub ‘SAMEORIGIN’ – oznaczający, że może zostać wyświetlona tylko na stronach posiadających ten sam \textit{origin}. Nagłówek ten chroni strony internetowe przed atakami typu \textit{clickjacking} i \textit{framebusting}.

**1.3** **Znacznik iframe, jako rozwiązanie problemów z Same-origin policy \\ i X-Frame-Options**

W celu umożliwienia użytkownikowi wygodnego wybrania elementu strony do monitorowania, strona ta zostaje wyświetlona w aplikacji wraz z zastosowaniem znajdujących się na niej plików \textit{css}. Wstrzyknięcie kodu tworzących ją plików bezpośrednio do aplikacji \textit{ReactJs} mogłoby jednak spowodować problemy w poprawnym jej wyświetleniu, ponieważ SOP mogłaby zablokować dynamicznie dodawane do niej elementy, będące danymi ściąganymi z innych źródeł. Źródła te, odnotowywałyby, że pochodzenie kierowanych do nich zapytań jest równe \textit{originowi} aplikacji \textit{WebsiteMonitor}, a nie \textit{originowi} strony którą użytkownik chce monitorować, więc zostałby one z dużym prawdopodobieństwem zablokowane. Kolejnym problemem byłby potencjalny konflikt nakładania się na siebie reguł stylowania z plików \textit{css} wyświetlanej strony oraz aplikacji \textit{WebsiteMonitor}. Na przykład w przypadku, gdy wyświetlana strona dla każdego napisu ustala kolor czcionki na czerwony, podczas gdy w aplikacji jest on czarny.

W celu ominięcia obu tych problemów kod wyświetlanej strony nie zostanie wstrzyknięty bezpośrednio do kodu aplikacji, lecz zostanie on ściągnięty przez serwer API z wykorzystaniem przeglądarki \textit{Headless Chromium}, co zapewni poprawne wyświetlenie dynamicznie dodawanych danych z innych domen. Tak ściągnięta strona zostanie wystawiona przez serwer API pod adresem, który zostanie wysłany do aplikacji \textit{ReactJs}. Strona zostanie następnie wyświetlona w znaczniku \textit{html iframe}. Znacznik ten umożliwia wyświetlanie stron internetowych. Ściągnięcie strony najpierw przez serwer API, a nie wyświetlenie go bezpośrednio w znaczniku \textit{iframe} zapobiegnie dodatkowo potencjalnym problemom z nagłówkiem X-Frame-Options. Strona, którą użytkownik chce monitorować, może być przez zwracający ją serwer wysyłana z ustawioną flagą X-Frame-Options. Oznaczałoby to, iż nie może być ona wyświetlona bezpośrednio w znaczniku \textit{iframe} w aplikacji \textit{WebsiteMonitor}. Aby temu zapobiec, strona będzie zwracana przez serwer API do aplikacji \textit{ReactJs} bez flagi X-Frame-Options.

**1.4 Wstrzyknięcie skryptu umożliwiającego wygodny wybór elementu do obserwacji**

Element na stronie można jednoznacznie zidentyfikować za pomocą \textit{xpath}, czyli języka ścieżek plików \textit{XML}, „służącego do elastycznego adresowania (wskazywania) różnych części dokumentu XML”\cite{XPath}. Wyszukanie takiego adresu przez użytkownika na stronie, którą chce on monitorować, a następnie powrót do aplikacji i podanie ścieżki elementu do monitorowania, byłoby jednak dla niego bardzo uciążliwe. By tego uniknąć, serwera API po pobraniu podanej przez użytkownika strony, odpowiednio ją parsuje, a następnie wstrzykuje na koniec znacznika \textit{body} własny skrypt w języku \textit{javascript}. Tak zmodyfikowaną stronę, wraz ze wszystkimi, zawartymi w niej plikami, w tym plikami css, bibliotekami oraz zdjęciami serwuje on pod własną domeną, a adres do niej wysyła klientowi, tak, aby mógł on wyświetlić ją w znaczniku \textit{iframe} w części frontendowej aplikacji WebsiteMonitor. Wstrzyknięty skrypt powoduje, że, gdy klient wskaże za pomocą kursora element strony wyświetlanej w znaczniku \textit{iframe}, to zostanie on otoczony ramką, a po dwukrotnym kliknięciu na taki element ramka zmieni kolor i na stałe zostanie wyświetlona przy wskazanym elemencie. Sprawi to, że element zostanie oznaczony jako element do obserwacji. Dodatkowo wstrzyknięty skrypt umożliwia przesłanie informacji o wybranym elemencie wyświetlanym w znaczniku \textit{iframe} do aplikacji \textit{ReactJs}. Odbywa się to za pomocą metody \textit{Window.postMessage()}, która umożliwia komunikację pomiędzy obiektami \textit{Window} stron internetowych mających różny \textit{origin}. Jest to jedyny dopuszczalny sposób na komunikację pomiędzy skryptami z aplikacji \textit{ReactJs} oraz tym wstrzykniętym do strony wyświetlanej w znaczniku \textit{iframe}. Jest tak, ponieważ komunikacja pomiędzy skryptami pochodzącymi z różnych stron (o różnym originie) jest blokowana przez przeglądarkę internetową, zgodnie z SOP, a aplikacja ReactJs oraz serwer API, przez który jest serwowana strona wyświetlana w \textit{iframe}, pochodzą z różnych źródeł (mają różne \textit{originy}).

**4. Porównanie plików tekstowych**

\section{Porównanie plików tekstowych}

Przedstawienie, w sposób czytelne i łatwy do odczytania dla użytkownika, różnic w tekstach zmieniających się na monitorowanej stronie internetowej jest jedną z kluczowych funkcjonalności aplikacji WebsiteMonitor. Podanie samych danych liczbowych dotyczących wykrytych zmian, na przykład: liczby nowych słów w tekście lub numerów linii, w których wystąpiły zmiany, nie jest wystarczająco przejrzystą formą przedstawienia różnic w tekstach. Aby zaprezentować te różnice bardziej wyraziście, aplikacja WebsiteMonitor wygeneruje, w postaci pliku \textit{html} dla każdego przeprowadzonego skanu strony, tabelę porównawczą. Tabela będzie porównywać stronę z jej poprzednią wersją linia po linii i dodatkowo oznaczać je kolorami zielonym, czerwonym lub żółtym, oznaczającymi odpowiednio dodaną, usuniętą lub zmodyfikowaną linię. Biały kolor oznacza, że linia nie została zmieniona.

Porównanie plików odbędzie się w serwerze API, użyta zostanie biblioteka \textit{Selenium} do obsługi przeglądarki internetowej \textit{Headless Chromium} i ściągnięcia za jej pomocą pliku \textit{html żądanej} strony. Następnie przy pomocy biblioteki \textit{bs4} i obiektu \textit{BeautifulSoup} zostaną z niego wydzielone występujące na stronie napisy. Dodatkowo jeśli użytkownik, utworzył monitoring, obserwujący tylko dany element strony, to zostanie on uprzednio wyszukany, a następnie wydzielone zostaną napisy występujące tylko w nim. Napisy zostaną zapisane do pliku, z zakodowanym swojej nazwie numerem identyfikującym monitoring, a także numerem przeprowadzonego skanu. Następnie zostaną wczytane dane z analogicznego pliku, stworzonego po wykonaniu poprzedniego skanu strony. Przy pomocy biblioteki \textit{difflib}, zostanie wygenerowana tabela, w postaci pliku \textit{html}, porównująca oba pliki linia po linii, wraz z oznaczeniem ich odpowiednimi kolorami, zgodnie z opisem powyżej. Tak wygenerowany plik z tabelą zostanie zapisany w pamięci serwera API, a ścieżka do niego zostanie zapisana w bazie danych. Plik przechowujący dane tekstowe z poprzedniego skanu strony zostanie usunięty, tak aby w pamięci przechowywany był tylko ten z najnowszego skanowania.

**Porównywanie tekstów wykonane przez bibliotekę difflib**

Biblioteka \textit{difflib} używana jest do porównywania plików tekstowych. Implementuje ona zmodyfikowany algorytm „Gestalt Pattern Matching” opublikowany pod koniec lat 80-tych przez Ratcliffa i Obershelpa. Idea zmodyfikowanego algorytmu polega na znalezieniu najdłuższego ciągłego podciągu, który nie zawiera elementów, które są w pewnym sensie nieinteresujące, takie jak puste linie lub spacje. Po znalezieniu takiego podciągu proces powtarzany jest rekurencyjnie dla fragmentów sekwencji po jego lewej i prawej stronie. Algorytm ten nie gwarantuje znalezienia minimalnej liczby edycji, ale zwykle daje dopasowania, które są czytelne dla ludzi. \cite{Difflib}

**5. Wykrywanie zmian we wszystkich plikach tworzących stronę**

Wykrywanie zmian we wszystkich plikach tworzących stronę składa się zasadniczo z 2 kroków. Najpierw wszystkie pliki używane do wyświetlenia danej strony ściągane są przez serwer API za pomocą biblioteki pywebcopy. W następnym kroku, obliczany jest hash każdego pliku, przy pomocy biblioteki hashlib i funkcji hashującej blake2b. Hash każdego pliku podczas danego skanu zapisywany jest w bazie danych.

\subsubsection{Funkcja hashująca \textit{BLAKE2b}}

Funkcja hashująca \textit{BLAKE2b} została wybrana przede wszystkim ze względu na swoją szybkość, jest ona szybsza od funkcji \textit{MD5}, \textit{SHA-1}, \textit{SHA-2}, \textit{SHA-3}, co możemy zaobserwować na wykresie poniżej.

**3. Projekt systemu**

W niniejszym rozdziale została przedstawiona architektura aplikacji webowej \textit{WebsiteMonitor}, a także

jej szczegółowy projekt w notacji UML, uwzględniający wymagania funkcjonalne opisane w rozdziale~\ref{rozdzial1}. W celu opisu relacji pomiędzy składowymi aplikacji zaprezentowane zostały również diagramy: przypadków użycia, klas, aktywności oraz sekwencji.

Architektura aplikacji \textit{WebsiteMonitor} jest wielowarstwowa i rozproszona. Składa się ona z czterech części:

\begin{enumerate}

\item serwera proxy \textit{nginx} odpowiedzialnego za połączenie aplikacji ze światem zewnętrznym, wydajną obsługę kierowanych do aplikacji zapytań, a także pełniącego rolę odwróconego proxy (reverse proxy), ukrywającego strukturę znajdującej się za nim aplikacji

\item aplikacji \textit{React.js}, odpowiedzialnej za bezpośrednią interakcję z użytkownikiem oraz wyświetlanie mu graficznego interfejsu aplikacji

\item serwera REST API, stanowiącego centralną część aplikacji, odpowiedzialnego za wykonywanie wszystkich jej zadań, dotyczących analizy zachodzących na stronach internetowych zmian, a także obsługi bazy danych

\item relacyjnej bazy danych \textit{MySQL}, odpowiedzialnej za wydajny dostęp do zapisywanych danych

\caption{{\color{dgray} Diagram przypadków użycia}}

\label{use-case-uml}

\end{figure}

Na rysunku~\ref{use-case-uml} przedstawiony jest diagram przypadków użycia. W aplikacji możemy wyróżnić 2 typy użytkowników. Pierwszy typ stanowią użytkownicy niezalogowani, którzy mogą tylko obserwować napływające wyniki monitoringu, dla którego udostępniony został im link. Drugi typ stanowią użytkownicy zalogowani, dla których dostępne są wszystkie funkcjonalności aplikacji w obrębie monitoringów stworzonych przez nich samych.

\caption{{\color{dgray} Diagram klas serwera API}}

\end{figure}

Na rysunku \ref{frontend-class-uml} widoczny jest diagram komponentów aplikacji \textit{ReactJs} odpowiedzialnej za wizualną prezentację aplikacji \textit{WebsiteMonitor} w przeglądarce użytkownika. Komponent definiowany jest za pomocą klasy lub funkcji napisanych w języku \textit{TypeScript}. Komponenty opisane za pomocą klasy dziedziczą na diagramie po klasie \textit{React.Component}, natomiast komponenty funkcyjne kończą swoją nazwę symbolem "( )". Komponent przyjmuje zdefiniowane parametry i zwraca, jak powinna wyglądać opisywana przez niego część interfejsu użytkownika. Interfejs podzielony jest na wiele niezależne części (komponentów), dzięki temu, gdy zmienią się dane w jednym komponencie, to nie jest odświeżana cała strona, a jedynie on sam. Dodatkowo, zdefiniowane komponenty mogą być użyte ponownie w innych miejscach aplikacji. Poza nimi na diagramie przedstawione są 2 klasy, które odpowiedzialne są za komunikację z serwerem API, są to \textit{authService} oraz \textit{monitorService}. \cite{React-component}

\section{Diagramy aktywności}

Na rysunkach \ref{sequence-uml-create-new-monitor} oraz \ref{sequence-uml-observe} przedstawiono diagramy sekwencji odpowiadające aktywnościom przedstawionym na diagramach aktywności w poprzednim podrozdziale.

Rysunek \ref{sequence-uml-create-new-monitor} przedstawia diagram sekwencji dla aktywności stworzenia nowego monitoringu przez zalogowanego użytkownika.

Momentem szczególnie istotnym na diagramie jest otrzymanie przez serwer API żądania utworzenia nowego monitoringu, które jest wysyłane za pomocą metody \texttt{createMonitor()} aplikacji \textit{React.js}. Po walidacji danych za pomocą metody \texttt{is\\_data\\_valid()} oraz zapisaniu danych nowego monitoringu do bazy danych za pomocą metody \texttt{insert\\_monitor()} serwer API tworzy nowy obiekt klasy \texttt{Scheduler}. Jest to kluczowa dla działania całej aplikacji klasa, która odpowiedzialna jest za równoległe i cykliczne przeprowadzanie analiz zmian występujących na monitorowanej stronie internetowej. Po utworzeniu obiektu klasy \texttt{Scheduler} uruchamiany jest za pomocą metody \texttt{run()} nowy wątek, który w pętli, co ustalony podczas tworzenia monitoringu interwał czasowy, ściąga pliki monitorowanej strony, a następnie porównuje je z ich poprzednią wersją. Wygenerowane w ten sposób dane zapisuje on w bazie danych, raport porównujący teksty występujące na stronie zapisuje w folderze \texttt{diffs}, a na koniec wysyła do użytkownika e-maila, z pełnym raportem zawierającym wyniki przeprowadzonego skanowania strony, a w tym: porównanie tekstów występujących na stronie, porównanie wszystkich plików tworzących stronę oraz wystąpienie na niej słów kluczowych, podanych przez użytkownika podczas tworzenia monitoringu. Jeżeli data zakończenia monitoringu minęła lub użytkownik samodzielnie usunął monitoring, to wątek kończy swoją pracę.

Rysunek \ref{sequence-uml-observe} przedstawia diagram sekwencji dla aktywności wejście do pokoju obserwacji. Cała sekwencja składa się z 2 głównych części. Pierwsza część, polega na pobraniu przez aplikację \textit{React.js} za pomocą metody \texttt{getExistingScans()} wyników wszystkich dotychczas przeprowadzonych skanów monitorowanej strony. Druga część polega na stworzeniu za pomocą metody \texttt{setIntervalApiPing()}, cyklicznego wysyłania zapytań (\texttt{getScan()}) o wyniki nowych skanowań. Wyniki obu części generowane są przez serwer API za pomocą metody \texttt{prepare\\_scan\\_raport()}, a następnie przedstawiane są użytkownikowi za pomocą metod aplikacji \textit{React.js} \texttt{handlePDF()}, \texttt{render()} oraz \texttt{addNotification()}.

\section{Projekt bazy danych}

Na rysunku \ref{db-uml} został przedstawiony schemat relacyjnej bazy danych \textit{MySQL}, używanej w aplikacji \textit{WebsiteMonitor} do wydajnego przechowywania i odczytywania danych. Baza danych składa się z 4 tabel:

\begin{enumerate}

\item users - przechowuje dane o zarejestrowanych użytkownikach, a także zahashowane hasła wraz z solą, użytą do ich zakodowania.

Kluczem głównym w tej tabeli jest mail użytkownika, podawany podczas rejestracji. Jego unikatowość zapewniona jest przez serwer API, który odrzuci każdą próbę utworzenia konta, na już istniejący w bazie mail.

\item monitors - przechowuje dane o utworzonych monitoringach stron internetowych. Każdy użytkownik może być autorem 0 lub większej liczby monitoringów, ale pojedynczy monitoring może mieć tylko jednego autora. Nie może istnieć monitoring bez autora, zatem relacja użytkownik - monitoring jest relacją 1 do wielu. Każdy monitoring w tabeli posiada swój unikatowy numer id, będący kluczem głównym tej tabeli. Przechowywane są również informacje o:

\begin{itemize}

\item mailu autora monitoringu, będącego kluczem obcym w tej tabeli

\item adresie url monitorowanej strony

\item elemencie strony wybranym do monitorowania

\item słowach kluczowych, o których wystąpieniu użytkownik chce zostać poinformowany

\item interwale czasowym, w którym strona powinna być sprawdzana pod względem zmian na niej występujących

\item mailu, na który użytkownik chce otrzymywać powiadomienia o przeprowadzonym skanowaniu strony

\item dacie i godzinie rozpoczęcia monitoringu

\item dacie i godzinie zakończenia monitoringu

\item rodzaju przeprowadzanej na stronie analizy: czy szukać zmian w tekstach na stronie

\item rodzaju przeprowadzanej na stronie analizy: czy porównywać wszystkie pliki tworzące stronę

\item statucie monitoringu: czy jest on aktywny

\end{itemize}

\item scans - przechowuje dane o przeprowadzonych skanach stron internetowych.

Każdy monitoring, może mieć przeprowadzonych przez niego samego 0 lub większą liczbę skanów. Skan mógł zostać przeprowadzony przez tylko jeden monitoring. Nie może istnieć skan bez przepisanego mu monitoringu, zatem jest to relacja 1 do wielu. Tabela przechowuje informacje o:

\begin{itemize}

\item id, oznaczającym numer wykonanego skanu. Skany są numerowane kolejnymi liczbami naturalnymi

\item id monitoringu, który wykonał dany skan. Jest to klucz obcy tabeli

\item fladze, czy na stronie zostały znalezione różnice, w stosunku do jej poprzedniej wersji

\item lini tektów, w których znalezione zostały słowa kluczowe

\end{itemize}

Klucz główny tej tabeli stanowi para id oraz id monitoringu.

\item files - przechowuje dane o plikach tworzących monitorowaną stronę. Każdy skan może posiadać wiele plików. Pojedynczy plik może należeć tylko do jednego skanu, jest to więc relacja 1 do wielu. Para id skanu oraz id monitoringu stanowi klucz obcy tabeli, a wraz z nazwą pliku stanowią jej klucz główny. Poza nimi każdy plik posiada informację o

\begin{itemize}

\item statusie pliku: czy jest on względem poprzedniego skanu dodany, usunięty, zmodyfikowany lub niezmieniony

\item hash pliku, wykorzystywanego przy jego porównywaniu z poprzednią wersją strony

\end{itemize}

\end{enumerate}

**IMPLEMENTACJA SYSTEMU**

\section{Opis technologii}

Aplikacja \textit{WebsiteMonitor} składa się 4 głównych komponentów, każdy z nich pełni inną funkcję i został zaimplementowany przy użyciu innej technologii.

\begin{enumerate}

\item Serwer REST API - został zaimplementowany w języku \textit{Python} \cite{Python}, przy użyciu frameworku do tworzenia aplikacji webowych \textit{Flask} \cite{Flask}. Zostały również wykorzystane biblioteki:

\begin{itemize}

\item \textit{difflib} - do porównywania plików

\item \textit{pywebcopy} - do pobierania stron internetowych

\item \textit{flaskmysql\\_db} - do łączenia z bazą danych

\item \textit{flask\\_jwt\\_extended} - do generowania tokenów autoryzacyjnych

\item \textit{flask\\_mail} - do wysyłania maili

\item \textit{bs4} - do parsowania plików \textit{html}

\item \textit{hashlib} - do obliczania hashy plików

\end{itemize}

\item Aplikacja webowa uruchamiana w przeglądarce użytkownika - została stworzona przy pomocy biblioteki \textit{React.js} \cite{React}. Aplikacja napisana została w języku \textit{typescript} \cite{TypeScript}, który to jest nadzbiorem języka \textit{javascript} i rozszerza go o możliwość statycznego typowania.

% javascript czy Javascript

\item Baza danych - została stworzona przy pomocy systemu zarządzania relacyjnymi bazami danych \textit{MySQL}. \cite{MySQL}

\item Serwer proxy \textit{ngnix} - został skonfigurowany poprzez stworzenie dedykowanego pliku konfiguracyjnego \textit{nginx.conf}. \cite{Nginx}

\end{enumerate}

\section{Dokumentacja ścieżek serwera API}

Serwer API umożliwia dostęp do swoich zasobów nie tylko aplikacji \textit{React.js}, ale również zewnętrznym aplikacjom lub skryptom, które nie są częścią systemu \textit{WebsiteMonitor}. W celu wykonania poprawnego żądania do serwera API należy w nagłówku żądania \textit{http} dołączyć w polu \texttt{Authorization} wygenerowany podczas logowania lub rejestracji token autoryzacyjny. Zakodowany jest w nim mail użytkownika, dzięki czemu, serwer może ustalić tożsamość osoby wysyłającej żądanie i potwierdzić, że ma ona prawo wykonać żądaną akcję. Wyjątkiem są ścieżki \texttt{get-monitor}, \texttt{get-existing-scans} oraz \texttt{get-scan}, które to nie wymagają tokenu autoryzacyjnego. Serwer API oferuje dostęp do swoich zasobów za pomocą poniższych ścieżek:

\begin{enumerate}

\item \textbf{GET} \texttt{monitors/get-monitor?\underline{monitorId}=' '}

Pobranie informacji o danym monitoringu.

\item \textbf{GET} monitors/\texttt{get-user-monitors}

Pobranie informacji o wszystkich monitoringach stworzonych przez użytkownika.

\item \textbf{GET} monitors/\texttt{get-existing-scans?\underline{monitorId}=' '}

Pobranie wyników wszystkich dotychczas przeprowadzonych skanów danego monitoringu.

\item \textbf{GET} \texttt{monitors/get-scan?\underline{monitorId}=' '\&\underline{scanId}=' '}

Pobranie wyników 1 skanu danego monitoringu.

\item \textbf{GET} \texttt{monitors/page-to-monitor?\underline{adress}=' '}

Pobranie linku do strony z wstrzykniętym skryptem umożliwiającym wygodny wybór elementu „do obserwacji”.

\item \textbf{POST} \texttt{/login}

Wygenerowania tokenu autoryzacyjnego.

Żądanie musi zawierać element \texttt{body} z następującymi polami:

\begin{itemize}

\item \texttt{mail}

\item \texttt{password}

\end{itemize}

\item \textbf{POST} \texttt{/register}

Utworzenie nowego konta i pobranie tokenu autoryzacyjnego.

Żądanie musi zawierać element \texttt{body} z następującymi polami:

\begin{itemize}

\item \texttt{name}

\item \texttt{surname}

\item \texttt{mail}

\item \texttt{password}

\end{itemize}

\item \textbf{POST} \texttt{monitors/create-monitor}

Utworzenie nowego monitoringu.

Żądanie musi zawierać element \texttt{body} z następującymi polami:

\begin{itemize}

\item \texttt{start}

\item \texttt{end}

\item \texttt{choosenElement}

\begin{itemize}

\item \texttt{tag}

\item \texttt{index}

\end{itemize}

\item \texttt{keyWords}

\item \texttt{intervalMinutes}

\item \texttt{textChange}

\item \texttt{allFilesChange}

\item \texttt{mailNotification}

\item \texttt{url}

\end{itemize}

\item \textbf{PUT} \texttt{monitors/update-monitor}

Zmiana parametrów już stworzonego monitoringu.

Żądanie musi zawierać element \texttt{body} z następującymi polami:

\begin{itemize}

\item \texttt{id}

\item \texttt{start}

\item \texttt{end}

\item \texttt{choosenElement}

\begin{itemize}

\item \texttt{tag}

\item \texttt{index}

\end{itemize}

\item \texttt{keyWords}

\item \texttt{intervalMinutes}

\item \texttt{textChange}

\item \texttt{allFilesChange}

\item \texttt{mailNotification}

\item \texttt{url}

\end{itemize}

\item \textbf{DELETE} \texttt{monitors/delete-monitor?\underline{monitorId}=' '}

Usunięcie danego monitoringu.

\end{enumerate}