**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 protokołu WebSocket, a historia zmian będzie prezentowana w czytelny sposób, wraz z możliwością wygenerowania pliku pdf.

Zakres działania aplikacji obejmuje przede wszystkim cykliczne pobieranie 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 …

**1. Analiza problemu**

…

Należy uwzględnić i uzasadnić czemu używa się iframe i ze inaczej się nie da, bo dr Krzywiecki sie dopytywal, ze to przestarzale

Jedną z kluczowych funkcjonalności aplikacji, jest wybranie konkretnego elementu, do obserwacji na podanej przez klienta stronie. Wymagającym zagadnieniem jest sposób w jaki należy wyświetlić w aplikacji stronę podaną przez klienta np. *www.pwr.pl,* tak jak widzi on ją w swojej przeglądarce i umożliwić mu wygodne wybrane elementu ‘do obserwacji’ (np. poprzez kliknięcie lub zaznaczenie go) , mając równocześnie na uwadze obostrzenie ze strony SOP.

Podstawowym mechanizmem obronnym nowoczesnych przeglądarek jest Same-Origin Policy.

[Same-Origin Policy (SOP)](https://en.wikipedia.org/wiki/Same-origin_policy):

* Protokół (inaczej – Schemat)
* Host (sprawdzany rygorystycznie – to znaczy subdomena nie jest tożsama z domeną!)
* Port

dwie aplikacje charakteryzujące się różnymi Originami nie mogą używać (ściągać, osadzać, odpytywać) swoich wzajemnych elementów.

Oznacza to na przykład:

- Nie można zapisywać i odczytywać ciasteczek Originu A, będąc na stronie innego Originu B

**- Nie można wywoływać zapytań HTTP z Originu A do Originu B (na przykład element** **<form> który jest wysyłany pod inny adres)**

Podejscie 1.

Sciagniecie kodu strony, i wklejenie go na nasza. Nie ma problemu jeśli chodzi o tresc, ale jeśli dynamicznie wkleilibysm reguly css, to mogly by one nakladac się na te usyte do stylowania mojej aplikacji. Co jeśli na podanej stronie czcionka ustawiona jest na kolor czarny a na mojej na bialy, reguly jednej bądź drugiej strony zostaly by nadpisane w zaleznosci od specyfikacji danej reguly.

Poza tym wystepuje tu problem z SOP. Strony często laduja się dynamicznie: czyli do pliku html są często dodawane tresci za pomoca skryptu js, który te tresci pobiera za pomoca np.

zapytn https do innej domeny B. Domena B ma zapisany adres strony A i akceptuje jej zapytania, zapytania ze skopiowanego skryptu na moja strone lecialby z domena C i z duzym prawopodobienstwem zostalby odrzucane przez poityke SOP.

Podejscie 2.

Idealnym do tego elementu html iframe.   
Wyswitlenie w iframie, bezposrednio na stronie, X-frame-option false na to nie pozwoli

Poza tym SOP zablokuje skrypt próbujący modyfikować elementy iframe, jeżeli strona posiadajaca iframe i strona na nim wyswietlana maja inny origin.

Rozwiazanie.

Sciagniecie strony już z zaladowanym dynamicznymi elementami, tak jakbysmy otwierali ja w przegladarce na serwer, dzieki pywebcopy, która taka przegladarke symuluje, nastepnie wystawienie do niej route na serwerze, z tej routy iframe z frontendu wyrenderowanej strony ponieważ mój sewer nie ma X-frame-option true.

Wstrzykniecie js na poziomie skryput na backendzie czemu? Latwiejsze w developmencie ponieważ wtedy backend i front są na innych portach więc SOP blokuje wstrzykniecie skryptu

W dockerze za reverse proxy nie byloby problemu

Do komunikacji pomiedzy skryptami w aplikacji WebSiteMonitor oraz skryptem wstrzyknietym w iframe, jest wykorzystana funkcja window.postMessage() (tutaj opis)

<https://sekurak.pl/czym-jest-cors-cross-origin-resource-sharing-i-jak-wplywa-na-bezpieczenstwo/>

- w jaki sposob porownac pliki ? Tak zeby było to ergonomiczne i czytelne ?

difflib

Nie chodzi tylko o to żeby powiedzieć ze 2 pliki sa rozne albo ze roznia się ilomas słowami,

Zależy nam żeby odnosci się do każdej linijki poprzedniej wersji i informować czy została usunieta lub zmieniona,

te „śmieciowe” elementy to takie, które są w pewnym sensie nieinteresujące, takie jak puste linie lub spacje Ta sama idea jest następnie stosowana rekurencyjnie do fragmentów sekwencji po lewej i po prawej stronie pasującego podciągu. Nie daje to minimalnej liczby sekwencji edycji, ale zwykle daje dopasowania, które „wyglądają dobrze” dla ludzi.

**IFRAME SOP**

<https://stackoverflow.com/questions/25098021/securityerror-blocked-a-frame-with-origin-from-accessing-a-cross-origin-frame>

<https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>

The **window.postMessage()** method safely enables cross-origin communication between [Window](https://developer.mozilla.org/en-US/docs/Web/API/Window) objects; e.g., between a page and a pop-up that it spawned, or between a page and an iframe embedded within it.

Normally, scripts on different pages are allowed to access each other if and only if the pages they originate from share the same protocol, port number, and host (also known as the "[same-origin policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy)"). window.postMessage() provides a controlled mechanism to securely circumvent this restriction (if used properly).

Czemu