## Opis Snail Project

Agile  
W trakcie tworzenia Projektu wykorzystane zostało programowanie zwinne (ang. Agile software development). Ponieważ założeniem było jak najszybsze otrzymywanie wyników poprzez wykonywanie skryptów, layout był tworzony „w trakcie” programowania innych zadań. Programowanie iteracyjno-przyrostowe, powstałe jako alternatywa do tradycyjnych metod typu waterfall, świetnie się sprawdziło podczas implementowania różnego rodzaju skryptów np. Insight. Jednym z założeń rozwiązania Agile jest praca w zespołach, które mogą niezależnie od siebie tworzyć pewne części danego projektu. W tym przypadku zespół składał się z jednej osoby oraz „odbiorcy”. Współpraca owocowała wieloma pomysłami oraz zmianami w trakcie tworzenia projektu, podejście typu Waterfall mogłoby się zakończyć niepowodzeniem projektu, lub brakiem możliwości wprowadzania jakichkolwiek zmian po fazie projektowej.

Agile bardzo często znajduje zastosowanie w małych zespołach programistycznych, w których jest dobra komunikacja, tak by nie tworzyć rozbudowanej dokumentacji kodu. Kolejne etapy wytwarzania oprogramowania zamknięte są w iteracjach, w których za każdym razem przeprowadza się testowanie wytworzonego kodu, zebranie wymagań, planowanie rozwiązań itd. Metoda nastawiona jest na szybkie wytwarzanie oprogramowania wysokiej jakości oraz regularna adaptacja do zmieniających się wymagań.

## Pierwsza iteracja Snail Project

Zanim zaimplementowano skrypty badające wydajność serwisów webowych, stworzony został *layout* Projektu. Po wpisaniu odpowiedniego adresu URL w przeglądarce wyświetli się górny pasek szerokości okna przeglądarki, na którym znajduje się nazwa projektu oraz adres IP użytkownika, który odwiedził stronę. Pozostałą część okna zajmuje obraz ślimaka, stanowiący symbol projektu, a na jego środku widnieje miejsce, w którym użytkownik powinien wprowadzić adres URL w postaci „[www.pwr.edu.pl](http://www.pwr.edu.pl)” bez poprzedzającego adres prefiksu [http://](NULL) oraz przycisk „Test”, który po wciśnięciu uruchomi wszystkie skrypty dokonujące odpowiednich analiz, parsując wyniki i wyświetli je w określony odpowiednimi stylami sposób, aby były czytelne i zrozumiałe dla odbiorcy. Podczas oczekiwania na wykonanie się skryptów testujących do momentu wyświetlenia wyników widoczny jest „loader” w postaci obracających się kropek sugerujący, że właśnie trwa wykonywanie testu i należy poczekać na jego wynik. Rozmiar obrazu ze ślimakiem jest przystosowany to szerokości i wysokości okna przeglądarki a wyniki prezentowane są na białym półprzezroczystym tle, które jest nałożone na estetyczny gradient, tło całej witryny.

Do nadania Projektowi ostatecznego kształtu wykorzystane zostały takie technologie i frameworki jak PHP, JavaScript, Ajax, Bootstrap, Mustache.js, jQuery oraz oczywiście HTML oraz CSS. Wszystko służyło odpowiedniemu ułożeniu i pozycjonowaniu elementów, oraz odpowiedniemu parsowaniu wyników zwracanych przez skrypty.

Cała aplikacja webowa jest anglojęzyczna ponieważ czyni to ją bardziej uniwersalną i dostępną również dla programistów z zza granicy, którzy chcą dokonywać testów wydajności i optymalizacji stron internetowych.

## PageSpeed Insight

Kolejną iteracją, zgodnie z założeniami programowania zwinnego, lecz pierwszym zaimplementowanym skryptem w projekcie został Page Speed Insights, który mierzy wydajność stron na komputerach. Skrypt w projekcie pobiera adres strony internetowej podanej przez użytkownika, parsuje go a następnie przesyła do serwera Google, gdzie następuje analiza strony. Po wykonaniu testu wyniki są zwracane do skryptu zaimplementowanego w Snail. Są to wyniki w postaci obiektów, następnie są one parsowane i wyświetlone użytkownikowi w sposób czytelny i zrozumiały w postaci tabeli.

Rezultat działania PageSpeed mieści się w zakresie od 0 do 100 punktów. Im wyższy wynik, tym lepszy. Wynik na poziomie co najmniej 85 punktów oznacza, że strona działa dobrze. Jednak na potrzeby Projektu, wynik jest wyświetlany w postaci procentowej. Po krótkim badaniu na grupie około 30 studentów, określono, że procent zoptymalizowania serwisu webowego jest dla odbiorców bardziej znaczącym wynikiem niż punkty. Google ostrzega, że narzędzie PageSpeed Insights jest nieustannie usprawniane, więc wynik może się zmieniać, gdy zostaną dodane nowe reguły oceny serwisów.

Insights sprawdza, jak można poprawić wydajność strony w następujących aspektach:

* czas wczytywania części strony widocznej na ekranie: czas, który upływa od momentu wysłania żądania nowej strony do momentu wyrenderowania części strony widocznej na ekranie przez przeglądarkę.
* czas pełnego wczytania strony: czas, który upływa od momentu, gdy użytkownik wysyła żądanie nowej strony do momentu, gdy strona zostanie w pełni wyrenderowana przez przeglądarkę.

Ze względu na dużą zmienność wydajności połączenia sieciowego PageSpeed Insights uwzględnia wyłącznie aspekty wydajności renderowania strony niezależne od sieci: konfigurację serwera, strukturę kodu HTML strony, a także korzystanie z zasobów zewnętrznych, takich jak obrazy, pliki JavaScript i CSS. Zastosowanie się do sugestii powinno poprawić względną wydajność strony, jednak wydajność bezwzględna wciąż pozostanie zależna od połączenia sieciowego użytkownika.

Skrypt zwraca wynik działania zawsze tych samych wartości:

* Unikanie zbędnych przekierowań
* Włączenie kompresji danych
* Wykorzystanie pamięci podręcznej przeglądarki
* Skrócenie czasu odpowiedzi serwera
* Minimalizowanie plików CSS
* Minimalizowanie plików HTML
* Minimalizowanie plików JavaScript
* Eliminowanie blokujących skryptów
* Optymalizacja obrazów
* Priorytetyzacja widocznej treści

Insights w Snail Project określa stopień wydajności strony w procentach. Rezultat „This site is optimized in 70%” należy czytać jako „Strona jest zoptymalizowana w 70 procentach”. Im wyższy wynik osiąga testowana aplikacja, tym mniej „kroków” należy wykonać aby zoptymalizować jej wydajność i jakość działania. PageSpeed dokonuje analizy stron internetowych za pomocą reguł ustalonych przez Google. Każda reguła opiera się na ogólnych zasadach działania strony internetowej, takie jak buforowanie zasobów, przekazywanie danych i wielkości pobierania danych, odpowiednią minimalizację plików ze stylami, skryptami itp.

Każda reguła generuje zmiennoprzecinkową wagę, priorytet, z jakim należy uwzględnić pierwszeństwo wprowadzania zmian aby osiągnąć lepszą wydajność. Na przykład, jeśli optymalizacja obrazów na testowanej stronie internetowej pomoże zmniejszyć ich rozmiar o 1 MB, to ta reguła otrzyma większą przykładowo o 5,5845 wagę niż zminimalizowanie rozmiaru plików CSS, które np. zmniejszą rozmiar przesyłanych danych o 300 KB. Ważne jest jednak aby zastosować wszystkie sugestie, które przy ponownym wykonaniu mogą dać lepszy rezultat punktowy. Innym przykładem może być sugestia włączenia kompresji danych w ustawieniach serwera, która będzie dwa razy ważniejsza od optymalizacji samych obrazów na stronie. Tabela wyników jest ustalona względem najważniejszych cech jakie należy wprowadzić (według Google), lecz w drugiej kolejności należy zwracać uwagę na odpowiednie wagi, które wyświetlają się po wykonaniu testu.

## GTMetrix

Zgodnie z zasadą programowania zwinnego, kolejną iteracją w projekcie Snail było zaimplementowanie skryptu, który niezależnie od miejsca przebywania użytkownika (w tym przypadku, osoby, która dokonuje testu) wykona analizę strony pod względem długości ładowania się elementów, czasu odpowiedzi, dokona pomiaru rozmiaru strony oraz utworzy wykres typu *waterfall*, z wynikami aby były one zrozumiałe i czytelne dla odbiorcy.

GTmetrix jest projektem rozwijanym przez wyspecjalizowanych programistów w firmie „Gossamer Threads” w Vancouver. Firma posiada ponad 19 lat doświadczenia w pracy z technologiami webowymi. Serwis ten nie posiada swojego własnego systemu oceny strony, korzysta z PageSpeed Insight oraz narzędzia Yslow do prezentowania wyników użytkownikowi. Serwis ten udostępnia jednak API do skryptu, który spełnia założenia analizy strony, oraz umożliwia implementację na własnej stronie internetowej, co sprawiło, że został on wykorzystany w Snail Project. Aby móc skorzystać z tego rozwiązania należało dokonać rejestracji na stronie [www.gtmetrix.com](http://www.gtmetrix.com) oraz wygenerować indywidualny „API Key” dla developera.

Skrypt pobiera adres z paska na głównej stronie Projektu, a następnie przesyła go na własny serwer i dokonuje weryfikacji, przed nieuprawnionym użyciem. Następnie żądanie serwera WWW jest przesyłane przez obiekt XMLHttpRequest do skryptu na serwerze Projektu Snail. JavaScript przetwarza odpowiednio ten obiekt prezentując zwrócony wynik w formie graficznej w odpowiednim miejscu na stronie. Cały test jest wykonywany przez „user-agent: Mozilla/5.0 (X11; Linux i686 on x86\_64; rv:25.0) Gecko/20100101 Firefox/25.0”

## Moderniz

Korzystanie z nowych technologii internetowych może wydawać się świetną zabawą, dopóki nie muszą wspierać przeglądarek, które często nie obsługują najnowszych trendów, celowo bądź do kolejnej aktualizacji przeglądarki. Modernizr to biblioteka, która ułatwia pisanie warunkowego kodu JavaScript i CSS do obsługi użycia w każdej sytuacji. Modernizr dokonuje sprawdzenia obsługi nowych trendów (np. obsługi cieni, responsywności, HTML Audio/Video) przez przeglądarkę a następnie tworzy obiekt JavaScript z wynikami, i dodaje klasy do elementu HTML.

Biblioteka może testować obsługę, funkcjonalności HTML5, CSS3 dla przeglądarki gdzie została uruchomiona, dlatego wynik jaki jest zwracany na stronie Snail Project dotyczy przeglądarki przez, którą użytkownik wszedł na stronę projektu.

## Walidacja HTML przez W3C

Zgodnie z duchem Agile, po konsultacjach z Profesorem Leszkiem Borzemskim ustalono następną iterację powstawania projektu o nazwie Snail jaką jest sprawdzenie poprawności kodu HTML testowanej strony z wykorzystaniem Validatora konsorcjum W3C.

W3C to skrót od World Wide Web Consortium, które zajmuje się ustanawianiem standardów sieci WWW takich jak skrypty, style, znaczniki HTML itp.

Walidator, jest dostępny na stronie internetowej konsorcjum jednak możliwe jest także wykorzystanie go w projektach bez implementowania całego systemu testującego. Działa to na zasadzie zadania zapytania AJAX serwera konsorcjum, a następnie odpowiednie sparsowanie wyniku, który jest zwracany w formacie JSON. Po pobraniu adresu testowanej strony z odpowiedniego miejsca na stronie głównej projektu Snail, wysyłane jest odpowiednie żądanie a po jego przetworzeniu wynik jest wyświetlany w odpowiednim miejscu na stronie.

Wynikiem jest lista błędów (o ile takie wystąpią), z krótkim komentarzem do problemu jaki nastąpił, oraz linia i kolumna w kodzie testowanej strony, w której dany błąd się znajduje.

## Unikanie zbędnych przekierowań

Przekierowania URL są stosowane aby user-agent został przeniesiony w inne miejsce niż nastąpiło poprzednie wywołanie strony. Najczęściej stosowana technika przy skracaniu adresów internetowych oraz przenoszeniu serwisów webowych pod inny adres.

Przekierowanie za pomocą metatagów „Refresh”:

<html><head>  
 <meta http-equiv="Refresh" content="0; url=http://www.pwr.edu.pl/" />  
</head><body>  
 <p>Nowy adres <a href="http://www.wiz.pwr.edu.pl/">link</a>!</p>  
</body></html>

Liczba 0 w atrybucie „content” określa liczbę sekund w czasie których ma być wyświetlana strona.

Drugim sposobem jest przekierowanie z wykorzystaniem JavaScript:

<script language="JavaScript" type="text/javascript">   
location.href="pwr.php";   
</script>

Trzeci sposób to użycie ramek:

<frameset rows="100%">  
 <frame src="http://www.pwr.edu.pl/">  
</frameset>  
<noframes>  
 <body>Please follow <a href="http://www.wiz.pwr.edu.pl/">link</a>!</body>  
</noframes>

Dzięki temu rozwiązaniu strona spod nowego adresu wyświetli się, ale w pasku przeglądarki adres się nie zmieni.

Przekierowanie w PHP:

<?php  
 header('HTTP/1.1 301 Moved Permanently');  
 header('Location: http://www.wiz.pwr.edu.pl/');  
 header('Connection: close');  
 exit();  
?>

Powyższy kod w języku PHP wygeneruje nagłówek:

HTTP/1.1 301 Moved Permanently  
Location: http://www.wiz.pwr.edu.pl/  
Content-Type: text/html  
Content-Length: 174

Inne kody stanu 3xx protokołu HTTP jakie można wykorzystać podczas przekieorwania:

* 300 – Wiele możliwości, przeważnie wykorzystywany przy stronach wielojęzycznych
* 301 – stałe przekierowanie
* 302 – przekierowanie tymczasowe (znaleziony)
* 303 – zobacz inne
* 307 – przekierowanie tymczasowe

Z punktu widzenia SEO przekierowanie 301 jest ważne, ponieważ przekierowuje ono siłę czy też wagę linków ze starej na nową stronę. Dzięki czemu pozycja w Google nie spada po przeniesieniu strony pod nowy adres.

Ostatnim sposobem jest przekierowanie za pomocą .htaccess:

Poniższą linię kodu należy umieścić w pliku .htaccess:

Redirect 301 / http://www.example.com/

## Włączenie kompresji danych

Korzystanie z kompresji przesyłanych danych pomiędzy klientem a serwerem ma na celu bardziej efektywne wykorzystanie dostępnej przepustowości łącza. Włączenie kompresji HTTP zapewnia krótszy czas transmisji danych pomiędzy przeglądarkami internetowymi a serwerami, na których dane się znajdują. Kompresowane mogą być dane zarówno statyczne jak i te generowane dynamicznie przez aplikacje.

Nagłówek Accept-Encoding wskazuje na to, że przeglądarka WWW obsługuje kompresję, natomiast nagłówek Content-Encoding jest stosowany przez serwer (jeśli obsługuje kompresję), który określa skompresowaną odpowiedź.

Najpopularniejszą metodą kompresji danych jest użycie formatu *gzip*. Jest to otwarty format, czyli taki, który nie został objęty żadnymi patentami oraz prawem ograniczającym jego wykorzystywanie. Mniej popularnym i zarazem mniej efektywnym formatem kompresji danych jest *deflate*. Opisane poniżej wyniki testów dziesięciu najpopularniejszych serwisów webowych w Polsce pokazały, że najczęściej stosowaną metodą kompresji przesyłanych danych jest użycie *gzip*.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcja kompresji | Typ danych | Podstawowa konfiguracja | Metoda konfiguracji |
| Format kompresowanych plików | Statyczny | .txt, .htm, and .html | Metabase |
|  | Dynamiczny | .exe, .dll, and .asp | Metabase |
| Format kompresji | Statyczny | gzip oraz deflate | Metabase |
|  | Dynamiczny | gzip oraz deflate | Metabase |
| Poziom kompresji | Statyczny | 10 | Metabase |
|  | Dynamiczny | 0 | Metabase |

## Wykorzystanie pamięci podręcznej przeglądarki

Zapisywanie zasobów statycznych w pamięci podręcznej przeglądarki pomaga oszczędzić czas użytkownikowi, który wielokrotnie odwiedza daną witrynę. Nagłówki zapisywania w pamięci podręcznej powinny dotyczyć wszystkich zasobów statycznych, które można zapisać, a nie tylko ich niewielkich podzbiorów na przykład obrazów. Zasoby, które można zapisać w pamięci podręcznej to pliki JS i CSS, pliki obrazów oraz inne pliki zawierające obiekty binarne, a w tym, multimedia, pliki PDF itp. Sam HTML nie jest statyczny i nie powinien być domyślnie uwzględniany w zapisywaniu w pamięci podręcznej, ale można zmniejszyć jego rozmiar używając narzędzi do minimalizowania rozmiaru plików .html.

Należy włączyć funkcję zapisywania materiałów z serwera w pamięci podręcznej przeglądarki. Czas przechowywania zasobów statycznych w pamięci podręcznej powinien wynosić co najmniej tydzień. Zasoby zewnętrzne, takie jak reklamy czy widżety, powinny być przechowywane w pamięci podręcznej co najmniej jeden dzień.

Dla wszystkich zasobów zapisywanych w pamięci podręcznej Google zaleca następujące ustawienia:

„Ustaw wartość Expires na co najmniej tydzień, a najlepiej na rok w przód. (Preferujemy ustawienie Expires zamiast Cache-Control: max-age, ponieważ jest częściej obsługiwane). Nie ustawiaj wartości dłuższej niż rok w przód, ponieważ będzie to naruszeniem wytycznych RFC.

Jeżeli wiesz dokładnie, kiedy dane zasoby się zmienią, możesz ustawić krótszy okres ważności. Jeżeli jednak wiesz tylko, że „wkrótce mogą się zmienić”, ale bez konkretów, ustaw długi okres ważności i użyj odcisku cyfrowego w URL-u (technika opisana poniżej).”

Nagłówki *Expires* i *Cache-Control: max-age*, określają okres, w którym przeglądarka może używać zasobów zapisanych w pamięci podręcznej bez sprawdzania, czy na serwerze WWW nie znajduje się ich nowsza wersja. Gdy zostaną ustawione, po pobraniu zasobu przeglądarka nie będzie wysyłała żądań GET do upływu terminu ich ważności, osiągnięcia przez nie maksymalnego wieku bądź do czasu opróżnienia pamięci podręcznej przez użytkownika.

Nagłówki *Last-Modifed* i *ETag, o*kreślają sposób decydowania przez przeglądarkę, czy pliki są takie same, na potrzeby zapisania w pamięci podręcznej. W przypadku nagłówka Last-Modified takim wyznacznikiem jest data. W przypadku nagłówka ETag wskazówką może być dowolna wartość jednoznacznie identyfikująca zasób (standardowo stosowane są wersje pliku i identyfikatory numeryczne zawartości). Last-Modified to „słaby” nagłówek, ponieważ przeglądarka określa, czy pobrać dany element z pamięci podręcznej czy nie w oparciu o mechanizmy heurystyczne. Te nagłówki pozwalają przeglądarce skutecznie aktualizować zasoby zapisane w pamięci podręcznej poprzez wysyłanie warunkowych żądań GET, gdy użytkownik wyraźnie ponownie załaduje stronę. Warunkowe żądania GET nie zwracają pełnej odpowiedzi, jeżeli zasób na serwerze się nie zmienił, i z tego względu charakteryzują się krótszym czasem oczekiwania niż pełne żądanie GET.

Ważne jest, by określić jeden z nagłówków Expires i Cache-Control max-age, oraz jeden z Last-Modified i ETag, w odniesieniu do wszystkich zasobów, które można zapisać w pamięci podręcznej. Nie ma potrzeby określać zarówno nagłówka Expires, jak i Cache-Control: max-age, albo i Last-Modified, i ETag.

## Skrócenie czasu odpowiedzi serwera

Czas odpowiedzi serwera to czas wczytywania kodu HTML niezbędnego do rozpoczęcia renderowania strony z serwera, po odjęciu czasu oczekiwania między Google a Twoim serwerem. Mogą wystąpić różnice pomiędzy poszczególnymi cyklami, ale nie powinny one być zbyt duże. W rzeczywistości duża zmienność czasu odpowiedzi serwera może sugerować problem z wydajnością.

Czas odpowiedzi serwer powinien wynosić mniej niż 200 ms. Spowolnienie może być spowodowane przez wiele czynników: mało wydajne algorytmy aplikacji lub zapytania do bazy danych, powolny routing, architekturę, biblioteki, nadmierne wykorzystanie zasobów obliczeniowych procesora bądź pamięci. Poprawiając czas odpowiedzi serwera, należy wziąć wszystkie te czynniki pod uwagę. Po wykonaniu testów oraz interpretacji wyników należy wprowadzać stopniowo różne rozwiązania mające na celu poprawę wydajności. Wykorzystanie różnych popularnych rozwiązań takich jak CMS, MVP, MVC itp. Skutkuje również opisaniem problemów z wydajnością nawet e specyfikacji technicznej danego rozwiązania.

## Minimalizowanie plików CSS

Minimalizowanie rozmiaru plików polega na usuwaniu zbędnych bajtów, na przykład dodatkowych odstępów, komentarzy, znaków łamania wiersza i wcięć. Kompaktowy kod CSS, HTML i JavaScript pomaga skrócić czas pobierania tych plików, ich analizowania oraz wykonywania. Dodatkowo w przypadku CSS i JavaScriptu możliwe jest dalsze zmniejszanie rozmiaru pliku poprzez zmianę nazw zmiennych, stosowanie metodyki DRY (Ang. Don’t Reapeat Yourself).

W celu zmniejszenia rozmiaru kodu CSS można użyć narzędzi takich jak YUI Compressor i cssmin.js, które są polecane przez Google. Istnieje także sporo narzędzi programistycznych, które już w trakcie tworzenia kodu HTML, styli CSS czy skryptów JavaScript dbają i automatyczne modyfikowanie kodu to jak najmniejszego rozmiaru pliku. Są to takie narzędzia jak Koala czy Codekit.

## Minimalizowanie plików HTML

Aby zmniejszyć rozmiaru kodu HTML można wykorzystać rozszerzenie PageSpeed Insights dla przeglądarki Chrome aby wygenerować kompaktową wersję kodu HTML. Należy uruchomić analizę HTML danej strony oraz wybrać odpowiednią regułę zmniejszania kodu.

## Minimalizowanie plików JavaScript

Narzędzie sugerowane przez Google do zmniejszenia ich rozmiaru celem szybszego przesyłania poprzez sieć są np. Closure Compiler, JSMin lub YUI Compressor. Tak jak wspomniano przy opisie minimalizowania rozmiaru plików kaskadowych arkuszy styli, można użyć także narzędzi typu Koala czy Codekit, które „w locie” dbają o odpowiednie formatowanie kodu i zmniejszają jego rozmiar.

## Eliminowanie blokujących skryptów

Zanim przeglądarka wyrenderuje widok strony, w pierwszej kolejności musi dokonać jej analizy. Jeżeli w jej trakcie napotka blokujący zewnętrzny skrypt, musi go zinterpretować zanim dokona dalszej analizy kodu samej strony. Każdy taki skrypt oznacza dodatkowe połączenia z serwerem w celu pobrania kodu JavaScript i przesłanie wyniku jego zawartości do klienta WWW, co znacznie pogarsza User eXperience.

Krótkie skrypty JavaScript wymagane do wyrenderowania części strony widocznej na ekranie powinny zostać umieszczone pomiędzy znacznikami HTML, natomiast pobranie kodu JavaScript niezbędnego do realizacji dodatkowych funkcji takich jak ruchome elementy, pokazy slajdów, itp. powinno odbywać się dopiero po dostarczeniu treści.

Krótkie skrypty zewnętrzne możesz włączyć bezpośrednio do dokumentu HTML. Wbudowanie niewielkich plików pozwoli przeglądarkom kontynuować renderowanie strony.

Aby zapobiec blokowaniu wczytywania stron przez kod JavaScript, przy wczytywaniu zasobów zaleca się użycie w kodzie HTML atrybutu *async*.

Na przykład:

<script async src="my.js">

Rozwiązanie to jednak nie sprawdza się jeżeli w kodzie JavaScript używana jest komenda document.write. Należy wtedy przepisać kod w taki sposób aby nie wykorzystywać document.write do wpisywania tekstu do dokumentu HTML. Przy asynchronicznym wczytywaniu skryptów, należy z zwrócić uwagę, by aplikacja uwzględniała wczytywanie ich w odpowiedniej kolejności, jeżeli jest to kod JavaScript zależący od siebie nawzajem.

Wykorzystanie różnych bibliotek JavaScript, na przykład jQuery, umożliwia wzbogacenie strony o dodatkowe funkcje takiej jak animacje, przejścia, zmianę styli CSS i inne efekty. Większość takich skryptów można zastosować już po wyrenderowaniu części widocznej na ekranie dla użytkownika. Ponieważ na doświadczenia użytkownika, oraz czas ładowania się strony ma to duży wpływ, warto stosować tą radę w większości projektów aplikacji webowych.

## Optymalizacja obrazów

Stosowanie jak najmniejszych rozmiarów obrazów, ma na celu skrócenie czasu oczekiwania na załadowanie przez użytkownika. Właściwe sformatowanie i skompresowanie obrazów może zaoszczędzić wiele bajtów danych. Skraca to czas wczytywania strony przy wolnym połączeniu, i jest szczególnie cenione przez użytkowników „mobilnych”, ze względu na transfer danych i opóźnienia sieci.

Warto przeprowadzić zarówno podstawową, jak i zaawansowaną optymalizację wszystkich obrazów. Podstawowa optymalizacja to wycinanie zbędnych obszarów, maksymalna akceptowalna redukcja głębi kolorów, usuwanie komentarzy do obrazów i korzystanie z odpowiedniego formatu plików. Te czynności można wykonać przy pomocy dowolnego programu do edycji obrazów, takiego jak GIMP czy Adobe Photoshop.

W sieci jest dostępnych jest wiele narzędzi umożliwiających bezstratną (tzn. bez szkody dla jakości obrazów) kompresję plików JPEG i PNG. Są to zarówno narzędzie darmowe (jpegtran, optipng, czy Gimp) oraz komercyjne (Adobe Photoshop – potężne narzędzie służące zarówno do edycji jak i tworzenia grafiki, nie tylko na potrzeby internetu)

* Pliki PNG są niemal zawsze lepsze od plików GIF, jednak niektóre starsze wersje przeglądarek mogą tylko częściowo obsługiwać pliki PNG charakteryzujące się obsługą kanału alfa.
* Pliki GIF stają się coraz mniej popularne w sieci, ponieważ bezcelowe jest ich wykorzystywanie do grafik przedstawiających statyczne obrazy, ze względu na niską jakość oraz większy rozmiar w porównaniu z formatem PNG.
* Formatu JPG najlepiej używać w przypadku wszystkich plików, które nie wymagają przezroczystości czyli przedstawiające zdjęcia, portrety, krajobrazy.

## Priorytetyzacja widocznej treści

W momencie gdy wymagana ilość danych przekracza początkowy rozmiar okna przeciążenia, konieczna jest dodatkowa wymiana informacji między serwerem a przeglądarką użytkownika co znacząco wpływa na czas odpowiedzi. W przypadku sieci o długim czasie oczekiwania, na przykład sieci komórkowych takich jak 3G, 4G, może to powodować poważne opóźnienia wczytywania stron.

Krok jaki należy wykonać aby strony wczytywały się szybciej, to ograniczenie rozmiaru plików zawierających znaczniki HTML, obrazy, style CSS czy kod JavaScript, które są wymagane do wyrenderowania widocznej dla użytkowników części strony.

Już na etapie projektowania serwisu webowego można zastosować pewne reguły, które będą miały wpływ na jego wydajność:

* Należy zaplanować strukturę strony w taki sposób aby jej najważniejsza część, widoczna na ekranie, wczytywała się w pierwszej kolejności. Jeżeli HTML wczytuje zewnętrzne skrypty np. jQuery przed główną zawartością, najważniejsze jest aby zmień kolejność w taki sposób, aby główna zawartość wczytywała się jako pierwsza.
* Należy zmniejszyć ilość danych wykorzystywanych przez zasoby serwisu.
* Po przeprojektowaniu witryny pod kątem poprawnego wyświetlania na różnych urządzeniach i wczytywania najważniejszej zawartości na początku zmniejsz ilość danych niezbędnych do wyrenderowania stron, posługując się następującymi metodami:
* Należy stosować style CSS zamiast obrazów tam, gdzie jest to możliwe np. jako tło dla serwisu, czy przycisków itp.
* Należy włączyć kompresję danych używając np. gzip
* Wczytywanie skryptów należy umieścić na końcu kodu HTML
* Należy zoptymalizować fonty używane w całym serwisie webowym

## Optymalizacja fontów

Aby nie narazić serwisu webowego na opóźnienia związane ze zbyt długim wczytywaniem fontów należy odpowiednio je zoptymalizować. Dzięki temu doświadczenia użytkowników będą lepsze i pozwolą odczuć pozytywne skojarzenia z daną stroną. Odpowiednia użycie czcionek pozwala zwiększyć wygodę użytkowników: rozszerzyć rozpoznawalność marki, polepszyć czytelność, funkcjonalność i sprawność wyszukiwania, zapewniając przy tym możliwość zmiany skali i pracę przy wielu różnych rozdzielczościach i formatach ekranu, od wielocalowych monitorów po wyświetlacze urządzeń przenośnych.

Przede wszystkim nie należy używać zbyt wielu krojów pisma na jednej czy kilku stronach serwisu. Pozwoli to na skrócenie czasu oczekiwania użytkowników na wyświetlenie danej treści.

Należy używać podzbiorów fontów. W przypadku wielu czcionek można wydzielić podzbiory, rozdzielić je na wiele zakresów Unicode i dostarczyć tylko glify konkretnie wymagane na danej stronie – pozwala to ograniczyć rozmiar pliku i zwiększyć prędkość pobierania zasobów. Przy określaniu podzbiorów należy mieć na uwadze optymalizację czcionek pod kątem ponownego użycia. Jeżeli na kilku podstronach serwisu wykorzystywane są różne znaki alfabetu łacińskiego czy też cyrylicy, dobrze dobrana czcionka będzie wszystkie te znaki obsługiwać. Czcionki należy przesyłać w formatach dostosowanych do każdej z przeglądarek: każda czcionka powinna być dostarczana w formatach WOFF2, WOFF, EOT i TTF. Ważne jest aby mieć pewność, że do formatów EOT i TTF stosowana będzie kompresja GZIP, ponieważ formaty te nie są domyślnie kompresowane, co może skutkować wydłużonym czasem oczekiwanie na wyrenderowanie strony.

Jedną z najważniejszych czynności podczas optymalizacji fontów jest określenie zasad ponownej walidacji i optymalnego buforowania, poniważ czcionki to zasoby statyczne. Są rzadko aktualizowane dlatego warto zmienić ustawienia serwera tak aby zapewniał długi okres ważności max-age i wysyłał tokeny walidacji, co umożliwi efektywne ponowne używanie czcionek na różnych stronach w witrynie.

@font-face jest regułą CSS umożliwiającą określenie lokalizacji konkretnego zasobu czcionki, jej stylu i kodów Unicode, dla których powinien obowiązywać. Połączenie takich deklaracji reguł @font-face można wykorzystać do utworzenia rodziny czcionek, dzięki której przeglądarka może określić, które czcionki trzeba pobrać i zastosować na bieżącej stronie.

W każdej deklaracji reguły @font-face określa się nazwę rodziny czcionek, dzięki której może ona pełnić rolę logicznej grupy wielu deklaracji, własności czcionki, takie jak styl, grubość, rozciągnięcie i deskryptor źródłowy zawierający uporządkowaną według ważności listę lokalizacji tego zasobu czcionki.

@font-face {  
font-family: 'PWR Font';  
font-style: normal;  
font-weight: 450;  
src: local('PWR Font'),  
url('/fonts/pwr.woff2') format('woff2'),   
url('/fonts/pwr.woff') format('woff'),  
url('/fonts/pwr.ttf') format('ttf'),  
url('/fonts/pwr.eot') format('eot');  
}