UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Dominika Pienczyn
nr albumu: 236 389

System rejestrowania usterek i napraw

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
dr W. Bzyl

Gdańsk 2017

Streszczenie

Praca przedstawia aplikację internetową o nazwie „Awaria”, która jest systemem rejestrującym usterki oraz naprawy sprzętu. Została stworzona w grupie 3-osobowej (Kamil Pek, Marcin Dawidowski, Dominika Pienczyn). W projekcie wykonałam szatę graficzną, tak zwany frontend aplikacji. Podczas tworzenia skupiałam się na tym, aby strona dostosowana była do urządzeń mobilnych, co obecnie jest standardem. Zajmowałam się również przeprowadzaniem testów strony, które uświadomiły mi jakie elementy należy jeszcze poprawić. Zaimplementowałam elementy, które poprawiły funkcjonowanie aplikacji: automatyzacja powiadomień, filtracja danych, uaktywnienie resetowania hasła oraz e-maili powitalnych podczas rejestracji.

Do stworzenia aplikacji wykorzystano system Linux, bazę danych PostgreSQL, framework Ruby on Rails oraz Bootstrap, a następnie wdrożono ją na serwer Heroku.

Wgląd do szerszej dokumentacji możliwy jest pod adresem https://github.com/dpienczyn/awaria1 [1] i przechowywany jest w repozytorium GitHuba razem z kodem aplikacji.

Aplikacja dostępna jest pod adresem https://awaria-system.herokuapp.com/.

Słowa kluczowe

aplikacja internetowa, aplikacja mobilna, ruby on rails, system rejestrowania, naprawa, bootstrap, interfejs graficzny użytkownika

[1] Dostęp do wdrożonego projektu, jest możliwy za pomocą następujących danych: konto administartora: ***awaria.kontakt@gmail.com*** z hasłem do strony i e-maila ***awaria1234567***, konto pracownika: ***pracownik.awaria@gmail.com*** z hasłem do strony i e-maila ***pracownikawaria***, konto użytkownika: ***user.awaria@gmail.com*** z hasłem do strony i e-maila ***userawaria***

Spis treści

Wprowadzenie

Graficzny interfejs użytkownika to ogólne określenie sposobu prezentacji informacji przez komputer oraz interakcji z użytkownikiem, polegające na rysowaniu i obsługiwaniu widżetów. Głównym aspektem tworzonego interfejsu jest optymalne rozmieszczenie elementów na stronie, zgodnie z ergonomią pracy oraz szatą graficzną, która pełni ważną rolę dopełniającą i pomaga prowadzić użytkowników przez strukturę informacji prezentowaną w serwisie. Kolejnym istotnym elementem jest to, aby aplikacja była responsywna. W dzisiejszym świecie korzystamy z różnego typu urządzeń nośnych tzn tabletów, smartfonów itp, dlatego tak ważne jest to, aby aplikacja umiała dopasować się do każdej rozdzielczości i każdego nośnika automatycznie. Od prawidłowo zaprojektowanego interfejsu zależy sukces strony, wygoda, intuicyjność oraz odpowiednia funkcjonalność. W mojej pracy zamierzam przedstawić to jak w prosty i skuteczny sposób można zmienić szatę graficzną aplikacji oraz kilka innych elementów, które umożliwiły zwiększenie funkcjonowania danego systemu.

ROZDZIAŁ 1

Wykorzystane technologie

1.1. Ruby on Rails

W utworzonym projekcie wykorzystano język Ruby wersje 2.3 oraz framework Rails wersja 5.0.0. Ruby on rails jest frameworkiem open source i wykorzystuje się go do tworzenia aplikacji webowych. Napisany został z wykorzystaniem architektury MVC *(ang. Model-View-Controller)*.

Modele *(ang. Model)* reprezentują dane aplikacji i służą do manipulowania tymi danymi. Model odpowiada jednej tabeli w bazie danych.

Widoki *(ang. View)* tworzą interfejs użytkownika aplikacji i służą do dostarczania danych do przeglądarki internetowej, bądź innego urządzenia. Są to pliki zawierające kod w języku Ruby i HTML.

Kontrolery *(ang. Controller)* w nich znajduje się cała logika aplikacji. Mają za zadanie połączyć model i widok. Odpowiadają za przetwarzanie żądań przychodzących z przeglądarki internetowej, za pozyskiwanie danych z modeli oraz przekazanie ich do widoków w celu ich reprezentacji.

1.2. Bootstrap

Bootstrap, framework CSS, zawiera wiele narzędzi, które przydają się podczas tworzenia interfejsu graficznego stron oraz aplikacji internetowych. Jest bardzo prosty w obsłudze, nie potrzeba wiele umiejętności żeby zacząć z nim pracować. Wystarczy podstawowa wiedza, by rozpocząć tworzenie czegoś własnego. Bootstrap bazuje głównie na gotowych rozwiązaniach HTML i CSS. Może być używany do stylizacji m.in. przycisków, formularzy, wykresów, nawigacji oraz innych komponentów wyświetlanych na stronie. Framework korzysta również z języka JavaScripts. By zacząć korzystać z platformy Bootstrap, należy w pliku Gemfile dodać gem, który odpowiedzialny jest za odpowiednie funkcjonowanie frameworka.

```
gem 'bootstrap-sass', '~> 3.3.7'
```

Listing 1.1. Plik *Gemfile*

Bootstrap jest platformą stylów CSS, więc każdy kod powinien być zapisany w pliku o dowolnej nazwie z rozszerzeniem **css.scss*. Pliki musza być umieszczone w przeznaczonym do tego katalogu */app/assets/stylesheets*.

Wymagane są również referencje do skryptów JavaScripts, które wykorzystywane są przez platformę.

```
//= require bootstrap-sprockets
//= require bootstrap
```

Listing 1.2. Referencje do skryptów JavaScripts

ROZDZIAŁ 2

Implementacja

2.1. Interfejs graficzny użytkownika

Pierwszy interfejs graficzny został stworzony w latach 70, XX wieku przez firme Xerox. Służy on do komunikowania się człowieka z oprogramowaniem komputera, wykorzystując obiekty wyświetlane na monitorze w trybie graficznym. Interfejs graficzny określa wygląd oraz funkcjonalność obiektów.

Składa sie zazwyczaj z:

- menu
- wyświetlanych na ekarnie ikonek, które oznaczają obiekty i polecenia
- okien wyświetlanych na ekranie
- funkcji dialogowych np. zapytań potwierdzajacych usunięcie lub zmiane wydanego polecenia

Projektowanie responsywne *(ang. responsive)* [1] w polskim tłumaczeniu oznacza to coś czułego, wrażliwego i ma na celu tworzenie stron internetowych, które dynamicznie adaptują się do swojego środowiska. Dzięki temu strona ma płynną, elastyczną strukturę i dopasowuje się do dowlonego urządzenia: smartfona, tableta, telewizora lub komputera. Ponadto strony responsywne, są kompatybilne z interfejsami dotykowymi urządzeń mobilnych.

2.2. Panel logowania

Panel logowania użytkownika jest dla aplikacji istotnym elementem, ponieważ odbiorca wchodząc na strone zostaje automatycznie do niego przekierowany. Dlatego zadbałam o to, by strona w projekcie była atrakcyjna graficznie, a zarazem prosta w obsłudze. Główną zasadą, której trzymałam się podczas tworzenia frontendu była estetyka oraz niezróżnicowany wygląd aplikacji.

Wygląd panelu logowania przed ostylowaniem kodem HTML wyglądał przeciętnie, zawierał tylko jedną klase *div class="field"*. Format panelu został przedstawiony poniżej:

Rysunek 2.1. Panel logowania przed ostylowaniem kodem HTML

```
<div class="field">
  <%= f.label :email %><br/>
  <%= f.email_field :email, autofocus: true %>
</div>
<div class="field">
  <%= f.label :password %><br/>
  <%= f.password_field :password, autocomplete: "off" %>
</div>
```

Listing 2.1. Kod odpowiedzialny za panel logowania

Rysunek 2.2. Ostylowany panel logowania

```
<div class="container">
<div class="row">
<div class="col-xs-12 col-sm-8 col-md-6 col-sm-offset-2
   col-md-offset-3">
<fieldset>
<h2 class="text-center">Zaloguj sie</h2>
<hr class="colorgraph">
<div class="form-group">
  <%= f.email_field :email, autofocus: true, class:
     "form-control input-lg", placeholder: "Adres email" %>
</div>
<div class="form-group">
  <%= f.password_field :password, autocomplete: "off",
     class: "form-control input-lg", placeholder: "Haslo"
     %>
</div>
```

Listing 2.2. Kod obudowany kodem HTML

W kodzie zastosowano poniższą klase ***div class=”col-xs-12 col-sm-8 col-md-6 col-sm-offset-2 col-md-offset-3”***, dzięki której uzyskujemy efekt strony responsywnej [10] i możliwej do obejrzenia na dowolnym wyświetlaczu:

- *col-xs* stosuje się dla bardzo małych wyświetlaczy (szerokość ekranu mniejsza niż 768 pikseli)
- *col-sm* stosuje się dla małych wyświetlaczy (szerokość ekranu większa równa 768 pikseli)
- *col-md* stosuje się dla średnich wyświetlaczy (szerokość ekranu wieksza, równa 992 pikseli)

Klasy służące do przesuwania kolumn względem siebie, służą do zwiększenia odstępu po lewej stronie kolumny:

- *col-sm-offset-2* w przypadku stron dla małych wyświetlaczy, jeżeli chcemy przenieść kolumnę obejmującą dwie kolumny w prawą strone
- *col-md-offset-3* w przypadku stron dla średnich wyświetlaczy, jeżeli chcemy przenieść kolumnę obejmującą trzy kolumny w prawą strone

Aby była możliwość tworzenia rzędów, to ***div class="col-xs-12 col-sm-8 col-md-6 col-sm-offset-2 col-md-offset-3"*** musi zostać umieszczony w kontenerze *div class="container"* oraz wewnątrz kontenera musi znaleźć się klasa *div class="row"*.

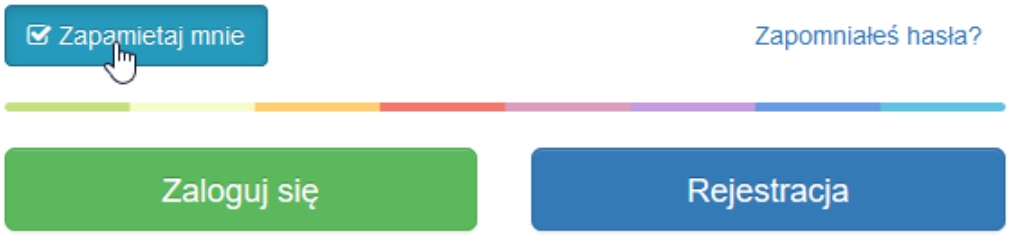
Klasa *form-group* służy do tworzenia formularzy, natomiast klasa *class: "form-control input-lg"* znajdująca się pomiedzy znacznikami kodu Ruby sprawia, że element rozciaga się na całą dostępną szerkość ekranu.

W pliku CSS odnosimy sie do styli klasy *hr class="colorgraph"* w której określamy rozmiar wysokości obszaru *(height)*, cechy górnego obramowania *(border-top)* lub ustawiamy kolor tła elementu *(background)*.

```
.colorgraph {
border-top: 70px;
height: 5px;
border-top: 0;
background: #c4e17f;
border-radius: 5px;
}
```

Listing 2.3. Kod CSS dla panelu logowania

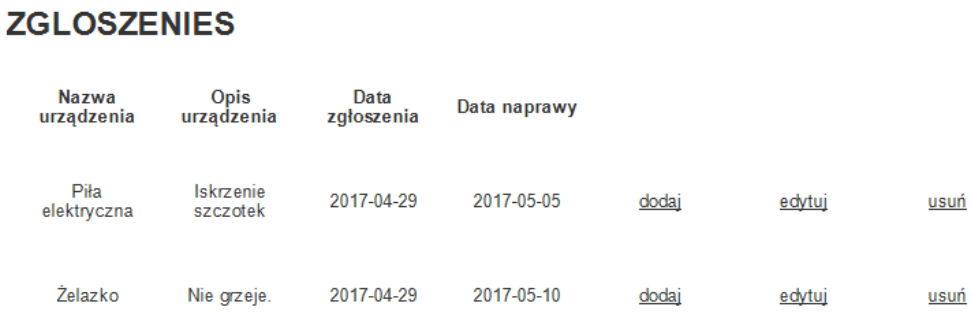
W panelu logowania wykorzystano kod JavaScript obsługujący element *checkbox*.



Rysunek 2.3. Checkbox w panelu logowania

2.3. Tabela zgłoszeń

Tabela zgłoszeń, po wygenerowniu kodu nie posiadała przycisków. Jej kolumny i wiersze nie były oddzielone linią w żaden sposób. Pod wpływem zmniejszania obrazu, tabela nie dopasowywała się do narzuconej jej rozdzielczości, czego efektem był brak możliwości odczytu niektórych danych, znajdujących sie w tabelce.



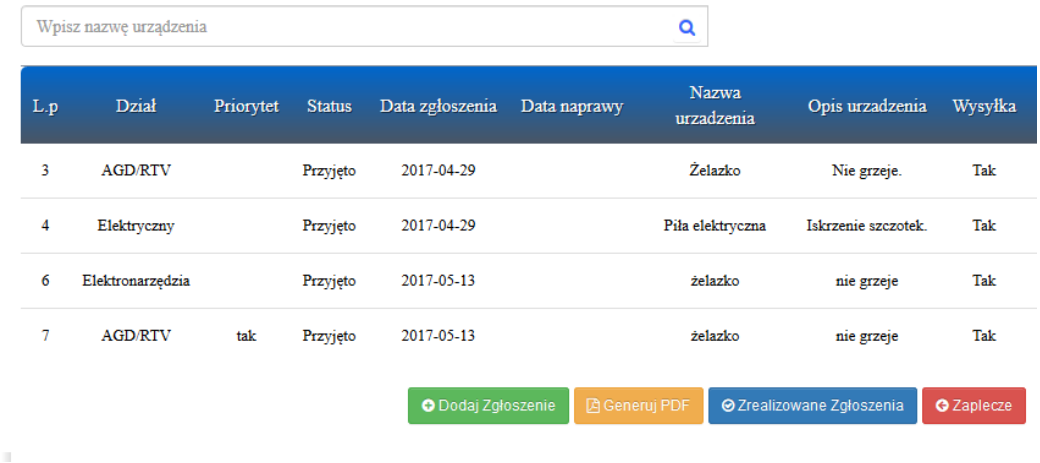
Rysunek 2.4. Tabela zgłoszeń po wygenerowaniu kodu

```
<%= link_to('Usun', zgloszeny, method: :delete,
   data: { confirm: 'Jestes pewny?' }, class:'btn
   btn-danger btn-xs') %>
```

Listing 2.4. Element klikalny w tabeli zgłoszeń

Do stworzenia przycisku należało użyć klasy *btn*. W klasie widzimy również *btn-danger*, dzięki czemu element klikalny jest koloru czerwonego oraz *btn-xs* co określa wielkość przycisku w tym przypadku jest to element bardzo mały.

Do stworzenia tabelki wykorzystano klasę *div class="table-responsive"*, dzięki której tabela zamyka się w kontenerze z poziomym paskiem przewijania podczas mniejszych rozdzielczości.



Rysunek 2.5. Ostylowana kodem HTML tabela zgłoszeń

ROZDZIAŁ 3

Elementy funkcjonalne systemu

3.1. Uaktywnienie przypominania hasła na e-mail

Konfiguracji przpominania hasła na adres e-mail użytkownika [9], dokonuje się w pliku konfiguracyjnym *config/environments/production.rb*. Ze względu na wdrożenie aplikacji na serwer Heroku, wykorzystałam dodatkowy element tej platformy do korespondencji z użytkownikiem o nazwie Sendgrid.

```
config.action_mailer.smtp_settings = {
user_name: ENV['SENDGRID_USERNAME'],
password: ENV['SENDGRID_PASSWORD'],
domain: 'awaria-system.herokuapp.com',
address: 'smtp.sendgrid.net',
port: 587,
authentication: :plain,
enable_starttls_auto: true
}
```

Listing 3.1. Konfiguracja pliku *production.rb*

W pliku, dane na temat hasła oraz nazwy użytkownika, zostały ukryte i dostępne są tylko dla użytkowników do tego uprawnionych. Wszystkie dane znajdują się na platformie Heroku. W sekcji *domain* deklarujemy adres url strony pod który została wdrożona.

3.2. Uaktywnienie e-maili powitalnych podczas rejestracji

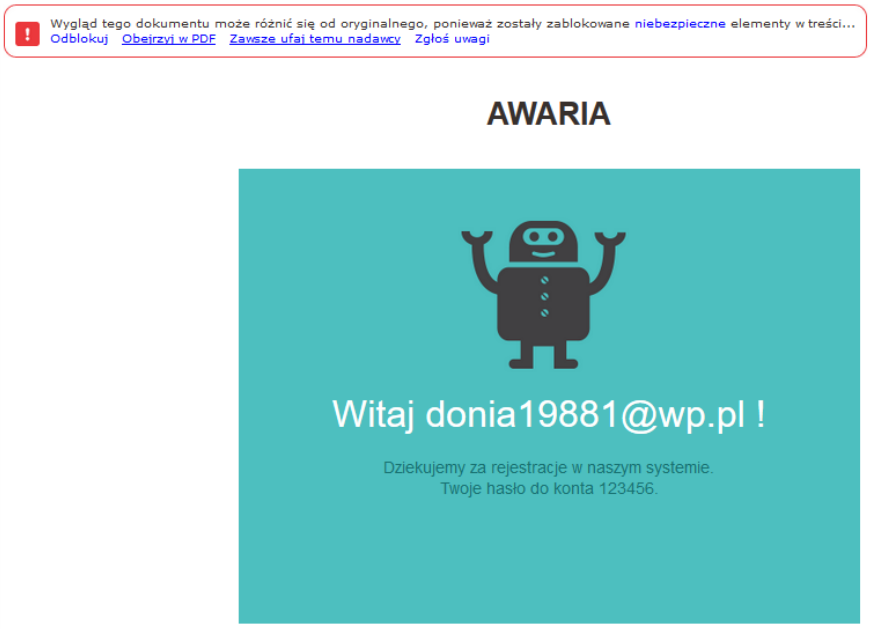
W pracy wykorzystałam element e-maili powitalnych, podczas pierwszej wizyty na stronie, które informują użytkownika o pozytywnym przejściu rejestracji. Wiadomości powitalne wysyłane są automatycznie, tuż po aktywacji adresu mailowego. Są elementem podtrzymania dalszej komunikacji z klientem. Do zaimplementowania tej funkcjonalności wykorzystałam plik *app/models/user.rb*.

```
class User < ApplicationRecord
  after_create :welcome_send
  def welcome_send
        WelcomeMailer.welcome_send(self).deliver
  end
```

Listing 3.2. Kod odpowiedzialny za wysyłanie e-mail powitalnych

W treści wiadomości, na adres użytkownika zostaje wysłane hasło, wprowadzone podczas rejestracji oraz logo firmy [4]. Wiadomości przesłane zostają w formie *html* oraz *txt*. Wybrałam te dwa formaty, ponieważ *txt* z pewnością trafi do każdego odbiorcy, natomiast wiadomości w formie *html* mogą być nieczytelne dla niektórych przeglądarek pocztowych. Różnica polega na tym, że dzięki kodzie HTML wiadomości wyglądają atrakcyjniej graficznie. Treści wiadomości powitalnych w formie *.html* oraz *.txt* zostały zawarte w pliku *app/views/welcome_mailer*.

Na kolejnej stronie można zobaczyć, jak graficznie prezentują się wiadomości powitalne oraz jaką zawierają treść.



Rysunek 3.1. Format *.html*

3.3. Przeszukiwanie danych w czasie rzeczywistym

W aplikacji wykonano dodatkowo przeszukiwanie danych w czasie rzeczywistym [12], umożliwiające wyszukanie danych w tabeli. Używając filtru, można wyświetlić tylko dane, które nas interesują i ukryć pozostałe. Po przefiltrowaniu danych w zakresie tabel można ponownie zastosować filtr w celu otrzymania aktualnych danych, badź można wyczyścić filtr w celu ponownego wyświetlenia wszystkich danych w tabeli. Filtr, który zastosowałam w projekcie został stworzony dla tabel: użytkownika, stanowiska, działu, zgłoszeń. W przeszukiwniu danych dla jednej tabeli, zastosowałam kilka warunków. Podczas wpisywania fraz, program nie rozróżnia wielkości liter za co odpowiada operator ILIKE .

Tabela użytkowników, filtrowanie według:

– imię *(ang. first_name)*
– nazwisko *(ang. last_name)*
– adres e-mail *(ang. email)*

Tabela zgłoszeń, filtrowanie według:

– nazwa urządzenia
– adres e-mail
– imię *(ang. first_name)*
– nazwisko *(ang. last_name)*

Tabela stanowisk, filtrowanie według:

– nazwa

Tabela działów, filtrowanie według:

– nazwa

Do wykonania tego elementu funkcjonalnego, wykorzystałam plik *app/controllers/zgloszenies_controller.rb.*

```
@zgloszenies = Zgloszenie.includes(:user)
.where("zgloszenies.nazwa_urzadzenia ILIKE :q OR
   users.email ILIKE :q OR users.first_name ILIKE :q OR
   users.last_name ILIKE :q",
q: "%#{params[:search]}%").references(:users)
```

Listing 3.3. Dane według których nastepuje przeszukiwanie

W tym pliku zostały zawarte wszystkie dane, które umożliwiają przeszukanie tabeli „zgłoszenie". Referencje użytkownika zostały powiązane z tabelą zgłoszeń, co umożliwia nie tylko wyszukanie interesującego nas zgłoszenia, jak również na podstawie zgłoszeń możemy wyszukać w bazie użytkownika oraz sprawdzić jakie zgłoszenie złożył i w jakim czasie. Funkcjonowanie przeszukiwania danych w czasie rzeczywistym opiera się także na kodzie JavaScript:

app/assets/javascripts/custom.js

```
$(document).on('keyup', '.search_form_z input', function()
   {
  $('.search_form_z').delay(200).submit();
});
```

Listing 3.4. JavaScripts - przeszukiwanie w czasie rzeczywistym

Dzięki wykorzystanemu kodzie JavaScripts użytkownik podczas wpisywania frazy nie musi dodatkowo klikać na przycisk, proces odbywa się w momencie, kiedy w wyszukiwarke zostaje wprowadzona pierwsza litera. Czas wyszukiwania to 200 milisekund.

3.4. Automatyzacja powiadomień

W projekcie utworzyłam automatyzacje powiadomień, która jest w stałym kontakcie z użytkownikiem zgłaszającym usterkę w systemie. Poprzez rejestrację w systemie, za pomocą adresu e-mail, na ten sam adres otrzymuje powiadomienia zawierające aktualny status swojego produktu. Na powiadomienia składają się trzy etapy:

- **started** *(przyjęcie usterki)* etap na którym, następuje zgłoszenie usterki w systemie
- **inprogress** *(rozpoczęcie naprawy)* etap na którym, rozpoczyna się naprawę usterki
- **finished** *(zakończenie naprawy)* etap na którym, zakończono naprawę usterki

Do zaimplementowania tego elementu wykorzystałam plik znajdujący się w *app/mailers/customer_notification_mailer.rb* w którym zawarłam kod:

```
class CustomerNotificationMailer < ApplicationMailer
  def started(zgloszenie_id, user_id)
        @zgloszenie = Zgloszenie.find(zgloszenie_id)
        user = User.find(user_id)
        mail(subject: default_i18n_subject, to: user.email)
  end
```

Listing 3.5. Opis statusów w modelu

W pliku *zgłoszenies_controller.rb* znajdujący się w ścieżce *app/controllers* znjadują się kody:

```
def create
  CustomerNotificationMailer.started(@zgloszeny.id,
     @zgloszeny.user_id).deliver_later
end
```

Listing 3.6. Powiadomienia o przyjęciu usterki do naprawy

Metoda *def create* odpowiada za tworzenie nowych zgłoszeń do której przypisano status *started*.

```
def update
  CustomerNotificationMailer.send(@zgloszeny.status,
     @zgloszeny.id, @zgloszeny.user_id).deliver_later if
     !@zgloszeny.started?
end
```

Listing 3.7. Powiadomienia o rozpoczeciu i zakończeniu naprawy usterki

Metoda *def update* odpowiada za edycję zgłoszeń do której przypisano status *inprogress* i *finished*.

ROZDZIAŁ 4

Testowanie aplikacji na urządzeniach mobilnych

Test optymalizacji mobilnej jest jedną z wielu rzeczy, które są bardzo istotne a z tego względu że więcej osób na dzień dzisiejszy korzysta z internetu za pomocą urządzeń mobilnych.

Aby strona nadawała się do użytku na urządzeniach mobilnych trzeba pamietać o tym by:

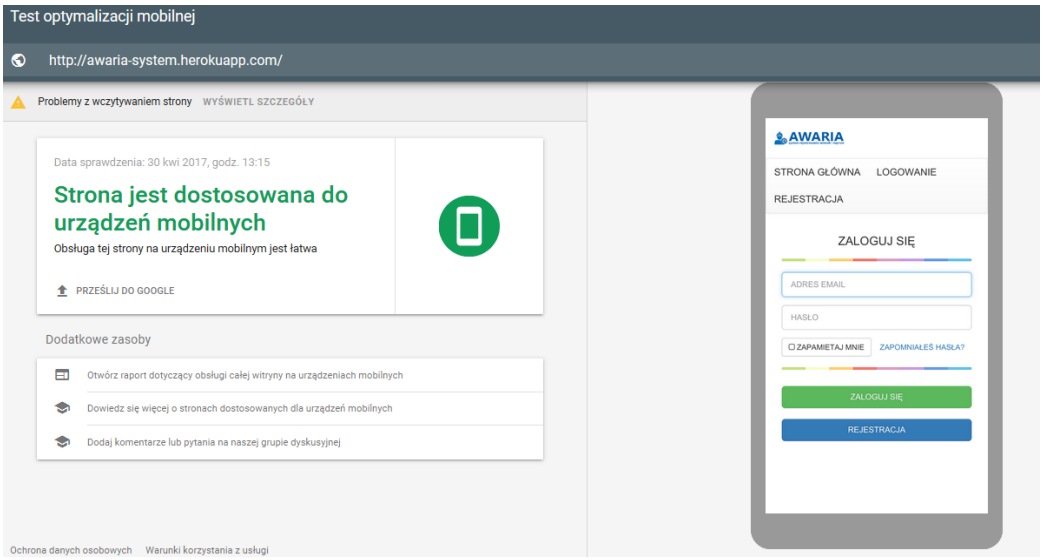
- Czcionka nie była zbyt mała, ponieważ strona staje się nieczytelna i wymaga powiekszenia widoku w celu odczytania jej treści.
- Elementy dotykowe, czyli przyciski i linki nawigacyjne, nie były zbyt blisko siebie, ponieważ użytkownicy korzystając z takiej strony mają problemy z naciśnięciem wybranego elementu, bez dotknięcia elementu sąsiadującego.
- Jeżeli rozmiar zawartości jest niedopasowany do widocznego obszaru, należy upewnić się czy strona korzysta z wartości względnych szerokości i pozycji elementów css. Ważne jest również to, by obrazy się skalowały.

Aby przeglądarka odpowiednio przeskalowała zaprojektowaną stronę do rozmiarów okna w pliku *app/views/layouts/application.html.erb* należy dodać meta-tag [1]:

```
<meta name="viewport" content="width=device-width,
    initial-scale=1.0">
```

Listing 4.1. Meta tag odpowiedzialny za skalowanie

Testy na optymalizacje mobilną, wykonałam za pomocą narzędzia **Mobile Friendly Test Google Search Console**.



Rysunek 4.1. Test optymalizacji mobilnej

ROZDZIAŁ 5

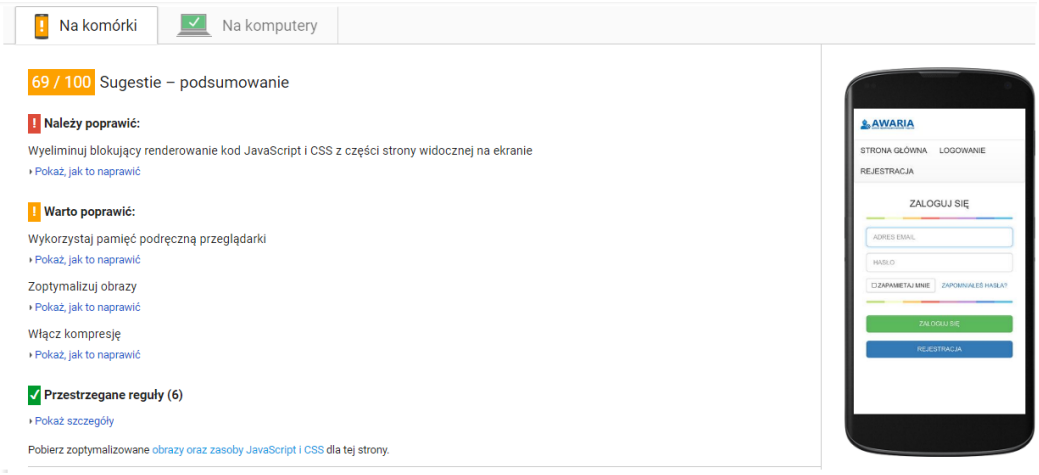
Test szybkości ładowania strony

Test szybkości ładowania strony polega na zbadaniu w jakim czasie ładuje się aplikacja. Im krótszy czas tym wyższa pozycja w Google. Według przeprowadzonych badań, im wolniej ładuje się strona, tym mniej czasu spędza na niej użytkownik. Wywiera także istotny wpływ na przeglądanie podstron przez użytkownika i szybkie dotarcie przez niego do porządanych informacji. Przekłada się to na oszczędność czasu oraz oszczedność transferu w przypadku urządzeń mobilnych. Dzięki szybko działającej stronie, roboty wyszukiwarek mogą szybciej przeskanować strone i wyszukać nowe treści.

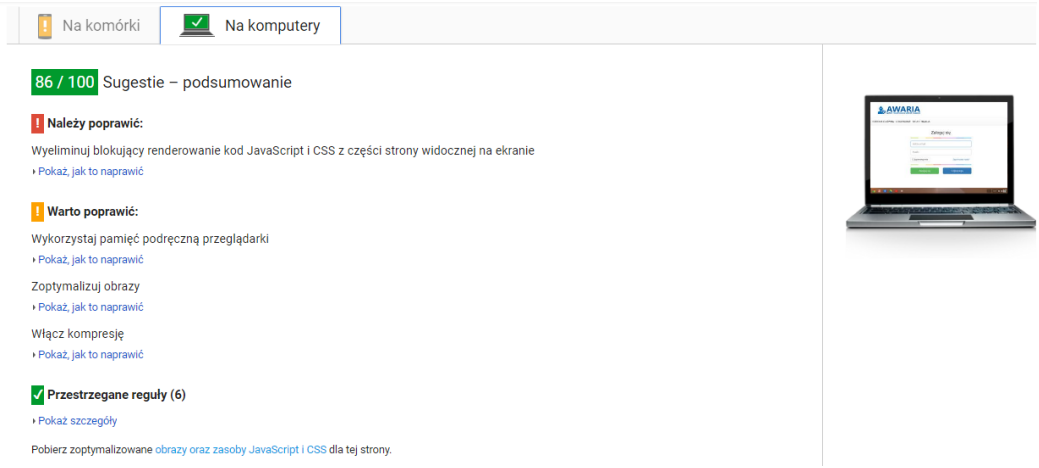
Po wykonaniu testów, narzędzie wskazało obszary, w których miejscach należałoby użyć udoskonaleń by strona ładowała się o wiele szybciej. Obszary dzielone są według trzech priorytetów: wysokim (*kolor czerwony*), średnim (*kolor żółty*), niskim (*kolor zielony*).

Według **PageSpeed Insights**, stronę można uznać za działającą dobrze, kiedy wynik w obu testach osiągnie conajmniej 85 punktów.

Na podstawie badań **PageSpeed Insights** aplikacja otrzymała następujące wyniki:



Rysunek 5.1. Wynik szybkości działania strony na urządzeniach mobilnych

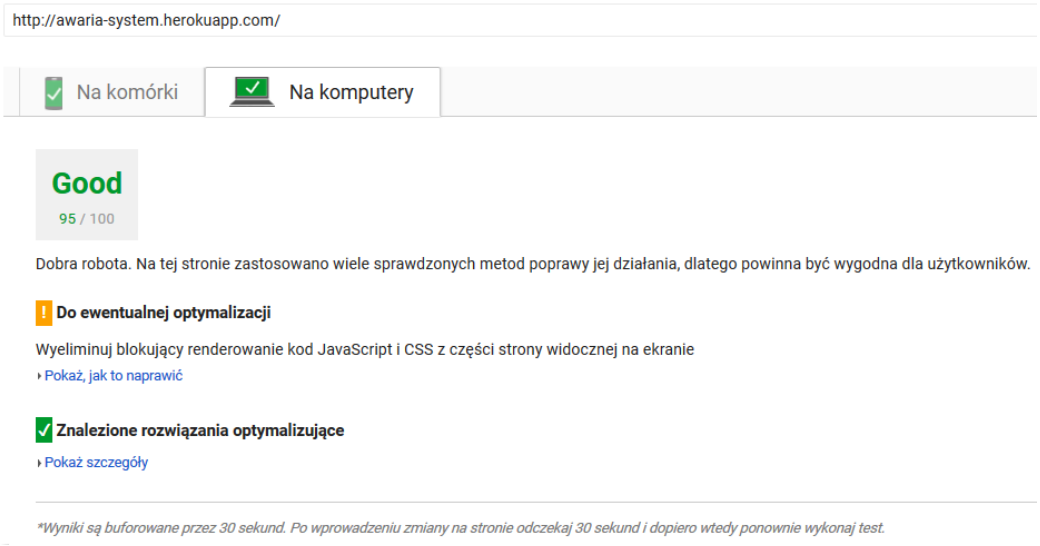


Rysunek 5.2. Wynik szybkości działania strony na komputerze

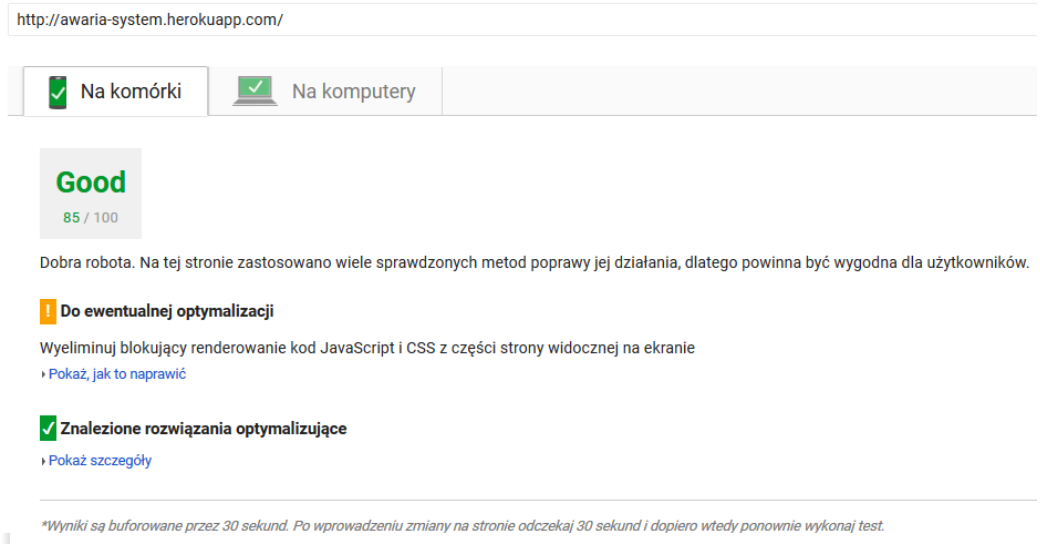
Podczas wykonywania testów, aplikcja otrzymała kilka sugesti w celu poprawienia jakości, szybkości strony. W tabeli w punktach, wyszczególniono elementy, na które składa się szybsze ładowanie strony.

Tabela 5.1. Podsumowanie

	Komputer	Telefon
Skróć czas odpowiedzi serwera		
Wyeliminuj blokujący renderowanie kod JavaScript i CSS		
Wykorzystaj pamięć podręczną przeglądarki		
Zoptymalizuj obrazy		
Włącz kompresję		
Nadaj priorytet widocznej treści	tak	tak
Unikaj przekierowań stron docelowych	tak	tak
Zmniejsz CSS	tak	tak
Zmniejsz HTML	tak	tak
Zmniejsz JavaScript	tak	tak



Rysunek 5.3. Wynik szybkości działania strony na komputerze po naprawie błędów



Rysunek 5.4. Wynik szybkości działania strony na telefonie po naprawie błędów

Po optymalizacji błędów, strona na komputer i telefon uzyskła więcej punktów. Tabelka po optymailizacji błędów, prezentuje się następująco.

Tabela 5.2. Podsumowanie po naprawie błędów

	Komputer	Telefon
Skróć czas odpowiedzi serwera	tak	tak
Wyeliminuj blokujący renderowanie kod JavaScript i CSS		
Wykorzystaj pamięć podręczną przeglądarki	tak	tak
Zoptymalizuj obrazy	tak	tak
Włącz kompresję	tak	tak
Nadaj priorytet widocznej treści	tak	tak
Unikaj przekierowań stron docelowych	tak	tak
Zmniejsz CSS	tak	tak
Zmniejsz HTML	tak	tak
Zmniejsz JavaScript	tak	tak

ROZDZIAŁ 6

Walidacja kodu

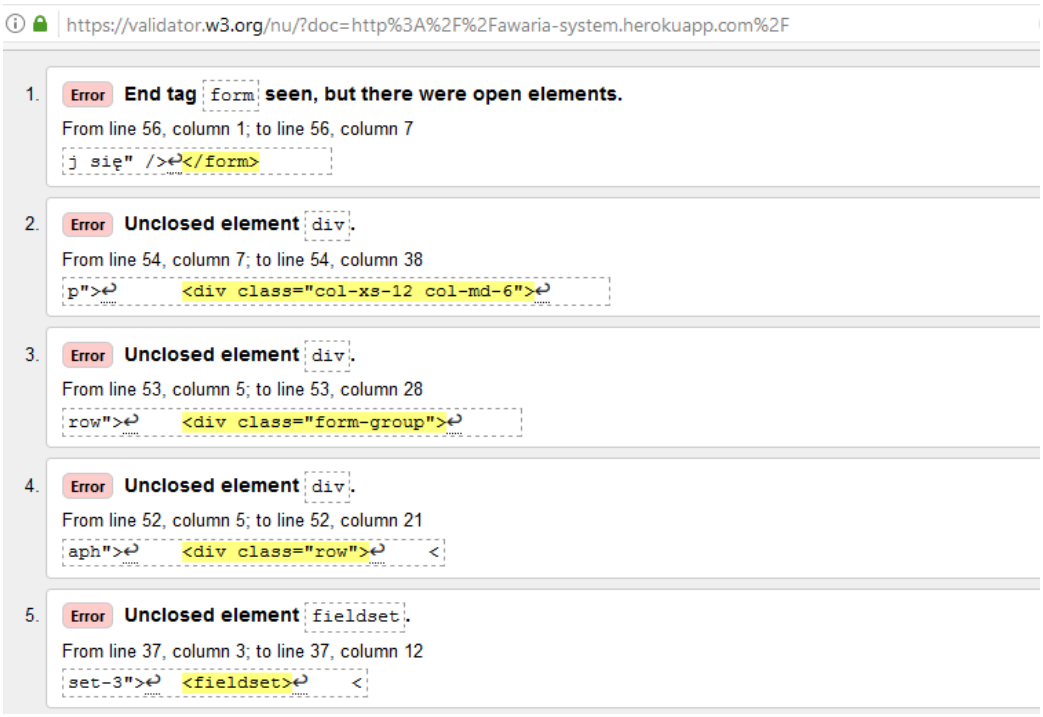
Walidacja to inaczej proces weryfikowania poprawności składniowej pliku XHTML. Wyróżnia się dwa rodzaje takiej poprawności składniowej: połączone z kontrolą zgodności z oficjalną specyfikacją XHTML, gdzie zastosowane zostają serwisy sieciowe, tzw. parasery oraz sprawdzanie wyłącznie poprawności składniowej. W tym przypadku stosuje się specjalne programy do tego przeznaczone, czyli walidatory.

Dlaczego zatem należy używać walidacji ?

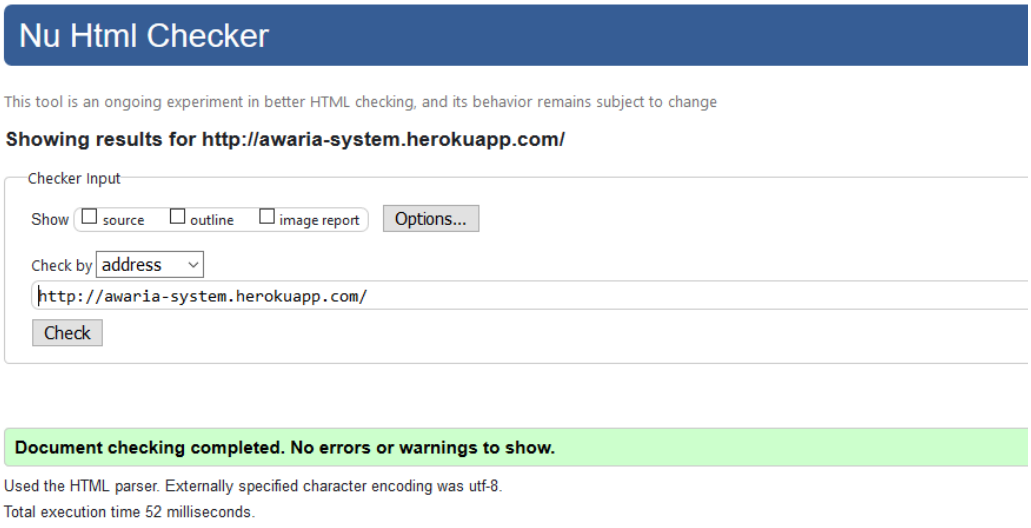
- Ponieważ, jeżeli kod strony jest poprawny to prawdopodobieństwo, że strona będzie dobrze wyświetlana na większości przeglądarek jest większa.
- Gdy strona nie zawiera błędów, to szybciej się ładuje, gdyż nie musi zastanawiać się jak interpretować nie właściwie zamieszczone znaczniki.
- Możliwość wykrycia i poprawy błedów przed oddaniem strony do użytku.
- Dzięki walidacji możemy nabyć dodatkowej wiedzy o języku XHTML oraz w przypadku zmiany specyfikacji.

Do walidacji kodu użyłam narzędzia dostępnego online o nazwie **W3C** znajdującego się pod adresem https://validator.w3.org.

Wyniki walidacji wykazały niewielką ilość popełnionych błędów, co nie oznacza że nie należy ich wyeliminować.



Rysunek 6.1. Wynik walidacji przed naprawą błędów



Rysunek 6.2. Wynik walidacji po naprawie błędów

Zakończenie

Tworzenie aplikacji w Ruby on Rails sprawiło, że nabyłam kolejne, nowe umiejetności. Nauczyłam się tworzenia elementów funkcjonalnych, nadawania szaty graficznej, testowania oraz dopasowywania strony do różnej rozdzielczości. Mam nadzieje, że nabyte umiejętności wykorzystam w kolejnym, samodzielnym projekcie, który zamierzam zrealizować w najbliższym czasie.

DODATEK A

Płyta DVD z kodem źródłowym projektu

DODATEK B

Płyta DVD z plikami pracy licencjackiej

Bibliografia

[1] Syed Fazle Rahman, *Bootstrap. Tworzenie interfejsów stron WWW. Technologia na start!*, Wydawnictwa Helion, 2015.

[2] John Elder, *Ruby on Rails. Tworzenie aplikacji WWW.*, Wydawnictwa Helion, 2016.

[3] Noel Rappin, *Professional Ruby on Rails.*, Wydawnictwa Wrox, 2008.

[4] Ruby on Rails API. http://api.rubyonrails.org/
(Listopad 20, 2016).

[5] Kurs frameworka Bootstrap. https://kursbootstrap.pl/
(Grudzień 2, 2016).

[6] Strona zawierająca kody HTML/CSS/JS. http://bootsnipp.com/
(Grudzień 2, 2016).

[7] Search and Filter Rails Models Without Bloating Your Controller.
http://www.justinweiss.com/articles/
search-and-filter-rails-models-without-bloating-your-controller/
(Grudzień 2, 2016).

[8] Action Mailer Basics.
http://guides.rubyonrails.org/action_mailer_basics.html
(Grudzień 2, 2016).

[9] Konfiguracja pliku production.rb.
http://wbzyl.inf.ug.edu.pl/rails4/mail
(Grudzień 2, 2016).

[10] Kurs CSS.
http://webkod.pl/
(Grudzień 2, 2016).

[11] Open Source Email Templates.
https://www.sendwithus.com/resources/templates/neopolitan
(Grudzień 2, 2016).

[12] Search, Sort, Paginate with AJAX.
http://railscasts.com/episodes/240-search-sort-paginate-with-ajax?view=asciicast
(Grudzień 2, 2016).

Spis rysunków

Spis kodów źródłowych

Spis tabel

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis