

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

¹Dostęp do wdrożonego projektu, jest możliwy za pomocą następujących danych: konto administratora: *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	7
1. Wykorzystane technologie	9
1.1. Ruby on Rails	9
1.2. Bootstrap	10
2. Implementacja	11
2.1. Interfejs graficzny użytkownika	11
2.2. Panel logowania	12
2.3. Tabela zgłoszeń	16
3. Elementy funkcjonalne systemu	19
3.1. Uaktywnienie przypominania hasła na e-mail	19
3.2. Uaktywnienie e-maili powitalnych podczas rejestracji	20
3.3. Przeszukiwanie danych w czasie rzeczywistym	22
3.4. Automatyzacja powiadomień	24
4. Testowanie aplikacji na urządzeniach mobilnych	27
5. Test szybkości ładowania strony	29
6. Walidacja kodu	33
Zakończenie	35
A. Płyta DVD z kodem źródłowym projektu	37
B. Płyta DVD z plikami pracy licencjackiej	39
Bibliografia	41
Spis rysunków	43

Spis kodów źródłowych	45
Spis tabel	47
Oświadczenie	49

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 wersji 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.

Listing 1.1. Plik *Gemfile*

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

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

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

Listing 1.2. Referencje do skryptów JavaScripts

```
1 //= require bootstrap-sprockets
2 //= require bootstrap
```

ROZDZIAŁ 2

Implementacja

2.1. Interfejs graficzny użytkownika

Pierwszy interfejs graficzny został stworzony w latach 70, XX wieku przez firmę 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 się zazwyczaj z:

- menu
- wyświetlanych na ekranie ikon, które oznaczają obiekty i polecenia
- okien wyświetlanych na ekranie
- funkcji dialogowych np. zapytań potwierdzających usunięcie lub zmianę 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 dowolnego 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 stronę 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 ostyleowaniem kodem HTML wyglądał przeciętnie, zawierał tylko jedną klasę *div class="field"*. Format panelu został przedstawiony poniżej:





STRONA GŁÓWNA LOGOWANIE REJESTRACJA

Adres e-mail

Hasło

Zaloguj się Rejestracja

Rysunek 2.1. Panel logowania przed ostyleowaniem kodem HTML

Listing 2.1. Kod odpowiedzialny za panel logowania

```
1 <div class="field">
2   <%= f.label :email %><br/>
3   <%= f.email_field :email, autofocus: true %>
4 </div>
5 <div class="field">
6   <%= f.label :password %><br/>
7   <%= f.password_field :password, autocomplete: "off" %>
8 </div>
```

**Rysunek 2.2.** Ostylowany panel logowania

Listing 2.2. Kod obudowany kodem HTML

```

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

```

W kodzie zastosowano poniższą klasę ***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 większa, 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 stronę
- *col-md-offset-3* w przypadku stron dla średnich wyświetlaczy, jeżeli chcemy przenieść kolumnę obejmującą trzy kolumny w prawą stronę

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"`**.

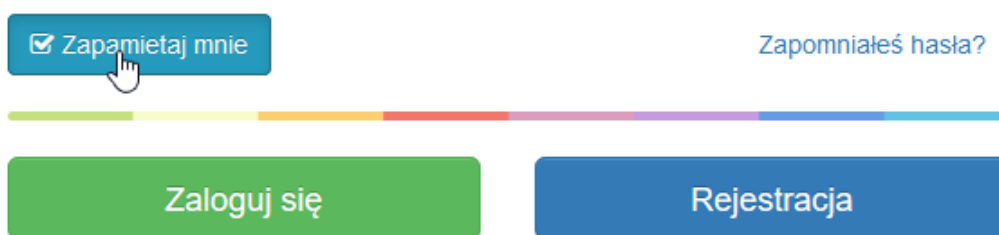
Klasa **`form-group`** służy do tworzenia formularzy, natomiast klasa **`class="form-control input-lg"`** znajdująca się pomiędzy znacznikami kodu Ruby sprawia, że element rozciąga się na całą dostępną szerokość ekranu.

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

Listing 2.3. Kod CSS dla panelu logowania

```
1 .colorgraph {  
2 border-top: 70px;  
3 height: 5px;  
4 border-top: 0;  
5 background: #c4e17f;  
6 border-radius: 5px;  
7 }
```

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



Rysunek 2.3. Checkbox w panelu logowania

2.3. Tabela zgłoszeń

Tabela zgłoszeń, po wygenerowaniu 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 się w tabelce.

ZGLOSZENIES

Nazwa urządzenia	Opis urządzenia	Data zgłoszenia	Data naprawy			
Piła elektryczna	Iskrzenie szczotek	2017-04-29	2017-05-05	dodaj	edytuj	usuń
Żelazko	Nie grzeje.	2017-04-29	2017-05-10	dodaj	edytuj	usuń


Rysunek 2.4. Tabela zgłoszeń po wygenerowaniu kodu

Listing 2.4. Element klikalny w tabeli zgłoszeń


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


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.




L.p	Dział	Priorytet	Status	Data zgłoszenia	Data naprawy	Nazwa urządzenia	Opis urządzenia	Wysyłka
3	AGD/RTV		Przyjęto	2017-04-29		Żelazko	Nie grzeje.	Tak
4	Elektryczny		Przyjęto	2017-04-29		Piła elektryczna	Iskrzenie szczotek.	Tak
6	Elektronarzędzia		Przyjęto	2017-05-13		żelazko	nie grzeje	Tak
7	AGD/RTV	tak	Przyjęto	2017-05-13		żelazko	nie grzeje	Tak

 Dodaj Zgłoszenie

 Generuj PDF

 Zrealizowane Zgłoszenia

 Zaplecze

Rysunek 2.5. Ostylowana kodem HTML tabela zgłoszeń

ROZDZIAŁ 3

Elementy funkcjonalne systemu

3.1. Uaktywnienie przypominania hasła na e-mail

Konfiguracji przypominania 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.

Listing 3.1. Konfiguracja pliku *production.rb*

```
1 config.action_mailer.smtp_settings = {  
2   user_name: ENV['SENDGRID_USERNAME'],  
3   password: ENV['SENDGRID_PASSWORD'],  
4   domain: 'awaria-system.herokuapp.com',  
5   address: 'smtp.sendgrid.net',  
6   port: 587,  
7   authentication: :plain,  
8   enable_starttls_auto: true  
9 }
```

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*.

Listing 3.2. Kod odpowiedzialny za wysyłanie e-mail powitalnych

```
1 class User < ApplicationRecord
2   after_create :welcome_send
3   def welcome_send
4     WelcomeMailer.welcome_send(self).deliver
5   end
end
```

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.

Wygląd tego dokumentu może różnić się od oryginalnego, ponieważ zostały zablokowane **niebezpieczne** elementy w treści...
[Odblokuj](#) [Obejrzyj w PDF](#) [Zawsze ufaj temu nadawcy](#) [Zgłoś uwagi](#)

AWARIA



Witaj donia19881@wp.pl !

Dziekujemy za rejestrację w naszym systemie.
Twoje hasło do konta 123456.

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, bądź 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 przeszukiwaniu 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 urzędu
- 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*.

Listing 3.3. Dane według których następuje przeszukiwanie

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

W tym pliku zostały zawarte wszystkie dane, które umożliwiają przeszukiwanie 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

Listing 3.4. JavaScripts - przeszukiwanie w czasie rzeczywistym

```
1 $(document).on('keyup', '.search_form_z input', function() {
2   $('.search_form_z').delay(200).submit();
3 });
```

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

3.4. Automatyzacja powiadomień

W projekcie utworzyłam automatyzację 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:

Listing 3.5. Opis statusów w modelu

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

W pliku `zgloszenies_controller.rb` znajdujący się w ścieżce `app/controllers` znajdują się kody:

Listing 3.6. Powiadomienia o przyjęciu usterki do naprawy

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


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

Listing 3.7. Powiadomienia o rozpoczęciu i zakończeniu naprawy usterki

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

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 pamiętać o tym by:

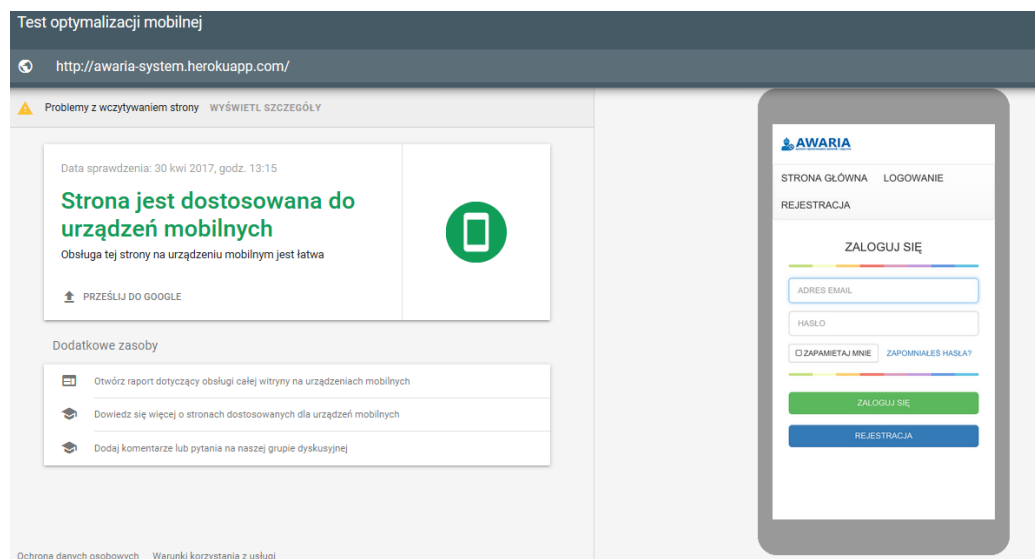
- Czcionka nie była zbyt mała, ponieważ strona staje się nieczytelna i wymaga powiększenia widoku w celu odczytania jej treści.
- Elementy dotykowe, czyli przyciski i linki nawigacyjne, nie były zbyt blisko siebie, ponieważ użytkownicy korzystając z takiej strony mają problemy z naciśnięciem wybranego 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]:

Listing 4.1. Meta tag odpowiedzialny za skalowanie

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

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



Rysunek 4.1. Test optymalizacji mobilnej

ROZDZIAŁ 5

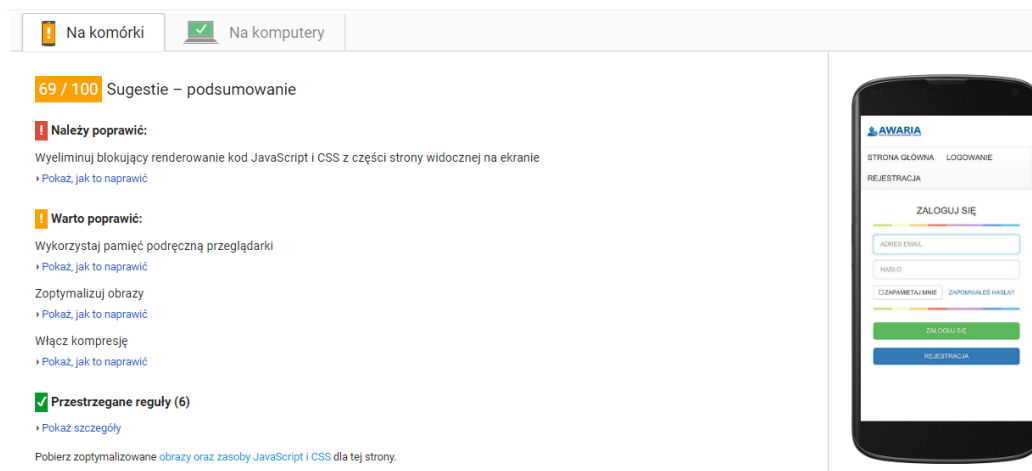
Test szybkości ładowania strony

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

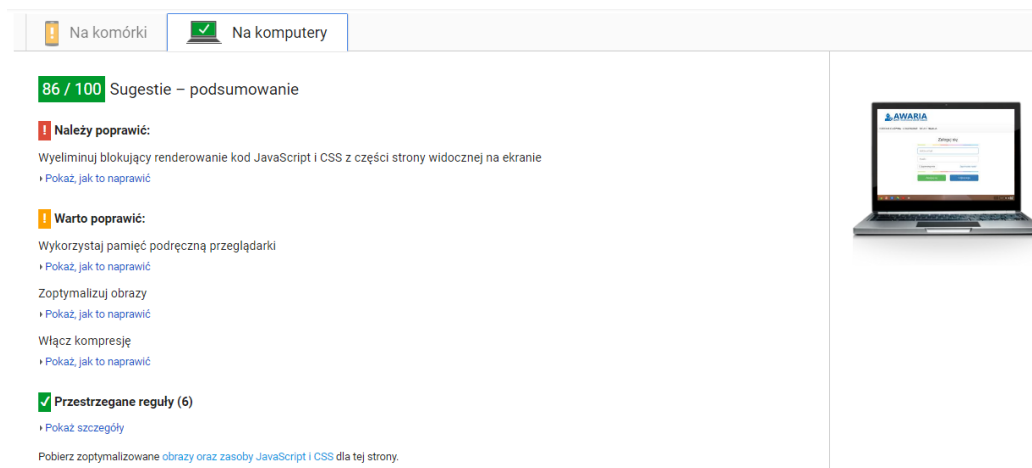
Po wykonaniu testów, narzędzie wskazało obszary, w których miejscach należałoby użyć udoskonaleń by strona łądowała się o wiele szybciej. Obszary dzielone są według trzech priorytetów: wysokim (*kolor czerwony*), średnim (*kolor żółty*), niskim (*kolor zielony*).

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, aplikacja otrzymała kilka sugestii 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

http://awaria-system.herokuapp.com/

✓ Na komórki

✓ Na komputery

Good

95 / 100

Dobra robota. Na tej stronie zastosowano wiele sprawdzonych metod poprawy jej działania, dlatego powinna być wygodna dla użytkowników.

! Do ewentualnej optymalizacji

Wyeliminuj blokujący renderowanie kod JavaScript i CSS z części strony widocznej na ekranie

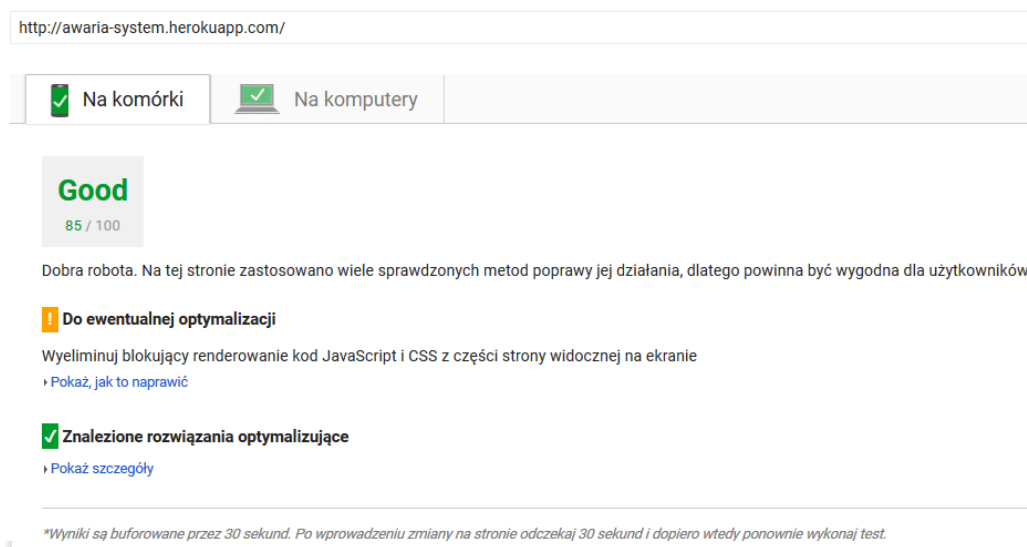
[Pokaż, jak to naprawić](#)

✓ Znalezione rozwiązania optymalizujące

[Pokaż szczegóły](#)

*Wyniki są buforowane przez 30 sekund. Po wprowadzeniu zmiany na stronie odczekaj 30 sekund i dopiero wtedy ponownie wykonaj test.

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 uzyskała więcej punktów. Tabelka po optymalizacji błędów, prezentuje się następująco.

Tabela 5.2. Podsumowanie po naprawie błędów

	Komputer	Telefon
Skróć czas odpowiedzi serwera	tak	tak
Wypełnij 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 ?

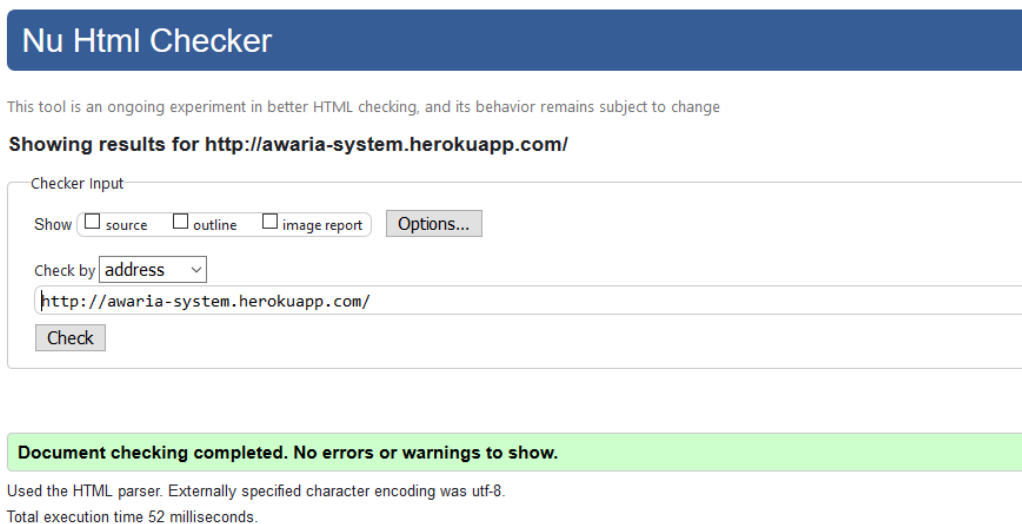
- Ponieważ, jeżeli kod strony jest poprawny to prawdopodobieństwo, że strona będzie dobrze wyświetlana na większości przeglądarek jest większa.
- Gdy strona nie zawiera błędów, to szybciej się ładuje, gdyż nie musi zastanawiać się jak interpretować nie właściwie zamieszczone znaczniki.
- Możliwość wykrycia i poprawy błędów 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 umiejętności. Nauczyłam się tworzenia elementów funkcjonalnych, nadawania szaty graficznej, testowania oraz dopasowywania strony do różnej rozdzielczości. Mam nadzieję, 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/](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

2.1.	Panel logowania przed ostyleowaniem kodem HTML	12
2.2.	Ostyleowany panel logowania	13
2.3.	Checkbox w panelu logowania	15
2.4.	Tabela zgłoszeń po wygenerowaniu kodu	16
2.5.	Ostylewana kodem HTML tabela zgłoszeń	17
3.1.	Format <i>.html</i>	21
4.1.	Test optymalizacji mobilnej	28
5.1.	Wynik szybkości działania strony na urządzeniach mobilnych .	30
5.2.	Wynik szybkości działania strony na komputerze	30
5.3.	Wynik szybkości działania strony na komputerze po naprawie błędów	31
5.4.	Wynik szybkości działania strony na telefonie po naprawie błędów	32
6.1.	Wynik walidacji przed naprawą błędów	34
6.2.	Wynik walidacji po naprawie błędów	34

Spis kodów źródłowych

1.1. Plik <i>Gemfile</i>	10
1.2. Referencje do skryptów JavaScripts	10
2.1. Kod odpowiedzialny za panel logowania	13
2.2. Kod obudowany kodem HTML	14
2.3. Kod CSS dla panelu logowania	15
2.4. Element klikalny w tabeli zgłoszeń	16
3.1. Konfiguracja pliku <i>production.rb</i>	19
3.2. Kod odpowiedzialny za wysyłanie e-mail powitalnych	20
3.3. Dane według których następuje przeszukiwanie	23
3.4. JavaScripts - przeszukiwanie w czasie rzeczywistym	23
3.5. Opis statusów w modelu	24
3.6. Powiadomienia o przyjęciu usterki do naprawy	24
3.7. Powiadomienia o rozpoczęciu i zakończeniu naprawy usterki	25
4.1. Meta tag odpowiedzialny za skalowanie	28

Spis tabel

5.1. Podsumowanie	31
5.2. Podsumowanie po naprawie błędów	32

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