

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 i została stworzona w grupie 3-osobowej. W pracy zajmowałam się wykonaniem frontendu aplikacji oraz stworzeniem poszczególnych elementów takich jak: automatyzacja powiadomień, filtracja danych, uaktywnienie resetowania hasła oraz e-maili powitalnych podczas rejestracji. Do stworzenia aplikacji wykorzystano system Linux, baze 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>, przechowywany w repozytorium Githuba razem z kodem aplikacji. Aplikacja dostępna jest pod adresem <https://awaria-system.herokuapp.com/>.

Słowa kluczowe

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

Spis treści

Wprowadzenie	7
1. Wykorzystane technologie	9
1.1. Ruby on Rails	9
1.2. Bootstrap	9
1.3. Baza danych	10
2. Implementacja	13
2.0.1. Interfejs graficzny użytkownika	13
2.1. Panel logowania	14
2.2. Tabela zgłoszeń	18
3. Elementy funkcjonalne systemu	21
3.0.1. Uaktywnienie przypominania hasła na e-mail	21
3.1. Uaktywnienie e-maili powitalnych podczas rejestracji	22
3.2. Przeszukiwanie danych w czasie rzeczywistym	24
3.3. Automatyzacja powiadomień	26
4. Testy	29
4.0.1. Testowanie aplikacji na urządzeniach mobilnych	29
4.1. Test szybkości ładowania strony	30
4.2. Walidacja kodu	32
Zakończenie	35
A. Tytuł załącznika jeden	37
B. Tytuł załącznika dwa	39
Bibliografia	41
Spis rysunków	43

Spis kodów źródłowych	45
Oświadczenie	47

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 szata graficzna 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 w jak 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 tworzyć coś 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'

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

W plikach z rozszerzeniem muszą znaleźć się dwa kody:

@import „Bootstrap-sprockets”

@import „Bootstrap”

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

//= require Bootstrap-sprockets

//= require Bootstrap

1.3. Baza danych

W projekcie posłużono się bazą danych PostgreSQL w wersji 9.5. Jest to jedna z trzech najpopularniejszych wolnodostępnych systemów zarządzania danymi. W Ruby on Rails wszystkie ustawienia bazy danych odbywają się w domyślnym pliku konfiguracyjnym *config/database.yml*. W pliku znajdują się trzy środowiska:

- **rozwojowe** (*ang. development*) - jest używane na komputerze programisty aby mógł kontrolować zmiany które zaistaniały w projekcie
- **testowe** (*ang. test*) - służy do uruchamiania testów automatycznych
- **produkcyjne** (*ang. production*) - jest stosowane wtedy kiedy aplikacja uruchomiona jest na serwerze produkcyjnym

Poniżej znajdują się kod środowiska produkcyjnego, zamieszczonego w projekcie:

Listing 1.1. Zawartość pliku *database.yml*

```
1 production:
2 <<: *default
3 database: awaria_production
4 username: awaria
5 password: PG#sysawa88!!
```


ROZDZIAŁ 2

Implementacja

2.0.1. 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*) 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 albo komputera. Ponadto strony responsywne są kompatybilne z interfejsami dotykowymi urządzeń mobilnych.

2.1. 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 front-endu to estetyka oraz niezróżnicowany wygląd aplikacji.

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



AWARIA
system rejestrowania usterek i napraw

STRONA GŁÓWNA LOGOWANIE REJESTRACJA

Adres e-mail

Hasło

Zaloguj się Rejestracja

Rysunek 2.1. Przykładowy dolny pasek nawigacji.

Źródło: Opracowanie własne

Listing 2.1. Cześć kodu Ruby nieostylowanego kodem HTML

```
1 <%= form_for(resource, as: resource_name,  
2   url: session_path(resource_name)) do |f| %>  
3   <div class="field">  
4     <%= f.label :email %><br />  
5     <%= f.email_field :email, autofocus: true %>  
6   </div>  
7   <div class="field">  
8     <%= f.label :password %><br />  
9     <%= f.password_field :password, autocomplete: "off" %>  
10  </div>
```

Powyższy kod znajduje się w pliku *views/devise/sessions/new.html.erb*.





STRONA GŁÓWNA LOGOWANIE REJESTRACJA

Zaloguj się

☐ Zapamiętaj mnie [Zapomniałeś hasła?](#)

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>Zaloguj ęsi</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: "Hasło"
        %>
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 i możliwej do obejrzenia na dowolnym wyświetlaczu:

1. *col-xs* stosuje się dla bardzo małych wyświetlaczy (szerokość ekranu mniejsza niż 768 pikseli)
2. *col-sm* stosuje się dla małych wyświetlaczy (szerokość ekranu większa równa 768 pikseli)
3. *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:

1. *col-sm-offset-2* w przypadku stron dla małych wyświetlaczy, jeżeli chcemy przenieść kolumnę, obejmującą dwie kolumny w prawą stronę
2. *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"`**.

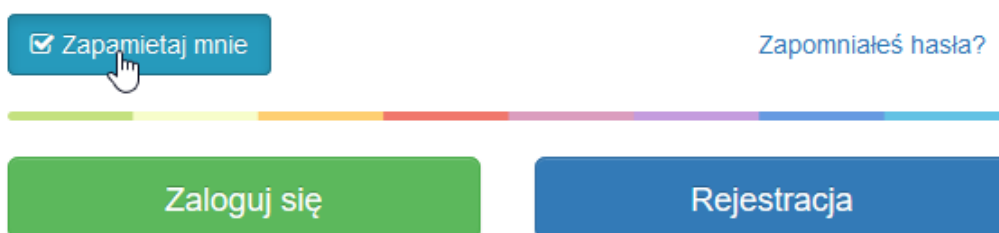
Klasa **`form-group`** służy do tworzenia formularzy, natomiast klasa **`class="form-control input-lg"`** znajdująca się pomiędzy znacznikami kodu Ruby sprawia że element rozciąga się na całą dostępną szerokość ekranu.

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

2.2. Tabela zgłoszeń

Tabele tworzy się poprzez znacznik `<table> ... </table>` i wystarczy dodać do nich klasę `.acrylic` by uzyskać dopełnienie oraz linie oddzielające wiersze. Między znacznikami `<tr>`, `<th>` umieszczamy kategorie na które ma dzielić się dana tabelka. Wszystkie kategorie obudowane znacznikiem `<th>` należy umieścić w sekcji znacznika `<thead>`. Natomiast zawartość danych tabeli umieszczamy między znacznikami `<td>`, ważne jest to by każda dana odpowiadała prawidłowej kolejności danej kategorii.

Wszystkie dane obudowane znacznikami `<td>` należy umieścić w sekcji znacznika `<tbody>`.

Tabela zgłoszeń po wygenerowaniu kodu nie posiadała przycisków jej kolumny i wiersze nie były oddzielone linią w żaden sposób a 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

L.p	Dział	Priorytet	Status	Data zgłoszenia	Data naprawy	Nazwa urządzenia	Opis urządzenia	Wysyłka	User	Pracownik	Akcje
3	AGD/RTV		Przyjęto	2017-04-29		Żelazko	Nie grzeje.	Tak			<input type="button" value="Usuń"/> <input type="button" value="Pokaż"/> <input type="button" value="Edytuj"/>
4	Elektryczny		Przyjęto	2017-04-29		Piła elektryczna	Iskrzenie szczotek.	Tak			<input type="button" value="Usuń"/> <input type="button" value="Pokaż"/> <input type="button" value="Edytuj"/>

Rysunek 2.5. Ostylowana kodem HTML tabela zgłoszeń

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

Listing 2.4. Element klikalny

```
1 <td>{{= link_to('Usuń', zgloszeny, method: :delete,
    data: { confirm: 'Jesteś pewny?' }, class: 'btn
    btn-danger btn-xs')}}</td>
```

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.

ROZDZIAŁ 3

Elementy funkcjonalne systemu

3.0.1. Uaktywnienie przypominania hasła na e-mail

Konfiguracji przypominania hasła na adres e-mail użytkownika, 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.

3.1. 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
```

W treści wiadomości, na adres użytkownika zostaje wysłane hasło, wprowadzone podczas rejestracji oraz logo firmy. 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.

AWARIA



Rysunek 3.1. Format *.html*

3.2. Przeszukiwanie danych w czasie rzeczywistym

W aplikacji wykonano dodatkowo przeszukiwanie danych w czasie rzeczywistym, 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
3     users.email ILIKE :q OR
4     users.first_name ILIKE :q OR users.last_name ILIKE :q",
5     q: "%#{params[:search]}%").references(:users)
end
```

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',
2   function() {
3     $('.search_form_z').delay(200).submit();
4   });
```

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.

3.3. Automatyzacja powiadomień

W projekcie utworzyłam automatyzację powiadomień która polega na byciu w stałym kontakcie z użytkownikiem który zgłosił 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
7
8   def inprogress(zgloszenie_id, user_id)
9     @zgloszenie = Zgloszenie.find(zgloszenie_id)
10    user = User.find(user_id)
11    mail(subject: default_i18n_subject, to: user.email)
12  end
13
14  def finished(zgloszenie_id, user_id)
15    @zgloszenie = Zgloszenie.find(zgloszenie_id)
16    user = User.find(user_id)
17    mail(subject: default_i18n_subject, to: user.email)
18  end
19 end
```

W pliku *zgłoszenies_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 ,
    @zgloszeny.user_id).deliver_later
```

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?
```

Metoda *def update* odpowiada za edycję zgłoszeń do której przypisano status *inprogress* i *finished*, biorąc pod uwagę to że status *started* podczas edycji, przy kolejnej edycji, nie może uzyskać już statusu początkowego.

ROZDZIAŁ 4

Testy

4.0.1. 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:

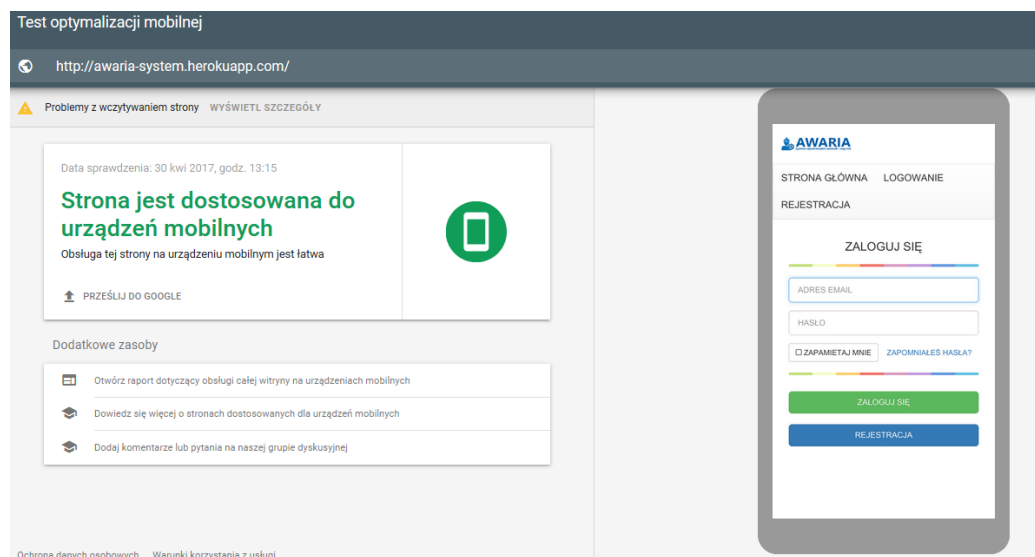
- Czcionka nie była zbyt mała - ponieważ wtedy strona staje się nieczytelna i wymaga powiększenia widoku w celu odczytania jej treści
- Elementy dotykowe - czyli przyciski i linki nawigacyjne, nie były zbyt blisko siebie, ponieważ użytkownicy korzystając z takiej strony mają problemy z naciśnięciem wybranego elementu bez dotknięcia elementu sąsiadującego.
- Rozmiar zawartości niedopasowany do widocznego obszaru - aby rozwiązać taki problem 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:

Listing 4.1. Meta tag odpowiedzialny za skalowanie

```
1 <head>
2 <title>Awaria <%= yield(:title) %></title>
3 <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

4.1. Test szybkości ładowania strony

Test szybkości ładowania strony polega na zbadaniu, to 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 porządzanych informacji co przekłada się na oszczędność czasu oraz oszczędność transferu w przypadku urządzeń mobilnych. Dzięki szybko działającej stronie roboty wyszukiwarek mogą szybciej przeskanować strone i wyszukać nowe treści.

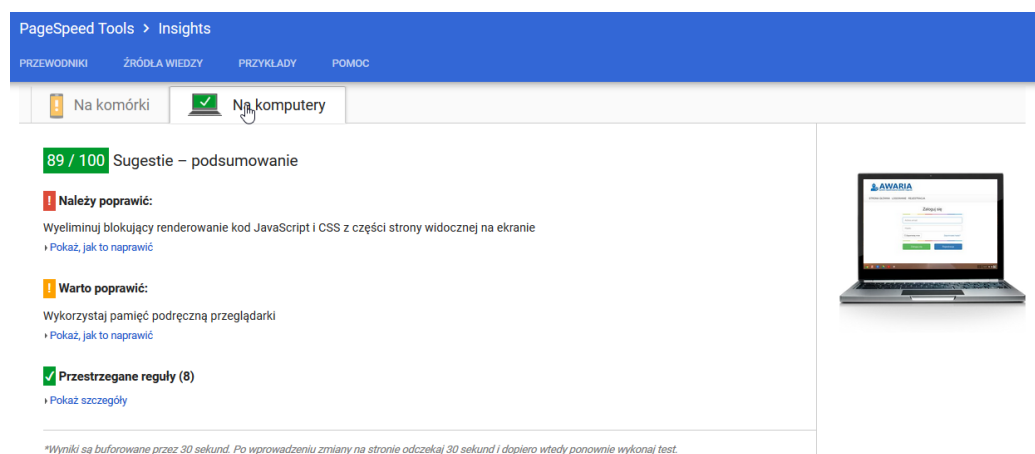
Po wykonaniu testów narzędzie wskazało obszary w których miejscach należałoby użyć udoskonaleń, by strona ładowała się o wiele szybciej. Obszary dzielone są według trzech priorytetów: wysokim, średnim, niskim.

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

Szybkość działania strony na komputerach PC 89/100 - udoskonalenia na poziomie niskim

Szybkość działania strony na urządzeniach mobilnych 71/100 - udoskonalenia na poziomie średnim

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



Rysunek 4.2. Wynik szybkości działania strony na komputerze

Sugestie dotyczące poprawek jakie należałoby przeprowadzić:

- Wyeliminuj blokujący renderowanie kod JavaScript i CSS z części strony widocznej na ekranie. Strona zawiera blokujące skrypty (3) i blokujące zasoby CSS (3). Powoduje to opóźnienia w renderowaniu strony.
- Wykorzystaj pamięć podręczną przeglądarki. Ustawienie daty wygaśnięcia lub maksymalnego wieku zasobów statycznych w nagłówkach HTTP powoduje, że przeglądarka wczytuje z lokalnego dysku twardego zasoby pobrane wcześniej, zamiast ponownie pobierać je z sieci.

Przestrzegane reguły, które zostały wykonane zgodnie z wymaganiami:

- Nadaj priorytet widocznej treści: Część strony widoczna na ekranie ma odpowiedni priorytet.
- Skróć czas odpowiedzi serwera: Serwer odpowiedział szybko.
- Unikaj przekierowań stron docelowych: Strona nie zawiera przekierowań.
- Włącz kompresję: Kompresja jest stosowana.
- Zmniejsz CSS: Kod CSS jest zmniejszony.
- Zmniejsz HTML: Kod HTML jest zmniejszony.
- Zmniejsz JavaScript: Zawartość pliku JavaScript jest zmniejszona.
- Zoptymalizuj obrazy: Grafiki są zoptymalizowane.

4.2. 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 tak zwane parasery oraz sprawdzanie wyłącznie poprawności składniowej w tym przypadku sotosuje się specjalne programy do tego przeznaczone czyli walidatory.

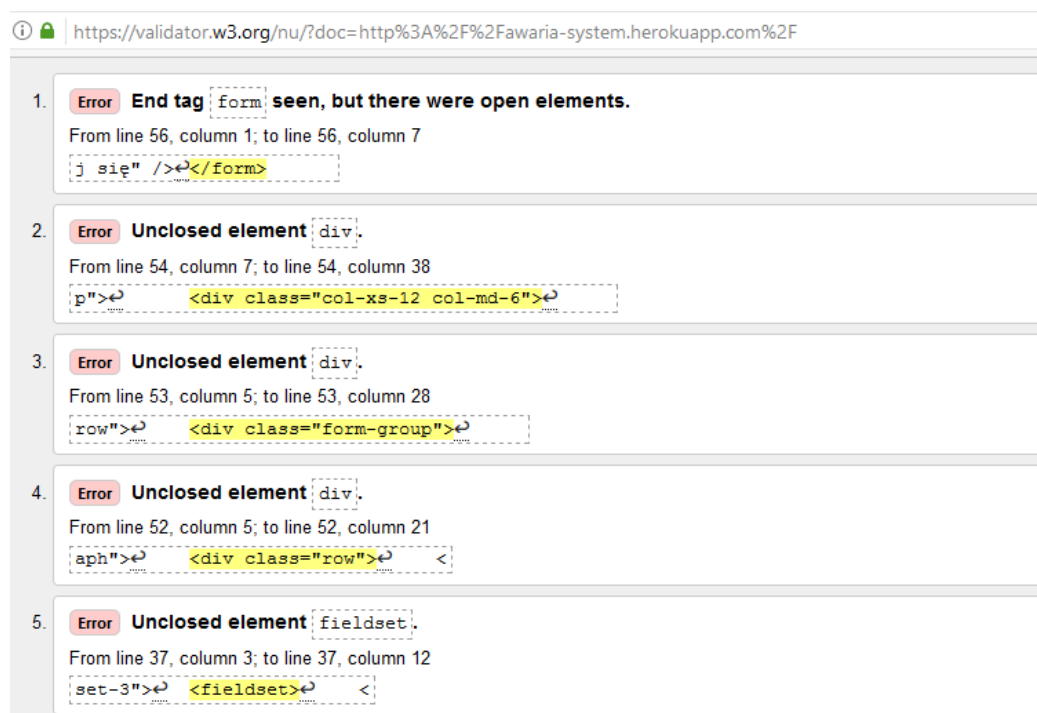
Dlaczego zatem należy używać walidacji ?

- 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 4.3. Wynik walidacji

Zakończenie

Tworzenie aplikacji w Ruby on Rails sprawiło że nabyłam kolejnych, nowych umiejętności. Nauczyłam się nie tylko tworzenia elementów funkcjonalnych, nadawania szaty graficznej, testowania czy dopasowania strony do różnej rozdzielczości ale także rozwiązywania problemów z którymi borykałam się podczas tworzenia aplikacji a które umożliwiły mi poszerzenie wiedzy. Mam nadzieję że nabyte umiejętności wykorzystam w kolejnym samodzielnym projekcie który zamierzam zrealizować w najbliższym czasie.

DODATEK A

Tytuł załącznika jeden

Treść załącznika jeden.

DODATEK B

Tytuł załącznika dwa

Treść załącznika dwa.

Bibliografia

- [1] Syed Fazle Rahman, *Bootstrap. Tworzenie interfejsów stron WWW. Technologia na start!*, Wydawnictwa Helion.
- [2] John Elder, *Ruby on Rails. Tworzenie aplikacji WWW.*, Wydawnictwa Helion.
- [3] Noel Rappin, *Professional Ruby on Rails.*, Wydawnictwa Wrox.
- [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).

Spis rysunków

2.1. Przykładowy dolny pasek nawigacji.	14
2.2. Ostylowany panel logowania	15
2.3. Checkbox w panelu logowania	17
2.4. Tabela zgłoszeń po wygenerowaniu kodu	18
2.5. Ostylowana kodem HTML tabela zgłoszeń	19
3.1. Format <i>.html</i>	23
4.1. Test optymalizacji mobilnej	30
4.2. Wynik szybkości działania strony na komputerze	31
4.3. Wynik walidacji	33

Spis kodów źródłowych

1.1. Zawartość pliku <i>database.yml</i>	11
2.1. Część kodu Ruby nieostylowanego kodem HTML	15
2.2. Kod obudowany kodem HTML	16
2.3. Kod CSS dla panelu logowania	17
2.4. Element klikalny	19
3.1. Konfiguracja pliku <i>production.rb</i>	21
3.2. Kod odpowiedzialny za wysyłanie e-mail powitalnych	22
3.3. Dane według których następuje przeszukiwanie	25
3.4. JavaScripts - przeszukiwanie w czasie rzeczywistym	25
3.5. Opis statusów w modelu	26
3.6. Powiadomienia o przyjęciu usterki do naprawy	27
3.7. Powiadomienia o rozpoczęciu i zakończeniu naprawy usterki	27
4.1. Meta tag odpowiedzialny za skalowanie	29

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