



UNIwersytet Gdański  
Wydział Matematyki  
Fizyki i Informatyki



PRACA LICENCJACKA

INFORMATYKA

# **Awaria - System rejestrowania usterek i napraw**

Autor:

**Dominika Pienczyn**

Promotor: dr Włodzimierz Bzyl

Gdańsk 2017

.....

podpis promotora

.....

podpis autora

## **Streszczenie**

Praca przedstawia aplikację internetową o nazwie "Awaria" która jest systemem rejestrującym usterki oraz naprawy sprzętu i została stworzona w grupie 3-osobowej. Jako cel, postawiłam sobie wykonanie frontendu aplikacji oraz stworzenie 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.

# Rozdział 1

## 1. Wstęp

Graficzny interfejs użytkownika to ogólne określenie sposobu prezentacji informacji przez komputer oraz interakcji z użytkownikiem, polegające na rysowaniu i obsłudze 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 2

## 2. Wykorzystane technologie

### 2.0.1 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. W Railsach jest tak że jeden 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.

### 2.0.2 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 musza 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
```

### 2.0.3 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:

- ✓ development (rozwojowe) - jest używane na komputerze programisty aby mógł kontrolować zmiany które zaistaniały w projekcie

- ✓ test (testowe) - służy do uruchamiania testów automatycznych

- ✓ production (produkcyjne) - jest stosowane wtedy kiedy aplikacja uruchomiona jest na serwerze produkcyjnym

Poniżej znajduje się kod środowiska produkcyjnego, zamieszczonego w projekcie:

```
production:
<<: *default
database: awaria_production
username: awaria
password: PG#sysawa88!!
```



# Rozdział 3

## Implementacja

### 3.0.1 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
- ✓ funkcje dialogowe np. zapytania potwierdzające usunięcie lub zmianę wydanego polecenia

W projekcie do stworzenia interfejsu wykorzystano framework Bootstrap który korzysta również z języka JavaScripts.

## Rozdział 4

# Elementy funkcjonalne systemu

### 4.0.1 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.

W projekcie zastosowano poniższą konfigurację:

```
config.action_mailer.smtp_settings = {  
  :user_name => 'apikey',  
  :password => 'SG.FjS4F15CSZyQetUWLdMIig  
.Ef2wfMbrWsoc2vgRbSBqT0D_mpiMfe0PpqMo_DrabOA',  
  :domain => "awaria-system.herokuapp.com",  
  :address => 'smtp.sendgrid.net',  
  :port => 587,  
  :authentication => :plain,  
  :enable_starttls_auto => true  
}
```

### 4.0.2 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. Do stworzenia tej funkcjonalności wykorzystałam plik *app/models/user.rb*.

Kod zawarty w pliku wygląda następująco:

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

W wiadomości zawarłam dodatkowy element, którym jest logo aplikacji oraz tytuł wiadomości. Kod znajduje się w pliku *app/mailers/welcome\_mailer.rb*

```
class WelcomeMailer < ApplicationMailer
  def welcome_send(user)
    @user = user
    attachments.inline['logo.png'] = File.read('app/assets/images/logo.png')
    mail to: user.email, subject: "Witamy na naszej stronie", from: 'awaria.kont
  end
end
```

Treści wiadomości powitalnych w formie *.html* oraz *.txt* zostały zawarte w pliku *app/views/welcome\_mailer*.

### 4.0.3 Filtrowanie danych w tabelach

W aplikacji wykonano dodatkowo przeszukiwanie danych, umożliwiające wyszukiwanie danych w tabeli. Używając autofiltru, 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 wszystkich tabel: użytkowników, stanowiska, działu, zgłoszeń, zgłoszeń-aktywności. W przeszukiwaniu danych dla jednej tabeli zastosowałam kilka warunków.

#### **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)
end
```

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 ale również użytkownika który znajduje się w systemie.

#### 4.0.4 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 stworzenie tego elementu funkcjonalnego wykorzystałam między innymi 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

  def inprogress(zgloszenie_id, user_id)
    @zgloszenie = Zgloszenie.find(zgloszenie_id)
    user = User.find(user_id)
    mail(subject: default_i18n_subject, to: user.email)
  end

  def finished(zgloszenie_id, user_id)
```

```
@zgloszenie = Zgloszenie.find(zgloszenie_id)
user = User.find(user_id)
mail(subject: default_i18n_subject, to: user.email)
end
end
```

Wykorzystałam również plik *zgłoszenies\_controller.rb* znajdujący się w ścieżce *app/controllers*.

Metoda odpowiadająca za tworzenie nowych zgłoszeń oraz tworzenie powiadomień 'przyjecie usterki':

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

Metoda odpowiadająca za edycje zgłoszeń oraz tworzenie powiadomień 'rozpoczęcie naprawy', 'zakończenie naprawy':



# Rozdział 5

## Testy

### 5.0.1 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:

- 1 Czcionka nie była zbyt mała - ponieważ wtedy strona staje się nieczytelna i wymaga powiększenia widoku w celu odczytania jej treści
- 2 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.
- 3 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 strona odpowiednio skalowała się na urządzeniach mobilnych, należy w pliku `app/views/layouts/application.html.erb` dodać meta-tag:

```
<!DOCTYPE html>
<html>
<head>
<title>Awaria <%= yield(:title) %></title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Testy na optymalizację mobilną wykonałam za pomocą **Google Search Console**. Narzędzie to, znajduje się pod adresem <https://search.google.com/search-console/mobile-friendly>.

### 5.0.2 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**, strone można uznać za działającą dobrze, kiedy wynik w obu testach osiągnie conajmniej 85 punktów.

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:**

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

**5.0.3 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 parasyery oraz sprawdzanie wyłącznie poprawności składniowej w tym przypadku sotosuje się specjalne programy do tego przeznaczone czyli walidatory.

**Dlaczego zatem należy używać walidacji:**

- 1 Ponieważ jeżeli kod strony jest poprawny to prawdopodobieństwo że strona będzie dobrze wyświetlana na większości przeglądarek jest większa.
- 2 Gdy strona nie zawiera błędów to szybciej się ładuje, gdyż nie musi zastanawiać się jak interpretować nie właściwie zamieszczonych znaczników.
- 3 Możliwość wykrycia i poprawy błędów przed oddaniem strony do użytku.
- 4 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 na stronie <https://validator.w3.org>.

Wyniki walidacji wykazały niewielką ilość popełnionych błędów, łącznie siedem, niżej przedstawiam wybrane przez mnie trzy:

Error: Unclosed element div.

From line 52, column 7; to line 52, column 38

p"> <div class="col-xs-12 col-md-6">

Error: Unclosed element fieldset.

From line 35, column 3; to line 35,

column 12

set-3"> <fieldset> <

Error: Unclosed element div.

From line 33, column 1; to line 33, column 17

ntainer"><div class="row"><div

Kod CSS według walidatora został napisany bezbłędnie:

Aby pokazać czytelnikom swojej strony, że stworzyłeś stronę interoperacyjną

<p>

<a href="http://css-validator.org/check/referer">



</a>

</p>

# Bibliografia

- [1] Stanisław Białas, *Macierze. Wybrane problemy*, AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków, 2006.
- [2] Nicholas J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, Philadelphia 1996.
- [3] Nicholas J. Higham, *Functions of Matrices. Theory and Computation*, SIAM, Philadelphia 2008.
- [4] Maksymilian Dryja, Janina i Michał Jankowscy, *Przegląd metod i algorytmów numerycznych, część 2*, Wydawnictwa Naukowo-Techniczne, Warszawa 1982.

# Spis treści

<b>1</b>	<b>1. Wstęp</b>	<b>2</b>
<b>2</b>	<b>2. Wykorzystane technologie</b>	<b>3</b>
2.0.1	Ruby on Rails . . . . .	3
2.0.2	Bootstrap . . . . .	4
2.0.3	Baza danych . . . . .	5
<b>3</b>	<b>Implementacja</b>	<b>7</b>
3.0.1	Interfejs graficzny użytkownika . . . . .	7
<b>4</b>	<b>Elementy funkcjonalne systemu</b>	<b>9</b>
4.0.1	Uaktywnienie przypominania hasła na e-mail . . . . .	9
4.0.2	Uaktywnienie e-maili powitalnych podczas rejestracji . . . . .	10
4.0.3	Filtrowanie danych w tabelach . . . . .	11
4.0.4	Automatyzacja powiadomień . . . . .	13
<b>5</b>	<b>Testy</b>	<b>15</b>
5.0.1	Testowanie aplikacji na urządzeniach mobilnych . . . . .	15
5.0.2	Test szybkości ładowania strony . . . . .	16
5.0.3	Walidacja kodu . . . . .	18

Warszawa, dnia .....

## Oświadczenie

Oświadczam, że pracę licencjacką pod tytułem: „Awaria - System rejestrowania usterek i napraw”, której promotorem jest dr Włodzimierz Bzyl, wykonałem/am samodzielnie, co poświadczam własnoręcznym podpisem.

.....