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 aplikacje 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 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 sie 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 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. 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 musza 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 mogł 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 znajduję 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 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

✓funkcje dialogowe np. zapytania potwierdzajace usuniecie lub zmiane 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 przpominania 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 konfiguracje:

```
config.action_mailer.smtp_settings = {
:user_name => 'apikey',
:password => 'SG.FjS4F15CSZyQetUWLdMIig.Ef2wfMbrWsoc2vgRbSBqT0D_mpiMfe0E
: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
```

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 Przeszukiwanie danych w czasie rzeczywistym

4.0.4 Automatyzacja powiadomień

W projekcie utworzyłam automatyzacje powiadomień która polega na byciu w stałym kontakcie z użytkownikiem, który zgłosił usterke w systemie. Poprzez rejestracje 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).delive
```

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

Rozdział 5

Testy

5.0.1 Testowanie aplikacji na urządzeniach mobilnych

5.0.2 Test szybkości ładowania strony

5.0.3 Walidacja kodu

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-Techiczne, Warszawa 1982.

Spis treści

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.

.....