Projekt Giełdy Terminów

Koncepcja

Michał Begejowicz Bartosz Żurkowski

19 czerwca 2015

AGH

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Spis treści

1 Wstęp **3**
1.1 Ogólny opis 3

2 Architektura systemu **4**
2.1 Model 4
2.2 Widoki 5
2.3 Kontrolery 6
2.4 Resque 6
2.5 Rack Middleware 6
2.6 Zasoby 7

3 Struktura podsystemów i komponentów **7**
3.1 Autoryzacja użytkowników 7
3.2 Panel administratorski 8
3.3 Internacjonalizacja 8
3.4 Obsługa planu zajęć 8
3.4.1 Integracja z systemem EnrollMe 9
3.5 Zarządzanie ofertami 9
3.6 Zarządzanie wymianami 10
3.7 System powiadomień 10

4 Inne technologie **11**

5 Metodyka pracy nad projektem **12**

1 Wstęp

System Giełdy Terminów będzie realizowany korzystając z frameworku Ruby on Rails. Wymusza to jego podstawowe cechy:

- Architektura klient-serwer
- Aplikacja webowa z interfejsem RESTowym
- Struktura Model-View-Controller

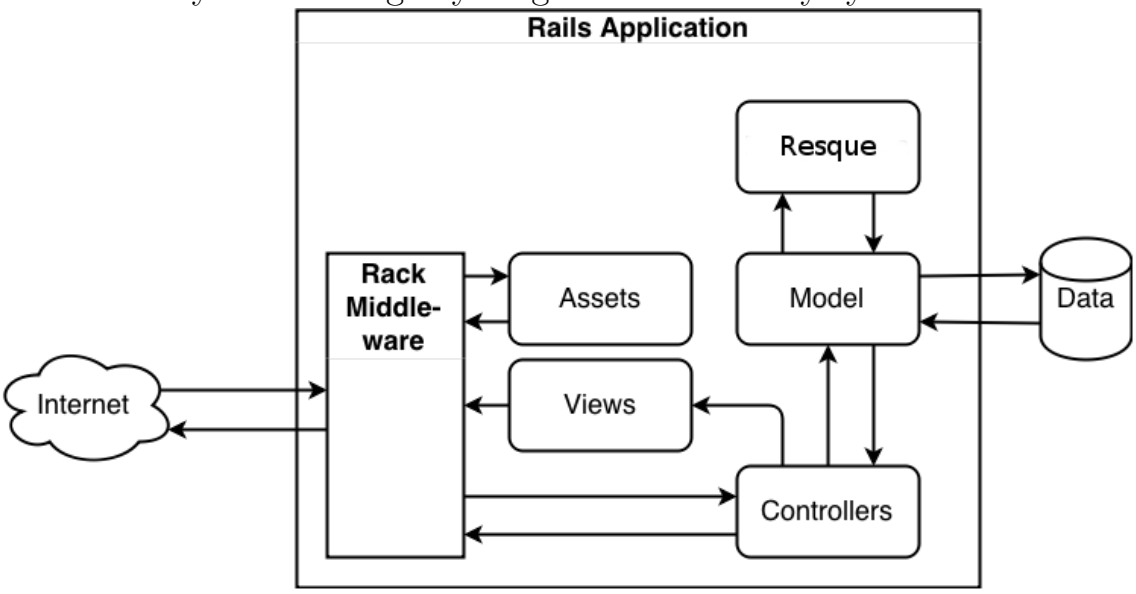
Wystawiony na zewnątrz zostanie również interfejs webowy.
W trakcie procesu projektowego aplikacja będzie dostępna na serwerach Heroku. Docelowo może znaleźć się na serwerach uczelnianych, jeżeli integracja z obecnie istniejącymi systemami okaże się możliwa.

1.1 Ogólny opis

Realizowany system będzie składał się z trzech głównych części: modelu, czyli opisu struktury bazy danych i definicji zasobów w niej zawartych, kontrolerów, odpowiedzialnych za logikę biznesową aplikacji i widoków, pozwalających reprezentować zasoby przechowywane w aplikacji.

2 Architektura systemu

Rysunek 1: Ogólny diagram architektury systemu



Budowa aplikacji zgodna jest z zasadami MVC. Oprócz standardowych elementów, na diagramie zamieszczono również blok "Resque" reprezentujący klasy wykonujące swoje zadania w tle, bez bezpośredniej ingerencji użytkownika.

2.1 Model

Model reprezentuje dane aplikacji i reguły do manipulowania tymi danymi. W przypadku Railsów, modele są głównie wykorzystywane do zarządzania zasadami interakcji z odpowiednią tabelą bazy danych. W większości przypadków jednej tabeli bazy danych odpowiada jeden model.

Dane systemu przechowywane będą w dwóch magazynach danych. Pierwszym z nich jest główna baza danych, obsługiwana przez Postgre-

SQL. Tam będą przechowywane informacje o użytkownikach systemu, planach zajęć, ofertach i wymianach. Drugim źródłem danych będzie Redis - magazyn klucz-wartość, służący jako cache dla dla aplikacji. Jest on niezbędny do poprawnego i wydajnego funkcjonowania Resque [1], czyli modułu obsługujące kolejkowanie i wykonywanie zadań.

Framework Ruby on Rails zapewnia dodatkową warstwę abstrakcji nałożoną na relacje bazy danych z pomocą biblioteki ActiveRecord[2]. Pozwala ona w prosty sposób modyfikować dane, definiować zależności między modelami i dodawać walidacje, gwarantujące poprawność - pod względem logiki aplikacji - danych w bazie.

2.2 Widoki

Widoki tworzą interfejs użytkownika aplikacji. W Railsach widoki są często plikami HTML zawierającymi kod w języku Ruby, wykonujący zadania związane wyłącznie z prezentacją danych. Przede wszystkim są one odpowiedzialne za dostarczanie danych do przeglądarki internetowej lub innego narzędzia, które jest używane do oglądania efektów działania aplikacji.

Interfejs w projektowanej aplikacji będzie podzielony na dwie części. Pierwszą z nich będzie widok podstawowy, umożliwiający logowanie się, umieszczanie i akceptowanie ofert czy przeglądanie planu zajęć. Drugim widokiem będzie widok admina, dostępny tylko dla wybranych użytkowników, pozwalający na dokładne poznanie i modyfikowanie stanu aplikacji.

[1] https://github.com/resque/resque/

[2] https://github.com/rails/rails/tree/master/activerecord

2.3 Kontrolery

Kontrolery "sklejają" modele i widoki. W Railsach odpowiadają za przetwarzanie żądań przychodzących z przeglądarki internetowej, pozyskiwanie danych z modeli i przekazywanie ich do widoków w celu ich prezentacji.

W tej części aplikacji znajduje się logika biznesowa. Oprócz samych klas kontrolerów, dziedziczących po ActionController::Base[3] (dostarczanym przez framework RoR), do tego fragmentu aplikacji należą też tak zwane serwisy - odpowiedzialne za pojedynczą funkcję systemu klasy pozwalajace lepiej organizować, testować i utrzymywać kod. Przykładem serwisu może być klasa budująca graf wymian dla danego przedmiotu.

2.4 Resque

Resque to moduł wspierający kontrolę asynchronicznych zadań, których wykonanie zajmuje dużo czasu. Zadania tego typu powinny być wykonywane w tle, aby nie obciążały głównych procesów serwera. Kolejkowane zadania są przetrzymywane w Redis.

Moduł zostanie zastosowany przede wszystkim do obliczania roszad terminów oraz importu planów zajęć.

2.5 Rack Middleware

Rack jest interfejsem pozwalającym tworzyć serwisy webowe w Rubym - filtrować zapytania oraz formować odpowiedzi wykorzystując powiązania pomiędzy wszystkimi określonymi modułami aplikacji (modularność). W skład obowiązków Rack wchodzi obsługa protokołu HTTP, zabezpieczanie ruchu internetowego i autentykacja użytkowników.

[3] http://guides.rubyonrails.org/action_controller_overview.html

2.6 Zasoby

Framework Ruby on Rails pozwala na swobodne organizowanie oraz dostęp do zgromadzonych zasobów - obrazów, arkuszy stylów, skryptów JavaScript itp.

Dodatkowo, mechanizm Assets Pipeline dokonuje kompilacji zasobów - kompresuje oraz łączy pliki oraz obsługuje potokową kompilację plików zgodnie z wieloma, powszechnie używanymi pre-procesorami.

3 Struktura podsystemów i komponentów

W tym rozdziale wymienione zostaną podstawowe komponenty tworzonej aplikacji. Na końcu każdej sekcji będzie można znaleźć listę bibliotek używanych w danym podsystemie.

3.1 Autoryzacja użytkowników

System będzie posiadał prosty system autoryzacji użytkowników oparty na bibliotece Devise.

Devise dostarcza logikę powiązaną z rejestracją, logowaniem, autoryzacją użytkowników czy przypominaniem haseł. Biblioteka ta, korzystając z generycznych kontrolerów, pozwala szybko zbudować system obsługi kont użytkowników, zapewniający odpowiednie przekierowania i obsługę błędów logowania.

Biblioteka ta dba również o bezpieczeństwo danych przechowywanych w bazie danych, szyfrując zapisywane hasła, implementując takie mechanizmy szyfrowania jak sól czy pieprz.

Wykorzystywane narzędzia i biblioteki:

- Devise[4]

[4]https://github.com/plataformatec/devise

3.2 Panel administratorski

System będzie rozpoznawał uprawnienia użytkowników. Wybranym osobom - adminom - udostępniony zostanie panel administratorski, w którym w łatwy sposób będzie można przejrzeć stan systemu i - w wyjątkowych sytuacjach - zmieniać go.

Panel administratorski będzie miał również znaczenie podczas prac programistycznych, pozwalając w łatwy sposób weryfikować wpływ wytwarzanego kodu na aplikację.

Wykorzystywane narzędzia i biblioteki:

- ActiveAdmin[5] - framework budujący panel admina w aplikacji na podstawie prostych definicji podstron panelu

3.3 Internacjonalizacja

System będzie korzystał z modułu I18n dostarczanego przez Ruby on Rails. Choć pierwsza wersja aplikacji będzie dostępna tylko w języku polskim, zakładamy, że każdy tekst wyświetlany powinien znajdować się w specjalnych plikach konfiguracyjnych obsługiwanych przez moduł I18n.

Dzięki temu modułowi będzie możliwe łatwiejsze utrzymanie kodu i ewentualne przetłumaczenie aplikacji na inne języki.

Wykorzystywane narzędzia i biblioteki:

- I18n[6]

3.4 Obsługa planu zajęć

W zakres odpowiedzialności tego modułu będzie wchodzić import planów zajęć zarejestrowanych studentów oraz ich graficzna wizualizacja.

Głównymi składowymi tej części systemu będą:

[5] https://github.com/activeadmin/activeadmin

[6] http://guides.rubyonrails.org/i18n.html

- model planu zajęć - reprezentacja w bazie danych informacji o terminach posiadanych przez konkretnych użytkowników
- widoki i kontrolery odpowiedzialne za wyświetlenie posiadanego planu zajęć
- moduł importujący dane o początkowym przypisaniu użytkowników do terminów z zewnętrznego systemu

3.4.1 Integracja z systemem EnrollMe

System EnrollMe to aplikacja zaimplementowana przez studentów mająca na celu ułatwienie przydzielania terminów ćwiczeń oraz zajęć laboratoryjnych.

Studenci logują się na swoje konta oraz przypisują określoną pulę punktów terminom zajęć, które najbardziej im odpowiadają. Następnie uruchamiany jest algorytm, którego celem jest przydzielenie terminów studentom w taki sposób, aby w jak najwięszym stopniu zgadzały się z ich preferencjami. W ten sposób każdy student otrzymuje gotowy plan zajęć.

Produktem końcowym opisanego algorytmu jest lista gotowych planów zajęć - przypisania terminów do poszczególych studentów. Wygenerowane dane umożliwią import planów zajęć do systemu Exchange.

3.5 Zarządzanie ofertami

Moduł będzie zawierał logikę związaną z tworzeniem i modyfikowaniem wymian. Odpowiedni kontroler umożliwi też wyświetlanie istniejących ofert.

Na tę część składać się będą:

- model oferty - reprezentacja w bazie danych informacji o chęci wymiany jednego terminu na jeden z innych, wybranych terminów

- widoki i kontrolery odpowiedzialne za wyświetlenie, tworzenie i usuwanie ofert

3.6 Zarządzanie wymianami

System akceptacji oraz odrzucania ofert wymian. W zakres modułu wchodzi również implementacja algorytmu wyznaczania roszad terminów.

Jest to moduł zawierający najwięcej logiki biznesowej. Jego poprawne wykonanie jest niezbędne, by aplikacja była użyteczna.

Główne elementy modułu:

- model proponowanej wymiany - reprezentacja w bazie danych informacji o możliwej wymianie - lista ofert i żądań które mogą zostać jednocześnie spełnione, by wymiana była poprawna i wszyscy jej uczestnicy dostali jeden z terminów, o które prosili
- serwis budujący i interpretujący graf ofert - algorytm analizujący aktywne oferty użytkowników pod kątem potencjalnych wymian
- widoki i kontrolery umożliwiające użytkownikom przeglądanie i akceptowanie proponowanych wymian

3.7 System powiadomień

Serwis odpowiedzialny za wysyłanie powiadomień o zajściu ważnych z punktu widzenia systemu zdarzeń do użytkowników o odpowiednich rolach.

Gdy pojawia się nowa wymiana, którą użytkownik może być zainteresowany, powinien dostać informację e-mailową o tym fakcie.

Wykorzystywane narzędzia i biblioteki:

- ActionMailer[7] - część frameworka Ruby on Rails, pozwalająca na łatwe generowanie i wysyłanie dynamicznych wiadomości e-mail
- mailgunner[8] - biblioteka ułatwiająca integrację z serwisem Mailgun[9] udostępniającym API do wysyłania e-maili

4 Inne technologie

W tej sekcji znajduje się lista technologii, które są używane w całej aplikacji.

- Bootstrap[10] - framework CSS zawierający zestaw gotowych reguł oraz styli - znaczneie przyspiesza projektowanie estetycznych UI
- simple_form[11] - biblioteka obsługująca automatyczne generowanie formularzy na podstawie definicji modeli ActiveRecord
- Relacyjna baza danych PostgreSQL[12]
- Puma[13] - szybki, wielowątkowy serwer przeznaczony dla aplikacji zaimplementowanych w Rubym
- RSpec[14] - framework testowy - udostępnia DSL ułatwiający pisanie testów jednostkowych, integracyjnych i akceptacyjnych

[7] http://guides.rubyonrails.org/action_mailer_basics.html
[8] https://github.com/timcraft/mailgunner
[9] http://www.mailgun.com/
[10] http://getbootstrap.com/
[11] https://github.com/plataformatec/simple_form
[12] http://www.postgresql.org/
[13] http://puma.io/
[14] http://rspec.info/

- Haml[15] - pre-procesor języka HTML pozwalający "przeplatać"jego struktury ze składnią Ruby'ego
- CoffeeScript[16] - pre-procesor języka JavaScript

5 Metodyka pracy nad projektem

Projekt będzie prowadzony zgodnie z metodyką Agile. Jedna iteracja będzie trwała tydzień, co dobrze pokrywa się z częstotliwością zajęć.

Zespół będzie korzystał z tablicy kanbanowej udostępnianej przez aplikację Trello, na której będą znajdować się zadania. Zadania będa między sobą powiązane, tworząc pewną hierarchię, podobną do Scrumowego podziału na epiki, historyjki i zadania.

Rozwój systemu będzie wersjonowany z pomocą systemu GIT, a repozytorium zdalne przechowywane będzie na serwerach GitHub.

[15] http://haml.info/
[16] http://coffeescript.org/