

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka (INF)
SPECJALNOŚĆ: Inżynieria systemów informatycznych (INS)

**PRACA DYPLOMOWA
MAGISTERSKA**

Analiza porównawcza popularnych frameworków
webowych.

Comparative analysis of most popular web
frameworks.

AUTOR:
Marcin Mantke

PROWADZĄCY PRACĘ:
dr inż. Roman Ptak

OCENA PRACY:

Spis treści

1	Wstęp	3
1.1	Ogólny opis pracy	3
1.2	Cel pracy	4
1.3	Wymagania	4
1.4	Zarys koncepcji	4
2	Przedstawienie omawianych technologii	5
2.1	Historia i rozwój technologii webowych	5
2.2	Aplikacja internetowa	6
2.3	Framework webowy	6
3	Przegląd wybranych rozwiązań	7
3.1	Ruby on Rails	7
3.1.1	Doktryna Ruby on Rails	7
3.2	Phoenix	9
3.3	Express	9
4	Analiza porównawcza	10
4.1	Opisowe porównanie charakterystyki wybranych frameworków	10
4.1.1	Ruby on Rails vs Phoenix	10
4.1.2	Ruby on Rails vs Express	10
4.1.3	Phoenix vs Express	10
4.2	Subiektywna ocena	10
5	Implementacja	11
5.1	Ruby on Rails	11
5.2	Phoenix	11
5.3	Express	11
6	Projekt komputerowego środowiska eksperymentalnego	12
6.1	Plan prowadzenia eksperymentów	12

SPIS TREŚCI	2
6.2 Wykorzystane narzędzia	12
6.2.1 Nginx	12
6.2.2 Docker	12
6.3 Benchmarki	12
6.3.1 Ruby	12
6.3.2 Elixir	12
6.3.3 JavaScript	12
7 Analiza wydajnościowa	13
7.1 Metody mierzenia wydajności aplikacji internetowych	13
7.2 Wyniki badań	13
8 Podsumowanie	14

Rozdział 1

Wstęp

1.1 Ogólny opis pracy

W momencie pisania pracy istnieje niezliczona ilość frameworków webowych. Prawie codziennie, dla samego języka JavaScript, powstaje jeden nowy (micro) framework. Przyczyn takiego stanu rzeczy jest kilka. Po pierwsze, problemy, z jakimi spotykają się programiści są bardzo zróżnicowane oraz skomplikowane. Wynika to z nacisku biznesu, czyli klientów, na to, aby nowo powstały produkt był innowacyjny. Programista oczywiście posiada narzędzia, które powinny być wystarczające do rozwiązania powierzonego mu problemu, jednakże niekiedy korzystanie z narzuconych przez framework rozwiązań wręcz utrudnia wykonanie powierzonej pracy. Z tego powodu powstają nowe frameworki, które udostępniają zestaw narzędzi ukierunkowany pod rozwiązanie nowego, konkretnego problemu. Druga przyczyna jest w pewien sposób powiązana z pierwszą. Jest to stworzenie frameworku nie w odpowiedzi na potrzebę, ale wygenerowanie potrzeby poprzez stworzenie frameworku, który ułatwia rozwiązanie przykładowego problemu. Kolejnym powodem jest próba odchudzenia istniejących frameworków. Przykładem może być Ruby on Rails, który jest frameworkiem kompletnym, ale niekiedy posiadającym zbyt dużo wbudowanych funkcji, z których trudno jest zrezygnować, a które w danym projekcie nie zostaną wykorzystane. Zwiększa to rozmiar oraz złożoność projektu, co oczywiście jest niekorzystne.

Ilość dostępnych frameworków pokazuje jak ważnym elementem stały się one dla programistów. Niestety, tak dynamiczny rozwój rozwiązań tego typu powoduje spory problem jeśli chodzi o wybór technologii. Niniejsza praca ma na celu zaprezentowanie wybranych frameworków webowych oraz dokonanie porównania ich funkcjonalności oraz wydajności. Analiza porównawcza ma na celu wyszczególnienie cech frameworków, na które programista powinien zwrócić szczególną uwagę przy doborze frameworku.

Jako że framework dodaje do aplikacji pewną warstwę abstrakcji, czyli kod, naturalny jest narzut wydajnościowy na aplikację. Z tego powodu, poza różnymi cechami odnośnie

budowy i dostarczanych funkcjonalności, frameworki różnią się również wydajnością.

1.2 Cel pracy

1.3 Wymagania

1.4 Zarys koncepcji

Rozdział 2

Przedstawienie omawianych technologii

2.1 Historia i rozwój technologii webowych

Od lat 90 XX wieku świat obserwuje bardzo dynamiczny rozwój technologii związanych z Internetem. Począwszy od roku 1991, kiedy to naukowcy z instytutu badawczego **CERN** (ang. *European Organization for Nuclear Research*) opracowali standard WWW, przed programistami zaczęła się otwierać nowa gałąź tworzenia aplikacji, którą są aplikacje internetowe. Początkowo aplikacje te były jedynie statycznymi stronami WWW, na których znajdował się jedynie tekst. Wprowadzenie kaskadowych arkuszy stylów (*CSS*) w roku 1996 sprawiło, że strony internetowe przybrały graficzną formę. Rok 1997 przyniósł obsługę języka *JavaScript* w przeglądarkach internetowych. Oznaczało to, że strony internetowe, poza statycznymi elementami, zyskały elementy dynamiczne, np. reagujące na akcje użytkownika.

Wraz ze wzrostem dostępu ludzi do Internetu rozwijały się technologie odpowiedzialne za strony internetowe. Za punkt początkowy istnienia nie stron, a aplikacji internetowych, można przyjąć rok 1997 i powstanie języka *PHP*. Był to pierwszy interpretowany skryptowy język programowania, który służył do budowania aplikacji internetowych działających w czasie rzeczywistym. Wraz z rozwojem języka PHP oraz innych, podobnych mu języków, np. *Python* i *Ruby*, zmienił się sposób budowania aplikacji. Programiści zaczęli rezygnować ze standardowych klientów w postaci aplikacji desktopowych i przechodzili na tzw. cienkich klientów (ang. *thin client*). Trend ten przyspiesza rozwój oraz różnorodność aplikacji serwerowych posiadających interfejs graficzny w formie strony internetowej, które nazywane są aplikacjami internetowymi [?].

2.2 Aplikacja internetowa

Aplikacja internetowa (webowa) jest aplikacją znajdującą się nie na komputerze użytkownika, lecz na ogólnodostępnym serwerze. Komunikacja pomiędzy, niekiedy rozproszonymi, elementami aplikacji odbywa się poprzez sieć komputerową. Aplikacja webowa swój interfejs graficzny poprzez przeglądarkę internetową bądź np. aplikację mobilną.

2.3 Framework webowy

Aby ułatwić korzystanie z coraz liczniejszych technologii wykorzystywanych w tworzeniu aplikacji internetowych, powstały narzędzia nazywane frameworkami webowymi. Framework jest uniwersalnym środowiskiem programistycznym, które dostarcza niezbędne narzędzia wymagane do stworzenia aplikacji internetowej w wybranym języku programowania [?]. Każdy z frameworków dostarcza pewną abstrakcję, która znajduje się wokół kodu napisanego przez programistę. Przykładem takiej abstrakcji jest system mapowania ścieżki podstrony (np. `/users/3`) na konkretną akcję w aplikacji (zwykle akcja *SHOW* dla kontrolera *Users*), czyli *routing*. Twórcy frameworków zauważyli, że w każdej aplikacji webowej są stosowane te same typy rozwiązań, więc w wielu przypadkach wprowadzili dane rozwiązania jako integralne części frameworków. W efekcie programiści mogą korzystać z gotowych, dogłębnie przetestowanych rozwiązań, które znajdują się w 90% aplikacji webowych.

Rozdział 3

Przegląd wybranych rozwiązań

3.1 Ruby on Rails

Ruby on Rails jest open source’owym frameworkiem webowym. Został stworzony w głównej mierze przez duńskiego programistę Davida Heinemeiera Hanssona. Pierwsza wersja RoR ukazała się w lipcu 2004 roku. Fakt ten pokazuje, że jest to już dojrzały framework, z ugruntowaną pozycją na rynku. Potwierdzeniem tej pozycji jest liczne wsparcie społeczności. Rails’y zostały stworzone przez ponad 4,5 tysiąca osób, istnieje ok. 132 tysiące ogólnodostępnych paczek, które rozszerzają funkcjonalność tego frameworku. Ruby on Rails domyślnie korzysta z architektury MVC, czyli *Model - View - Controller*.

3.1.1 Doktryna Ruby on Rails

Tworzeniu i rozwojowi Ruby on Rails towarzyszy kilka podstawowych zasad. Jedne istnieją od początku, inne ewoluowały na przestrzeni lat. Spisane są one w *The Rails Doctrine* [?].

Optimize for programmer happiness

Pierwsza reguła odnosi się mocno do języka, z którego wywodzą się Rails’y. Sam fakt umieszczenia nazwy języka Ruby w nazwie frameworku pokazuje jak ważny jest on dla całego projektu. Motywacją stworzenia języka Ruby była chęć dostarczenia programistom radości z pisania kodu. W momencie powstawania Ruby’ego, większość popularnych wtedy języków programowania narzucało sposób pisania bądź stosowało liczne ograniczenia w sposobie pisania kodu. Ruby stał w opozycji do tych zasad, dając programistom pełną dowolność w sposobie tworzenia kodu.

Jako przykład owego „uwolnienia” programisty podawana jest *Zasada Najmniejszego Zaskoczenia* (ang. The Principle of Least Surprise).

Listing 3.1: Wyjście z interpretera Ruby'ego.

```
$ irb
irb(main):001:0> exit
$ irb
irb(main):001:0> quit
```

Listing 3.2: Wyjście z interpretera Python'a.

```
$ python
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
```

Ruby zaakceptuje oba polecenia opuszczenia interpretera. Python natomiast, pomimo odczytania intencji programisty (wyświetlenie instrukcji), opuści interpreter jedynie po wpisaniu komendy *exit()* lub po użyciu kombinacji klawiszy *Ctrl-D*, co oczywiście nie jest zgodne z oczekiwaniami programisty.

Wzorując się na zasadach, na których powstawał Ruby, Rails'y również miały umilać życie programistom. Jako przykład jest podawana klasa *Inflector*. Zapisane są w niej reguły oraz wyjątki od reguł w języku angielskim, które umożliwiają mapowanie klasy *Comment* na tabelę *Comments*, *Person* na *People* itp. Kolejnym przykładem może być dodatek do Ruby'owej klasy *Array*, który poza dostępnym w Rubym pierwszym elementem tablicy, umożliwia dostęp do kolejnych czterech elementów tablicy, poprzez wyrażenia *Array#second*, *Array#third* itd.

Oczywiście nie są to kluczowe cechy frameworku, ale stanowczo zaznaczają ważkość celu, którym jest przede wszystkim sprawianie radości z tworzenia oprogramowania.

Convention over Configuration

Twórcy Ruby on Rails starają się kłaść mocny nacisk na prostotę używania ich narzędzia. Zamiast zrzucać na programistów ciągłe podejmowanie decyzji w kwestiach mało istotnych, jak na przykład format klucza obcego w bazie danych, Rails'y narzucają konwencję nazewnictwa. Oczywiście jest możliwość konfiguracji narzuconych przez framework konwencji, lecz jest to raczej rzadko spotykane, a wręcz niewskazane. Zasada ta, poza zdjęciem odpowiedzialności za część decyzji z barków programisty, przynosi również inne korzyści. Dzięki „domyślnej” konwencji, możliwe jest stworzenie głębszej abstrakcji, na której operuje framework. Przykładem może być zastosowanie wcześniej wymienionej klasy *Inflector*. Jeśli możliwe jest zmapowanie klasy *Person* na tabelę *People*, to możliwe jest również zmapowanie relacji *has_many* : *people* tak i *posb*, aby domyślnie wykorzystywaną była klasa *Person*.

3.2 Phoenix

3.3 Express

Rozdział 4

Analiza porównawcza

4.1 Opisowe porównanie charakterystyki wybranych frameworków

4.1.1 Ruby on Rails vs Phoenix

4.1.2 Ruby on Rails vs Express

4.1.3 Phoenix vs Express

4.2 Subiektywna ocena

Rozdział 5

Implementacja

5.1 Ruby on Rails

5.2 Phoenix

5.3 Express

Rozdział 6

Projekt komputerowego środowiska eksperymentalnego

6.1 Plan prowadzenia eksperymentów

6.2 Wykorzystane narzędzia

6.2.1 Nginx

6.2.2 Docker

6.3 Benchmarki

6.3.1 Ruby

6.3.2 Elixir

6.3.3 JavaScript

Rozdział 7

Analiza wydajnościowa

7.1 Metody mierzenia wydajności aplikacji internetowych

7.2 Wyniki badań

Rozdział 8

Podsumowanie