

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

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

PRACA DYPLOMOWA
MAGISTERSKA

Analiza porównawcza frameworków
internetowych w języku Ruby w zastosowaniach
GISowych

Comparative analysis of Ruby's web frameworks
for Geographic Information Systems

AUTOR:
Mikołaj Grygiel

PROWADZĄCY PRACĘ:
dr inż. Roman Ptak

OCENA PRACY:

Spis treści

1 Wprowadzenie	3
1.1 Cel pracy	3
1.2 Zakres i koncepcja pracy	3
2 Podstawy teoretyczne	5
2.1 Charakterystyka Systemów Informacji Geograficznej	5
2.2 Charakterystyka języka Ruby	6
3 Istniejące systemy GIS w języku Ruby	7
3.1 OpenStreetMap	7
3.2 MangoMap	7
4 Frameworki internetowe w języku Ruby	9
4.1 Ruby on Rails	10
4.2 Roda	11
4.3 Hanami	12
5 Narzędzia dostępne do przetwarzania danych geograficznych w języku Ruby	15
5.1 Przechowywanie danych	15
5.1.1 PostgreSQL	15
5.1.2 PostGIS	15
5.1.3 MySQL Spatial	15
5.1.4 SpatiaLite	15
5.2 Obiektowe przetwarzanie danych	16
5.2.1 RGeo	16
5.2.2 GeoRuby	16
5.3 Prezentowanie danych	16
5.3.1 Leaflet	16
5.3.2 GoogleMaps	16
6 Badania	17
6.1 Plan badań	17
6.1.1 Aplikacja zaimplementowana na potrzeby badań	17
6.1.2 Narzędzia wykorzystane podczas badań	20
6.1.3 Środowisko badawcze	20
6.2 Porównanie funkcjonalności wybranych frameworków	20
6.2.1 Przechowywanie danych	20
6.2.2 Przetwarzanie danych	21
6.2.3 Prezentowanie danych	21

6.2.4	Podsumowanie	22
6.3	Porównanie struktur wykonanych projektów	23
6.4	Porównanie wydajności zaimplementowanych aplikacji	26
6.4.1	Interfejs użytkownika	26
6.4.2	Zapytania HTTP	29
6.4.3	Komunikacja z bazą danych	35
7	Podsumowanie	39
7.1	Analiza wyników badań	39
7.2	Realizacja celu projektu	39
Literatura		41

Rozdział 1

Wprowadzenie

1.1 Cel pracy

Język Ruby zajmuje 12 miejsce w rankingu popularności języków programowania *Tiobe*¹. Dużą popularnością wśród frameworków internetowych cieszy się Ruby on Rails, w rankingu *Hotframeworks*² zajmuje 3 miejsce wśród wszystkich frameworków. Ruby on Rails jest bez wątpienia najpopularniejszym frameworkm w języku Ruby, kolejne dwa frameworki w języku Ruby to Sinatra i Hanami, zajmują w wcześniej przytoczonym rankingu odpowiednio miejsca 25. i 73. Jednak w języku Ruby istnieje kilkanaście frameworków przeznaczonych do budowania aplikacji internetowych.

Celem niniejszej pracy jest poznanie wybranych frameworków w języku Ruby, ich porównanie w konkretnym zastosowaniu jakim są systemy informacji geograficznej oraz odpowiedź na pytanie jaki framework najlepiej wybrać do tworzenia systemu GIS.

1.2 Zakres i koncepcja pracy

”Framework” można zdefiniować jako szkielet służący do budowania aplikacji, czyli zbiór gotowych rozwiązań powtarzających się problemów i wzór do budowania nowych funkcjonalności.[15]

W niniejszej pracy zostaną omówione wybrane frameworki w języku Ruby w świetle ich użyteczności przy budowie systemu informacji geograficznej. Frameworki zostaną porównane na podstawie informacji zawartych w dostępnej dokumentacji narzędzia oraz zaimplementowanej przykładowej aplikacji, za pomocą każdego z wybranych narzędzi, spełniającej wymagania systemu GIS.

¹Dane z marca 2017 roku dostępne na stronie <https://www.tiobe.com/tiobe-index/>

²Ranking <https://hotframeworks.com> bierze pod uwagę liczbę repozytoriów kodu na platformie Github i ilość tematów na forum Stackoverflow dotyczących danego frameworku. Dane z dnia 26.03.2017 r.

Rozdział 2

Podstawy teoretyczne

2.1 Charakterystyka Systemów Informacji Geograficznej

System Informacji Geograficznej skrótnie nazywany GIS (ang. Geographic Information System) można zdefiniować na wiele sposobów. Michael Schmandt w swoim opracowaniu[14] podaje następujące definicje:

Definicja 1. GIS jest to system komputerowy składającym się z sprzętu i oprogramowania oraz ludzie, którzy wspomagają zbieranie, zarządzanie, analizowanie i wyświetlanie danych przestrzennych. Stosując tą definicję możemy podzielić system GIS na 4 moduły:

- moduł wprowadzania danych - zawiera narzędzia pozwalające na wprowadzanie i przechowywanie danych przestrzennych.
- moduł zarządzania danymi - ta część umożliwia edytowanie oraz przeglądanie zgromadzonych zbiorów danych
- moduł analizowania danych - podsystem, który odpowiada za analizowanie danych geograficznych i wyciągania z nich informacji
- moduł prezentowania danych - pozwala na tworzenie map, modeli, statystyk ilustrujących zgromadzone dane

Definicja 2. System informacji geograficznej to system komputerowy, który pozwala na przechowywanie danych powiązanych ze sobą geograficznie.

Definicja 3. GIS to narzędzie do wyszukiwania wzorców geograficznych(przestrzennych) w zbiorach danych.

Pierwsza definicja jest najbardziej szeroka i zawiera w sobie dwie następne - definicja druga to dwa pierwsze moduły z **Definicja 1.**, a definicja trzecia to moduł analizowania danych.

W podobny sposób do definicji nr 1 GIS jest zdefiniowany w *Principles of Geographic Information Systems*[11] jako zbiór narzędzi pozwalających operować na danych reprezentujących zjawiska geograficzne. Zbiór ten dzieli się na 4 grupy ze względu na funkcje:

- zbieranie i przygotowywanie danych

- zarządzanie i przechowywanie danych
- analiza danych
- prezentowanie danych

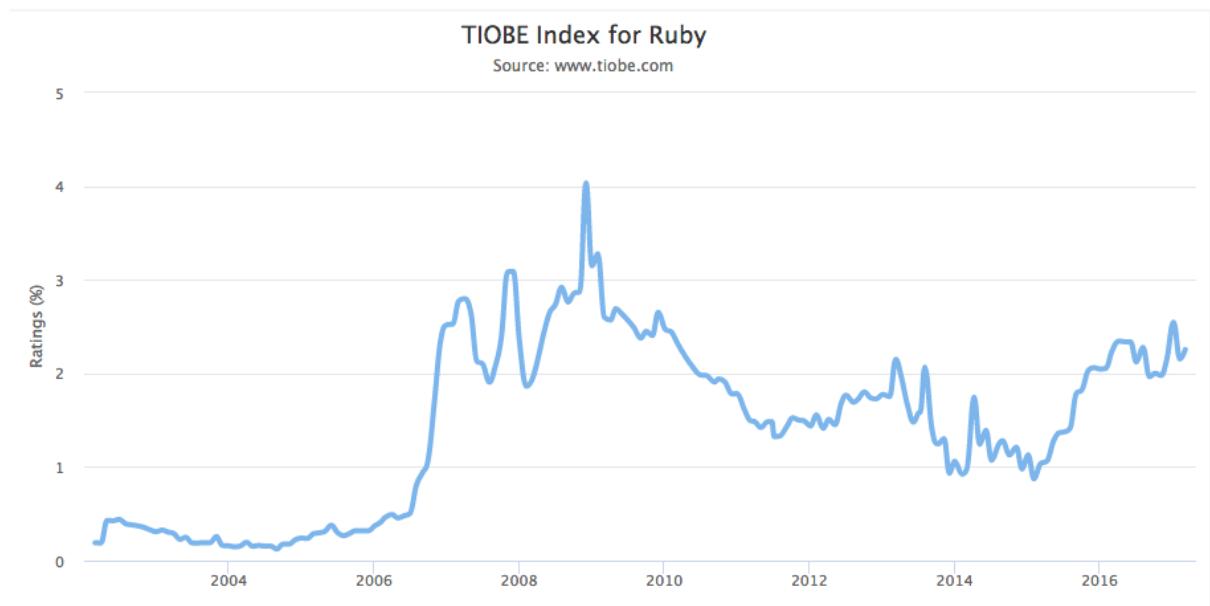
W poniższej pracy przyjmuje się pierwszą definicję Systemu Informacji Geograficznego - GIS to system informatyczny służący do wprowadzania, przechowywania, zarządzania, analizowania i prezentacji danych przestrzennych.

2.2 Charakterystyka języka Ruby

Język Ruby został wydany w 1995 roku. Twórca Rubiego, Yukihiro “Matz” Matsumoto, inspirował się takimi językami programowani jak Perl, Smalltalk, Eiffel, Ada i Lisp by stworzyć jego zdaniem język, który zbalansuje programowanie funkcyjne z programowaniem imperatywnym[3]. Składnia Rubiego ma przypominać język naturalny, autor języka opisuje go jako: *Ruby jest prosty z wyglądu, ale bardzo skomplikowany w środku, tak jak ciało ludzkie.*¹

Ruby jest językiem ścisłe obiektowym, wszystko postrzegane jest jako obiekt. Każda funkcja jest metodą, ponieważ musi być przyłączona do jakiegoś obiektu. Ruby posiada celowo tylko jednokrotne dziedziczenie, ale pozwala na dołączanie wielu modułów, które są zbiorami metod do klasy. Ruby jest bardzo elastycznym językiem, pozwala na usunięcie lub przeddefiniowanie dowolnej swojej części. Mimo silnie obiektowej natury, dostępne są również elementy programowania funkcyjnego takie jak funkcje anonimowe lub domknięcia.

Szerszą popularność Ruby zyskał w 2006r., 11 lat po publikacji. Swoją popularność zawdzięcza głównie frameworkowi Ruby on Rails. w rankingu popularności języków programowania Tiobe znajduje się aktualnie na 12 miejscu².



Rysunek 2.1 Historia popularności języka Ruby według rankingu Tiobe

¹wypowiedź w liście ruby-talk 12.05.2000 r., źródło: <http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/2773>

²dane z dnia 08.04.2016 r. <https://www.tiobe.com/tiobe-index/>

Rozdział 3

Istniejące systemy GIS w języku Ruby

3.1 OpenStreetMap

OpenStreetMap jest internetowym systemem informacji geograficznej z otwartym kodem źródłowym. System zbudowany z wykorzystaniem bazy danych PostgreSQL, frameworku Ruby on Rails oraz biblioteki javascriptowej Leaflet służącej do tworzenia interaktywnych widoków z mapami. Dostęp do danych jest otwarty, dane mogą być edytowane przez dowolnego użytkownika, dlatego mogą być niezgodne z rzeczywistością. OpenStreetMap definiuje 4 typy obiektów przestrzennych[5]:

- Węzeł (ang. node) - pojedynczy punkt geoprzestrzenny reprezentowany przez długość i szerokość geograficzną.
- Linia (ang. way) - jest to uporządkowany zbiór punktów, które mogą reprezentować funkcje liniowe(wektory) lub obszary.
- Relacja (ang. relation) - składa się z uporządkowanej listy węzłów, linii i innych relacji.
- Tag (ang. tag) - to jednostka informacji dołączona do obiektu jednego z powyżej opisanych typów. Tag składa się z klucza oraz wartości.

OpenStreetMap można wykorzystać przez utworzenie komponentu HTML z wybraną mapą, gotowego do zamieszczenia na dowolnej stronie internetowej lub przez pobranie danych z wybranej mapy. Skompresowane aktualne dane dla całej planety z pojedynczego dnia zajmują prawie 40GB. Można pobierać również dane historyczne.

3.2 MangoMap

MangoMap jest komercyjnym narzędziem do tworzenia map dostępnych przez internet z własnych danych geoprzestrzennych. Ceny za korzystanie z serwisu wynoszą 49-399\$ miesięcznie w zależności od liczby map i udostępnianego miejsca na serwerze do przechowywania danych. System zbudowany jest w oparciu o framework Ruby on Rails. Mapy tworzy się przy użyciu interfejsu graficznego. Stworzone mapy mogą być udostępnione na serwerze MangoMap przez unikalny link lub zamieszczone na zewnętrznej stronie WWW przez komponent HTML[4].

Rozdział 4

Frameworkki internetowe w języku Ruby

W języku Ruby istnieje kilkanaście wspieranych frameworków internetowych. Wybór wykorzystanych frameworków w niniejszej pracy dokonano w następujący sposób:

1. Podzielono frameworki według daty opublikowania pierwszej wersji na 3 grupy:
 - (a) opublikowane w latach 2004 - 2011 - frameworki o ugruntowanej pozycji
 - (b) opublikowane w latach 2012 - 2015 - stosunkowo nowe frameworki
 - (c) opublikowane w latach 2016 - 2017 - najnowsze frameworki
2. z każdej grupy wybrano framework z największą ilością pobrań.

Ta metoda ma na celu wyłonienie najpopularniejszych frameworków, które powstały w różnych etapach języka Ruby, jednocześnie każdy z nich współpracuje z najnowszą wersją języka. W ten sposób wybrano *Ruby on Rails*, *Trailblazer* i *Hanami*.

Tablica 4.1 Frameworki internetowe w języku Ruby¹

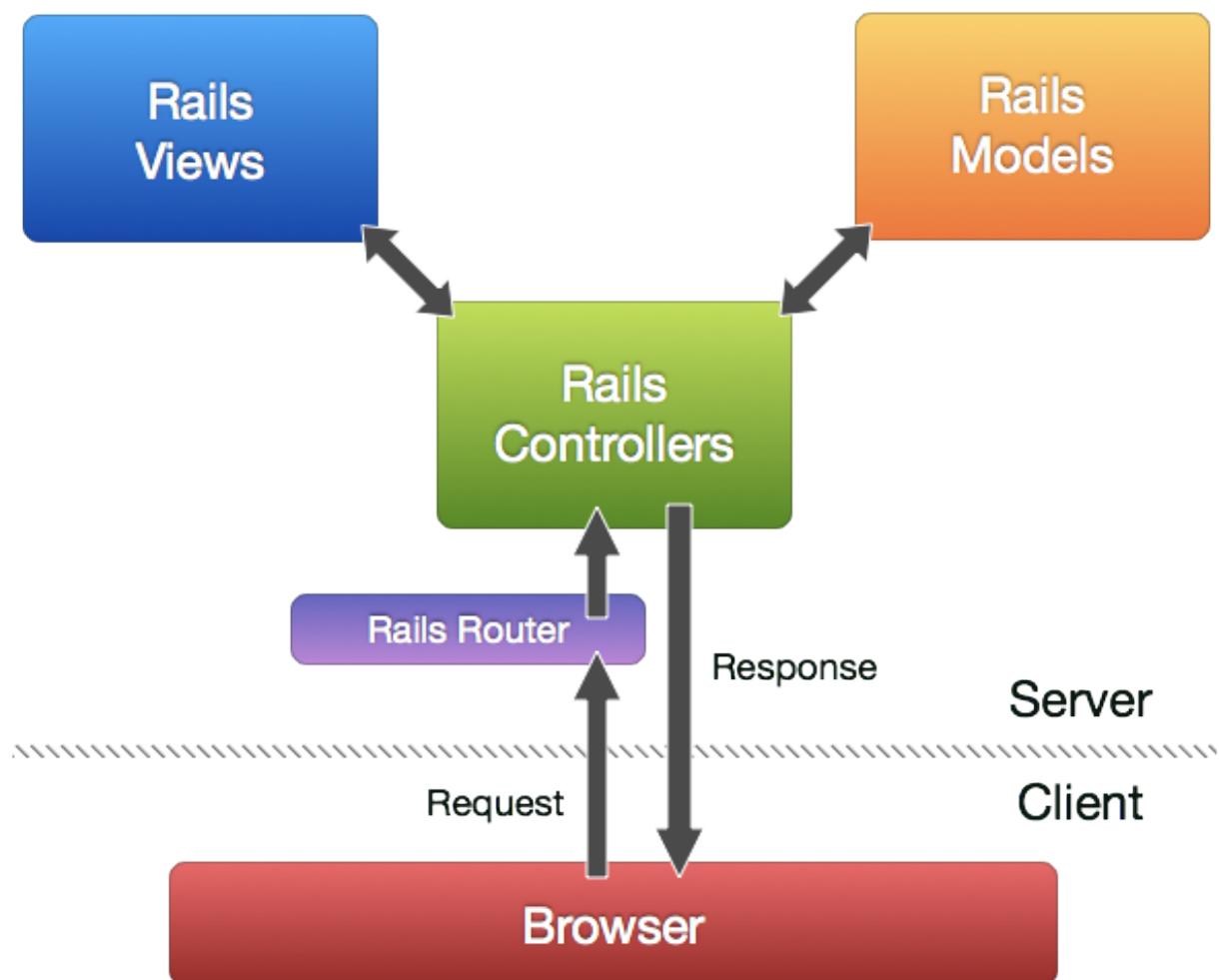
Nazwa	Data opublikowania pierwszej wersji	Data opublikowania najnowszej wersji	Ilość pobrań
Ruby on Rails	25.10.2004 r.	20.03.2017 r.	91 898 706
Hobo	29.04.2007 r.	07.05.2016 r.	211 042
Sinatra	04.10.2007 r.	19.03.2017 r.	45 207 501
Padrino	16.11.2009 r.	23.03.2017 r.	556 481
Cuba	25.04.2010 r.	01.07.2016 r.	124 371
Strelka	24.08.2011 r.	19.01.2017 r.	57 007
Pakyow	20.09.2011 r.	15.07.2016 r.	18 222
Scorched	03.03.2013 r.	12.10.2016 r.	26 929
Trailblazer	26.07.2013 r.	23.01.2017 r.	96 217
Roda	20.07.2014 r.	15.03.2017 r.	105 103
Vanilla	09.05.2015 r.	05.07.2016 r.	80 125
Hanami	20.01.2016 r.	06.04.2017 r.	28 885
Dry-web	21.04.2016 r.	02.02.2017 r.	7 190

¹Zestawienia przygotowano na podstawie danych z <https://rubygems.org/> oraz <https://www.ruby-toolbox.com>. Pominięto frameworki, których ostatnia wersja ukazała się ponad rok temu. Aktualne na dzień 08.04.2017 r.

4.1 Ruby on Rails

Pierwsza stabilna wersja (1.0.0) frameworku Ruby on Rails ukazała się pod koniec 2005 roku, aktualna wersja (5.0.2) została opublikowana 02.03.2017 r. Aplikacja zbudowana z wykorzystaniem RoR jest oparta o wzorzec projektowy Model-Widok-Kontroler[13] (ang. Model-View-Controller). Aplikacja oparta na tym wzorcu jest podzielona na 3 części:

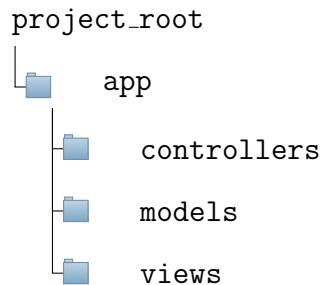
- Modele (ang. model) - reprezentują logikę biznesową. w tej warstwie znajdują się wszelkie obiekty, które służą do wykonywania operacji związanych z implementacją funkcjonalności aplikacji.
- Widoki (ang. view) - służą do prezentowania danych.
- Kontrolery (ang. controller) - obsługują zapytania użytkownika. Nie zawierają w sobie żadnej logiki biznesowej.



Rysunek 4.1 Architektura aplikacji w Ruby on Rails, źródło:
<http://blog.ifuturz.com/ruby-on-rails/ruby-on-rails-mvc-learn-with-fun.html>

Dwie główne zasady frameworku[8]:

- Nie powtarzaj się (ang. Don't Repeat Yourself) - każda informacja powinna mieć pojedynczą, jednoznaczną i autorytatywną reprezentację w kodzie źródłowym systemu. Ułatwia to utrzymywanie kodu.
- Konwencja ponad konfiguracje (ang. Convention Over Configuration) - RoR posiada ustalone zasady postępowania w różnych przypadkach. Aplikacja domyślnie zachowuje się według tych zasad, nie wymagając dodatkowej konfiguracji. Pozwala to zredukować ilość kodu źródłowego.



Rysunek 4.2 Podstawowa struktura projektu Ruby on Rails

4.2 Roda

Twórcy frameworku Roda jest przy tworzeniu narzędzia podstawili sobie 4 cele[9]:

- Prostota (ang. simplicity) - framework ma być prosty zarówno wewnętrzny (w implementacji), jak i zewnętrzny (dla użytkowników).
- Niezawodność (ang. reliability) - Roda wspiera i promuje projektowanie aplikacji z niemutowalnym stanem. Aplikacje zbudowane za pomocą Rody, są zaprojektowane tak, aby ograniczać zmienne, stałe i metody przypisane do instancji obiektów aby uniknąć konfliktów z kodem zaimplementowanym przez użytkownika.
- Rozszerzalność (ang. extensibility) - framework zbudowany jest całkowicie w oparciu o wtyczki, aby ułatwić dodawanie funkcjonalności do framework'a. Każda część Roda może być zastąpiona własną implementacją przez użytkownika.
- Wydajność (ang. performance) - Roda posiada mały koszt obsługi zapytań, drzewo trasowania i inteligentną obsługę pamięci podręcznej co sprawia, że Roda jest szybsza od innych popularnych frameworków języka Ruby.

Roda opiera się o drzewo trasowania, punkty dostępu aplikacji zdefiniowane są w strukturze drzewa. Przykład drzewa trasowań znajduje się we fragmencie kodu 4.1.

Fragment kodu 4.1 Proste drzewo trasowań

```

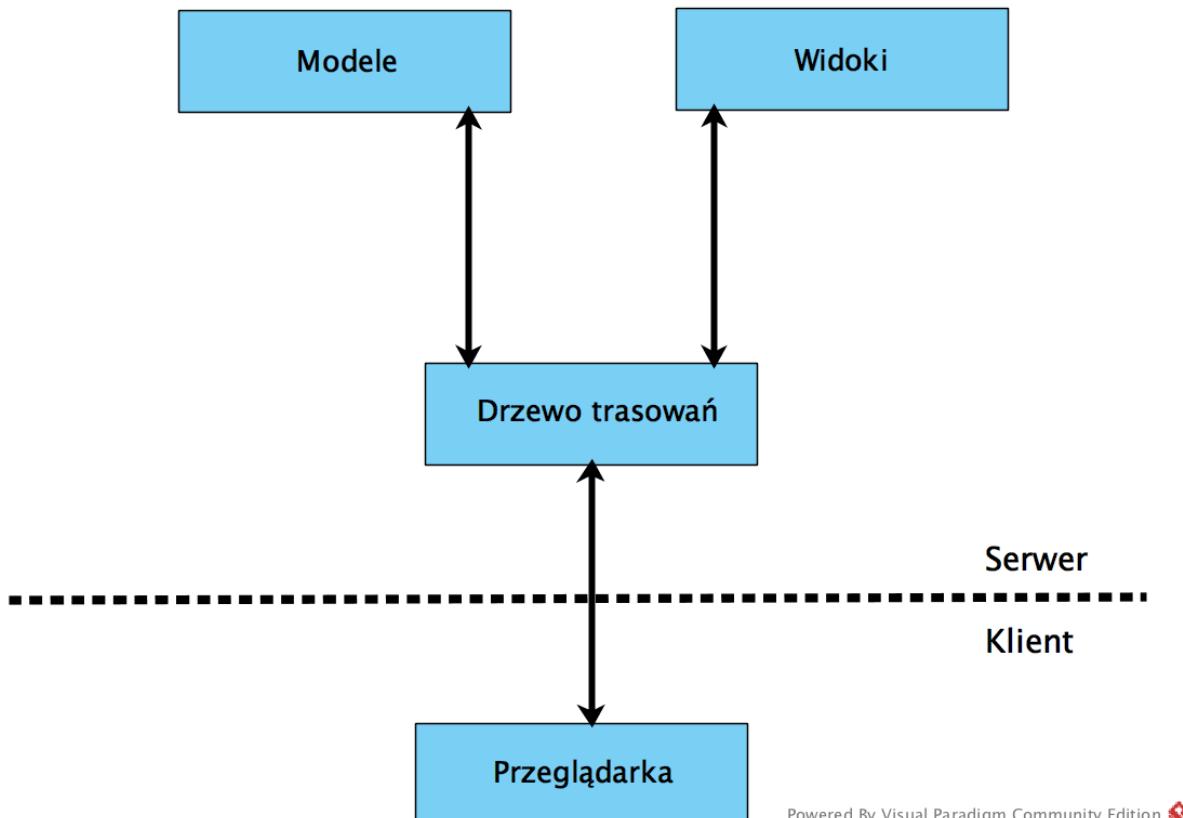
r.on "a" do          # /a gałęz
  r.on "b" do        # /a/b gałęz
    r.is "c" do      # /a/b/c zapytanie
      r.get do end   # GET /a/b/c zapytanie
      r.post do end  # POST /a/b/c zapytanie
  
```

```

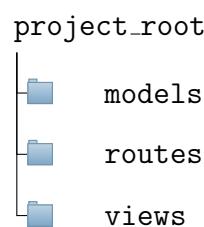
end
r.get "d" do end # GET /a/b/d zapytanie
r.post "e" do end # POST /a/b/e zapytanie
end
end

```

W przeciwieństwie do Ruby on Rails, Roda nie posiada warstwy kontrolerów i osobnego modułu obsługującego trasowanie, w Rodzie przy definicji danego punktu końcowego znajduje się od razu kod obsługujący otrzymane zapytanie.



Rysunek 4.3 Architektura aplikacji w frameworku Roda



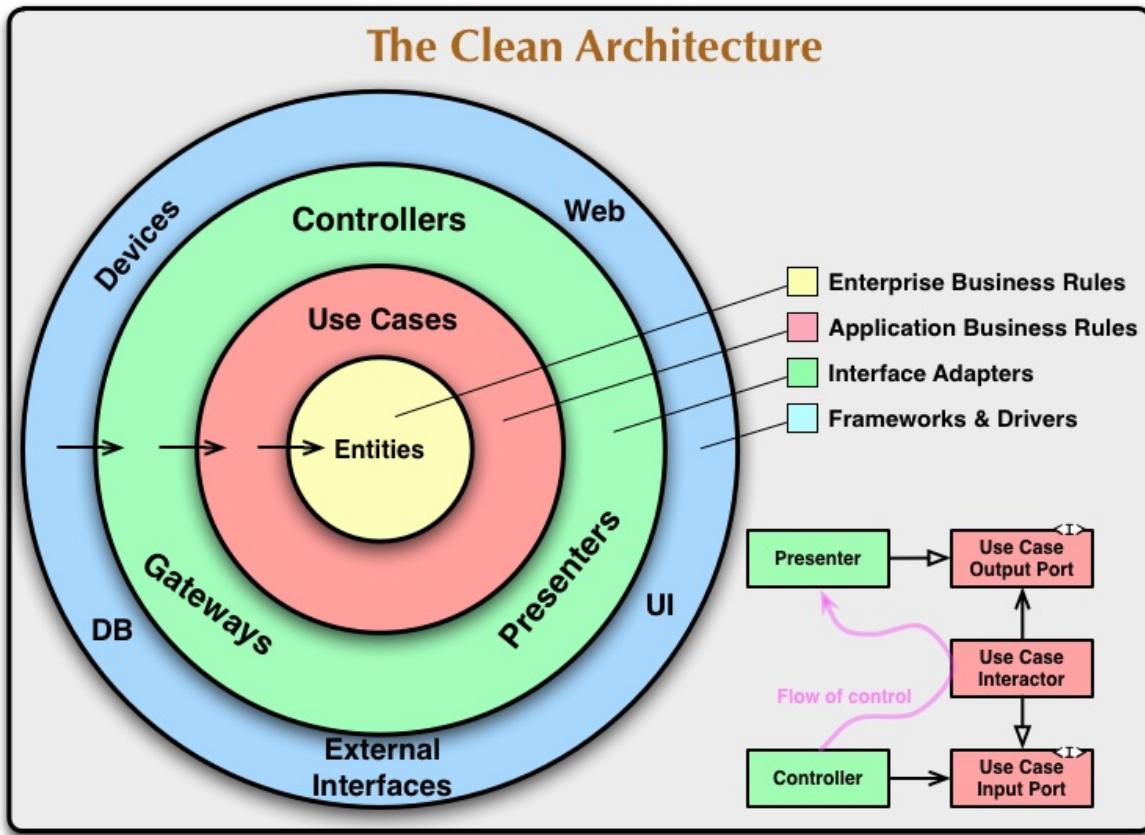
Rysunek 4.4 Struktura projektu Roda

4.3 Hanami

Hanami jest lekkim frameworkiem internetowym opartym o architekturę MVC, zbudowanym z wielu mikro-bibliotek. w przeciwieństwie do Ruby on Rails, Hanami nie stawia konwencji ponad konfiguracje, wszystkie informacje powinny być zawarte w napisanym

przez programistę kodzie, którego zrozumienie nie wymaga znajomości konwencji frameworka. Architektura projektu jest głównie inspirowana przez te dwa podejścia:

- Czysta architektura (ang. clean architecture) - schemat tej architektury składa się z kolejnych zawierających się w sobie kół. Każde wewnętrzne koło nie ma żadnych zależności w zewnętrznym kole, czyli w obiektach należących do wewnętrznego koła, nie ma odwołań do obiektów zewnętrznych kól[12]. To podstawowa zasada tego podejścia.



Rysunek 4.5 Schemat czystej architektury[12]

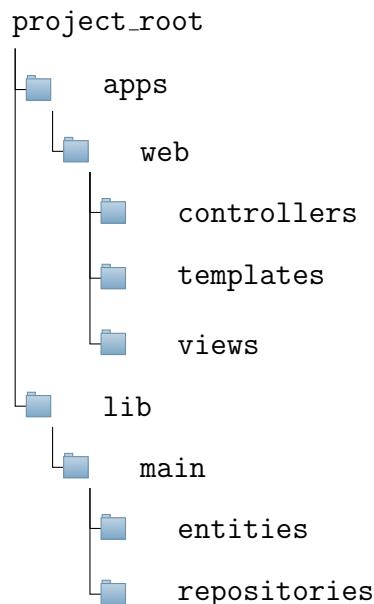
Na schemacie 4.5 można wyróżnić 4 byty:

- Encje (ang. entities) - zawierają ogólne reguły biznesowe systemu. Mogą być reprezentowane przez obiekty z metodami, struktury danych lub pojedyncze funkcje.
- Przypadki użycia (ang. use cases) - ta warstwa skupia w sobie wszystkie możliwe przypadki użycia systemu przez użytkownika. Znajduje się tu cała logika biznesowa zarządzania encjami. Zmiana w tej warstwie nie powinna wpływać na encje, interfejs użytkownika lub bazę danych.
- Adaptery interfejsów (ang. interface adapters) - na tym poziomie dane są konwertowane z formatu najbardziej dogodnego dla przypadków użycia i encji do formatu przyjmowanego przez zewnętrzne narzędzia takie jak baza danych lub przeglądarka internetowa.

- Frameworki oraz narzędzia (ang. frameworks and drivers) - w tej warstwie znajdują się zewnętrzne narzędzia takie jak inne frameworki, baza danych lub przeglądarka internetowa.
- Najpierw monolit (ang. monolith first) - według tej zasady budowę systemuaczyna się od monolitycznej aplikacji. Jednak budowa aplikacji od samego początku powinna być jak najbardziej modularna aby można było ją później rozbić na wiele mniejszych aplikacji.

Zgodnie z zasadą *czystej architektury* w utworzonym projekcie systemu z użyciem Hanami oddzielona jest warstwa logiki biznesowej znajdująca się w folderze *lib* projektu od mechanizmu komunikacji z innymi serwisami zawartego w folderze *apps*.

Hanami posiada kontener aplikacji, który pozwala nam utworzyć wiele aplikacji w ramach jednego projektu, które korzystają z tego samego zbioru encji i przypadków użycia, a następnie uruchomić je w jednym procesie Rubiego.



Rysunek 4.6 Podstawowa struktura projektu Hanami

Rozdział 5

Narzędzia dostępne do przetwarzania danych geograficznych w języku Ruby

5.1 Przechowywanie danych

Omawiane frameworki domyślnie wykorzystują relacyjną bazę danych do przechowywania informacji. Najpopularniejsze bazy danych to PostgreSQL, MySQL, SQLite.

5.1.1 PostgreSQL

PostgreSQL posiada kilka wbudowanych typów przestrzennych: *Point*, *Lines*, *Line Segment*, *Box*, *Path*, *Polygon*, *Circle*. Dane są zapisane jako łańcuch znaków zawierające współrzędne geograficzne. PostgreSQL dostarcza niewiele funkcji do wyszukiwania danych.

5.1.2 PostGIS

PostGIS jest biblioteką rozszerzającą możliwości PostgreSQL w zakresie przetwarzania danych przestrzennych. Biblioteka jest wspierana przez fundacje Open Source Geospatial. Dane są zapisane w formie binarnej jako współrzędne geograficzne lub geometryczne w zależności od wyboru.

5.1.3 MySQL Spatial

5.1.4 SpatiaLite

SpatiaLite jest rozszerzeniem dla bazy danych SQLite, które zawiera wsparcie dla typów przestrzennych zdefiniowanych przez grupę *Open Geospatial Consortium*[10]: *Point*, *Line*, *Multiline*, *Polygon*, *Multipolygon*. Cała baza danych zapisana jest w pojedynczym pliku. Biblioteka udostępniona jest na zasadzie otwartego źródła.

5.2 Obiektowe przetwarzanie danych

5.2.1 RGeo

RGeo wspiera typy danych zdefiniowane przez *Open Geospatial Consortium*, takie jak punkt, linia i wielokąt. Dane są reprezentowane w formie obiektowej. Biblioteka pozwala na podstawowe operacje analizowania danych przestrzennych, takie jak szukanie punktów przecięcia i obliczenia pola powierzchni. Biblioteka operuje na reprezentacji geograficznej i gemoetrycznej danych i pozwala konwertować dane pomiędzy dostępnymi reprezentacjami. RGeo udostępnia udostępnia jedynie interfejs w języku Ruby dla programistów, a do obsługi danych geograficznej korzysta z bibliotek GEOS i proj.4, te biblioteki są napisane w języku C++.

5.2.2 GeoRuby

GeoRuby

5.3 Prezentowanie danych

5.3.1 Leaflet

5.3.2 GoogleMaps

Rozdział 6

Badania

6.1 Plan badań

Na podstawie dostępnej dokumentacji oraz zaimplementowanych aplikacji w każdym z badanych narzędzi, frameworki zostaną porównane w następujących aspektach:

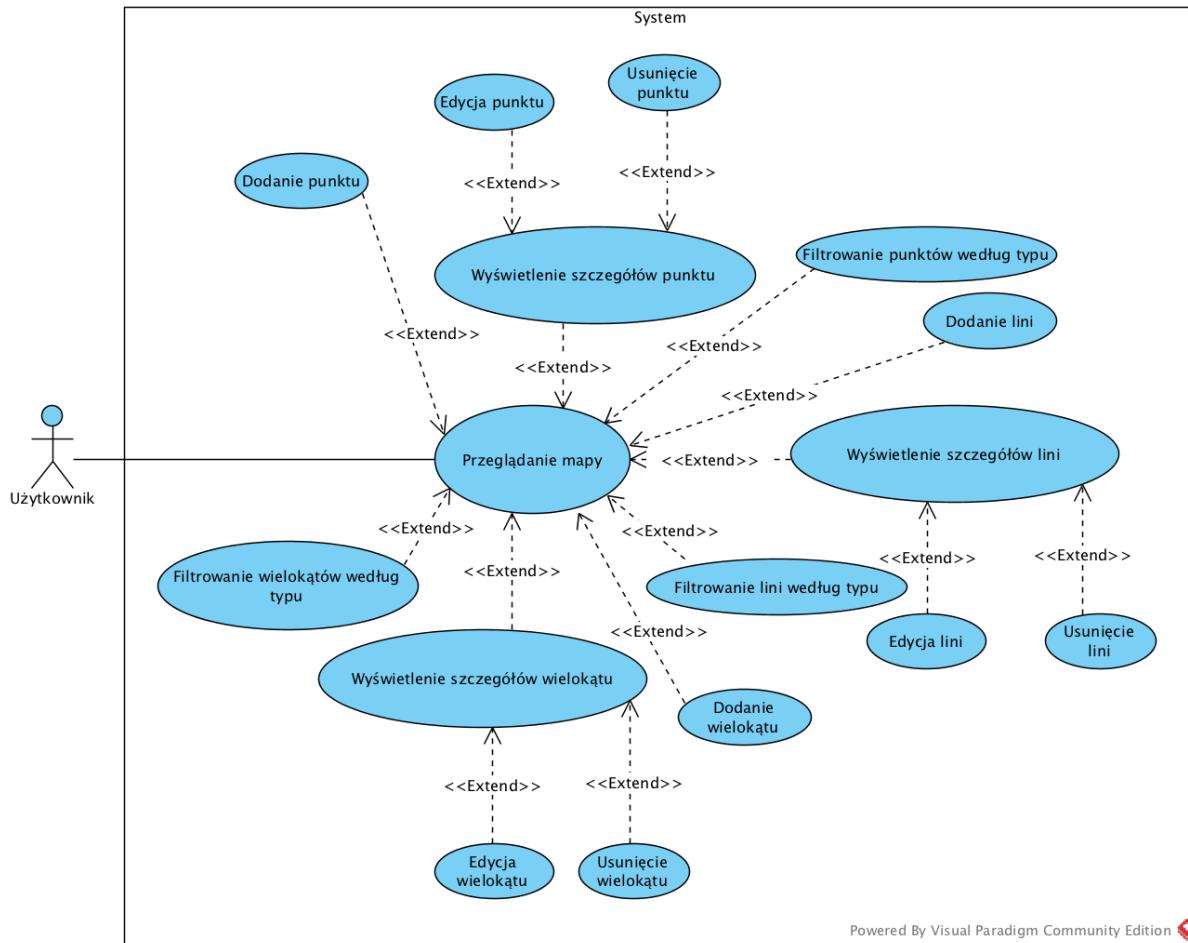
- dostępne funkcjonalności
- struktura utworzonego projektu
- wydajność zaimplementowanej aplikacji

6.1.1 Aplikacja zaimplementowana na potrzeby badań

W celu wykonania analizy porównawczej w każdym z wybranych frameworków została zaimplementowana aplikacja internetowa operująca na danych geograficznych posiadająca takie same funkcjonalności. Aplikacje zostały zrealizowane zgodnie z architekturą i konwencją wykorzystywanymi w danym frameworku zawartych w dokumentacji narzędzia. W projektach wykorzystano jedynie biblioteki niezbędną do zrealizowania wymaganych funkcjonalności. Nie używano bibliotek modyfikujących architekturę aplikacji lub funkcjonalności frameworka. Aplikacja ma za zadanie wykonywać 4 podstawowe operacje na danych geograficznych:

- tworzenie
- odczytywanie
- aktualizowanie
- usuwanie

Wszystkie funkcjonalności wykonanej aplikacji zostały zaprezentowane na rysunku 6.1.



Rysunek 6.1 Diagram przypadków użycia aplikacji

Aby przeprowadzić badania w realistycznych warunkach aplikacja korzysta z gotowych zbiorów danych:

- punkty - pochodzące ze zbioru "Państwowy rejestr nazw geograficznych - miejscowości", zbiór zawiera 256796 obiektów, dostępny jest pod adresem <http://www.codgik.gov.pl/index.php/dane/prng.html>
- linie - dane uzyskano z serwisu <http://download.geofabrik.de/europe/poland.html> pozwalającego pobrać drogi z terenu Polski z bazy danych OpenStreetMap. Zbiór obejmuje 2519660 obiektów
- wielokąty - obiekty pobrano z Państwowego Rejestru Granic dostępnego pod adresem <http://gis-support.pl/baza-wiedzy/dane-do-pobrania/>, zbiór liczy 59978 obiektów

Każda wersja aplikacji posiada taką samą bazę danych, której schemat znajduje się na rysunku 6.2

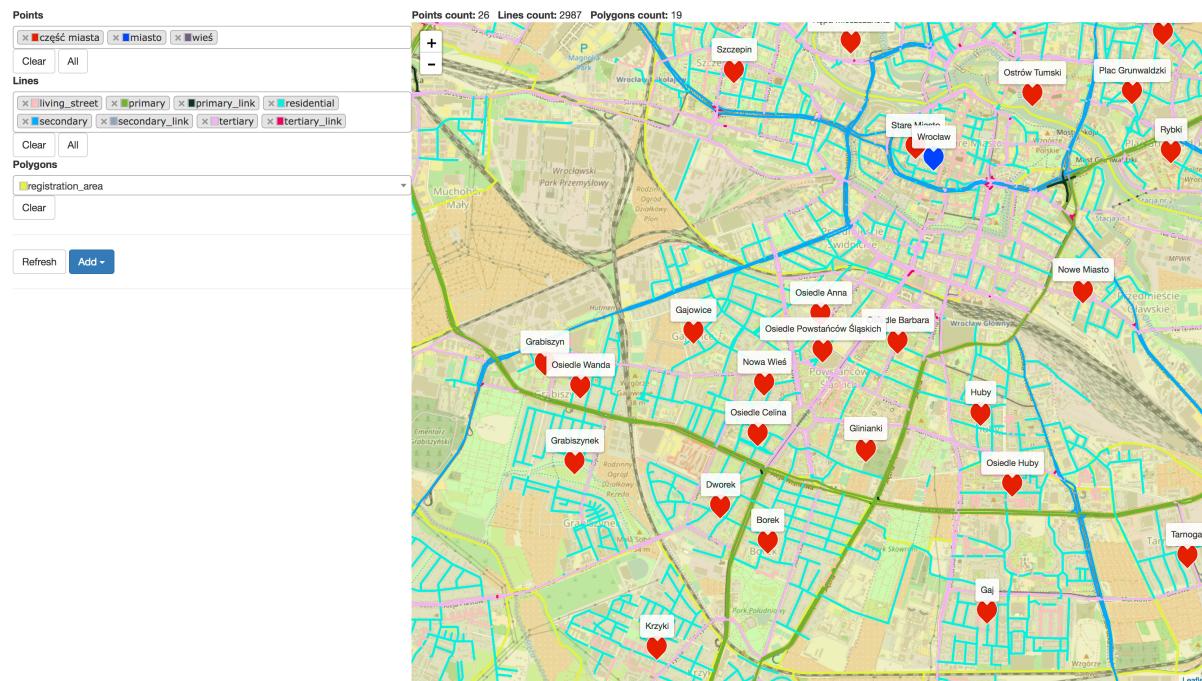
points	
id	SERIAL
coordinates	geometry
name	CHARACTER VARYING
object_type	CHARACTER VARYING
object_class	CHARACTER VARYING
voivodeship	CHARACTER VARYING
county	CHARACTER VARYING
commune	CHARACTER VARYING
terc	CHARACTER VARYING
created_at	TIMESTAMP(6) WITHOUT TIME ZONE
updated_at	TIMESTAMP(6) WITHOUT TIME ZONE

polygons	
id	SERIAL
name	CHARACTER VARYING
coordinates	geometry
terc	CHARACTER VARYING
unit_type	INTEGER
created_at	TIMESTAMP(6) WITHOUT TIME ZONE
updated_at	TIMESTAMP(6) WITHOUT TIME ZONE

lines	
id	SERIAL
name	CHARACTER VARYING
coordinates	geometry
road_type	CHARACTER VARYING
created_at	TIMESTAMP(6) WITHOUT TIME ZONE
updated_at	TIMESTAMP(6) WITHOUT TIME ZONE

Rysunek 6.2 Schemat ERD bazy danych

Do prezentacji danych na mapie wykorzystano bibliotekę LefletJS, ponieważ jest darmowa i ma otwarte źródła. Mapa pochodzi z serwisu OpenStreetMap. Widok gotowej aplikacji znajduje się na rysunku 6.3.



Rysunek 6.3 Widok gotowej aplikacji

6.1.2 Narzędzia wykorzystane podczas badań

selenium, capybara, jmeter, ruby benchmark

6.1.3 Środowisko badawcze

6.2 Porównanie funkcjonalności wybranych frameworków

6.2.1 Przechowywanie danych

Ruby on Rails posiada największe wsparcie w przechowywaniu danych geograficznych z rozpatrywanych narzędzi. Biblioteka ActiveRecord domyślnie wykorzystywana w Ruby on Rails do mapowania obiektowo-relacyjnego posiada adaptery dla następujących baz danych: PostgreSQL z dodatkiem PostGIS, MysqlSpatial, SpatiaLite. Za pomocą adapterów, w migracjach można używać typów kolumn z przestrzennych baz danych. Przy komunikacji typy przestrzenne z bazy danych mapowane są na typy zaimplementowane w bibliotece RGeo. Konfiguracja nie wymaga wiele pracy, wystarczy dodać odpowiednią bibliotekę do projektu i w konfiguracji bazy danych ustawić odpowiedni adapter. Od tego momentu można w migracjach przy definiowaniu kolumn używać wszystkich typów przestrzennych dostarczanych przez wybraną bazę danych. Dane przestrzenne podczas wykonywanych akcji między aplikacją, a bazą danych są automatycznie mapowane.

Hanami w migracjach pozwala używać kodu napisanego w języku sql do np. definiowania typów lub włączania dodatków w bazie danych, dlatego bez problemów można stworzyć i wersjonować bazę danych z typami przestrzennymi. Jednak framework, podobnie jak Ruby on Rails, posiada mechanizm automatycznego mapowania typów bazy danych do typów obiektów, który nie pozwala na definiowanie własnych typów lub własnych reguł mapowania. To ograniczenie powoduje otrzymanie wyjątku przy próbie uruchomienia aplikacji zawierającej nieznany typ. Framework wspiera jedynie typy przestrzenne zdefiniowane w bazie danych Postgresql, które są zapisywane postaci tekstowej zawierającej współrzędne. Możliwe jest zapisanie wszystkich 3 podstawowych typów przestrzennych: punktu, lini, wielokąta. Postgresql dostarcza podstawowe operacje wyszukiwania danych przestrzennych tylko dla punktów, dlatego zastosowanie Postgresql bez dodatku Postgis mocno ogranicza możliwości budowanej aplikacji. Dodanie obsługi typów geometrycznych dostarczonych przez dodatek Postgis nie wymaga dużej ilości kodu, wymagana jest dobra znajomość języka Ruby i mechanizmu przeddefiniowania istniejących już klas tzw. "monkey patching". Najłatwiejsze rozwiązanie pozwalające używać typów geometrycznych z biblioteki Postgis znajduje się na listingu 6.1. Przy takim rozwiążaniu wartości z kolumny geometrycznej mapowane są na typ String z wartością binarną.

Fragment kodu 6.1 Obsługa typów geometrycznych z biblioteki Postgis przez Hanami

```
require 'rom/sql/extensions/postgres/infererer'
require 'rom/sql/extensions/postgres/types'

ROM::SQL::Schema::PostgresInfererer.class_eval do
  alias map_db_type_original map_db_type

  def map_db_type(db_type)
    # najpierw następuje próba użycia oryginalnej metody mapowania typów
    # jeśli zakończy się ona nie powodzeniem, to następuje sprawdzenie
    # czy typ kolumny z bazy danych rozpoczyna się od słowa 'geometry'
    # wtedy dane są rzutowane do typu ROM::Types::String
    map_db_type_original(db_type) || (db_type.start_with?('geometry') ? ROM
      ::Types::String : nil)
  end
end
```

6.2.2 Przetwarzanie danych

Do obiektowego przetwarzania danych dostępne są dwie biblioteki Rgeo i GeoRuby. Ruby on Rails, dzięki wspomnianym wcześniej adapterom dużo wygodniej jest korzystać z Rgeo dzięki automatycznemu mapowaniu danych relacyjnych na obiekty. Framework Hanami w żaden sposób nie rozszerza możliwości bibliotek Rgeo i GeoRuby. W zaimplementowanej metodzie mapującej wartości przestrzenne na typ obiektowy otrzymujemy wartość binarną, którą dalej można zamienić na typ z biblioteki Rgeo, co zaprezentowano na listingu 6.2

Fragment kodu 6.2 Tworzenie obiektu Rgeo z zapisu binarnego danych przestrzennych

```
module CoordinatesHelper
  FACTORY = RGeo::Geographic.spherical_factory(:srid => 4326)

  # utworzenie obiektu Rgeo z binarnego zapisu danych przestrzennych
  def coordinates_object
    RGeo::WKRep::WKBParser.new(FACTORY, support_ewkb: true,
      default_srid: 4326).parse(coordinates)
  end

  # rzutowanie obiektu przestrzennego do postaci "well known text"
  def coordinates_text
    coordinates_object.as_text
  end
end
```

6.2.3 Prezentowanie danych

Framework Ruby on Rails posiada biblioteki do integracji obu narzędzi do prezentacji danych geograficznych wspomnianych w rozdziale 5.3:

1. leaflet-rails - dołącza do projektu Ruby on Rails bibliotekę leafletjs napisaną w języku javascript wraz ze stylami oraz dodaje metody do widoków pozwalające na dołączenie mapy i podstawową konfigurację bez potrzeby używania javascriptu
2. Google Maps for Rails - biblioteka opakowuje javascriptową bibliotekę google maps, ułatwiając jej dodanie do projektu

Dzięki takiej integracji zgodnie z konwencją w języku Ruby[3] wspomniane biblioteki są przechowywane centralnie w systemie zamiast w projekcie. Hanami w przeciwieństwie do Ruby on Rails nie posiada bibliotek wspomagających dołączenie do projektu biblioteki leafletjs lub Google Maps. Żeby skorzystać jednej z wymienionych bibliotek należy dołączyć jej kod do projektu.

Przy odczycie danych w formacie JSON, Ruby on Rails automatycznie serializuje model do postaci JSON. W projekcie wykorzystującym Hanami, niezbędne jest dodanie zewnętrznej biblioteki realizującej serializowanie danych np. Roar i zdefiniowanie odpowiedniego reprezentera. Przykładowy reprezenter został zaprezentowany na listingu 6.3

Fragment kodu 6.3 Reprezenter dla modelu Point

```
require 'roar/decorator'
require 'roar/json'

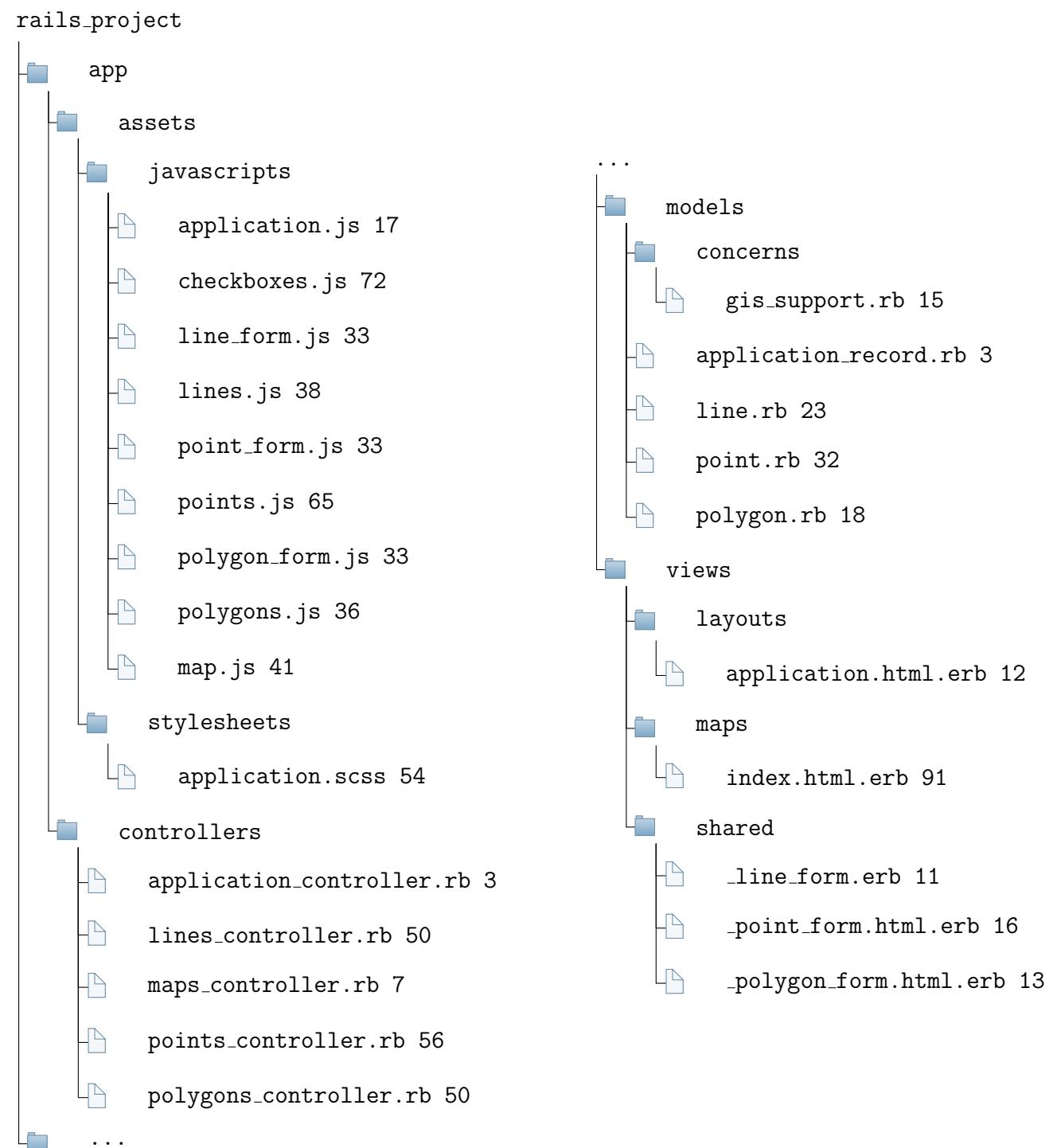
module Web::Representers
  class Point < Roar::Decorator
    include Roar::JSON

    property :id
    property :name
    property :object_type
    property :object_class
    property :voivodeship
    property :county
    property :commune
    property :terc
    property :coordinates_text
    property :color
  end
end
```

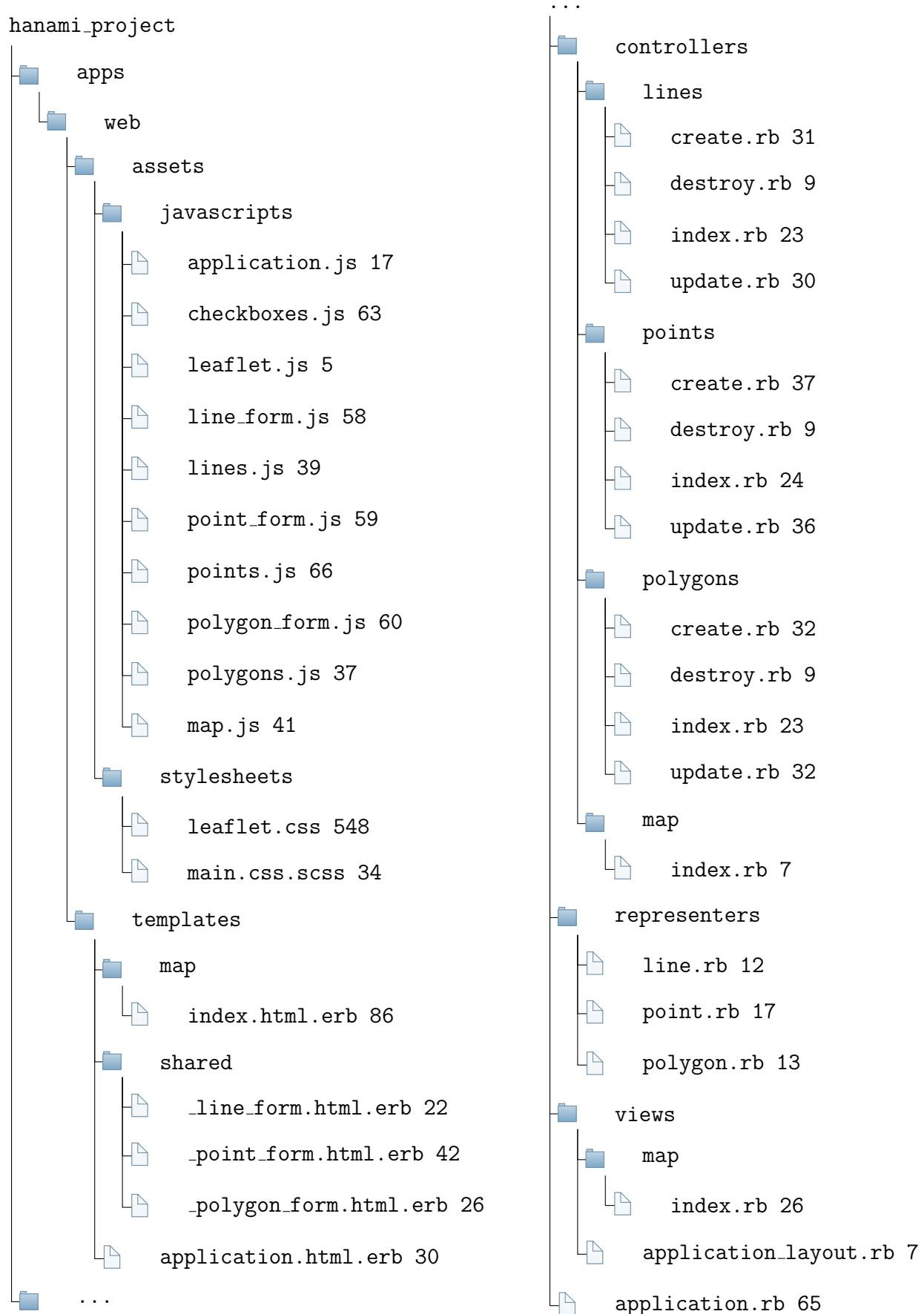
6.2.4 Podsumowanie

Ruby on Rails posiada zdecydowanie więcej gotowych narzędzi do pracy z danymi geograficznymi. Tworzenie aplikacji dzięki temu trwa mniej czasu. Implementacja systemu informacji geograficznej za pomocą frameworku Hanami wymaga od programisty znacznie więcej pracy i większych umiejętności programistycznych.

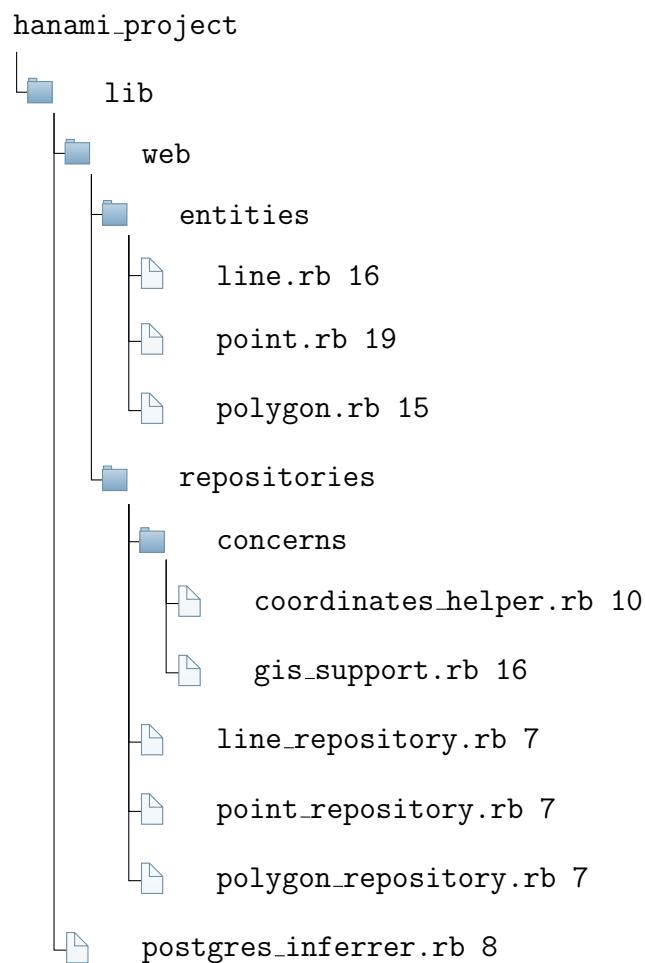
6.3 Porównianie struktur wykonanych projektów



Rysunek 6.4 Struktura projektu Rails



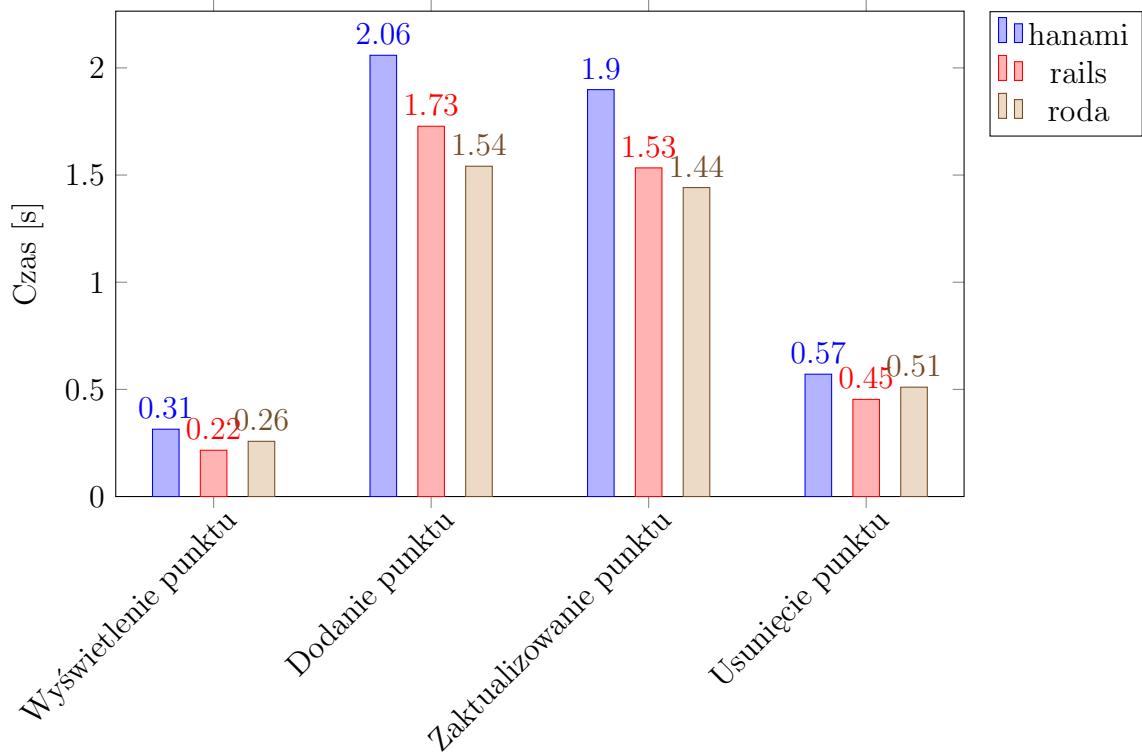
Rysunek 6.5 Struktura projektu Rails



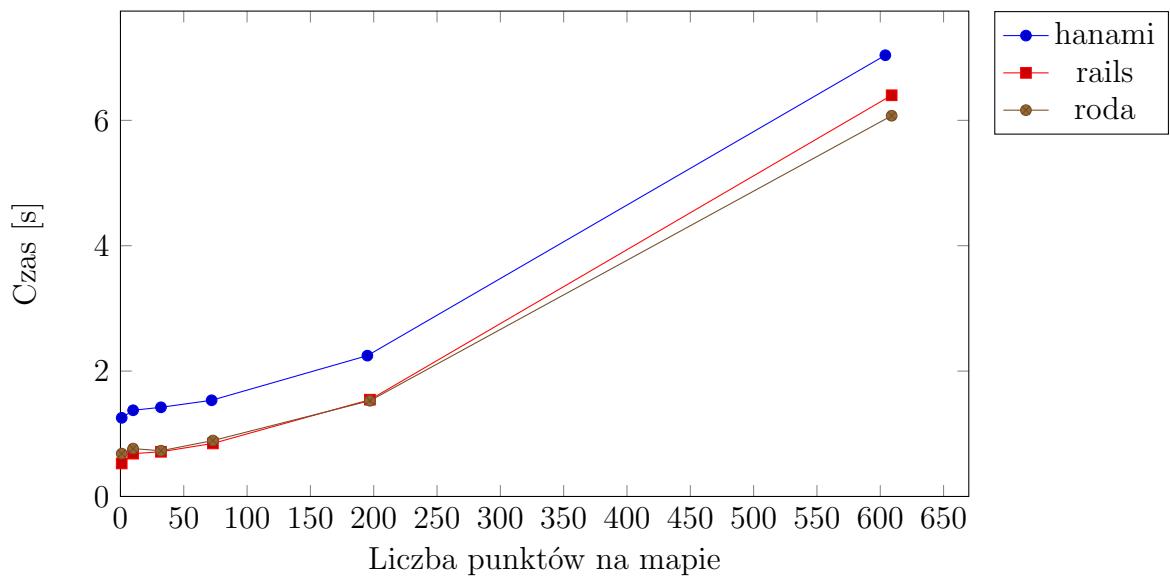
Rysunek 6.6 Struktura projektu Rails

6.4 Porównanie wydajności zaimplementowanych aplikacji

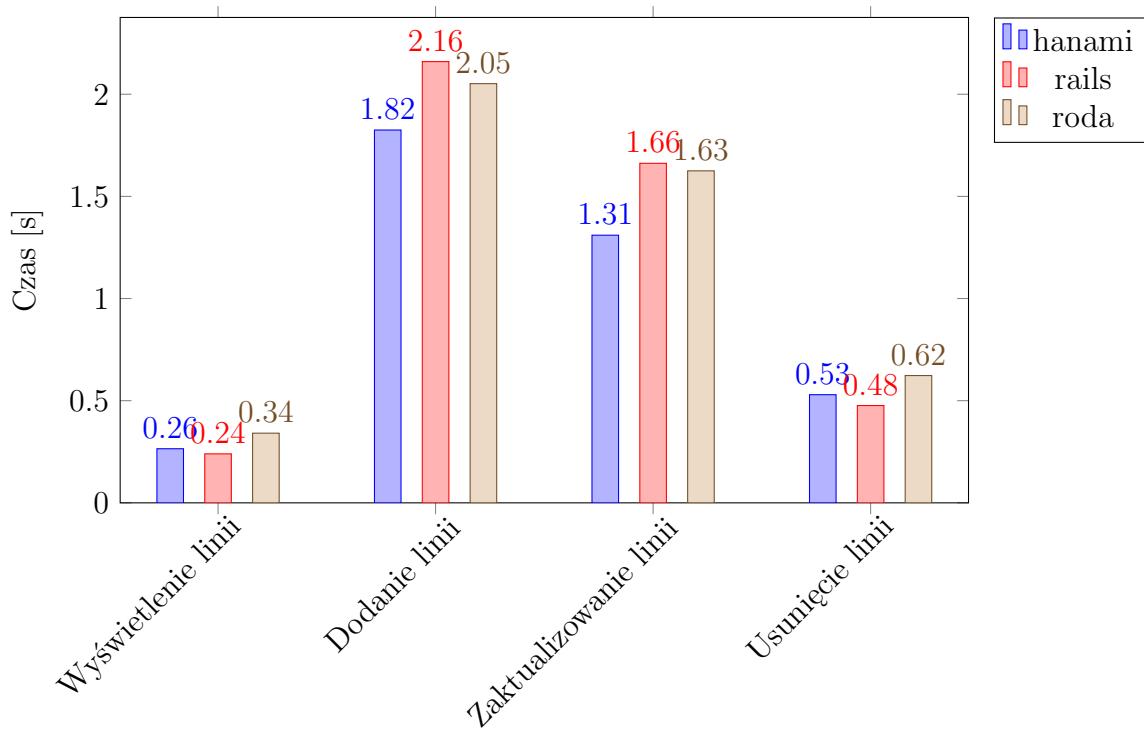
6.4.1 Interfejs użytkownika



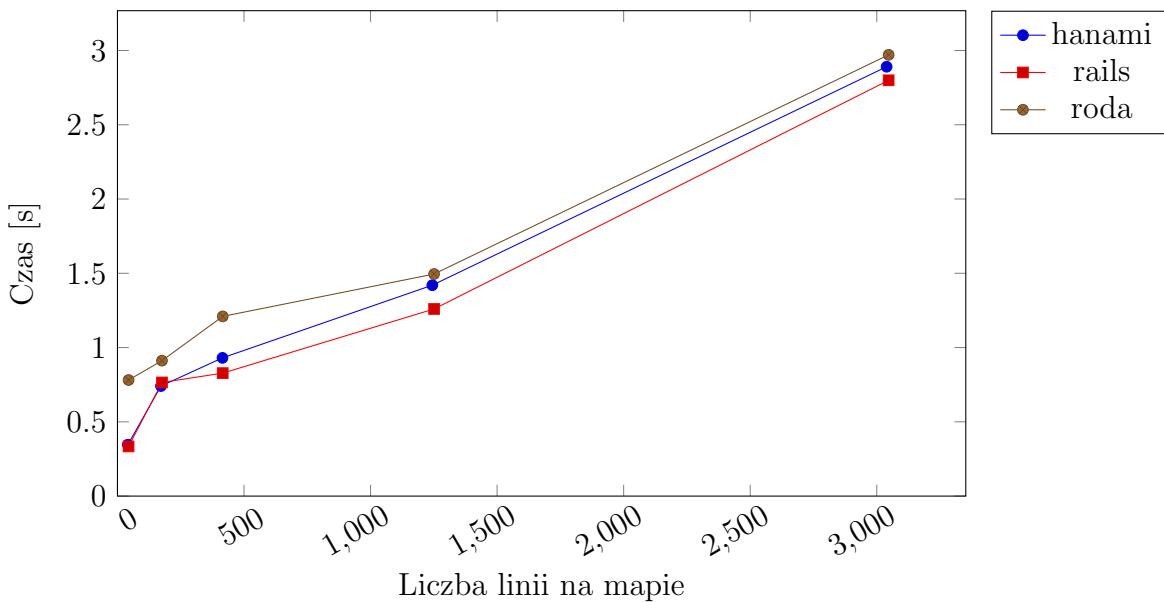
Rysunek 6.7 Czas wykonania operacji CRUD dla punktu



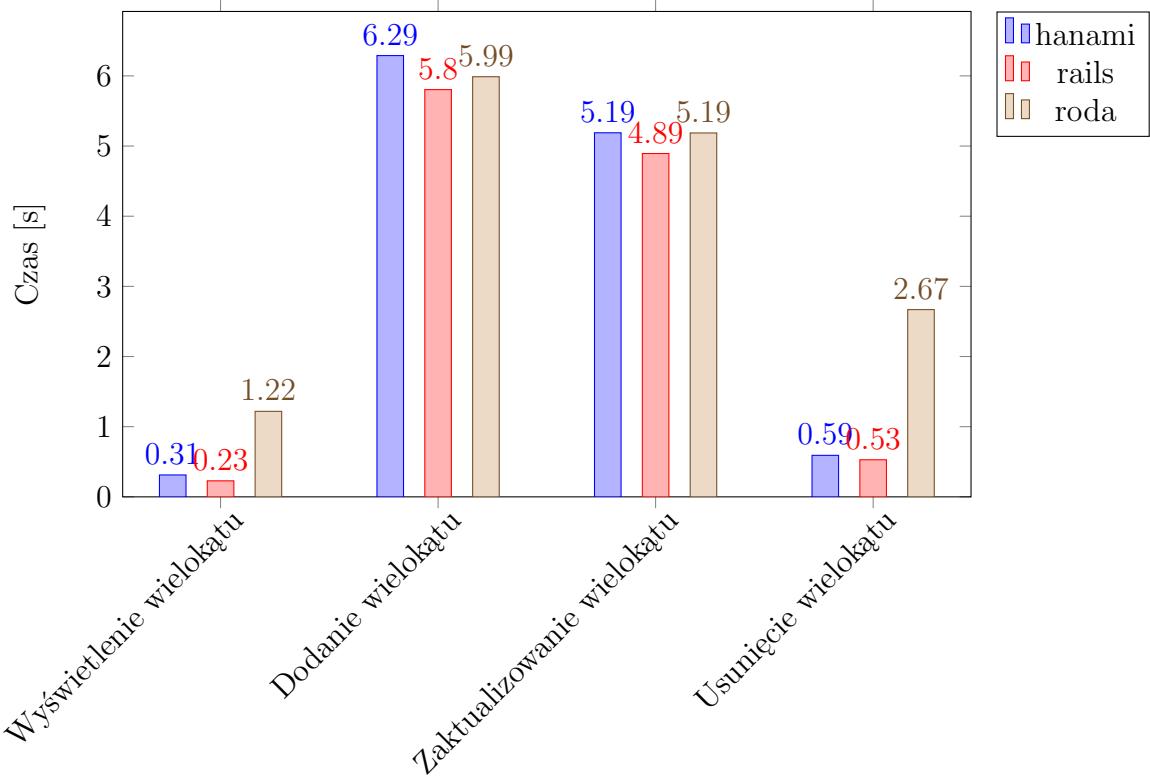
Rysunek 6.8 Czas ładowania widoku w zależności od liczby punktów



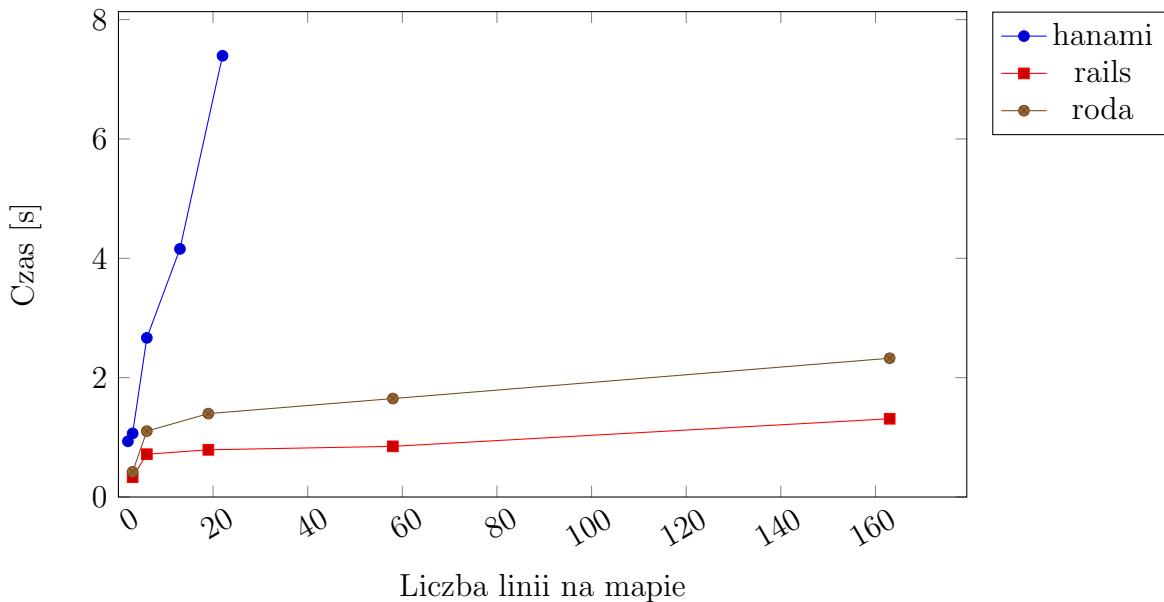
Rysunek 6.9 Czas wykonania operacji CRUD dla linii



Rysunek 6.10 Czas ładowania widoku w zależności od liczby linii

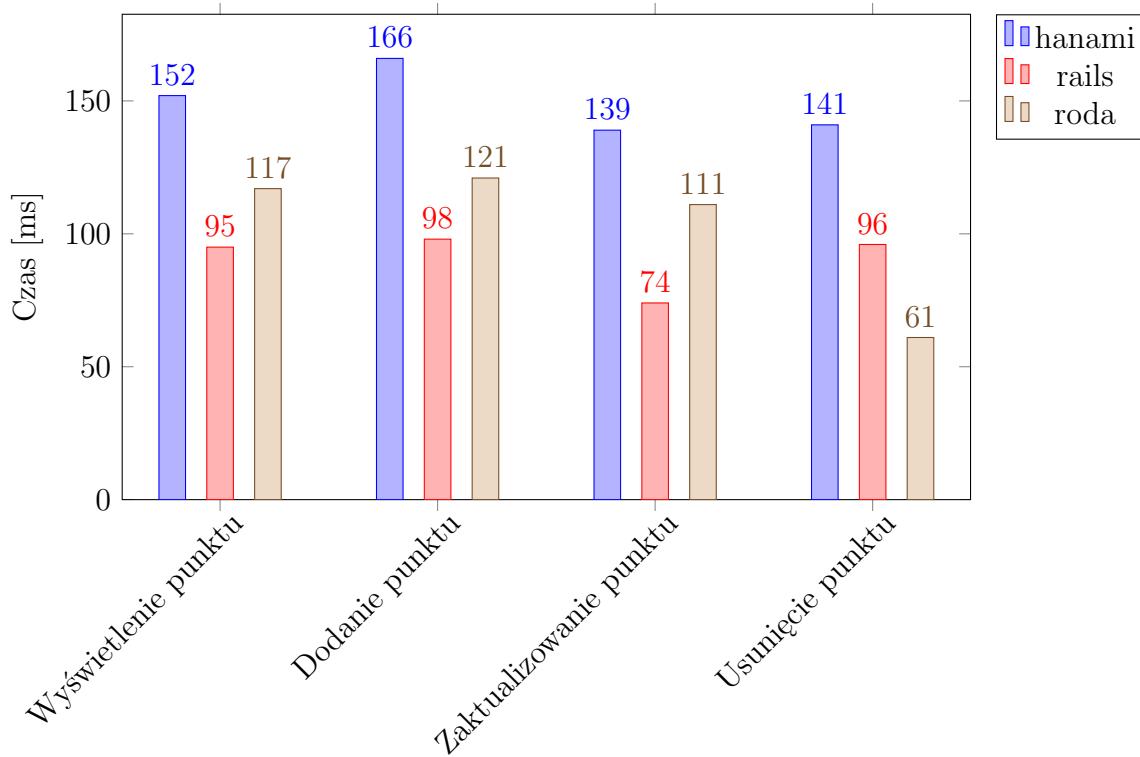


Rysunek 6.11 Czas wykonania operacji CRUD dla wielokątów

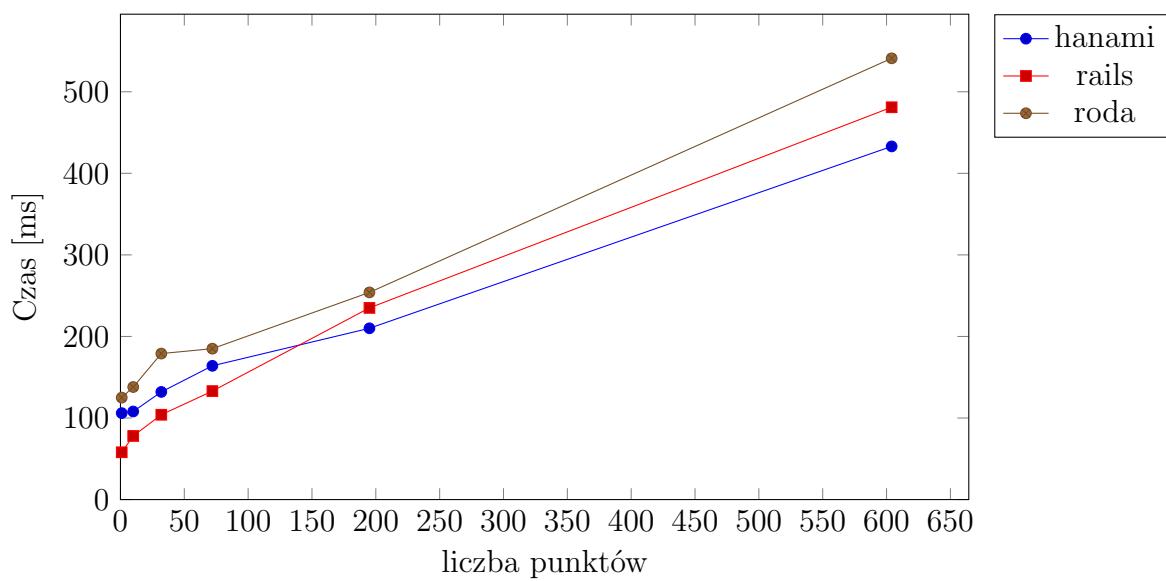


Rysunek 6.12 Czas ładowania widoku w zależności od liczby wielokątów

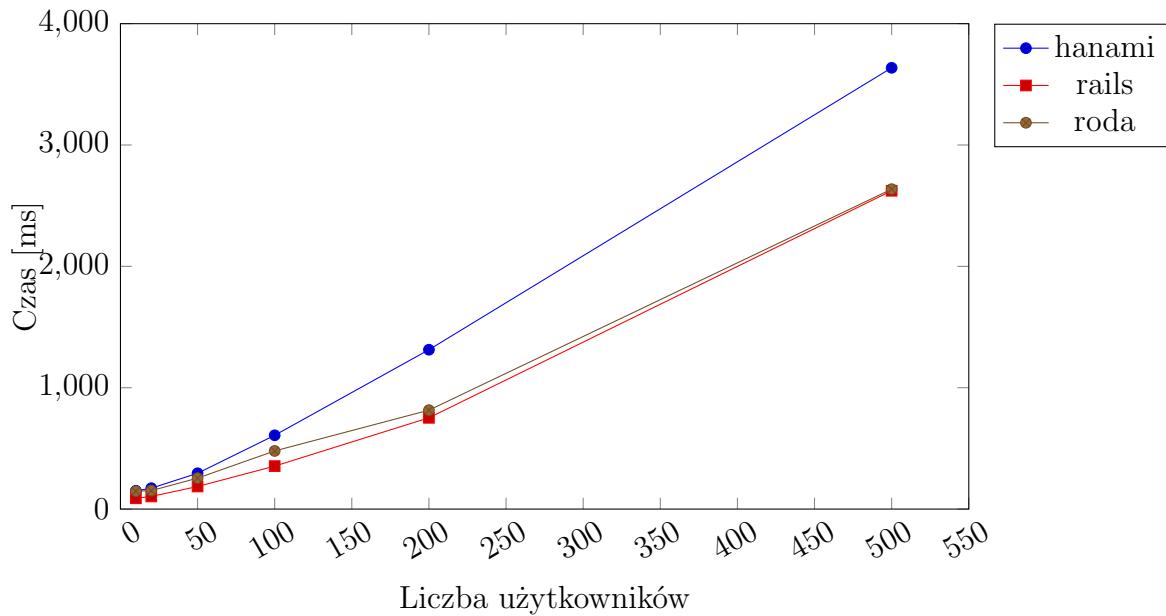
6.4.2 Zapytania HTTP



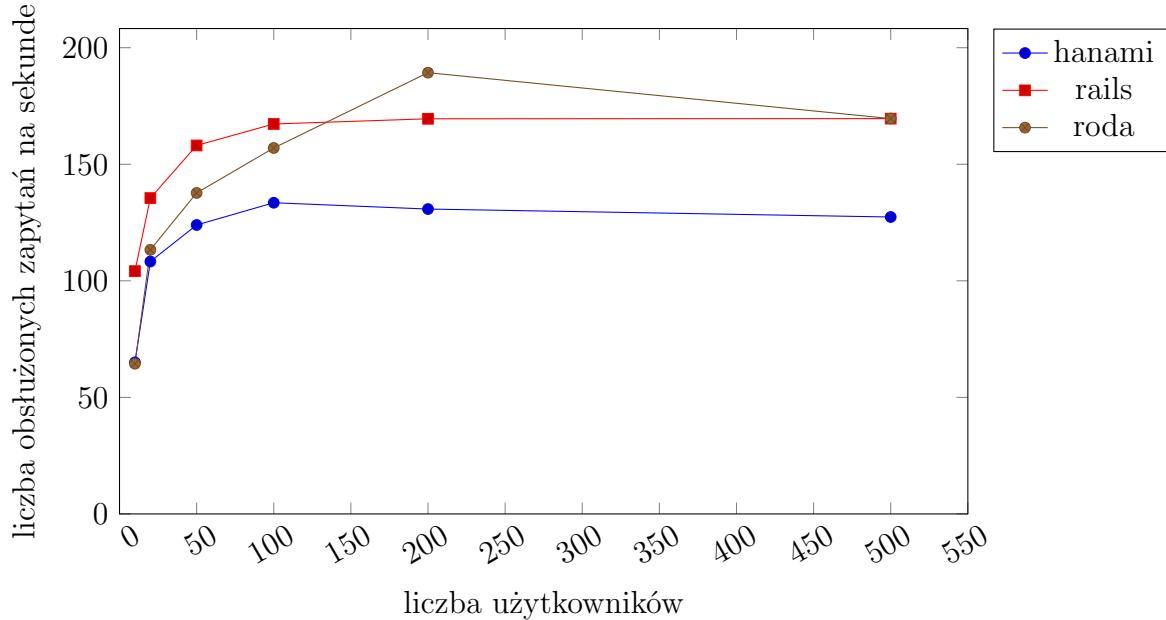
Rysunek 6.13 Czas wykonania zapytania HTTP dla punktu



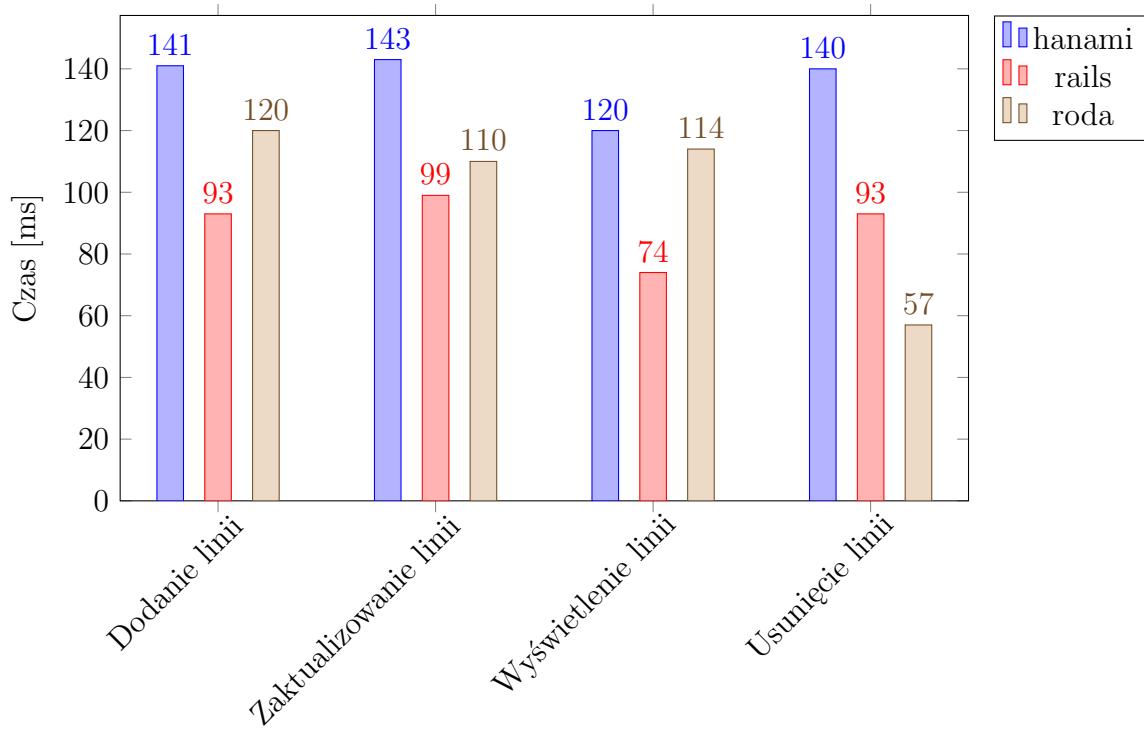
Rysunek 6.14 Czas ładowania danych w zależności od liczby punktów



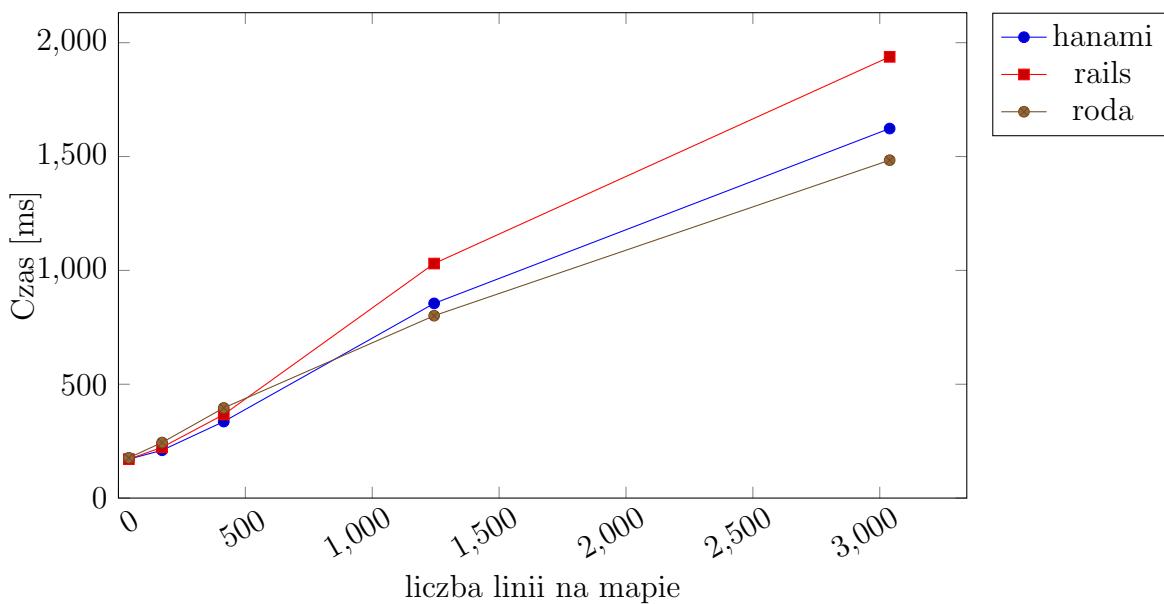
Rysunek 6.15 Czas ładowania danych w zależności od liczby użytkowników



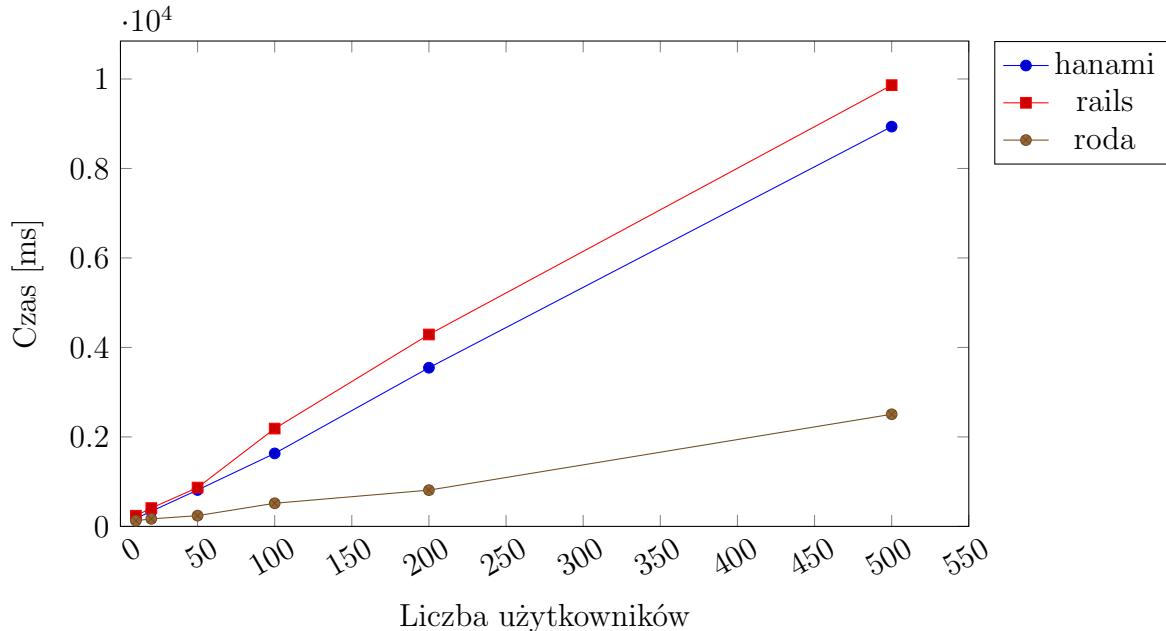
Rysunek 6.16 liczba obsłużonych zapytań na sekundę w zależności od liczby użytkowników



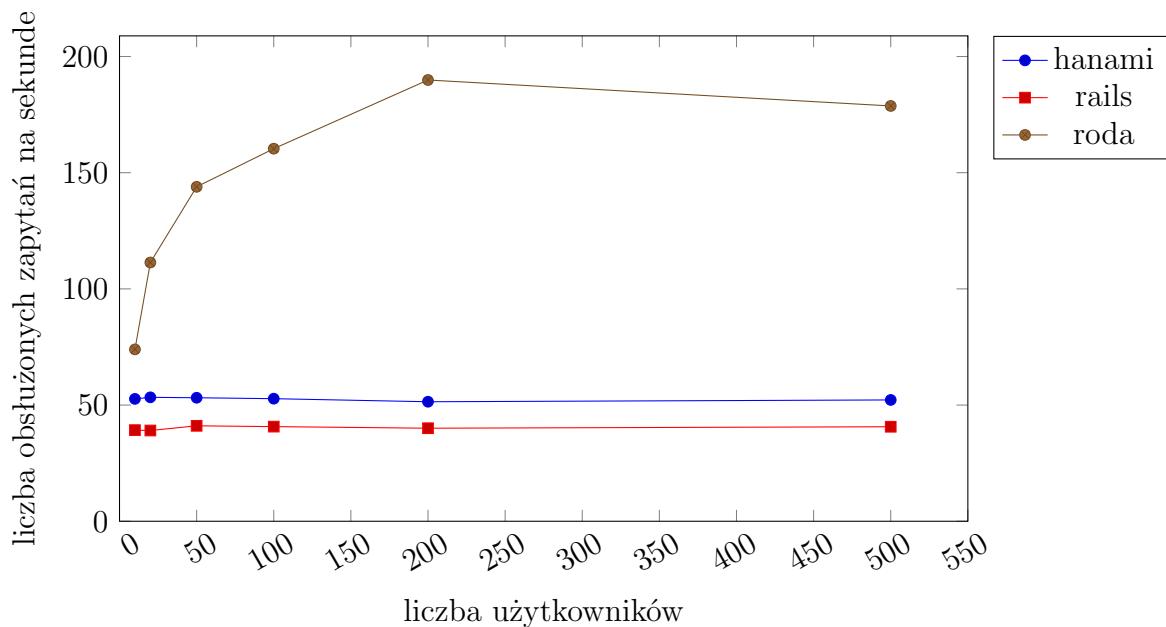
Rysunek 6.17 Czas wykonania operacji CRUD dla linii



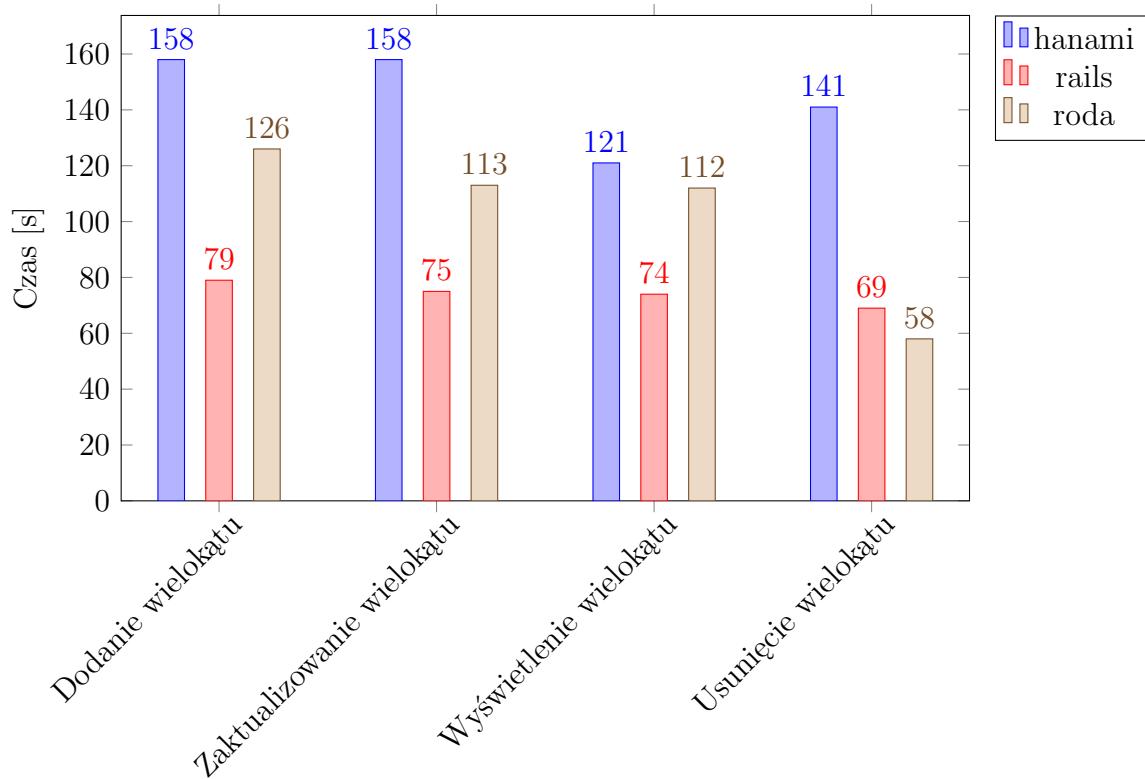
Rysunek 6.18 Czas ładowania danych w zależności od liczby linii



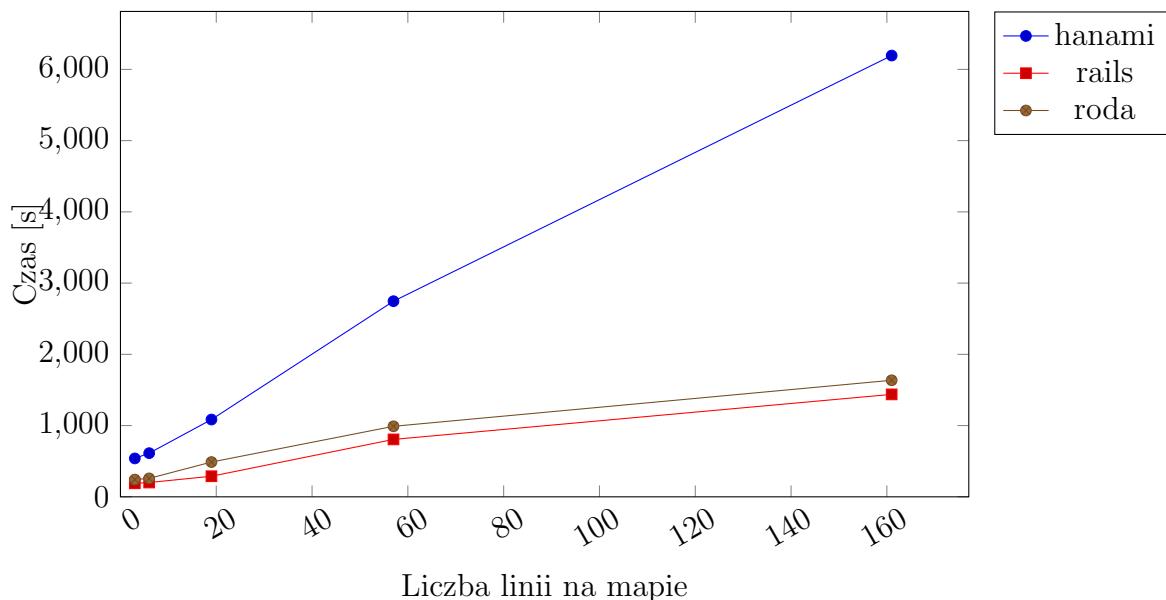
Rysunek 6.19 Czas ładowania danych w zależności od liczby użytkowników



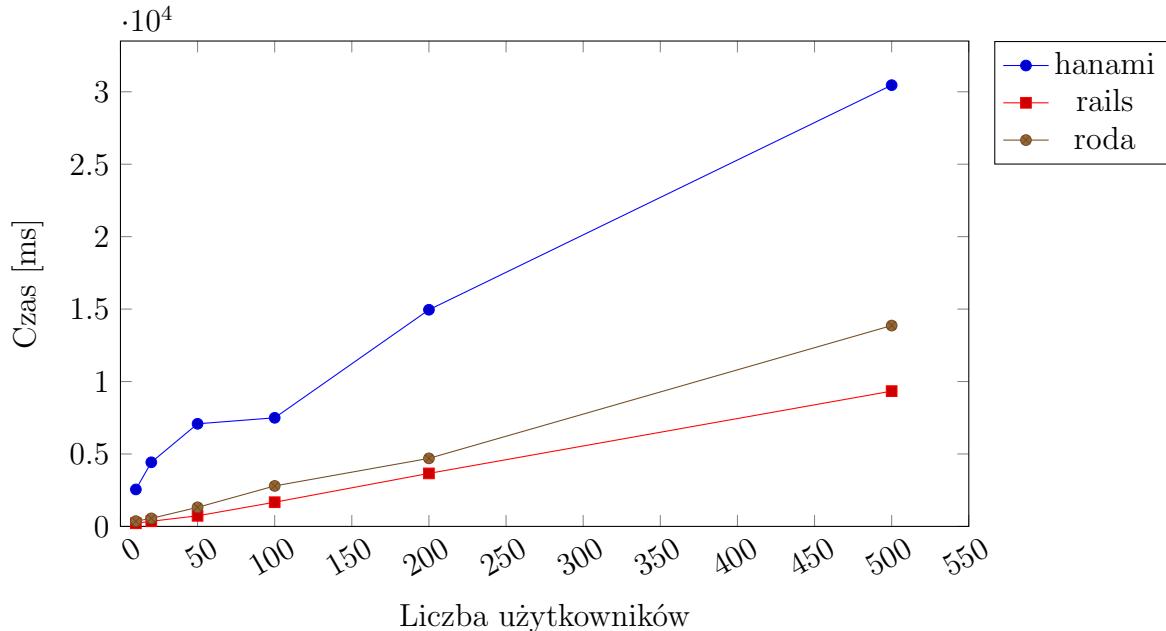
Rysunek 6.20 liczba obsłużonych zapytań na sekundę w zależności od liczby użytkowników



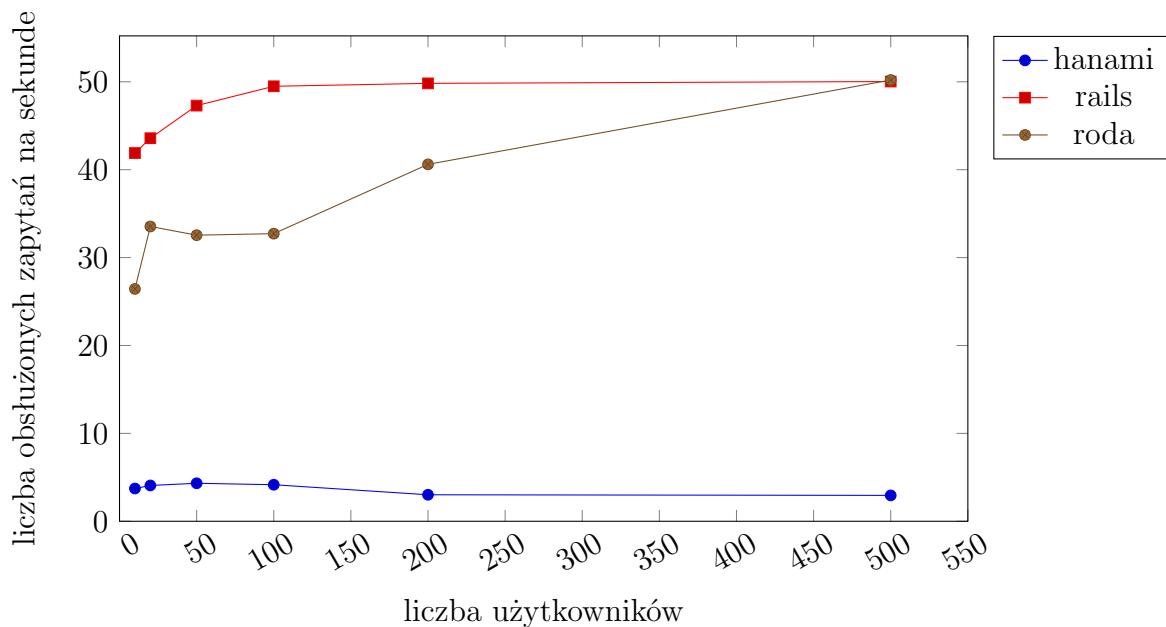
Rysunek 6.21 Czas wykonania operacji CRUD dla wielokątów



Rysunek 6.22 Czas ładowania danych w zależności od liczby wielokątów

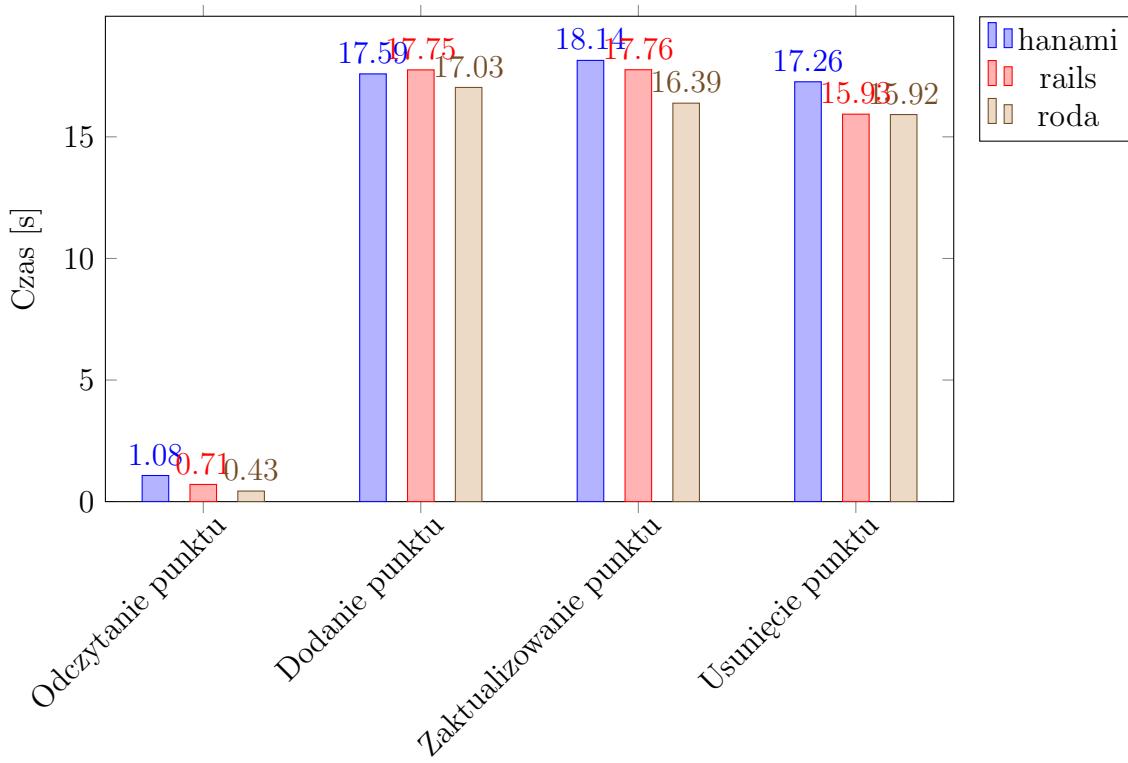


Rysunek 6.23 Czas ładowania danych w zależności od liczby użytkowników

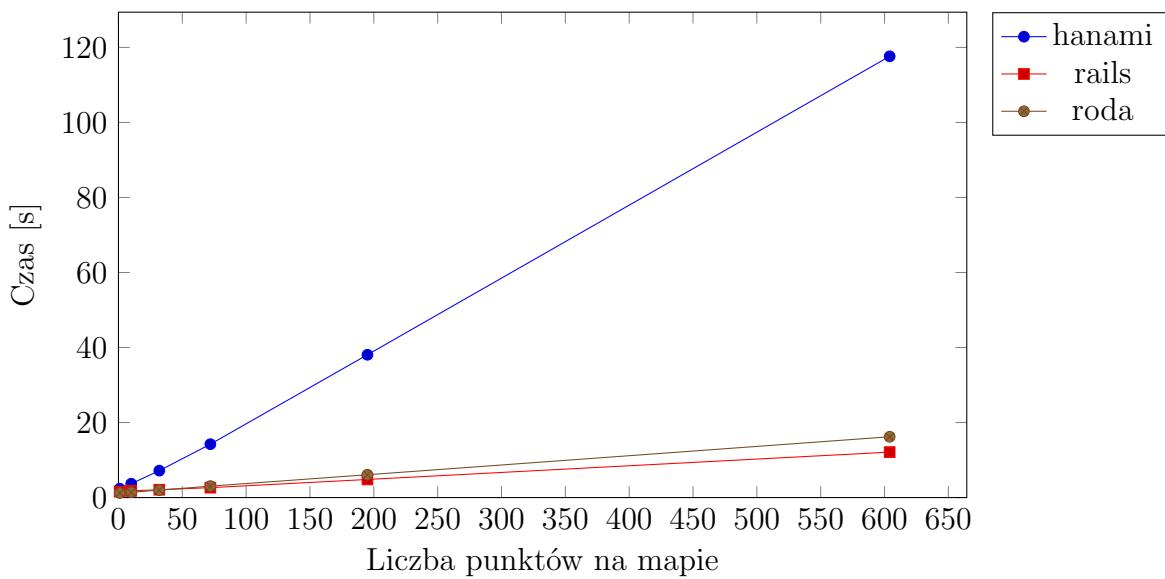


Rysunek 6.24 liczba obsłużonych zapytań na sekundę w zależności od liczby użytkowników

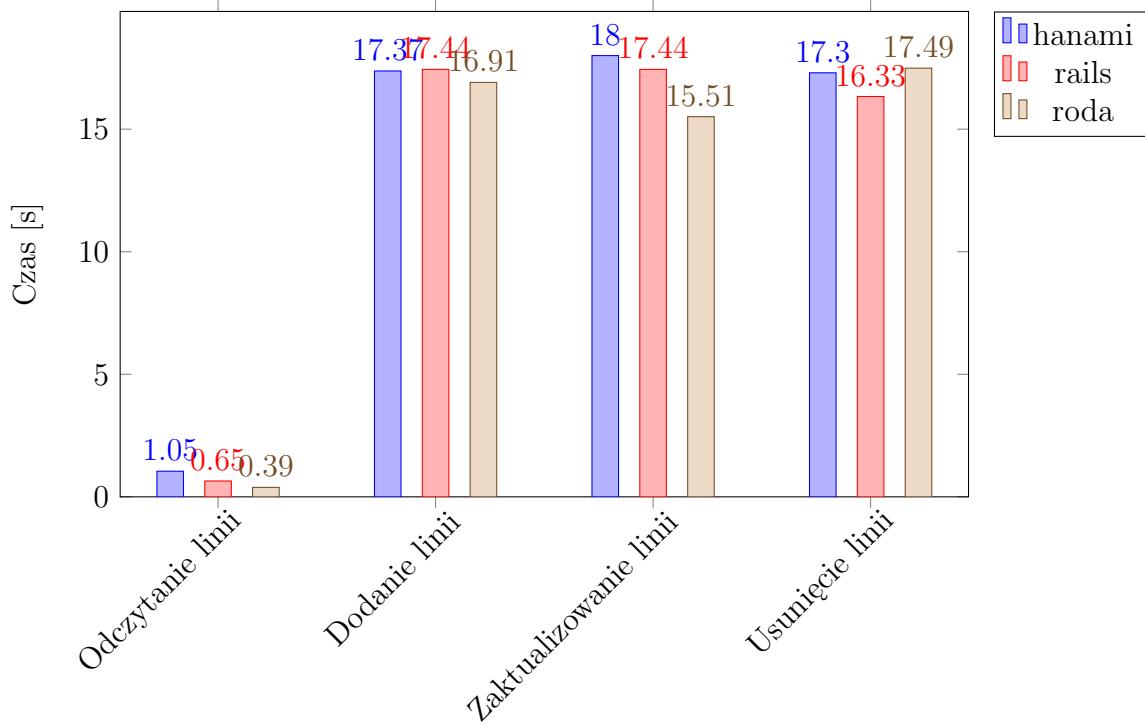
6.4.3 Komunikacja z bazą danych



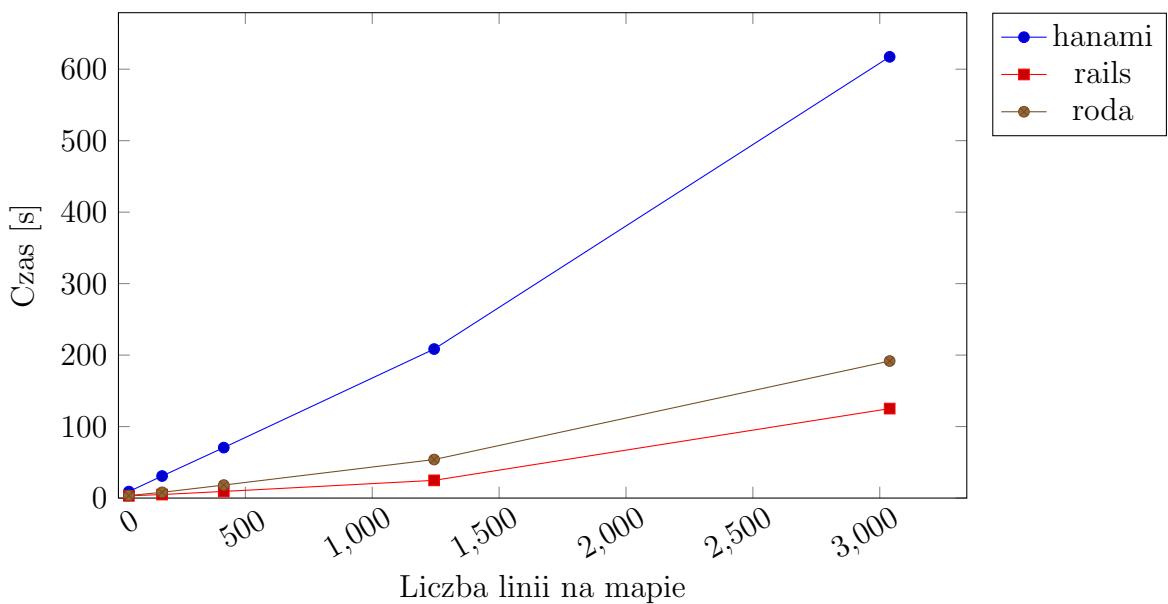
Rysunek 6.25 Czas wykonania operacji CRUD dla punktu



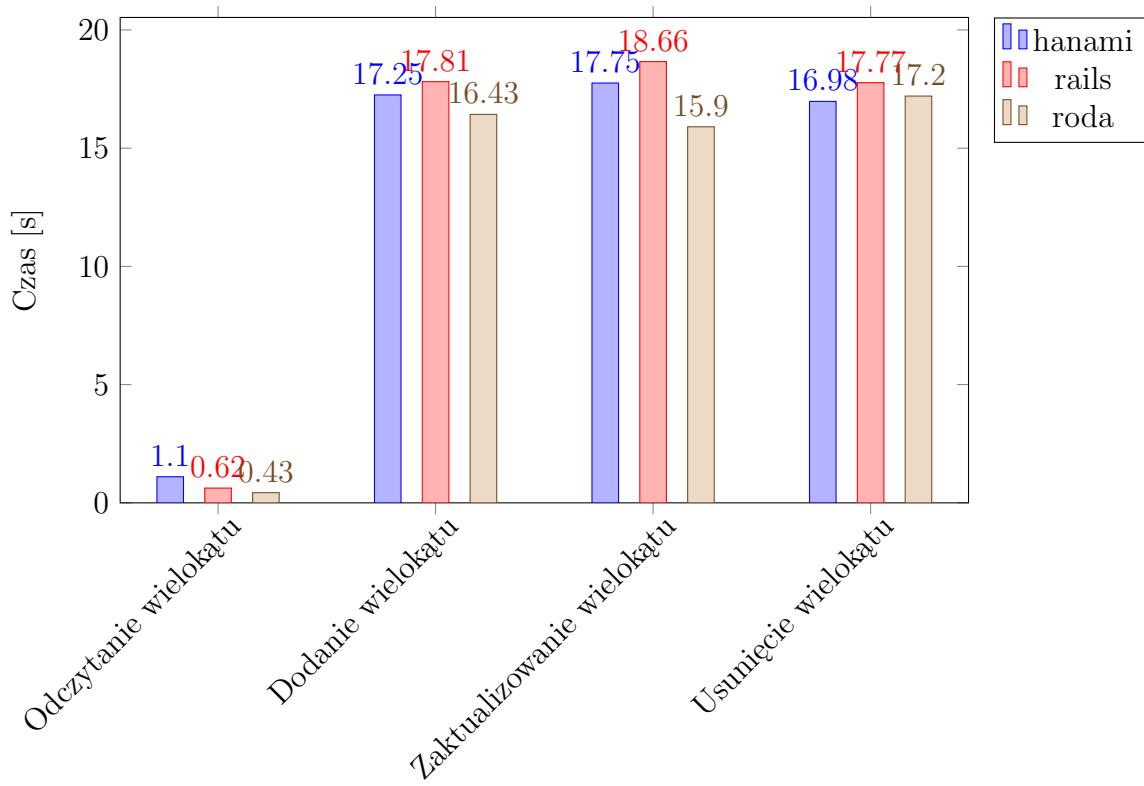
Rysunek 6.26 Czas wczytywania danych z bazy w zależności od liczby punktów



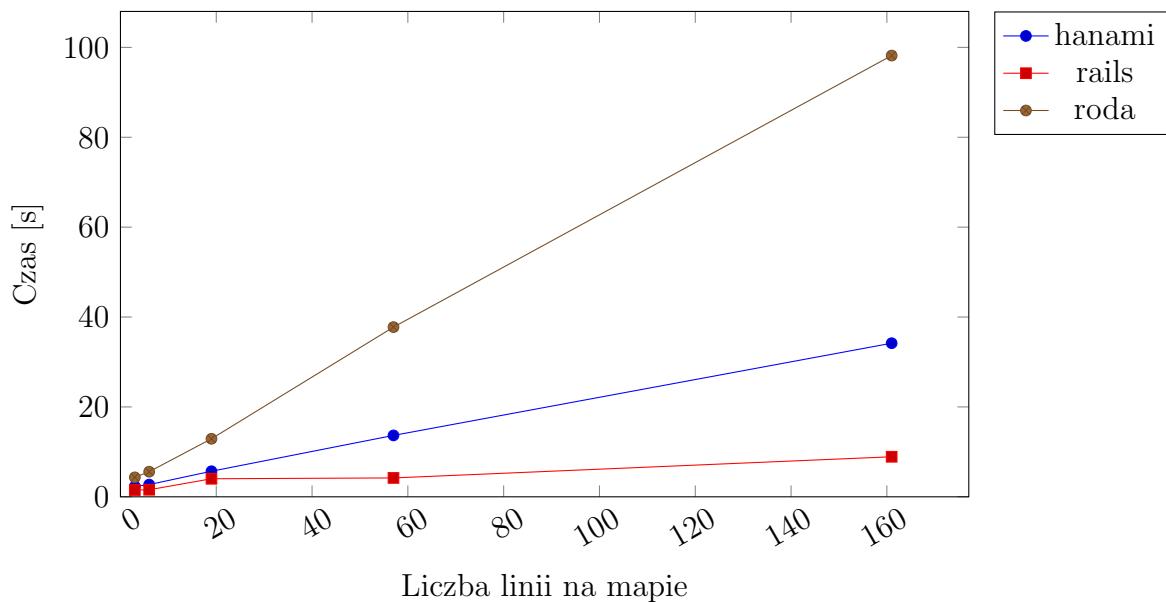
Rysunek 6.27 Czas wykonania operacji CRUD dla linii



Rysunek 6.28 Czas wczytywania danych z bazy w zależności od liczby linii



Rysunek 6.29 Czas wykonania operacji CRUD dla wielokątów



Rysunek 6.30 Czas wczytywania danych z bazy w zależności od liczby wielokątów

Rozdział 7

Podsumowanie

7.1 Analiza wyników badań

który framework jest najbardziej wydajny w rozpatrywanych przypadkach

7.2 Realizacja celu projektu

czy cel projektu został zrealizowany, napotkane trudności/przeszkody

Bibliografia

- [1] *Dokumentacja biblioteki Rgeo*, dostępna pod adresem:
<http://www.rubydoc.info/gems/rgeo/>, aktualne na dzień 08.03.2017r.
- [2] *Dokumentacja Hanami*, dostępna pod adresem:
<http://hanamirb.org/guides/>, aktualne na dzień 08.03.2017r.
- [3] *Dokumentacja języka Ruby*, dostępna pod adresem:
<https://www.ruby-lang.org/pl/documentation/>, aktualne na dzień 08.03.2017r.
- [4] *Dokumentacja MangoMap*, dostępna pod adresem:
<http://help.mangomap.com/>, aktualne na dzień 22.04.2017r.
- [5] *Dokumentacja OpenStreetMap*, dostępna pod adresem:
<http://wiki.openstreetmap.org/>, aktualne na dzień 08.03.2017r.
- [6] *Dokumentacja PostgreSQL*, dostępna pod adresem:
<https://www.postgresql.org/docs/9.6/static/index.html>, aktualne na dzień 09.06.2017r.
- [7] *Dokumentacja PostGIS*, dostępna pod adresem:
<http://postgis.net/documentation/>, aktualne na dzień 08.03.2017r.
- [8] *Dokumentacja Ruby on Rails*, dostępna pod adresem:
<http://guides.rubyonrails.org/>, aktualne na dzień 08.03.2017r.
- [9] *Dokumentacja Roda*, dostępna pod adresem:
<http://roda.jeremyevans.net/documentation.html>, aktualne na dzień 01.06.2017r.
- [10] *Dokumentacja SpatiaLite*, dostępna pod adresem:
<https://www.gaia-gis.it/fossil/libspatialite/index>, aktualne na dzień 09.06.2017r.
- [11] Huisman Otto, By (de) Rolf A., *Principles of Geographic Information Systems*, ITC, 2009
- [12] Martin Robert, *The Clean Architecture*, dostępna pod adresem:
<https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html>, aktualne na dzień 20.04.2017r.
- [13] Ruby Sam, Thomas Dave, Hansson Heinemeier David, *Agile Web Development with Rails 5*, Pragmatic Programmers, 2016

- [14] Schmandt Michael, *GIS Commons: An Introductory Textbook on Geographic Information Systems*, dostępne pod adresem:
<http://giscommons.org/>, aktualne na dzień 07.04.2017r.
- [15] Smyrdek Przemysław, *Czym jest framework i po co go używać*, dostępne pod adresem:
<http://poznajprogramowanie.pl/czym-jest-framework-i-po-co-go-uzywac/>, aktualne na dzień 20.04.2017r.