



POLITECHNIKA KRAKOWSKA im. T. Kościuszki
Wydział Mechaniczny
Instytut Informatyki



Kierunek studiów: Informatyka

Specjalność : Informatyka przemysłowa

STUDIA STACJONARNE

PRACA DYPLOMOWA

INŻYNIERSKA

Oskar Kapusta

**MOBILNY SYSTEM POMIARU CZASU
W ZAWODACH NARCIARSKICH**

**MOBILE TIMING SYSTEM FOR
SKIING COMPETITIONS**

Promotor:
prof. dr hab. inż. Leszek Wojnar

Kraków, rok akademicki 2012/2013

Autor pracy: Oskar Kapusta

Nr pracy:

OŚWIADCZENIE O SAMODZIELNYM WYKONANIU PRACY DYPLOMOWEJ

Oświadczam, że przedkładana przeze mnie praca dyplomowa inżynierska została napisana przeze mnie samodzielnie. Jednocześnie oświadczam, że ww. praca:

- 1) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem w sposób niedozwolony,
- 2) nie była wcześniej podstawą żadnej innej procedury związanej z nadawaniem tytułów zawodowych, stopni lub tytułów naukowych.

Jednocześnie przyjmuję do wiadomości, że w przypadku stwierdzenia popełnienia przeze mnie czynu polegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzej pracy, lub ustalenia naukowego, właściwy organ stwierdzi nieważność postępowania w sprawie nadania mi tytułu zawodowego (art. 193 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym, Dz.U. z 2012 r. poz. 572).

.....
data i podpis

Uzgodniona ocena pracy:

.....
podpis promotora

.....
podpis recenzenta

.....
podpis dyrektora instytutu
ds. dydaktyki

Spis treści

1	Cel i zakres pracy	5
2	Wstęp	5
3	Realizacja tematu	6
3.1	Architektura	6
3.2	Wybrane technologie	6
3.2.1	Memcached	6
3.2.2	Dependency Injection	6
3.2.3	Sprockets	7
3.2.4	Twitter Bootstrap	9
3.2.5	HAML	9
3.2.6	Backbone.js	9
3.3	Komunikacja	10
3.4	Podstawy działania	11
3.5	Implementacja	12
3.5.1	Aplikacja	12
3.5.2	Worker	21
3.6	Obudowa ochronna	21
4	Wnioski	21
5	Licencje Gemów	22
5.1	Wykaz gemów	22
5.2	Treści licencji	24
5.2.1	MIT	24
5.2.2	LGPLv3	25
5.2.3	Ruby	28
5.2.4	BSD-3	29
6	Summary	29
7	Dodatek A - Capistrano	29
8	Dodatek B - God	29
9	Dodatek C - YAGI	29

1 Cel i zakres pracy

Przedmiotem niniejszej pracy jest budowa mobilnego systemu pomiaru czasu dla zawodników narciarskich. System składa się z dwóch bramek: startowej oraz końcowej, które wykorzystują wiązkę laserową w celu uchwycenia dokładnego momentu przejechania zawodnika przez bramkę.

Niniejsza praca będzie się składać z trzech głównych części. Pierwsza z nich poświęcona zostanie architekturze systemu i wykorzystanych technologiach, a w szczególności, komunikacji pomiędzy komponentami oraz podstawą teoretycznym działania. Druga część tej pracy zawiera opis implementacji, zaś trzecia część jest poświęcona umieszczeniu systemu w obudowie ochronnej.

2 Wstęp

Obecnie na rynku istnieją podobne systemy do tego, którego budowę ta praca przedstawia, jednak często kosztują tysiące złotych. Praca ta jest próbą stworzenia rozwiązania spełniającego podobne zadanie do ww. systemów korzystając z ogólnie dostępnych podzespołów za nie wielkie pieniądze.

Jako szkielet systemu zostało wybrane Raspberry Pi - platforma komputerowa stworzona przez Raspberry Foundation. W momencie premiery (29 lutego 2012) model B użyty w tej pracy miał cenę początkową US\$ 35. Raspberry Pi oparte jest o chip BCM2835 zawierający procesor ARMv6. Urządzenie działa pod kontrolą dystrybucji systemu Linux Raspbian będącą portem Debiana Wheezy koniecznym z powodu braku kompatybilności (oficjalne wydanie Debiana Wheezy na platformę armhf działa jedynie z procesorami ARMv7 lub nowszymi).

Obie aplikacje (startowa i końcowa) zostały napisane przy użyciu języka Ruby 2.1.0 oraz dla aplikacji startowej stworzony został interfejs web umożliwiający wprowadzanie zawodników oraz podgląd wyników, jak również import oraz eksport. Napisany został on przy użyciu CoffeeScript oraz biblioteki JavaScript Backbone.js. CoffeeScript jest językiem inspirowanym elegancką składnią Ruby i Pythona, który kompiluje się do JavaScriptu. Backbone.js natomiast zapewnia strukturę aplikacji.

W pracy zostały użyte różne *gemy* - programy i biblioteki menedżera paczek *RubyGems*, których lista w raz z licencjami zostanie przedstawiona na końcu tej pracy.

Kompletny kod źródłowy pracy można znaleźć pod adresami:

<https://github.com/okapusta/skirace>

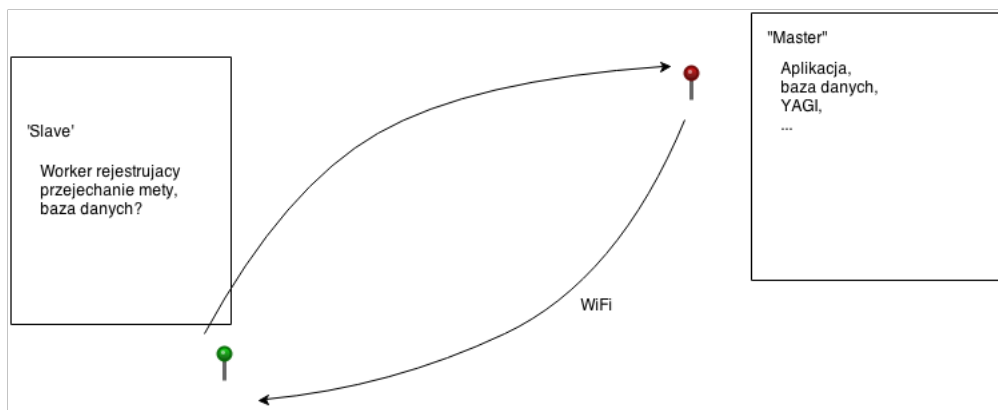
<https://github.com/okapusta/skirace-worker>

3 Realizacja tematu

3.1 Architektura

System którego ta praca dotyczy został zbudowany w myśl modelu master/slave gdzie mastertem jest aplikacja początkowa. To tam znajduje się interface, baza danych oraz serwer Memcached. W momencie uruchomienia aplikacji, oprócz startu serwera serwującego aplikację WEB tworzony jest nowy wątek zawierający event loop, który rejestruje przecięcie wiązki lasera, a kiedy to się stanie ustawia godzinę tego zdarzenia w cache.

Zadaniem slavea - workera - jest jedynie pobranie czasu startu z Memcached, obliczenie czasu końcowego oraz wypisanie go na ekranie LCD linii końcowej w momencie finishu.



Rysunek 1: Architektura systemu

Komunikacja pomiędzy dwiema bramkami odbywa się poprzez WiFi.

3.2 Wybrane technologie

3.2.1 Memcached

Memcached jest to rozproszony system buforowania pamięci podręcznej oryginalnie zaprojektowany na potrzeby serwisu LiveJournal. Pozwala on na przechowywanie obiektów w pamięci RAM przy pomocy kluczy (key-value store). W aplikacji wykorzystany został w celu przechowania czasów startu i finishu. Serwer Memcached jest uruchomiony tylko na jednej (startowej) bramce, worker natomiast łączy się z cache przy pomocy klienta i może pobierać lub pisać dane do pamięci drugiego urządzenia.

3.2.2 Dependency Injection

W niniejszej pracy został wykorzystany wzorzec projektowy *Dependency Injection (DI)* polegający na usuwaniu bezpośrednich zależności klas na rzecz *wstrzykiwania* ich w czasie konstruowania obiektu. Do osiągnięcia tego celu i uproszczenia DI użyty został gem *Dependor*, udostępniający zestaw metod i modułów przeznaczony do tego celu.

Poniżej zamieszczone są listingi przedstawiające normlne wstrzykiwanie zależności w Ruby oraz z wykorzystaniem gemu Dependor.

(a) Ruby

```
1 class A
2   attr_reader :obj
3
4   def initialize(obj)
5     @obj = obj
6   end
7
8   def do_b
9     obj.do_sth
10  end
11 end
12
13 a = A.new(B.new)
14 a.do_b
```

(b) Ruby + Dependor

```
class A
  takes :b

  def do_b
    b.do_sth
  end
end

a = A.new
a.do_b
```

3.2.3 Sprockets

Do kompilacji CoffeeScript oraz szablonów *.hamlc* (*Haml Coffee Assets*) został użyty gem *Sprockets* zawierający preprocessory dla języków takich jak CoffeeScript czy SCSS. Sprockets w środowisku developerskim pozwala na kompilację assetów (JavaScriptów i CSSów) 'w locie', natomiast w środowisku produkcyjnym assety są prekompilowane. Sprockets pozwala również na minifikację zasobów to jest zastąpienie nazw funkcji czy zmiennych pojedynczymi znakami w celu zmniejszenia rozmiaru kodu, który musi zostać pobrany przez przeglądarkę.

Sprockets działa w kontekście Rack - minimalnego interfejsu dla aplikacji Ruby do komunikacji z popularnymi serwerami WWW. Zasoby serwowane przez sprockets są montowane w pliku *config.ru*, będącym plikiem konfiguracyjnym dla interfejsu Rack poprzez który aplikacja jest uruchamiana. To tutaj tworzony jest wątek rejestrujący przejechanie linii startu oraz tutaj montowana jest ścieżka serwera WWW '/' tak aby pokazywała na aplikację Sinatra. Samo sprockets jest montowane w następujący sposób:

```
1 map Skirace::Application.assets_prefix do
2   run Skirace::Application.sprockets
3 end
```

Listing 1: config.ru

Wykorzystane tutaj zmienne klasowe (*assets_prefix*, *sprockets*) aplikacji są definiowane w pliku *application.rb* zawierającym klasę aplikacji Sinatra.

```
1 set :assets_prefix, '/assets'
2 set :assets_path, File.join(public_folder, assets_prefix)
3
4 set :sprockets, Sprockets::Environment.new(root)
```

Listing 2: Ustawienie zmiennych Sprockets

```
1 sprockets.append_path File.join(root, 'app', 'assets', 'javascripts')
2 sprockets.append_path File.join(root, 'app', 'assets', 'stylesheets')
3 sprockets.append_path File.dirname(HamlCoffeeAssets.helpers_path)
```

Listing 3: Przeszukiwane foldery

Powyższe listingi umieszczają skompilowane pliki z wyznaczonych ścieżek w folderze *public/assets* pod nazwami *application.js* oraz *application.css*. Pliki te zawierają jedynie asse-ty załączone poleceniem *require* w plikach *app/assets/javascripts/application.js* i *app/assets/stylesheets/applica*

```
1 //= require jquery-1.10.2.min
2 //= require jquery.ui.widget
3 //= require jquery.fileupload
4 //= require jquery.cookie
5 //= require bootstrap.min
6 //= require underscore-min
7 //= require backbone-min
8 //= require hamlcoffee
9 //= require app
10 //= require_tree ./lib
11 //= require_tree ./models
12 //= require_tree ./collections
13 //= require_tree ./templates
14 //= require_tree ./services
15 //= require_tree ./views
16 //= require_tree ./routers
17 Skirace.init();
```

Listing 4: *app/assets/javascripts/application.js*


```

1  /*
2    *= require bootstrap.min
3    *= require bootstrap-responsive.min
4    *= require app.css
5    *= require_tree .
6    */

```

Listing 5: app/assets/stylesheets/application.css

3.2.4 Twitter Bootstrap

Twitter Bootstrap jest frameworkiem front-end dostarczającym style CSS oraz kod JavaScript, który za zadanie ma przyspieszenie budowę front-endu aplikacji. Bootstrap dostarcza zestaw klas HTML, które posiadają określone style. W tej pracy wykorzystany został Twitter Bootstrap w wersji drugiej. Aktualna wersja Bootstrapa to 3.1.1.

3.2.5 HAML

W projekcie został użyty HAML (HTML Abstraction Markup Language), który sprawia że kod jest przyjemniejszy do pisania i czytania. W HAML używa się jedynie tagów otwierających a o tym jak osadzone są elementy decyduje indentacja. HAML posiada też skróty na sekcje div o podanym id lub klasie.

(c) HAML

(d) HTML

<pre> 1 .klasa 2 3 #id 4 %p 5 Paragraf </pre>	<pre> <div class="klasa"></div> <div id="id"> <p>Paragraf</p> </div> </pre>
---------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------

3.2.6 Backbone.js

Backbone.js jest lekką biblioteką JavaScript nadającą strukturę aplikacji JavaScript. Typowe użycie tej biblioteki jest to zazwyczaj trio pomiędzy samym backbonem a jQuery i underscore.js.

W tej pracy został też użyty gem haml-coffee-assets pozwalający na pisanie szablonów Backbone w języku HAML z osadzonym CoffeeScriptem (podobnie jak w eRuby).

Biblioteka Backbone.js stara się odtworzyć to co jest po stronie serwera w modelu MVC na stronę klienta (przeglądarki) i JavaScriptu. Model odzwierciedla zasób na serwerze i jest odpowiednikiem modelu po stronie serwera. Kolekcja jest grupą modeli pobieraną z serwera. Na kolekcji zdefiniowane są zdarzenia (eng. *events*), które ponownie renderują widok np. kiedy element zostanie dodany do kolekcji. W Backbone Router odpowiedzialny

jest za tłumaczenie adresów URL na widoki oraz za obsługę historii przeglądarki. Widok jest odpowiednikiem kontrolera znanego z Rails. Odpowiada na zdarzenia w oknie przeglądarki takie jak kliknięcie myszką i podejmuje odpowiednie akcje. Szablony natomiast są, w przypadku tej pracy, kodem HAML używanym przez widoki do renderowania treści aplikacji.

3.3 Komunikacja

Oba urządzenia komunikują się ze sobą przy pomocy WiFi w trybie pracy Ad Hoc. Jako karta sieciowa została wybrana karta USB TP-Link TL-WN722N ponieważ posiada antenę o zysku 4dBi, którą można odkręcić oraz zamontować mocniejszą antenę. Poniższy listing przedstawia konfigurację urządzenia do pracy w trybie Ad Hoc.

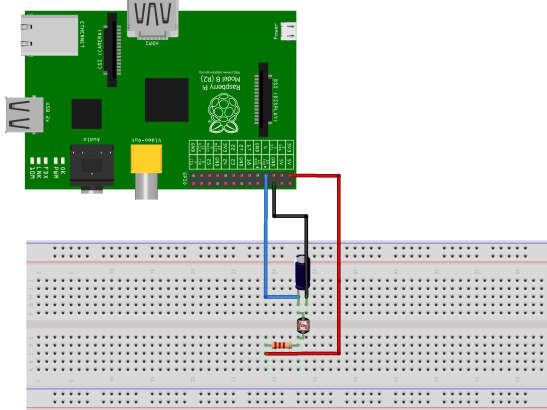
```
1 auto wlan0
2 iface wlan0 inet static
3     address 192.168.1.1
4     netmask 255.255.255.0
5     wireless-channel 1
6     wireless-essid SKIRACE
7     wireless-mode ad-hoc
```

Listing 6: `/etc/network/interfaces`

Plik `/etc/network/interfaces` drugiego urządzenia został skonfigurowany analogicznie. Różni się on jedynie adresem IP ustawionym na `192.168.1.2`. Urządzenia ustawione w trybie pracy Ad Hoc wykrywają się nawzajem i tworzą sieć o SSID SKIRACE.

3.4 Podstawy działania

Ten rozdział ma na celu przedstawienie budowy oraz podstaw teoretycznych działania czujnika rejestrującego przecięcie wiązki lasera. Składa się on z fotorezystora, rezystora oraz kondensatora. Poniższy rysunek przedstawia jego budowę.



Rysunek 2: Sensor

W układzie o którym mowa w tym rozdziale rezystor $2.2k\Omega$ działa jako zabezpieczenie przed zbyt dużym napięciem skierowanym na port GPIO. Podłączony jest do fotorezystora, którego rezystancja jest niska kiedy świeci na niego wiązka lasera. Kondensator $1\mu F$ jest ładowany a kiedy przekroczy wartość graniczną wynoszącą około 2V pin GPIO rejestruje wartość HIGH. Działanie to jest wykorzystane w kodzie aplikacji gdzie mierzony jest czas potrzebny na naładowanie kondensatora. Kiedy czas jest krótki znaczy to że na fotorezystor pada wiązka lasera, kiedy się zwiększy znaczy to że laser został przecięty tj. zawodnik przejechał przez start lub metę.

3.5 Implementacja

Niejszy rozdział zostanie poświęcony implementacji. Składa się on z dwóch podrozdziałów: Aplikacja, który opisuje budowę aplikacji startowej oraz rozdziału Worker opisującego worker rejestrujący przejechanie mety.

3.5.1 Aplikacja

Tak jak była o tym mowa we wstępie do ninejszej pracy aplikację startową można podzielić na dwa osobne komponenty. Jeden stanowi aplikacja napisana we frameworku Sinatra, która udostępnia API dla aplikacji front-end do komunikacji z bazą danych. Ona także serwuje skompilowane zasoby oraz dostarcza połączenia z Memcached. W momencie uruchomienia aplikacji jest też tworzony nowy wątek, którego zadaniem jest zarejestrowanie startu oraz ustawienie czasu tego zdarzenia w Memcached co przedstawia listing 7. Aplikacja front-end natomiast dostarcza interfejs dostępny przez przeglądarkę internetową służący to interakcji z back-endem.

```
1 require './application'
2
3 t = Thread.new(Injector.new(OpenStruct.new({}))) do |injector|
4   while true
5     injector.capacitor.discharge(injector.options.capacitor.pin)
6     sleep 0.01; reading = 0;
7
8     while injector.gpio.read(injector.options.capacitor.pin) == LOW
9       reading += 1
10    end
11
12    if reading > injector.options.activation_threshold
13      injector.caching_service.set('start_time', Time.now)
14    end
15  end
16 end
17 t.abort_on_exception = true
```

Listing 7: Utworzenie wątku w pliku config.ru

Wykorzystana tutaj klasa `Injector`, ma za zadanie budowanie obiektów używających dependency injection w raz z wszelkimi zależnościami przy pomocy gemu `Dependor`.

W swoim konstruktorsze oczekuje obiektów `Sinatra` dlatego tutaj inicjalizowana jest pustym obiektem `OpenStruct`. `OpenStruct` odpowiada na dowolne metody oraz zwraca `nil` jeśli nazwa metody wywołanej na rzecz tego obiektu nie zawiera się w haszu, który został użyty do inicjalizacji. Obiekt klasy `Injector` jest przekazywany w bloku, gdzie później dostarcza obiektów takich jak `capacitor`, którego klasa jest zdefiniowana w pliku `app/services/components/capacitor.rb`.

W klasie `Injector` są definiowane moduły które mają być przeszukane w poszukiwaniu klas, które mają być wstrzykiwane co przedstawia listing 8. Nazwy przyjmowane w argumentach metody `takes` oraz nazwy metod wywołanych na obiekcie `Injector` odpowiadają nazwą plików wstrzykiwanych klas. Warunkiem tego jest odpowiednie nazewnictwo klas i plików zgodnie z konwencjami przyjętymi w programowaniu Ruby.

W Ruby przyjęło się nazywanie nazw klas i modułów z dużej litery `CamelCase` a pliki zawierające te klasy powinny mieć taką samą nazwę jak klasa tylko zapisaną w `snake_case` z rozszerzeniem `*.rb`.

```
1 class Injector
2   include Dependor::AutoInject
3   include Dependor::Sinatra::Objects
4
5   look_in_modules ::Repositories,
6   ::Connections,
7   ::RaspberryPi,
8   ::Presenters,
9   ::Uploaders,
10  ::Parsers,
11  ::Components
12
13  def initialize(objects = nil)
14    sinatra_objects(objects)
15
16    io.wiringPiSetup
17  end
18 end
```

Listing 8: Klasa `injector` odpowiedzialna za tworzenie obiektów z wykorzystaniem DI

Powyższy listing przedstawia jedynie fragment klasy `Injector` ponieważ jest zbyt długa aby ją całą zamieścić.

Pozostałe metody klasy zawierają jedynie obiekty lub nazwy stałych, które są wstrzy-

kiwane. W konstruktorze klasy `Injector` jest też inicjalizowane GPIO (General Purpose Input Output).

Obsługę GPIO zapewnia `WiringPi-Ruby`—wrapper Ruby popularnej biblioteki C `WiringPi`. W klasie `Injector` definiuje go metoda pokazana na listingu 9

```
1 def io
2   Wiringpi
3 end
```

Listing 9: `WiringPi`

Metoda `io` jest wstrzykiwana do klasy `RaspberryPi::Gpio`, która jest odpowiedzialna za ustawienie odpowiedniego kierunku pinu (INPUT—OUTPUT) oraz napisanie lub pobranie z niego.

```
1 class RaspberryPi::Gpio
2   takes :io
3
4   def read(pin)
5     mode(pin, INPUT)
6     io.digitalRead(pin)
7   end
8
9   def write(pin, value)
10    mode(pin, OUTPUT)
11    io.digitalWrite(pin, value)
12  end
13
14  private
15
16  def mode(pin, mode)
17    io.pinMode(pin, mode)
18  end
19 end
```

Listing 10: `app/services/raspberry_pi/gpio.rb`

Ta sama klasa 10 użyta jest w workerze. Jest to jedna z zalet obiektowości a w szczególności dependency injection ponieważ klasa nie ma żadnych bezpośrednich zależności i może być łatwo przeniesiona do innej aplikacji. Klasa `RaspberryPi::Gpio` jest w tej aplikacji użyta jedynie w klasie `Components::Capacitor` 11.

```

1 class Components::Capacitor
2   takes :gpio
3
4   def discharge(pin)
5     gpio.write(pin, LOW)
6   end
7 end

```

Listing 11: app/services/components/capacitor.rb

Zadaniem tej klasy jest napisanie wartości LOW na pin w celu rozładowania ładunku na kondensatorze.

Dzieje się to na początku pętli *while* w wątku z listingu 7. Następnie w tej samej pętli *śpimy* przez 1ms oraz odczyt ustawiany jest na zero. W kolejnej pętli *while* pin GPIO jest ustawiany na wejście, pobierana jest z niego wartość a kiedy kondensator przekroczy wartość graniczną jest zwracany jest odczyt inkrementowany w tej pętli.

Jeśli jego wartość przekroczy zadany czas potrzebny na naładowanie kondensatora (tutaj próg aktywacji) w Memcached ustawiany jest dokładny czas tego zdarzenia pod kluczem *start_time*. Odpowiedzialna jest za to klasa *CachingService* przedstawiona na listingu 12.

```

1 class CachingService
2   takes :memcache_connection
3
4   def get(key)
5     memcache_connection.client.get(key)
6   end
7
8   def set(key, value)
9     if memcache_connection.client.set(key, value)
10      return value
11    end
12  end
13
14  def fetch(key, &block)
15    result = get(key)
16    return result if result
17
18    set(key, yield)
19  end
20 end

```

Listing 12: app/services/caching_service.rb

Klasa w konstruktorze jako argument przyjmuje obiekt klasy MemcacheConnection pokazanej na listingu 14. Metody tej klasy *get* oraz *set* mają oczywiste działanie zatem zostaną pominięte. Warto jednak zwrócić uwagę na metodę *fetch* która próbuje pobrać wartość z cache a kiedy nic nie znajduje się pod przekazanym kluczem wywołuje metodę *set* ustawiającą klucz wartością zwracaną przez przekazany blok. Przykładowe użycie tej metody przedstawiono poniżej.

```

1 caching_service.fetch('query') do
2   User.find(id: @id)
3 end
4

```

Listing 13: Przykład wykorzystania metody fetch.

Klasa `Connections::MemcacheConnection` przyjmuje tutaj w metodzie `takes` symbol `:dalli_client`, który jest nazwą metody zawartej w klasie `Injector` zwracającą klasę `Dalli::Client`.

```
1 class Connections::MemcacheConnection
2   takes :options, :dalli_client
3
4
5   def client
6     @client ||= dalli_client.new(options.memcache_server,
7     options.memcache_client)
8   end
9 end
```

Listing 14: `app/services/connections/memcache.rb`

3.5.1.1 API

API dla aplikacji Backbone (interfejsu) zostało napisane we frameworku Sinatra. Framework ten został zaprojektowany do pisania niewielkich, lekkich aplikacji web i został wybrany ponieważ autor uznał że Ruby on Rails (najpopularniejszy framework Ruby) jest zbyt rozbudowany i uruchomienie aplikacji RoR na RaspberryPi mogło by zużyć zbyt wiele zasobów urządzenia. Sama aplikacja Sinatra uruchamiana jest na porcie 9292. Z tego powodu znajdują się za proxy — serwerem Nginx, który przekierowuje zapytania przychodzące na port 80 na aplikację.

Typowo aplikacja Sinatra składa się z jednej klasy. W celu wprowadzenia ład i składu autor zdecydował się ją rozbić na mniejsze pliki. Jako pierwszy załączany jest plik `application.rb`, który min. zawiera deklarację modułów, konfigurację Sprockets oraz konfigurację Warden, która zostanie omówiona w rozdziale 3.5.1.2. Na końcu załączane są pozostałe pliki zawierające poszczególne end-pointy, które monkey patchują klasę. Znajdują się w folderze `app/routes` i zostaną przedstawione poniżej przy pomocy zapytań curl i odpowiedzi. W celu autentykacji najpierw jest POSTtowany login. Potem możemy autentykować się przy pomocy cookie.

```
curl -H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-X POST http://localhost:9292/login \
-d '{"username":"admin", "password":"password"}' \
-c cookie
```

W przypadku udanego logowania serwer odpowiada JSONem

```
{
  "user": {
    "authenticated":true,
    "auth_token":"4a59a1427841af26"
  }
}
```

Contestants

W pliku `app/routes/contestants.rb` (listing ??) zawarte są 2 endpointy:

GET `/contestants/:id` zwracający zawodnika o podanym id oraz,

POST `/contestants` służący do tworzenia nowych zawodników.

```
curl -H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-X GET http://localhost:9292/contestants/1 \
-b cookie
```

Listing 15: GET `/contestants/:id`

```
[
  {
    "contest_id":1,
    "first_name":"Kamil",
    "last_name":"Wójcik",
    "end_time":null
  }
]
```

Listing 16: Odpowiedź zwracana przez GET `/contestants/1`

Przy tworzeniu zawodnika jest zwracany jedynie kod odpowiedzi HTTP. Jeśli dane są poprawne i zawodnik zostanie zapisany do bazy zwracany jest kod 200 (OK) w przeciwnym wypadku zwracany jest kod błędu 422 (Unprocessable Entity).

```
curl -H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-X POST http://localhost:9292/contestants \
-d '{"contestant":
    {"first_name": "oskar", "last_name": "kapusta"},
  "contest":
    {"id": "1"}}'
-b cookie
```

Listing 17: POST */contestants*

Do metody `get '/contestants/:id'` przekazywane są w bloku 2 zmienne: `contestant_presenter` oraz `contestant_repository`. `contestant_presenter` jest obiektem klasy `Presenters::ContestantPresenter`, której zadaniem jest prezentacja kolekcji zawodników w formatach JSON, CSV, XML oraz jako Hasz.

Użyta tutaj metoda `as_json` mapuje przekazaną kolekcję na tablicę haszy oraz wywołuje na zwracanej wartości metodę `to_json` co przedstawia listing 18.

```
def as_json(collection)
  collection.map do |contestant|
    {
      contest_id: contestant.contest_id,
      first_name: contestant.first_name,
      last_name: contestant.last_name,
      end_time: contestant.end_time
    }
  end.to_json
end
```

Listing 18: *app/services/presenters/contestant_presenter.rb*

Przekazywana także w bloku metody `post` zmienna `contestant_repository` jest obiektem klasy `Repositories::ContestantRepository` mającej za zadanie budowę, zapis oraz pobieranie z bazy zawodników 19. W konstruktorze przyjmuję model zawodnika *app/models/contestant.rb*.

W bloku metody `post` przekazywane są też zmienne `hash` i `json_parser` metody klasy *Injector* zwracającej stałą JSON, która jest potem użyta do sparsowania ciała rzędu HTTP. `hash` jest obiektem klasy `Hash` modułu *Skirace*. Klasa ta dziedziczy po `Hash` i ma na celu dodanie pomocniczych metod znanych z Rails takich jak użyta tutaj `with_indifferent_access` pozwalająca na odwołanie się do klucza haszu przy pomocy stringa lub symbolu.

```
class Repositories::ContestantRepository
  takes :db_contestant

  def build(params)
    contestant = db_contestant.new(params[:contestant])
    contestant.contest_id = params[:contest][:id]
    contestant
  end

  def save(contestant)
    contestant.save
  end

  def all
    db_contestant.all
  end

  def get(id)
    db_contestant.where(id: id)
  end
end
```

Listing 19: *app/services/repositories/contestant_repository.rb*

Contests

W pliku `app/routes/contests.rb` znajdują się następujące endpointy:

GET `/contests`

Zwraca wszystkie zawody z bazy.

```
get '/contests' do |contest_presenter, contest_repository|
  contest_presenter.as_json(contest_repository.all)
end
```

Listing 20: `app/routes/contests.rb`

```
curl -H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-X GET http://localhost:9292/contests \
-b cookie
```

Listing 21: GET `/contests`

```
[
  {
    "id":1,
    "name":"Zakopane 2014"
  },
  {
    "id":2,
    "name":"Szawnica"
  }
]
```

Listing 22: JSON response

3.5.1.2 Warden

3.5.1.3 Front-end

3.5.2 Worker

3.6 Obudowa ochronna

4 Wnioski

5 Licencje Gemów

5.1 Wykaz gemów

- **execjs**
Autorzy: Sam Stephenson, Josh Peek
Licencja: MIT
- **haml** MIT
Autorzy: Hampton Catlin, Nathan Weizenbaum
Licencja: MIT
- **haml_coffee_assets**
Autor: Michael Kessler
Licencja: MIT
- **uglifyer**
Autor: Ville Lautanala
Licencja: MIT
- **sinatra**
Autorzy: Blake Mizerany, Konstantin Haase
Licencja: MIT
- **sequel**
Autorzy: Sharon Rosner, Jeremy Evans
Licencja: MIT
- **sprockets**
Autorzy: Sam Stephenson, Josua Peek
Licencja: MIT
- **sprockets-helpers**
Autor: Perer Browne
Licencja: MIT
- **dependor-sinatra**
Autor: Adam Pohorecki
Licencja: MIT
- **therubyracer**
Autor: Charles Lowell
Licencja: MIT
- **wiringpi** GNU LGPLv3

- **thin**
Autor: Marc-Andre Cournoyer
Licencja: Ruby
- **sqlite3**
Autorzy: Jamis Bluck, Luis Lavena, Aron Patterson
Licencja: BSD-3
- **sass**
Autorzy: Nathan Weizenbaum, Chris Eppstein, Hampton Catlin
Licencja: MIT
- **binding_of_caller**
Autor: John Mair
Licencja: MIT
- **pry**
Autorzy: John Mair, Conrad Irwin, Ryan Fitzgerald
Licencja: MIT
- **warden**
Autor: Daniel Neighman
Licencja: MIT
- **bcrypt-ruby**
Autor: Coda Hale
Licencja: MIT
- **memcache-client**
Autorzy: Eric Hodel, Robert Cottler, Mike Perham
Licencja: BSD-3
- **dalli**
Autor: Mike Perham
Licencja: MIT
- **nokogiri**
Autorzy: Aaron Patterson, Mike Dalessio, Yoko Harada, Tim Elliott, Akinori MURASHIMA
Licencja: MIT

5.2 Treści licencji

5.2.1 MIT

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.2.2 LGPLv3

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- (a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

- (b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- (a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- (b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- (a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- (b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- (c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- (d) Do one of the following:
 - . Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - i. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

- (e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- (a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- (b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

5.2.3 Ruby

Ruby is copyrighted free software by Yukihiro Matsumoto matz@netlab.jp. You can redistribute it and/or modify it under either the terms of the 2-clause BSD (see the file `BSD`), or the conditions below:

1. You may make and give away verbatim copies of the source form of the software without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may modify your copy of the software in any way, provided that you do at least ONE of the following:
 - (a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or by allowing the author to include your modifications in the software.
 - (b) use the modified software only within your corporation or organization.
 - (c) give non-standard binaries non-standard names, with instructions on where to get the original software distribution.
 - (d) make other distribution arrangements with the author.
3. You may distribute the software in object code or binary form, provided that you do at least ONE of the following:
 - (a) distribute the binaries and library files of the software, together with instructions (in the manual page or equivalent) on where to get the original distribution.
 - (b) accompany the distribution with the machine-readable source of the software.
 - (c) give non-standard binaries non-standard names, with instructions on where to get the original software distribution.
 - (d) make other distribution arrangements with the author.
4. You may modify and include the part of the software into any other software (possibly commercial). But some files in the distribution are not written by the author, so that they are not under these terms.

For the list of those files and their copying conditions, see the file `LEGAL`.

5. The scripts and library files supplied as input to or produced as output from the software do not automatically fall under the copyright of the software, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this software.
6. THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5.2.4 BSD-3

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6 Summary

7 Dodatek A - Capistrano

8 Dodatek B - God

9 Dodatek C - YAGI