



**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

<b>1</b>	<b>Cel i zakres pracy</b>	<b>5</b>
<b>2</b>	<b>Wstęp</b>	<b>5</b>
<b>3</b>	<b>Realizacja tematu</b>	<b>6</b>
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 . . . . .	25
3.6	Obudowa ochronna . . . . .	25
<b>4</b>	<b>Wnioski</b>	<b>25</b>
<b>5</b>	<b>Licencje Gemów</b>	<b>26</b>
5.1	Wykaz gemów . . . . .	26
<b>6</b>	<b>Summary</b>	<b>28</b>



## 1 Cel i zakres pracy

Przedmiotem niniejszej pracy jest budowa mobilnego systemu pomiaru czasu dla zawodów 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.

Jako szkielet systemu zostało wybrane Raspberry Pi - platforma komputerowa stworzona przez Raspberry Foundation. W momencie premiery (29 luty 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 interface 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>

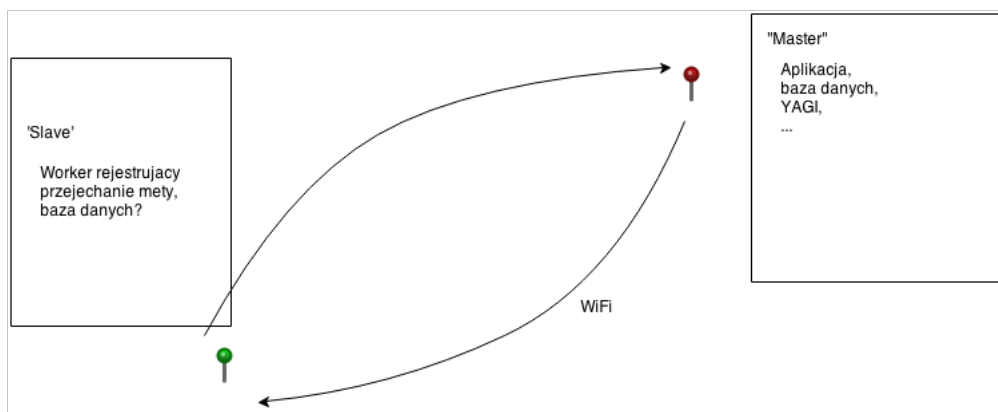
<https://github.com/okapusta/skirace-worker>

## 3 Realizacja tematu

### 3.1 Architektura

System którego ta praca dotyczy został zbudowany w myśl modelu master/slave gdzie masterem 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 bazie danych.

Zadaniem workera jest zarejestrowanie przecięcia linii mety a kiedy to się stanie worker wysyła rżądanie POST do aplikacji startowej, która w odpowiedzi zwraca obliczony końcowy czas oraz imię i nazwisko zawodnika wyświetlane na LCD.



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 id aktualnego wyścigu.

#### 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.

### (2.1) 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
```

### (2.2) 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/application.css*.

```

1  //= require_tree ./vendor
2  //= require app
3  //= require_tree ./lib
4  //= require_tree ./models
5  //= require_tree ./collections
6  //= require_tree ./templates
7  //= require_tree ./services
8  //= require_tree ./views
9  //= require_tree ./routers
10 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.

(2.3) HAML

```
1  .klasa
2
3  #id
4    %p
5      Paragraf
```

(2.4) HTML

```
<div class="klasa"></div>
<div id="id">
  <p>Paragraf</p>
</div>
```

Powyższe listingi przedstawiają przykładowy kod HAML oraz HTML do którego on się kompiluje.

### 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.

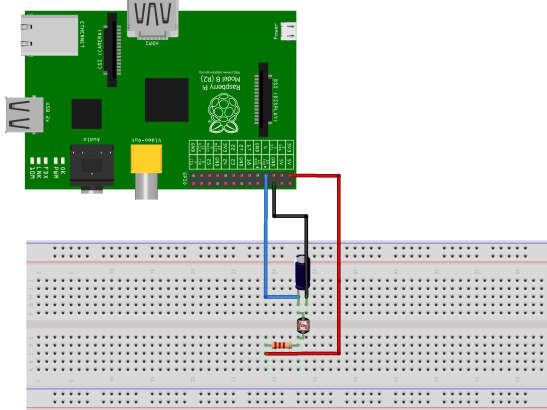
```
1 auto wlan0
2 iface wlan0 inet static
3     address 192.168.10.1
4     netmask 255.255.255.0
5     wireless-channel 1
6     wireless-essid SKIRACE
7     wireless-mode ad-hoc
```

Listing 6: */etc/network/interfaces*

Plik */etc/network/interfaces* drugiego urządzenia został skonfigurowany analogicznie. Różni się on jedynie adresem IP ustawionym na *192.168.10.2*. 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 niejszej 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. W momencie uruchomienia aplikacji jest też tworzony nowy wątek, którego zadaniem jest zarejestrowanie startu. Aplikacja front-end natomiast dostarcza interfejs dostępny przez przeglądarkę internetową służący do interakcji z back-endem.

```
1 module Skirace
2   module StartingLine
3     def self.registered(app)
4       Thread.new(Injector.new) do |i|
5         while true
6           reading = 0
7           i.capacitor.discharge(i.options.capacitor.pin)
8
9           while i.gpio.read(i.options.capacitor.pin) == LOW
10             reading +=1
11           end
12
13           if reading > i.options.activation_threshold
14             injector.contastant_repository.set_start_time
15           end
16
17           sleep i.options.measurement_accuracy
18         end
19       end
20     end
21   end
22 end
```

Listing 7: Wątek rejestrujący przejechanie mety

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

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 nazwom 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 Dendor::AutoInject
3   include Dendor::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 wstrzykiwane. W konstruktorze klasy `Injector` jest też inicjalizowane GPIO (General Purpose Input Output).

Obsługę GPIO zapewnia `WiringPi-Ruby`—wrapper Ruby popularnej biblioteki C `Wi-`

ringPi. 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 porządku 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 wykonane jest rżądanie POST z login oraz hasłem. 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* 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 dwie zmienne: `contestant_presenter` oraz `cotestant_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 żądania 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.

Takie podejście zostało wybrane ponieważ autor uważa że monkey patchowanie klas należących do *Ruby core* jest czymś czego nie powinno się robić.

```

class Repositories::ContestantRepository
  takes :db_contestant, :time_service, :time

  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).first
  end

  def first(options = {start_time_at: nil})
    db_contestant.where(options).order(:id).first
  end

  def set_start_time
    first.update(start_time_at: Time.now)
  end

  def set_end_time(end_time_at)
    contestant = first("start_time_at is not null and end_time is null")
    end_time = time_service.format(time.parse(end_time_at) - contestant.start_time_at)

    contestant.update(end_time_at: end_time_at, end_time: end_time)

    contestant
  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. Zmienna `contest_presenter` przekazana w bloku jest

```
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

obiektem klasy `Presenters::ContestPresenter` zawierającej jedynie metodę `as_json` działającą analogicznie do metody o tej samej nazwie klasy `Presenters::ContestantPresenter` przedstawionej wcześniej.

`contest_repository` jest tutaj obiektem klasy `Repositories::ContestRepository`. Metoda `all` najpierw sprawdza czy w bazie znajdują się zawody metodą `any?` Jeśli w bazie danych istnieją zawody są one zwracane w przeciwnym wypadku wywoływana jest prywatna metoda `create_default` (23).

```

class Repositories::ContestRepository
  takes :db_contest

  def all
    db_contest.any? ? db_contest.all : create_default
  end

  private

  def create_default
    [ OpenStruct.new(db_contest.create(name: 'Default').values) ]
  end
end

```

Listing 23: *app/services/repositories/contest\_repository.rb*

GET */contests/public*

Jeśli publicznie udostępnianie zawodów jest włączone zwrócony zostanie JSON zawierający id i nazwę zawodów oraz listę zawodników 26 w przeciwnym wypadku zwrócony zostanie JSON z odpowiedzią *no-public-contests* oraz statusem HTTP 404 (Not Found) 27.

```

get '/contests/public' do |public_contest_presenter, contest_repository|
  begin
    public_contest_presenter.as_json(contest_repository.get_public)
  rescue
    {response: 'no-public-contests', status: 404}.to_json
  end
end

```

Listing 24: *app/routes/contests.rb*

```

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

```

Listing 25: GET */contests/public*

```

{
  "id":1,"name":"Zakopane 2014",
  "contestants": [
    {
      "contest_id":1,
      "first_name":"Kamil",
      "last_name":"Wójcik",
      "end_time":null
    }, ...
  ]
}

```

Listing 26: Publiczne udostępnianie zawodów włączone

```

{"response":"no-public-contests","status":404}

```

Listing 27: Publiczne udostępnianie zawodów wyłączone

GET */contests/:id/contestants*

Zwraca listę zawodników dla zawodów o przekazanym id.

```

get '/contests/:id/contestants' do |contestant_presenter, contest_repository|
  env['warden'].authenticate!

  contestants = contest_repository.get(params[:id]).contestants
  contestant_presenter.as_json(contestants)
end

```

Listing 28: *app/routes/contests.rb*

POST */contests*

Tworzy nowe zawody.

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

Listing 29: GET `/contests/1/contestants`

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

Listing 30: Odpowiedź JSON

### 3.5.1.2 Warden

W celu autentykacji użytkowników aplikacji został wykorzystany gem Warden będący oprogramowaniem pośredniczącym (*eng. middleware*) dla aplikacji Rack. Warden wstrzymuje *leniwy obiekt* (który jest inicjalizowany tylko wtedy kiedy znajdzie na niego zapotrzebowanie) do `env['warden']`. Pozwala on np. na sprawdzenie czy użytkownik jest zalogowany `env['warden'].authenticated?` Warden pozwala na definiowanie strategii autentykowania użytkowników. Każda ze zdefiniowanych strategii będzie próbowała zautentykować użytkownika aplikacji aż któraś z nich się powiedzie. Strategie autentykacji, która została użyta w tym projekcie przedstawia listing 31.

```
Warden::Strategies.add(:password) do
  def valid?
    @params = parse_params
    @params['username'] && @params['password']
  end

  def authenticate!
    user = injector.user_repository.get_by_username(@params['username'])

    if injector.authentication_service.authenticate(user, @params['password'])
      success!(user)
    else
      fail!
    end
  end
end
```

Listing 31: Strategia autentykacji



### 3.5.1.3 Front-end

Interfejs aplikacji, tak jak była to tym mowa we wstępie, został napisany przy pomocy Backbone.js. Aplikacja frontend jest inicjalizowana ostatnią linią listingu 4. Obiekty aplikacji oraz funkcja inicjalizująca ją są zawarte w pliku *app.coffee* przedstawionym na listingu 32.

Listing 32: `app/assets/javascripts/app.coffee`

### 3.5.2 Worker

## 3.6 Obudowa ochronna

Obudowa urządzenia została wykonana ze skrzynek izolacyjnych D25 montowanych na rurach kanalizacyjnych PP 50/1000 mm przy pomocy opasek zaciskowych. W skrzynkach znajduje się RaspberryPI przykręcone do wygiętego fragmentu opakowania na płyty CD na warstwie ochronnej wycinku karimaty, koszynek na 6 baterii AA dostarczający zasilanie oraz układ UBEC - przetwornica impulsowa - redukująca napięcie z baterii do 5V 3A dla bramki startowej, oraz 5V 5A dla bramki końcowej.

## 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

## 6 Summary