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 2013/2014

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

Przedmiotem niniejszej pracy jest budowa mobilnego systemu pomiaru czasu dla zawodów narciarskich. System składa się z dwóch bramek: startowej oraz końcowej, które wykorzystują wiązke 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 systemy podobne do tego, którego budowę ta praca przedstawia, jednak często kosztują tysiące złotych. Praca ta jest próbą stowrzenia 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 Pi Fundation. 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 zoptymalizowaną wersją Debiana Wheezy dla procesorów ARMv6.

Obie aplikacjie (startowa i końcowa) zostały napistane 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 menadżera 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`

2.1 Założenia dotyczące systemu

Przed systemem, będącym celem tej pracy, stawia się szereg wymagań dotyczących jego funkcjonalności oraz budowy. Wymgania te zostały przedstawione poniżej, natomiast w rozdziale 3.1 autor przedstawi jak użyte materiały oraz technologie je spełniają.

1. Odporny na trudne warunki atmosferyczne
2. Lekki
3. Autonomiczny
4. Bezprzewodowy
5. Możliwość wprowadzania zawodników na starcie
6. Możliwość pomiaru czasu dla dwóch zawodników jednocześnie
7. Możliwość odczytu czasu po minięciu mety
8. Możliwość exportu danych do komputera

3 Realizacja tematu

3.1 Koncepcja wykonania

System pomiaru czasu został wymyślony tak aby jego budowa była prosta. Składa się on z dwóch identycznie wykonancyh bramek. Obie bramki składają się z urządzenia Raspberry Pi oraz modułu laserowego zamontowancyh na rurach kanalizacyjnych. Wiązka laserowa jest użyta w celu uchwycenia momentu przejechania bramki. Pada ona na fotorezystor, którego rezystancja jest niska kiedy jest oświetlony. Fakt ten został wykorzystany w układzie pomiarowym, który składa się jeszcze z kondensatora oraz rezystora. Kondensator jest ładowany i mierzony jest czas potrzebny na jego naładowanie. Kiedy jest wyższy niż zadana wartość graniczna znaczy to że wiązka została przecięta. Taka budowa oraz użyte materiały i podzespoły spełniają pierwsze trzy wymagania.

Aby spełnić czwarte wymaganie obie bramki komunikują się przy użyciu WiFi. Do Raspberry Pi podłączona jest karta WiFi a system skonfigurowany jest tak aby obie bramki wykryły się nawzajem i utworzyły śieć o SSID Skirace.

Wymagania 5 oraz 8 zostały spełnione poprzez stworzenie interfejsu webowego uruchamiangeo na bramce startowej. Aby z niego korzystać należy posiadać laptop oraz połączyć się z siecią Skirace, a następnie odwiedzić adres bramki (*192.168.10.1*). Alternatywną metodą wprowadzania zawodników jest użycie konsoli aplikacji. W tym celu należy się zalogować używając SSH do urządzenia na konto użytkownika app, posługując się hasłem skirace2014 lub kluczem publicznym. Następnie należy wejść do folderu */home/app/skirace/current* oraz poleceniem *rvmsudo bundle exec script/console* uruchmić konsolę.

Aby umożliwić natychmiastowy odczyt wyniku po przejechaniu mety (7) bramka końcowa została wyposażona w wyświetlacz LCD, na którym pokazywane są imię oraz nazwisko zawodnika oraz jego czas.

Ostanie wymaganie (6) spełnione jest w kodzie obsługującym działanie aplikacji startowej. W momencie przejechania mety zawsze brany jest z bazy danych pierwszy zawodnik, który ma ustawiony czas startowy a nie posiada czasu końcowego.

3.2 Budowa systemu

System pomiaru czasu składa się z czterech części. Tworzą one bramki, przez które zawodnik musi przejechać. Bramki zostały wykonane z rur kanalizacychnych aby urządzenie było lekkie i można było je łatwo wbić w śnieg. Kielichy rur nie zostały ucięte aby dało się je zatkać z góry zaślepką. Jako pierwszą autor przedstawi budowę elementu bramki, który zawiera moduł laserowy. Jest ona taka sama dla bramki startowej oraz końcowej.

Element zawierający moduł laserowy składa się z rury PCV, która stanowi obudowę ochronną dla elementów elektrycznych znajdujących się w środku. Stanowi też podporę oraz zapewnia odpowiednią wysokość dla lasera, taką żeby mógł on trafić w fotorezystor znajdujący się w otworze elementów odpowiedzialych za pomiar czasu. Wiązkę laserową zapewnia moduł laserowy składający się z diody laserowej oraz rezystora zapobiegającego

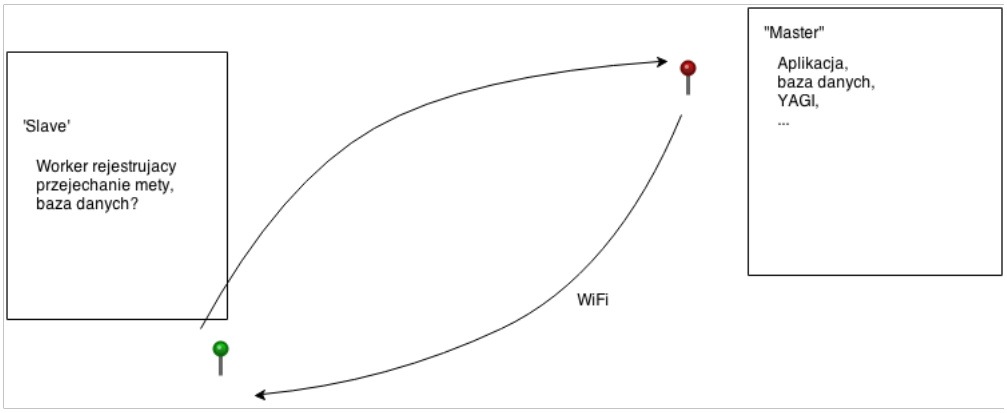
spaleniu diody. Emituje on wiązkę laserową o długości fali 650nm. Moduł zasilany jest z koszyczka na 6 baterii AA przymocowanego z tyłu obodowy (1). Napięcie dostarczane z baterii jest redukowane do 3.3V przez przetwornicę impulsową (układ UBEC). Koszyczek zamontowany z tyłu obudowy przymocowany jest opaską zaciskową w niewielkim wycięciu zapewniającym podporę. Laser jest uruchamiany przełącznikiem znajdującym się u góry.

Rysunek 1: Moduł laserowy

3.3 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 sie stanie, ustawia godzinę tego zdarzenia w bazie danych.

Zadaniem workera jest zarejestrowanie przecięcia linii mety. Worker wysyła rządanie POST do aplikacji startowej, która w odpowiedzi zwraca obliczony końcowy czas oraz imie i nazwisko zawodnika wyświetlane na LCD.



Rysunek 2: Architektura systemu

Komunikacjia pomiędzy dwiema bramkami odbywa się poprzez WiFi.

3.4 Wybrane technologie

3.4.1 Memcached

Memcached jest to rozproszony system buforowania pamięci podręczniej 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.

3.4.2 Dependency Injection

W ninejszej pracy został wykorzystany wzorzec projektowy *Dependency Injection (DI)* polegający na usuwaniu bespośrednich 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.

(3.1) Ruby

```ruby
class A
  attr_reader :obj

  def initialize(obj)
   @obj = obj
  end

  def do_b
    obj.do_sth
  end
end

a = A.new(B.new)
a.do_b
```

(3.2) Ruby + Dependor

```ruby
class A
  takes :b

  def do_b
    b.do_sth
  end
end

a = A.new
a.do_b
```

3.4.3 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 kompilacje assetów (JavaScriptów i CSSów) 'w locie', natomiast w środowisku produkcyjnym assety są prekompilowane. Sprockets pozwala również na minifikacje zasobów to jest zastąpienie nazw funkcji czy zmiennych pojedyńczymi znakami w celu zmiejszenia 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. Tutaj montowana jest ścieżka serwera WWW '/' tak aby pokazywała na aplikację Sinatra. Samo sprockets jest montowane w następujący sposób:

```ruby
map Skirace::Application.assets_prefix do
  run Skirace::Application.sprockets
end
```

Listing 1: config.ru

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

```
set :assets_prefix, '/assets'
set :assets_path, File.join(public_folder, assets_prefix)

set :sprockets, Sprockets::Environment.new(root)
```

Listing 2: Ustawienie zmiennych Sprockets

```
sprockets.append_path File.join(root, 'app', 'assets', 'javascripts')
sprockets.append_path File.join(root, 'app', 'assets', 'stylesheets')
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 assety załączone poleceniem *require* w plikach app/assets/javascripts/application.js (4) i app/assets/stylesheets/application.css (5).

```
//= require_tree ./vendor
//= require app
//= require_tree ./lib
//= require_tree ./models
//= require_tree ./collections
//= require_tree ./templates
//= require_tree ./services
//= require_tree ./views
//= require_tree ./routers
Skirace.init();
```

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

```
/*
 *= require bootstrap.min
 *= require bootstrap-responsive.min
 *= require app.css
 *= require_tree .
 */
```

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

3.4.4 Twitter Bootstrap

Twitter Bootstrap jest frameworkiem front-end dostarczającym style CSS oraz kod JavaScript, który za zadanie ma przyśpieszenie 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.

3.4.5 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.

(3.3) HAML

```haml
.klasa

#id
  %p
    Paragraf
```

(3.4) HTML

```html
<div class="klasa"></div>

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

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

3.4.6 Backbone.js

Backbone.js jest lekką biblioteką JavaScript nadającą strukturę aplikacją. 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.

3.5 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 konfiguracje urządzenia do pracy w trybie Ad Hoc.

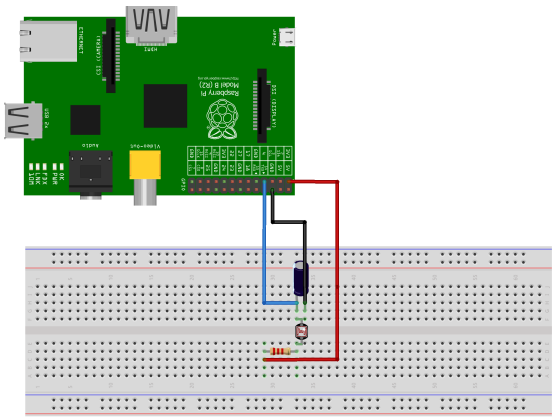
```
1 auto wlan0
2 iface wlan0 inet static
3   address 192.168.10.1
4   netmask 255.255.255.0
5   wireless-channel 1
6   wireless-essid SKIRACE
7   wireless-mode ad-hoc
```

Listing 6: /etc/network/interfaces

Plik */etc/network/interfaces* drugiego urządzenia został skonfigurowany analogicznie. Różni sie 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.

3.6 Podstawy działania

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



Rysunek 3: 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.

3.7 Implementacja

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

3.7.1 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. W momencie uruchomienia aplikacji jest też tworzony nowy wątek, którego zadaniem jest zarejestrowanie startu. Aplikacja front-end natomiast dostarcza interfejs dostępny przez przeglądarke internetową służący do interakcji z backendem.

```
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 maja być wstrzykiwane co przedstawia listing 8. Nazwy przyjmowane w argumentach metody *takes* oraz nazwy metod wywołanych na obiekcie Injector odpowiadają nazwom plików wstrszykiwanych 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 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 defniuje go metoda pokazana na listingu 9

```
def io
  Wiringpi
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.

```
class RaspberryPi::Gpio
  takes :io

  def read(pin)
    mode(pin, INPUT)
    io.digitalRead(pin)
  end

  def write(pin, value)
    mode(pin, OUTPUT)
    io.digitalWrite(pin, value)
  end

  private

    def mode(pin, mode)
      io.pinMode(pin, mode)
    end
end
```

Listing 10: app/services/rasperry_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 bezpośrenich 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) ustawiany jest dokładny czas tego zdarzenia.

3.7.1.1 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 (najpopularnieszy 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 znajuję się za proxy — serwerem Nginx, który przekierowuje zapytania przychodzące na port 80 na aplikację.

Typowo aplikacjia Sinatra składa się z jednej klasy. W celu wprowadzenia porządku autor zdecydował się ją rozbić na miejsze pliki. Jako pierwszy załączany jest plik *application.rb*, który min. zawiera deklarację modułów, konfigurację Sprockets oraz konfigurację Warden, która zostanie omówiona w rozdziale 3.7.1.2. Na końcu załączane są pozostałe piliki 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 rządanie POST z loginem 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 12: GET */contestants/:id*

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

Listing 13: 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). Do metody get

```
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 14: POST */contestants*

'/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 15.

Przekazywana także w bloku metody *post* zmienna contestant_repository jest obiektem klasy *Repositories::ContestantRepository* mającej za zadanie budowę, zapis oraz pobiera-

```
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 15: *app/services/presenters/contestant_presenter.rb*

nie z bazy zawodników 16. W konstruktorze przyjmuję między innymi model zawodnika.

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 Haszu 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.

```ruby
class Repositories::ContestantRepository
  takes :db_contestant, :time_service, :time, :caching_service

  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})
    current_contest = caching_service.get('current_contest')
    options.merge!({contest_id: current_contest}) if current_contest

    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 16: *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 17: *app/routes/contests.rb*

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

Listing 18: GET */contests*

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

Listing 19: JSON response

obiektem klasy *Presenters::ContestPresenter* zawieającej 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* (20).

```
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 20: *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 23 w przeciwnym wypadku zwrócony zostanie JSON z odpowiedzią *no-public-contests* oraz statusem HTTP 404 (Not Found) 24.

```
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 21: *app/routes/contests.rb*

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

Listing 22: GET */contests/public*

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

Listing 23: Publiczne udostępnianie zawodów włączone

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

Listing 24: 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 25: *app/routes/contests.rb*

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

Listing 26: GET */contests/1/contestants*

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

Listing 27: Odpowiedź JSON

POST */contests*
Tworzy nowe zawody.

```
curl -H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-X POST http://localhost:9292/contests \
-d '{"name":"Zawody"}' -b cookie --verbose
```

Listing 28: POST */contests*

Ten endpoint odpowiada jedynie kodem statusu HTTP 200 (OK) jeśli utworzenie nowych zawodów się powiedzie oraz 422 (Unprocessable entity) w przeciwnym wypadku.

PUT */contests/current*
Ma na celu ustawianie aktualnie wybranych zawodów w Memcached. Podobnie jak powyższy endpoint zwraca jedynie w odpowiedzi kod statusu.

```
curl -H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-X PUT http://localhost:9292/contests/current \
-d '{"id":"1"}' -b cookie --verbose
```

Listing 29: PUT */contests/current*

3.7.1.2 Warden

W celu autentykacji użytkowników aplikacji został wykorzystany gem Warden będący oprogramowaniem pośredniczącym (*eng. middleware*) dla aplikacji Rack. Warden wstrzykuje *leniwy obiekt* (który jest inicjalizowany tylko wtedy kiedy zajdzie 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 30.

```
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
```

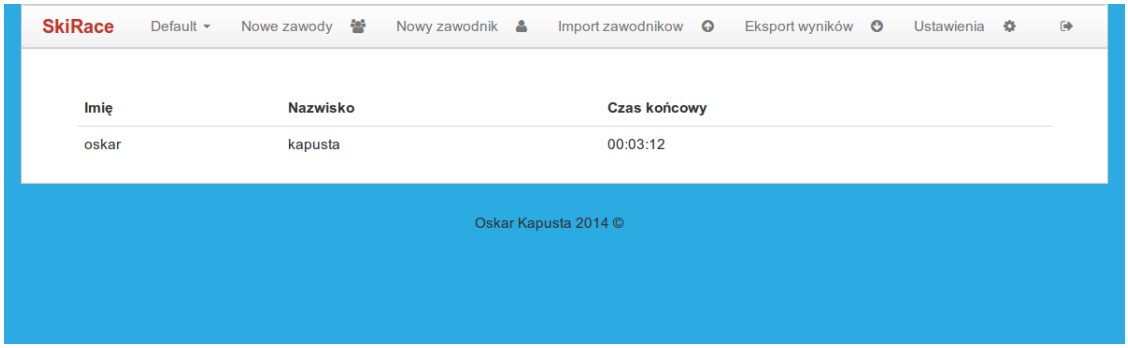
Listing 30: Strategia autentykacji

3.7.1.3 Front-end

Inerface apllikacji, 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 31. Rysunek 4 przedstawia jego wygląd.

```
window.Skirace =
  Models: {}
  Collections: {}
  Routers: {}
  Views:
    Contests: {}
    Contestants: {}
    Users: {}
    Export: {}
    Import: {}
    Settings: {}
  Services:
    Contestant: {}
  init: ->
    if typeof(Storage) == "undefined"
      alert "Twoja przegladarka nie obsluguje Web Storage\n\n" +
      "Obsługiwane przeglądarki: IE 8+, Firefox, Opera, Chrome, Safari."
      return
    app = new Skirace.Routers.Application()
    Backbone.history.start()
```

Listing 31: app/assets/javascripts/app.coffee



Rysunek 4: Interface

3.7.2 Worker

Worker o którym była mowa we wstępie do niniejszej pracy rejestruje przejechanie linii mety. Został on napisany w całości w języku Ruby jako *gem* - program menadżera paczek RubyGems. Po zainstalowaniu w systemie dostarcza on komendy *skirace-worker*, która go uruchamia.

Po uruchominiu startuje pętlę pobierającą wartość z czujnika a w momencie przejechania mety wysyła rządanie POST do aplikacji startowej oraz w odpowiedzi otrzymuje imię, nazwisko i czas końcowy zawodnika oraz wyświetla je na wyświetlaczu LCD zgodnym ze standardem HD44780. Strukturę oraz informacje o gemie definiuje plik *skirace-worker.gemspec*, który zawiera klasę ze specyfikacją.

Plikiem wykonywalnym gemu jest skrypt Ruby, który załącza plik *lib/skirace_worker.rb* zawierający definicje modułów workera. Dadaje on też kolejne pliki oraz klasy a następnie uruchamia worker (32).

```
1  #!/usr/bin/env ruby
2
3  require './lib/skirace_worker'
4  SkiraceWorker::Runner.run
```

Listing 32: bin/skirace-worker

Tak samo jak w aplikacji startowej w workerze również został wykorzystany wzorzec projektowy Dependency Injection. Obiekt klasy Injector jest inicjalizowany w klasie *SkiraceWorker::Runner* (33), która zawiera 2 publiczne metody klasowe (statyczne). Klasa *Injector* dostarcza obiektu klasy *Worker*, która posiada metodę *work* mającą analogiczne działanie do pętli z aplikacji startowej rejestrującej przecięcie wiązki lasera na starcie (7).

```
1  class SkiraceWorker::Runner
2    class << self
3      def run
4        injector.worker.work
5      end
6
7      def injector(opts = {})
8        Injector.new(opts)
9      end
10   end
11 end
```

Listing 33: lib/skirace_worker/runner.rb

Za wysłanie żądania do aplikaci startowej jest odpowiedzialna klasa *Connections::Application* (34). Jest ono skierowane do endpointu aplikacji */api/endtime*. Przykładowe zapytanie do API przedstawione jest na listingach 35 oraz 36.

```
class Connections::Application
  takes :http_client, :options

  def end_time(time)
    http_client.post options.api_url, end_time: time
  end

end
```

Listing 34: lib/skirace_worker/connections/application.rb

```
curl -X POST http://localhost:9292/api/endtime \
  -d "end_time=2014-09-28 12:54:57 +0200"
```

Listing 35: Żądanie wysyłane po przejechaniu mety

```
{
  "total_time":"00:01:07",
  "first_name":"oskar",
  "last_name":"kapusta"
}
```

Listing 36: Odpowiedź API

Odpowiedź API (36) jest następnie parsowana oraz formatowana przez klasę *Formatter (37)* a następnie wyświetlana na wyświetlaczu LCD.

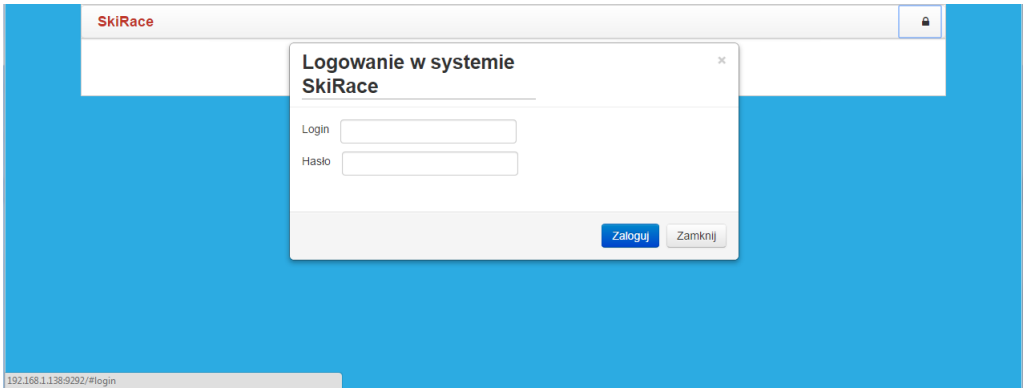
```
class Formatter
  takes :json_parser
  def time_format(response)
    parsed_response = json_parser.parse(response)

<<-eos
#{parsed_response['first_name']} #{parsed_response['last_name']} \n
time: #{parsed_response['total_time']}
eos
  end
end
```

Listing 37: lib/skirace_worker/formatter.rb

4 Sposób obsługi

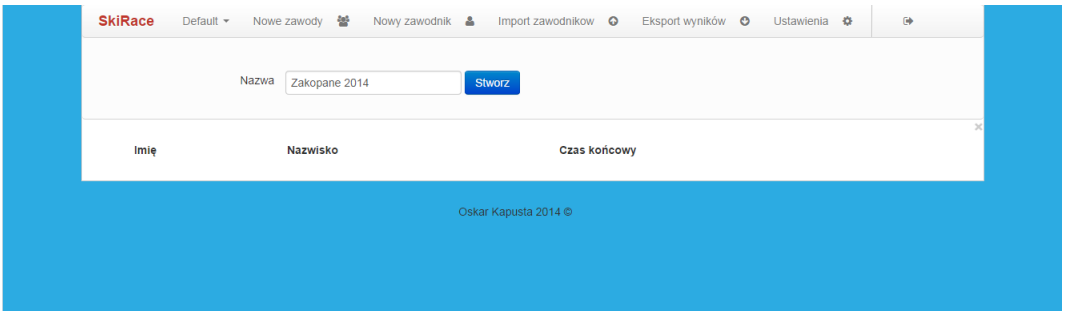
Urządzenie składa się z czterech elementów — startowego, końcowego oraz dwóch zawierających moduły laserowe. Aby z niego korzystać należy utworzyć dwie bramki wbijając je w śnieg tak aby wiązka lasera z częsci zawierającej moduł laserowy padała na czujnik znajdujący się w otworze wywierconym na bramce z systemem. Laser uruchamiany jest przełącznikiem znajdującym się na górze rur zawierających go. Po uruchomieniu wiązki laserowej i nakierowaniu jej na fotorezystor zależy uruchomić urządzenie przełącznikiem znajdującym się z boku skrzynki przymocowanej do elementu bramki.

Przed rozpoczęciem korzystania z systemu znależy utworzyć nowy wyścig oraz wprowadzić zawodników korzystając z interfejsu web lub konsoli aplikacji. W tym celu trzeba połączyć się z siecią bezprzewodową Skirace oraz otworzyć w przeglądarce internetowej aplikację znajdującą się pod adresem *192.168.10.1*. Następnie należy zalogować się w aplikacji klikając na ikonę kłódki (5) znajdującą się w prawym górnym rogu paska nawigacyjnego interfejsu używając do logowania użytkownika *admin* oraz hasła *password*.



Rysunek 5: Logowanie

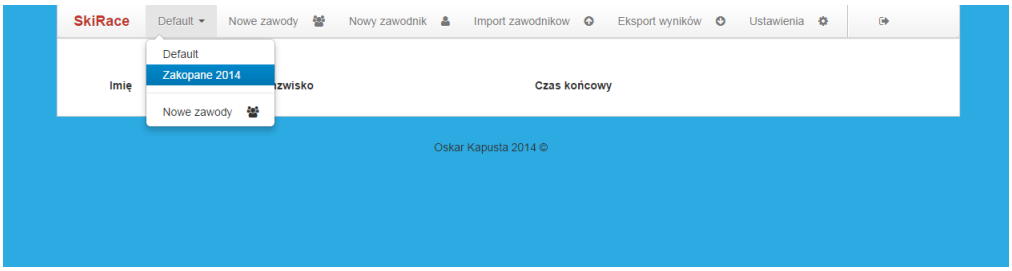
Jeśli w bazie danych nie isnieją zawody zostaną utworzone domyślne o nazwie *Default*. Aby utworzyć nowe zawody należy kliknąć na odnośnik *Nowe zawody* w pasku nawigacyjnym aplikacji (6).



Rysunek 6: Nowe zawody

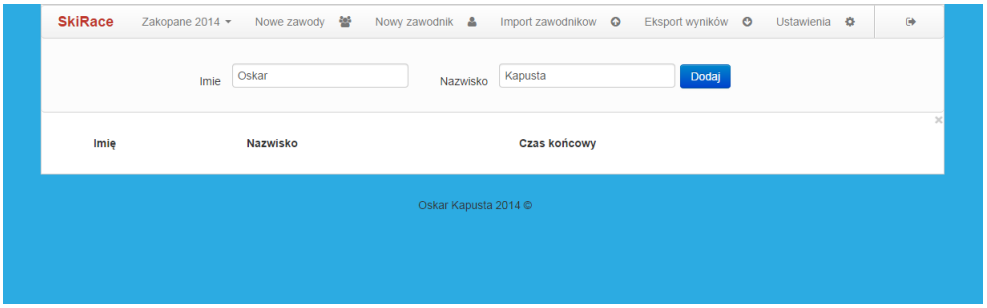
Następnie należy wybrać nowo utworzone zawody z menu rozwijanego znajdującego się

przed przyciskiem *Nowe zaowdy* (7).

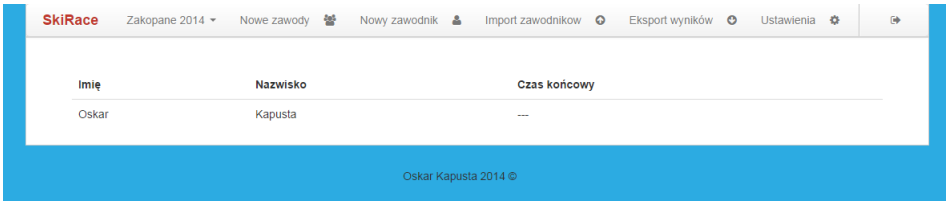


Rysunek 7: Wybór zawodów

W celu dodania nowego zawodnika należy wybrać *Nowy zawodnik* (8). Zostanie on dodany do aktualnie wybranych zawodów (9).

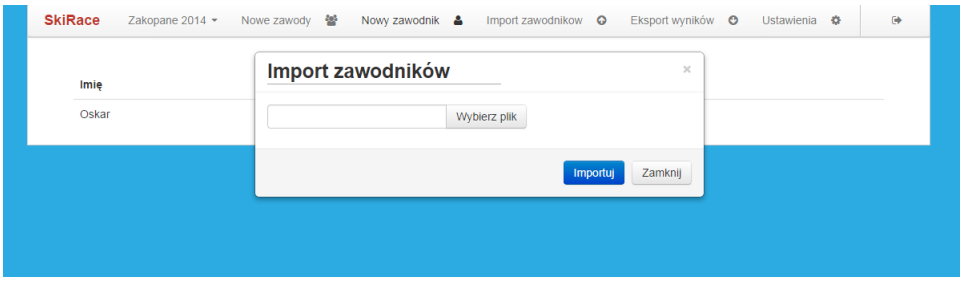


Rysunek 8: Dodanie nowego zawodnika

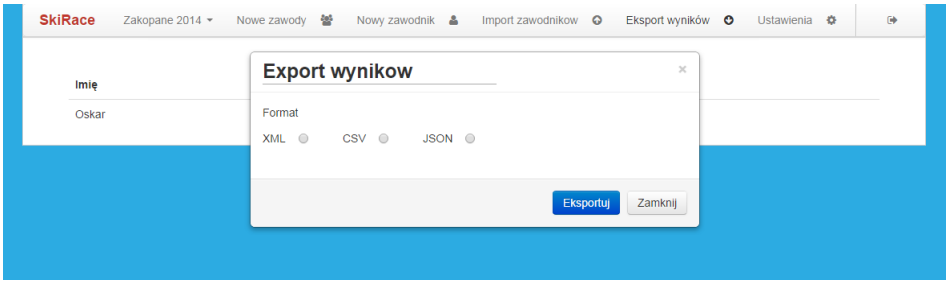


Rysunek 9: Nowy zawodnik

Import (10) oraz eksport (11) zawodników można wykonać w jednym z trzech obsługiwanych formatów: XML, CSV oraz JSON. W przypdaku eksportu eksportowani są zawodnicy w raz z czasami. Podczas importu można użyć wyeksportowanego pliku, jednak podczas parsowania czasy zawodników zostaną pominięte.



Rysunek 10: Import



Rysunek 11: Eksport

Ostatnią z funkcjonalności aplikacji web są ustawienia gdzie można włączyć publiczne udostępnianie zawodów (pokazanie wyników zawodników bez logowania). W ustawieniach znajduje się też zarządzanie użytkownikami aplikacji.

Ostatnia z ikon w pasku nawigacyjnym służy do wylogowania z systemu.

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

- **uglifier**
  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
  Licancja: 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 Eliott, Akinori MUSHA
  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 BSDL (see the file BSDL), 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

Subject of this paper is mobile timing system for skiing competitions. It consists of two gates that use laser beam to capture precise moment of crossing them by the contestant. Device which is subject of this paper was designed using widely available components. As backbone of this system Raspberry Pi was chosen. It's a cheap computer platform developed by Raspberry Pi Foundation that operates under control of Raspbian operating system – fork of Debian Wheezy GNU/Linux distribution. Gates communicate using wireless WiFi network working in Ad-Hoc mode and create network with ssid SKIRACE.

Both applications were written using Ruby 2.1.0 programming language. Starting application is built around Sinatra – micro framework designed for small-scale web applications and has additional web interface, written in CoffeeScript, that provides functionalities like creating contests or entering contestants. To give structure to front-end application Backbone.js was used – a lightweight JavaScript library that tries to reflect back-end MVC model into browser window. Sinatra application listens on port 9292 so Nginx was used – a reverse proxy server that redirects incoming requests on port 80 to back-end.

Finish line worker was built as gem – program of RubyGems package manager. When installed in the system it provides an executable that runs loop which measures time necessary to charge the capacitor. When capacitor is charged worker registers HIGH value on GPIO pin and makes request to starting application with exact time of this event. When that happens starting application sets end time in database and responds with name, surname and time of first contestant that hasn't have end time set. Response is then parsed and displayed on LCD screen.

Similar mechanism was employed in starting application to register time of crossing starting line. Most of classes, mainly responsible for GIPO, were copied from worker and closed in module included in main application file. When module is registered it starts a new thread that sets start time in database when beam is crossed.

This paper consists of three main parts. Subject of first of them is system architecture and different technologies that were employed in application, and in particular, communication between them and theoretical basis. Second part contains description of implementation of both applications and the third one covers putting system in protective casing.

At the very end of this paper author presented list off all gems that were used in both applications with their authors and licenses.