FitnessJunction.ie

Technical Report

John McCormack (x12100323), john.mccormack1@ncirl.ie

16 January 2013

# Executive Summary

FitnessJunction.ie is a social fitness website that allows users to record, analyse, and share their personal fitness data. It is intended to combine features of social networking sites such as Facebook or Twitter, with those of fitness tracking sites such as Runkeeper or Strava.

The web application is developed using the Ruby on Rails framework. It aims to use the features and principles of that framework to create a fully-featured site with separation of the presentation, logic, and data storage aspects. In common with many Rails applications, this development makes use of pre-existing code libraries, or “gems”, to integrate some features into the site. Features such as file upload, pagination, and Google maps mashups are integrated through the use of such gems.

Use is also made of AJAX technologies with XML being a core element of the data creation aspect of the site, while various Javascript libraries are used with the intention of improving the user interface of the site.

The site has a number of established competitors, and while it currently does not offer any unique selling point, I would hope to develop it further in the future by building on my knowledge of web technologies, particularly Ruby on Rails and the AJAX technologies.

# 1. Introduction

## Background

The project area was chosen for a number of reasons. I have a keen interest in sports, particularly cycling, and I make use of sites like Strava.com and MapMyRide.com to track my training. I have been interested for some time in developing a site that would potentially offer a different take on what these sites offer. While both are strong in the area of tracking fitness activities, I believe that the social networking aspect of such a site could be developed further, possibly by using events, such as a 10km run, or a triathlon for example, as drivers to create content on the site.

In developing this site, I wanted build on my knowledge of Ruby on Rails from Semester 1. I had found the experience of working with the framework an initially challenging experience, but was keen to learn more. This project offered the chance to put some of the lessons from last semester to use, and also to delve further in the many features offered by the Ruby on Rails framework.

I was also keen to develop a site that made use of AJAX technologies. My studies on this course to date have made it clear that Javascript and XML, together with the new HTML5 and CSS3 features, will be increasingly important to future web development. Internet users in the Web 2.0 era expect interactivity, responsiveness and flexibility from web applications. While not all of these goals may have been met yet in this project, I would hope to further develop the application to make maximum use of these technologies.

## Aims

In choosing to focus on the area of sporting and recreational activity, I wanted to deliver a website that combined features of a social networking site with those of a fitness tracking site. The goal was to create a framework for users to add content to the site. This would require an engaging user interface and features that made adding such content easy.

From the point of view of the server side technologies, I wanted to come away with a clearer understanding of the Model-View-Controller architecture of the Ruby on Rails framework. I also wanted to work with a MySQL database. Even though Rails’ Active Record class makes dealing with the database an abstraction, it was still an opportunity to use some of the MySQL commands that were taught in last semester’s Introduction to Computing module.

Another aim was to make use of third party APIs – to be able to research the available options for a particular problem, and then make use of the API itself by exploring its documentation and examples.

## Technologies

As described earlier in the introduction, the development framework was Ruby on Rails (version 3.2.8).

MySQL was used as the database rather than the default SQLite for a new Rails application.

The following gem libraries were used for various purposes:

* Twitter Bootstrap – styling and layout features
* Bcrypt-Ruby – password encryption
* Paperclip – file uploads
* Nokogiri & XPath– parsing XML files
* Faker – populating database with sample users
* Will-paginate – pagination
* Gmaps4rails – displaying data on embedded Google map
* jQuery-ui-rails - calendar date picker
* Rspec, Capybara & Factory\_Girl – testing
* jQuery-Fileupload – progress bars for file upload (to be completed)
* Morris.js & Raphael.js – barcharts & graphs display (to be completed)
* Pg – provide interface to PostgreSQL database (for deployment to Heroku)

In my project proposal, I specified that I intended to work directly with the Garmin Communicator & Google Maps APIs for the purposes of uploading data and mapping courses respectively. The Gmaps4rails gem took care of the mapping integration, while the Garmin Communicator API proved to be more difficult to decipher than I had hoped. The file format of the data that I hoped to use from a Garmin GPS device1 was also in a proprietary format (.fit) that I could not easily convert to XML. Rather than spend more time on their API, I uploaded files from the Garmin GPS device to the Garmin Connect website2 and then downloaded the data as XML. I have saved a number of such XML files in the root directory of the application for testing purposes3.

I used Adobe Flash to create button graphics for the home page which were then converted to .png format for use in the project.

My project proposal also mentioned that I intended using an integrated development environment (IDE) such as RubyMine or Aptana RadRails. Instead I used the Sublime Text 2 text editor which I found to be a less memory-intensive application with a good layout and set of features such as Multiple Selections and Distraction Free Mode.

# 2. Application Development

## Requirements

The project proposal set out a number of requirements in Section 4 - Requirements Specification (see Appendix)**.** These were broken down into the following areas:

* + - Interface requirements
    - Functional requirements
    - Data requirements
    - Non-Functional Requirements

Below, I will run through the individual requirements for each area, briefly re-stating what the requirement was, how implementation was approached, and the current status as of this report date.

### Interface Requirements

#### **Clear & Intuitive Layout**

*“The requirement is for a layout that is uncluttered and that makes it easy for the user to identify the features of the site.”*

I have used Twitter’s Bootstrap framework to provide the styling for the site. Bootstrap’s flexible grid system makes it quite easy to lay out elements on the page, while the built-in classes for buttons, drop-down menus, image placeholders and thumbnails are very convenient for quickly adding a consistent style to the elements of the site that the user will interact with. The Bootstrap library is included in the asset pipeline by importing it into a stylesheet (/app/assets/stylesheets/custom.css.scss).

#### **Advertising Space**

*“There should be sufficient space on the site for advertisements.”*

Sufficient whitespace has been allocated on most pages for possible use as advertising space. There may be less space available on the Profile page once all features are fully developed.

### Functional Requirements

#### **Registration & Log in**

*“This is a Registration/Login feature that allows new users to create a profile on the site, or allows existing users to log in.”*

This was an essential requirement for the project as all of the functionality of the site could only be accessed by logging in. In keeping with the requirement specification, the Registration form has been kept to a small number of fields as it was felt important to make it easy for a potential user to set up an account. An extended Registration form could potentially drive some users away. Only the name, email address, and password are collected at the Registration stage. Further personal details will be available to update from the user’s Profile page (currently, the user can add or edit a profile photo once logged in).

The Log In feature has also been kept simple, requiring the user to enter their email address and password to access their profile. Validations are used on both the Registration & Log In forms to ensure that valid email addresses and passwords are provided. Once logged in, the user is directed to their profile page.

#### **Profile Page**

*“A Profile Page will need to be created for each user. The user’s personal details should appear on the Profile Page, and there should be an option to edit some of these details. The page should show details of the user’s most recent activities as well as links to recent activities of site users to which they are linked.”*

The Profile page is a work in progress. Users can see a list of their activities (though the detail of each activity, for the time being, is minimal). There is a link to edit the user’s profile, where the user can update their name, email address & password, or add a profile photo. The Profile page also contains links to the user’s followers (i.e. other users of the site who are following that user) and following users (people the user is following). As this will be the main landing page for registered users of the site, this will require more development in terms of layout and content. It is envisaged that the user will be able to add comments to activities, which should facilitate a higher level of content generation.

#### **Create Activity**

*“A user should be able to create a new fitness activity and save it. Details of the activity should include the date, time, location, description, type, start time, end time, and optional details like comments, heart rate, and calories burned.”*

The Profile page contains a button link to the new Activity page. The Activity functionality has not yet been fully fleshed out, and for the time being the form for the new Activity looks for basic data such as the name, type, date and location. The newly created Activity then appears in the feed on the user’s Profile page. The next stage of development would likely be to update this feature of the site, for example using jQuery helpers for the Duration and Distance fields similar to the Date Chooser helper used for the date field. The Activity functionality would also have to be integrated with the Course functionality (see section *2.1.2.5 Map a Route*, below).

#### **Upload Data from GPS device**

*“A user may also choose to upload details of their fitness activity from a GPS device.”*

This functionality has not been fully implemented. The GPS device I am working with (a Garmin 500 cycling computer) does not store details of recorded bike rides in a format that I am able to work with. The .fit format that it uses stores data in a binary format which I have been unable to convert to XML. I have instead used the functionality of Garmin’s Connect website to upload the binary file and convert this to XML. Using an Upload function (see app/controllers/courses\_controller.rb), I have been able to access the data of the converted XML file which contains all the data recorded by the GPS device. The next stage to fulfil this requirement would be to delve into the Garmin API to work out how to include the .fit conversion into my site’s functionality. Further development would then see integration with other GPS devices, such as mobile phones.

#### **Map a Route**

*“When creating an activity, a user should have the option of mapping a route for that same activity. The route should display the start & end points, and a highlighted line on a map should show the route taken. Additionally, an elevation profile of the route may be available.”*

This functionality has been largely completed, though it has yet to be fully integrated into the intended workflow of the site. I have broken this functionality down into a few stages in order to work through the details of each before it is pieced together. A Course consists of a number of Trackpoints. Each Trackpoint has a longitude and latitude attribute. The value of these attributes is derived from values contained in the uploaded XML file:

<trkpt lon=**"-6.428207773715258"** lat=**"53.34898978471756">**

<ele>73.0</ele>

<time>2012-12-30T08:34:28.000Z</time>

<extensions>

<gpxtpx:TrackPointExtension>

<gpxtpx:atemp>15.0</gpxtpx:atemp>

<gpxtpx:hr>112</gpxtpx:hr>

<gpxtpx:cad>64</gpxtpx:cad>

</gpxtpx:TrackPointExtension>

</extensions>

</trkpt>

The XML file is initially uploaded, then parsed for the ‘lon’ and ‘lat’ values which are in turn assigned to new Trackpoints that are built for each pair of coordinates. The Trackpoints together are built into a Course. Using the Gmpas4Rails gem, the Course details are then passed to the view as a variable called ‘@course\_polylines’, which displays a line on a Google map tracing the route taken on the Course. A variable called ‘@markers’ contains the coordinates of the first and last Trackpoint, and these are used to display markers on the map.

The next step in developing this requirement is to build the Course into the Actiity model, so that a Course can only exist if it is attached to an Activity. The Activity feed in the user’s profile page would then include a thumbnail of a map for any Activity that consists of a Course.

Additional detail could be drawn from the XML file to display graphs for elevation, cadence, heart-rate, temperature, etc.

#### **Create Segments**

*“When a route has been created, it should be possible to select a portion of that route and create a segment.”*

This functionality has not yet been implemented, as the required first step, i.e. mapping the route, has only recently been completed.

#### **Calendar**

*“All activities completed by a user, and any events signed up to by the user will be stored in a calendar.”*

This requirement has yet to be completed.

#### **Search Facility**

*“A search feature should be available that allows a user to search for other users, or to search for routes that match criteria selected by the user such as distance, location, or type.”*

A limited Search function is available that enables the user to search for other users by name. The next stage of this requirement would be to allow the user to search for activities, courses, clubs, and events.

#### **Follow other Users**

*“Having searched for another user, it should then be possible to “follow” that user. A button or link will appear in the user’s profile page that provides this feature.”*

This requirement has been completed. Based on Michael Hartl’s Ruby on Rails Tutorial, this functionality allows a user to view another user’s profile, and then select that user as someone to follow. Users can be unfollowed also. Html partials are used to display Follower stats in the user’s profile page. The Activity feed includes details of Activities by followed users.

The next stage of this requirement would be to incorporate a ‘comment’ feature that would allow users to further interact with their followed and following users.

#### **Create Groups & Events**

*“It should be possible to create a group or event to which users can sign up.”*

This requirement has yet to be completed.

#### **Download route details to compatible GPS device**

*“A user should be able to search for and download a route to their GPS device.”*

This requirement has yet to be completed.

### Data Requirements

#### **Integrate data from GPX, TCX files into database**

*“When uploading activity details from a GPS fitness device, the site will have to be able to accept and work with the XML formats of GPX (GPS eXchange Format) & TCX (Training Center XML ) that are specific to some GPS fitness devices.”*

This requirement has been partially completed. While not yet able to work with the native .fit format of the GPS device, I have been able to get this file converted and have been able to take data from this file and integrate it to the database (see 2.1.2.5 above).

#### **Data Retention**

*“Data must be retained in order to provide some of the core functionality of the site.”*

Data is stored in a MySQL database.

### Non-Functional Requirements

#### **Performance requirement**

*“The performance of the site should be on a par with its competitors.”*

As the site is not deployed to a production server, this comparison cannot be carried out yet.

#### **Availability requirement**

*“Website should be available 24 hours. Server hosting should be reliable and consistent.”*

N/A.

#### **Security requirement**

*“As users are sharing some personal data with others, there is a requirement that the site is secure from a login perspective.”*

The application uses a gem called ‘bcrypt-ruby’ to encrypt each user’s password for security purposes. It also uses the built-in Rails method ‘has\_secure\_password’ to compare the encrypted password to the one supplied in the sign in form.

#### **Extendibility requirement**

*“Application should be easily scalable to cope with an increase in users.”*

The development is not yet at a stage where this requirement needs to be met.

## Design and Architecture

The architecture of the application was set out in the Requirements Specification document under the heading “System Architecture”. This is reproduced below. A couple of changes were made to this structure for the development. The Route model was renamed Course, while an additional Trackpoint model was added for the purposes of mapping the Course to a Google map.

## Implementation

The main classes and functions in the application are set out below.

**User Class (/app/models/user.rb):**

The User authentication, relationship to other users, validations, and search parameters are set out in the User model:

**class User < ActiveRecord::Base**

**attr\_accessible :email, :name, :password, :password\_confirmation, :photo**

**has\_secure\_password**

**has\_many :activities, dependent: :destroy**

**has\_many :relationships, foreign\_key: "follower\_id", dependent: :destroy**

**has\_many :followed\_users, through: :relationships, source: :followed # specifies the followed column in the relationships table as the source of the followed\_users array**

**has\_many :reverse\_relationships, foreign\_key: "followed\_id",**

**class\_name: "Relationship",**

**dependent: :destroy**

**has\_many :followers, through: :reverse\_relationships, source: :follower**

**has\_attached\_file :photo**

**# callback methods**

**before\_save { |user| user.email = email.downcase }**

**before\_save :create\_remember\_token**

**validates :name, presence: true, length: { maximum: 50 }**

**VALID\_EMAIL\_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i**

**validates :email, presence: true,**

**format: {with: VALID\_EMAIL\_REGEX},**

**uniqueness: { case\_sensitive: false }**

**validates :password, length: { minimum: 6 } # don't need to explicitly validate presence of password as this is done auotmatically by has\_secure\_password**

**validates :password\_confirmation, presence: true**

**def feed**

**Activity.from\_users\_followed\_by(self) # from\_users\_followed\_by method defined in activity.rb**

**end**

**def following?(other\_user)**

**relationships.find\_by\_followed\_id(other\_user.id) # takes in a user, called other\_user, and checks to see if a followed user with that id exists in the database**

**end**

**def follow!(other\_user)**

**relationships.create!(followed\_id: other\_user.id) # the follow! method calls create! through the relationships association to create the following relationship**

**end**

**def unfollow!(other\_user)**

**relationships.find\_by\_followed\_id(other\_user.id).destroy**

**end**

**def self.search(search)**

**if search**

**where 'name LIKE ?', "%#{search}%"**

**else**

**scoped**

**end**

**end**

**private**

**# use the urlsafe\_base64 method from the SecureRandom Ruby library module to create the token**

**def create\_remember\_token**

**self.remember\_token = SecureRandom.urlsafe\_base64**

**end**

**end**

**Course Controller Class (/app/controllers/courses\_controller.rb):**

The Course Controller specifies, among other things, how to deal with an uploaded XML file (in the Upload method). It also declares the variable for use in the view (the Show method). Below are the methods for uploading, parsing and creating the Course from an array of Trackpoints.

**def upload**

**# ref: http://stackoverflow.com/questions/2521053/how-to-read-a-user-uploaded-file-without-saving-it-to-the-database**

**file\_data = params[:file] # file received from upload form**

**map\_course = Nokogiri::XML(File.read(file\_data.path)) # creates Nokogiri document for parsing purposes**

**map\_course.remove\_namespaces!**

**@trkpt\_elements = map\_course.xpath("//trkpt") # uses Xpath to select all the trkpt elements from the document...**

**@trkpt\_attribute\_values = @trkpt\_elements.map do |n| #... and maps each value from those elements**

**n.values**

**end**

**@course = Course.new**

**@trkpt\_attribute\_values.each do |long, lat| # @trkpt\_attribute\_values is an array of arrays**

**@course.trackpoints.build(:longitude => long, :latitude => lat) # here the trackpoints for the new course are created using the values from the @trkpt\_attribute\_values array**

**@course.save**

**end**

**end**

The Show method below arranges for the latitude and longitude attributes to be passed to a json variable for use in the view.

**def show**

**@course = Course.find(params[:id])**

**@course\_polylines =[]**

**@course\_trackpoints = []**

**@course.trackpoints.each do |c| # create an array of hashes to stoe the lat & lon attributes from the trackpoint model**

**@course\_trackpoints << { :lng => c[:longitude], :lat => c[:latitude]}**

**end**

**@course\_polylines << @course\_trackpoints # passes the values into the @course\_polylines variable**

**@course\_polylines = @course\_polylines.to\_json # converts to json for use with the gmaps4rails gem**

**@markers = [@course\_trackpoints[0], @course\_trackpoints[-1]] # select the first & last trackpoints for use as markers for the start & finish of the course**

**@markers = @markers.to\_json**

**respond\_to do |format|**

**format.html**

**format.json { render json: @course}**

**end**

**end**

**Use of Partials to display Follower stats (app/views/shared/\_follow.html.erb)**

This partial is called in the User show page (app/views/users/show.html.erb) to render either the “Follow” or “Unfollow” buttons for other users of the site.

**<% unless current\_user?(@user) %>**

**<div id="follow\_form">**

**<% if current\_user.following?(@user) %>**

**<%= render 'unfollow' %>**

**<% else %>**

**<%= render 'follow' %>**

**<% end %>**

**</div>**

**<% end %>**

## Testing

A limited amount of testing was done using the RSpec framework. A suite of tests was drawn up following the guidelines in Michael Hartl’s Ruby on Rails Tutorial. These tested mostly for the appearance of content on pages, for instance, that the ‘title’ element matched the page. The tests adhered to the ‘fail-implement-pass’ development cycle, meaning that failing test were written first, then implementing application code to get the tests to pass.

For the purposes of speeding up the development of the prototype, I stopped following the tutorial’s guidelines on testing, and have continued the development since without writing new tests, or checking that existing tests were passing. The tests that do exist can be run by using the command:

**bundle exec rspec spec/**

(This will include a number of tests that currently fail.)

# 3. Conclusions

I believe that a lot of the aims of the project have been met. I feel that I have improved my understanding of the Ruby on Rails framework. Working with the various plugin code libraries has enabled me to integrate some rich user interfaces, for example with the date chooser widget on the New Activity page, or the pagination feature on the Users page.

My understanding of the Model-View-Controller structure has developed also, largely to the work involved in uploading and parsing an XML file, then redirecting the browser to display data in the view gathered by the controller.

I intend to keep working on the development past the end of the course to try and improve what functionality is there, and to add new features.

# Further development or research

The next phase of development would involve integrating some of the existing functionality to form a coherent whole. The Activity model needs to be connected with the Course model while further data needs to be gathered from the XML files to build the Course and Activity models.

Additional research will need to be done on integrating the various GPS devices into the application. Rather than have a user click on an upload button, the application should be able to recognise when there is a GPS device connected to the computer and then read any files contained on that device that have not already been uploaded.

A Comments model would have to be developed to increase the user-generated content of the site. This would also necessitate the re-working of the profile page layout, possibly making more use of the Bootstrap framework for styling purposes.

Another area that would need to be researched is that of deployment. The initial stages of the project were deployed to Heroku, but this has not been followed up on as it was not deemed to be a core part of the project. Exploring other options for deployment, such as Engine Yard or Blacknight would be an area that I would be looking at in the future.

Learning more about testing frameworks, including RSpec, Cucumber and Capybara is also something that will form part of the future development of this application. Experience of Test-driven development appears to be an important consideration for potential employers in this area.

# Notes

1. Garmin 500 computer: <https://buy.garmin.com/shop/shop.do?cID=160&pID=36728>
2. http://connect.garmin.com/
3. training1/2/3/4/5/6.xml

# Bibliography

Black, D. (2006) *Ruby For Rails.* Greenwich, CT.: Manning Publications Co.

Fernandez, O. (2011) *The Rails 3 Way.* Boston, MA.: Pearson Education, Inc.

Hartl, M. (2013) ‘Ruby on Rails Tutorial’ [Internet]. Available from <http://ruby.railstutorial.org/ruby-on-rails-tutorial-book> [Accessed November-December 2012]

# Appendix

1. Project Proposal (copy attached to printed version of this report, also included on CD).
2. Project Plan (Gantt Chart included in Project Management report on CD).
3. Requirement Specification (copy attached to printed version, and on CD).