ENSTA Bretagne

Rapport

CI 2016

October 1, 2015

Report of Internship

ZHENG Tao
tao.zheng@ensta-bretagne.org
Système logiciel et Sécurité

1   Introduction

1.1   Presentation of WRSC

The WRSC(World Robotic Sailing Championship) is a competition open to fully autonomous and unmanned sailing boats up to 4m in length. This year's WRSC/IRSC is being held in Finland from August 31st to September 4th, 2015. It is the eighth edition of the regatta, with previous events held in Austria (2008), Portugal (2009), Canada (2010), Germany (2011), Wales/UK (2012), France (2013), and Ireland (2014).

1.2   History Project SWARMON

The tracking system and website made to follow the boats have been first developed through the project SWARMON from 2013 to 2014. This project was realised by ENSTA Bretagne's students Quentin DESCOURS, Benoit BOURDON, Jean-Jacques BOYE, Simon STEPHAN and the responsible teacher Olivier REYNET.

The project was improved during summer 2014 by Bastien DROUOT and Benoit BOURDON in the Åland University of Applied Sciences, under the direction of Ronny ERIKSSON, vice rector of the University. It was used successfully for the WRSC 2014 in Galway (Ireland).

Then he tracker and the website were again enhanced by Quentin DESCOURS, Benoit BOURDON and Bastien DROUOT during their third year at ENSTA Bretage from 2014 to 2015.

Finally, the actual version is meant for the WRSC of 2015 that happened in Mariehamn in Åland (Finland). It was built up by Tao ZHENG and Sylvain Hunault, students from the ENSTA Bretagne, thanks to the help of the predecessors and under the guidance of Anna FRIEBE, Project Manager at Åland University of Applied Sciences, during summer 2015 in Mariehamn.

1.3   Internship Objectives

As all the competitive robots accomplish their missions in the sea, and it is really hard to supervise the whole traces of all robots, furthermore it is impossible to measure accurately all the positions of robots, just based on the crews' visions, especially under the extreme weather conditions. From this raison, it is necessary to build a reliable tracking system which can gather all GPS data and time stamps in order to record all the states during certain periods not only for controlling robots' states in case of intervening collision avoidance, but also for evaluating the final performance of all the robots.

With the efforts of our predecessors, we had already defined the usable API for our electronic cards, which means we had already defined the API for gathering GPS data including latitude, longitude, date time, course and speed, but with the antennas in the previous version, because the antennas were not suitable and specified for our desired frequency to gather GPS information, although we could also obtain all the GPS data what we want, the precision of measurement remains insufficient. So for the hardware part, we need to choose appropriate antennas, and we need to prepare all the materials for the tracking system for the final competition(like: battery, electronic cards, SIMCOM modules, SIM cards, and boxes which were used to put the whole tracker), unfortunately, at the beginning of our internship, the number of participated teams is still unknown, to make sure we got enough trackers, we booked 10 for each module. What's more, we had the Lua code which permits the electronic card to be run automatically.

For software part, the previous version of website based on the technology Ruby On Rails and the google map (Javascript APIs were provided and published on the Internet by Google Company freely.) And the performance of the previous version website in Galway (Ireland, WRSC 2014) proved that the website was built successfully. However, the old css and html were obsolete, and the admin marker creation need to be added. Since the website could display all the information gathered by the tracker in the google map, but the website did not handle of

these information further till this moment. So as for the website, several domains remain to be improved:

- The security of account need to be reinforced;

- Replay and real-time display methods need to be improved to reduce the pressure of server;

- Scoring and Ranking based on the data gathered by tracker;

- Admin marker creation need to be added, also need to add the markers to the page replay and page real-time.

- Css and html features need to be updated.

2 Tasks, Projects and Activities

2.1 Dairy tasks and activities

Actually, the development of website is function-oriented, the branch "master" is the principal branch which will be released to the public, besides every one or two weeks, a new branch which means a new feature was added into the website. And also every new feature need to be tested before merging into the "master" branch.
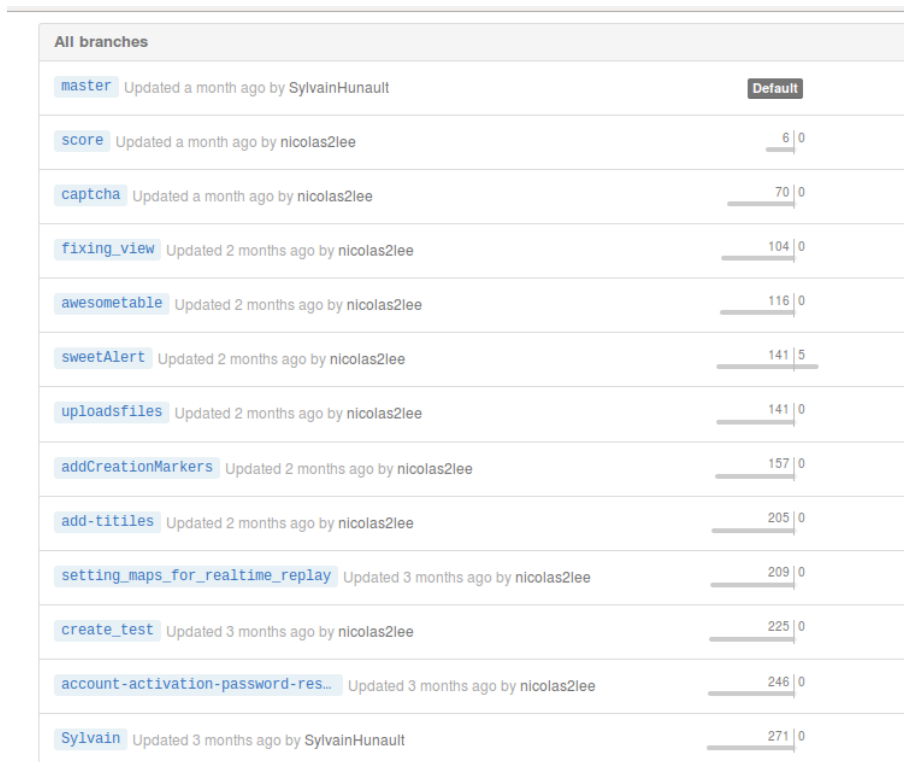

Figure 1: All function-oriented branches

2.2 New features

2.2.1 New style and New GUI

The web site created for the WRSC 2014 (project SWARMON) contains a no open sources CSS and Layout(left image in the figure before), and at the beginning of 2015, the web site was updated with the latest CSS3 which is open sources and more beautiful and customizable(right image in the figure before). Our current web site based on the latest CSS3.

In the previous version, the web site could only display the basic information about every team, like: team name, leader, members and description. In order to simplify the access, we add two fields by using two Ajax requests to permit the management of members and robots, so team leader could kick their members by itself rather than ask the administrator to do so.

Similar to the page team#show, the previous web site could only display the basic information about every robot, like: name, owner(team), category. These information are essential but insufficient, especially for the new visitors, the users do not know which mission their robot participated (if they want to know, they need to go to the page attempts, and check their robot
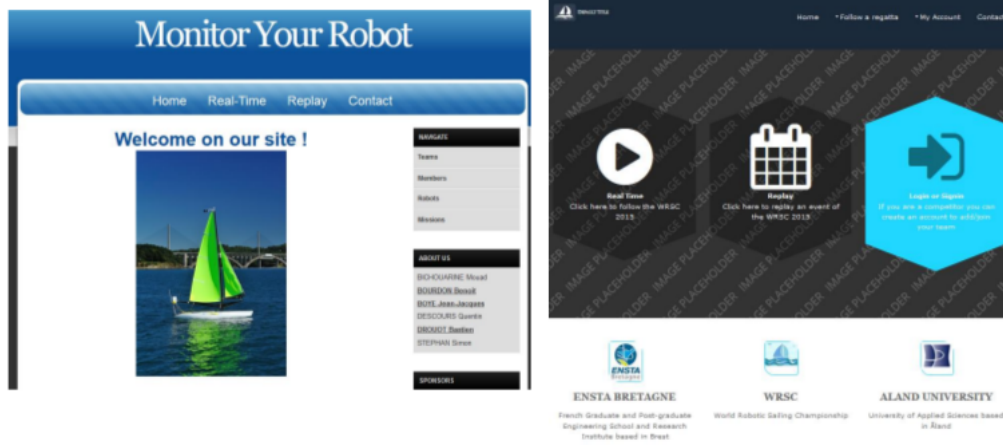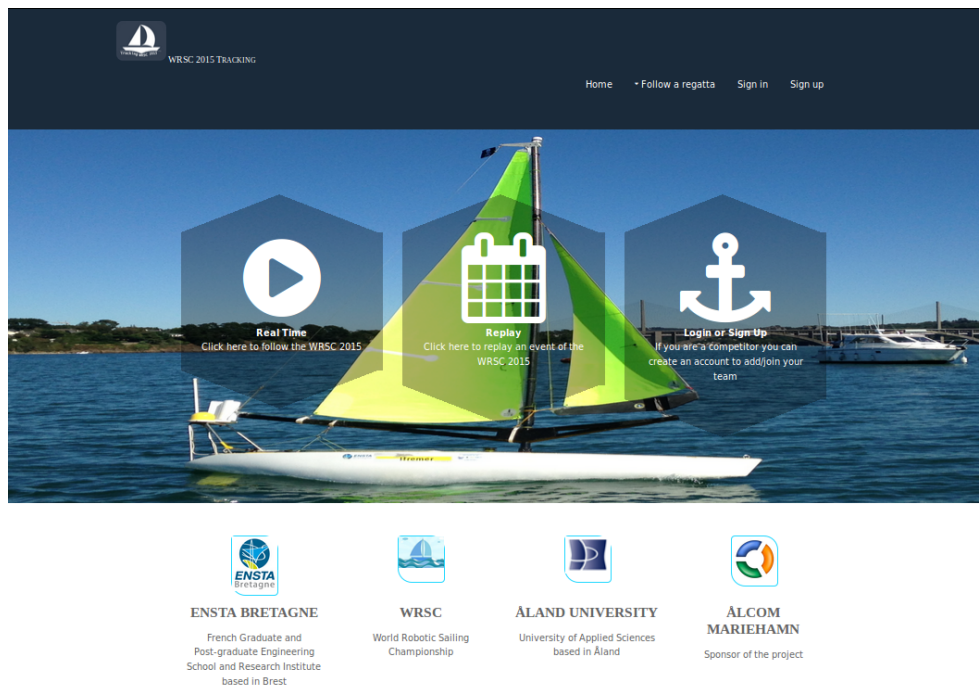
Figure 2: old home page, WRSC2014, MYR2015

Figure 3: New Home Page

name in the whole list one by one and also if they want to know the time for missions, or/and the time for every attempt, they need to go another page to check all the information). So in order to facilitate the use of web site, we decided to put all these information into the page robot#show. Also we had thought a lot about the form to display all these information to users, since the logic is that a robot participates several missions, and for every mission, there could be more attempts, and every score and tracker id were attached to an attempt. So based on this hierarchical order, it will better to choose a tree structure chart to show the logic. In order to meet our need, we had chosen google org charts which are open sources and all APIs are provided
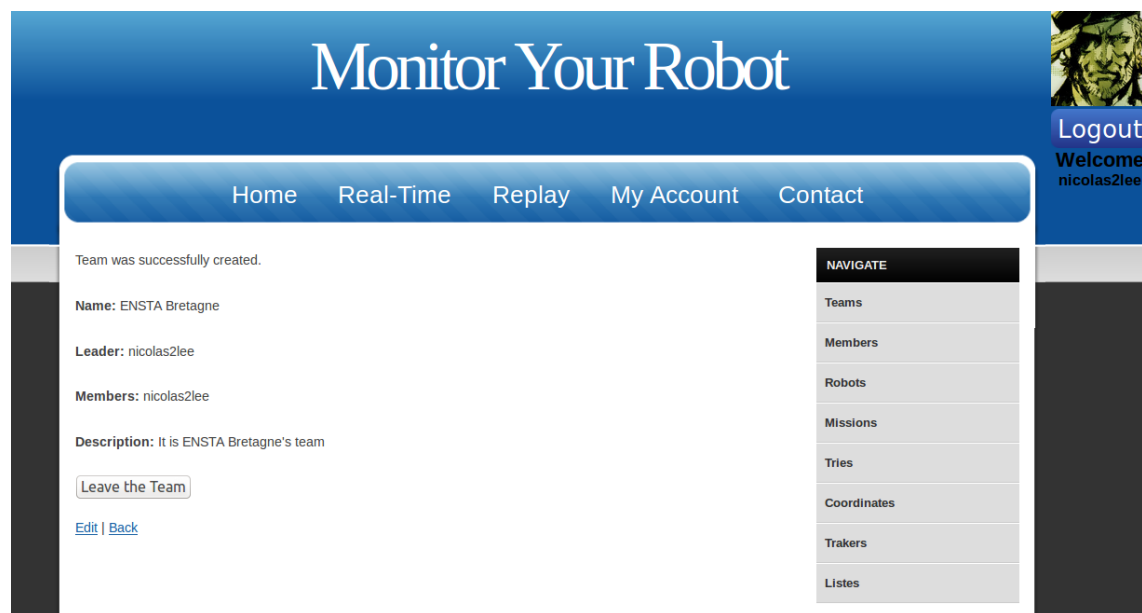
Figure 4: old team show

and published by Google company. Since our web site was built up by ruby on rails, and google org charts were released as JavaScript APIs, from this, every time when the site loading the robot#show page, it will recall a Ajax request and send all the associated information to the JavaScript part, then in the JavaScript part, we could use the org charts' APIs directly.
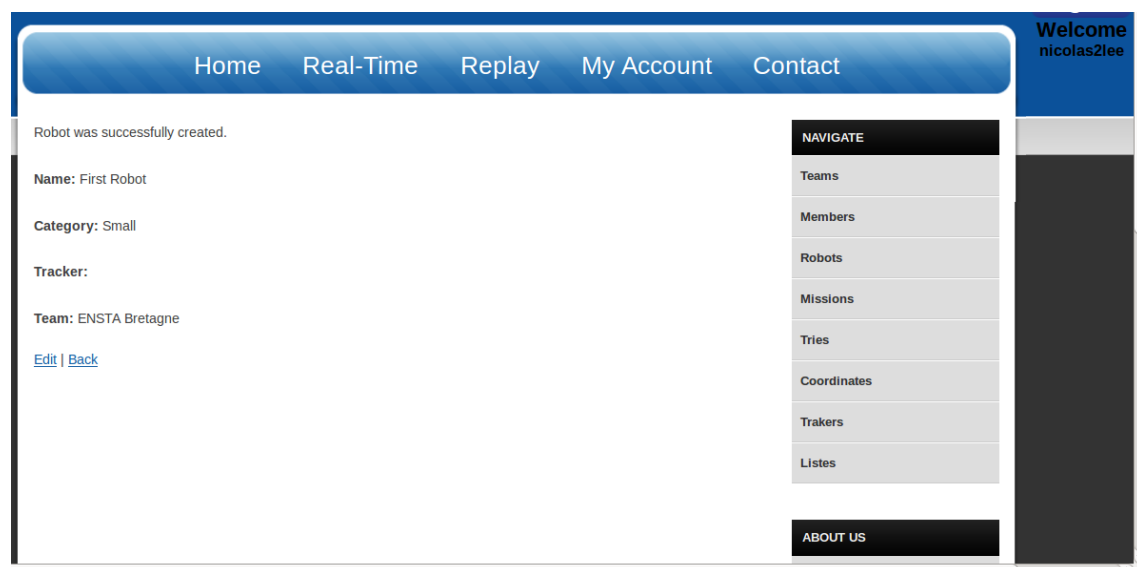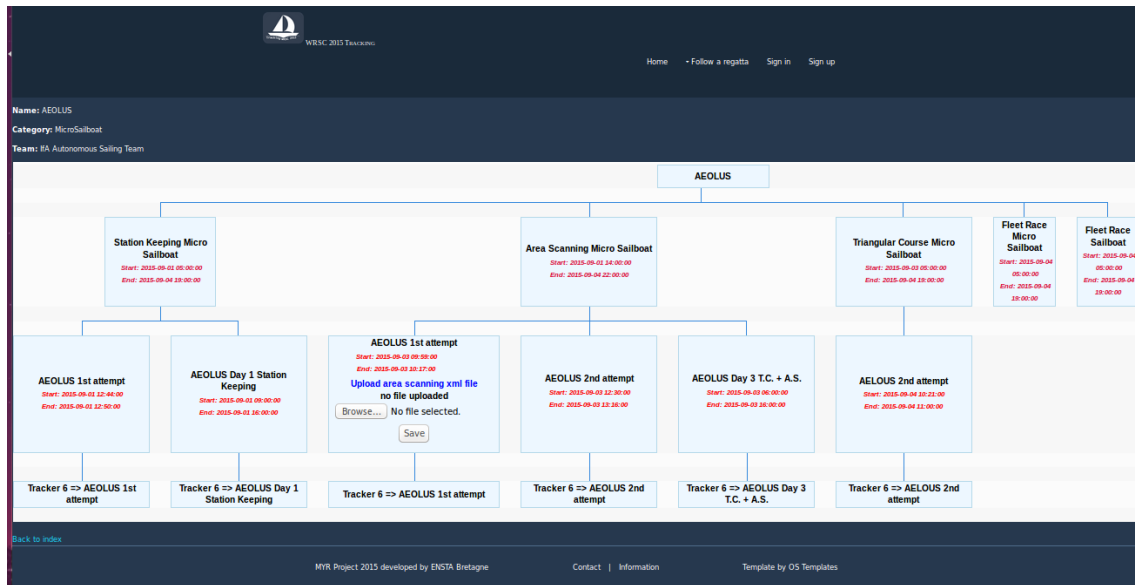

Figure 5: old robot show
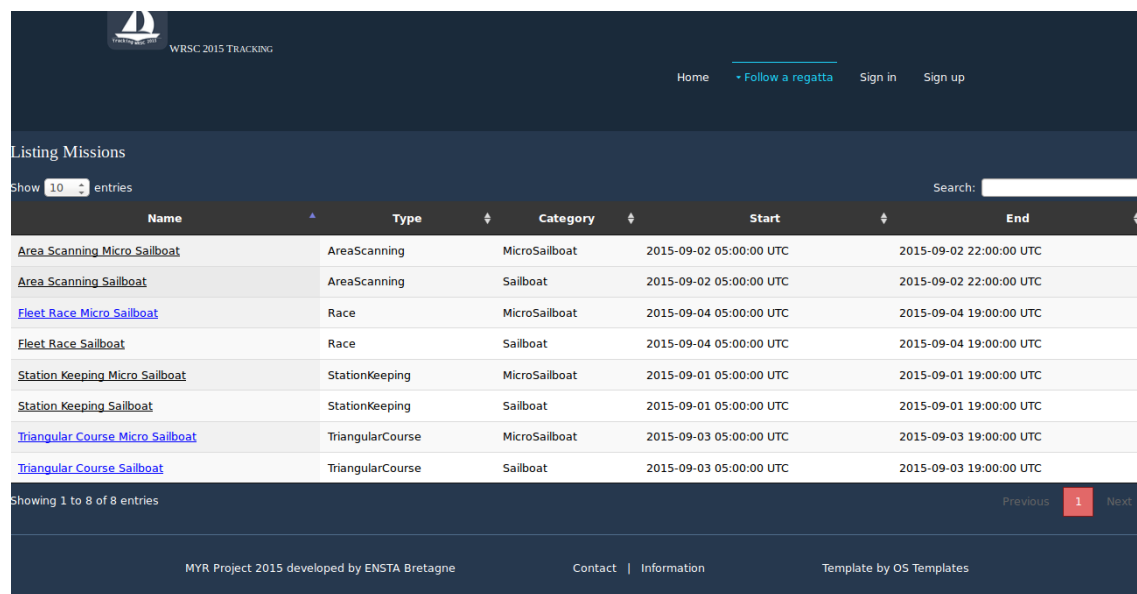
Figure 6: Google org charts example

2.2.2 Sortable & Searchable Table

The previous version web site(WRSC 2014 in Galway) used a table to list all the information for teams, robots, missions, members and etc. And all these tables are only sortable by the alphabetic order. The only sortable table works well when there is not enormous data, if we

Figure 7: WRSC 2014 searchable only table

suppose that there are more than 100 members in our database, and if you want to find the information about one member and you had only the name of this member, it will be stupid to check the whole list just by eyes. Furthermore, if the web site displays more than 10 records in one page, which often makes the user feeling uncomfortable and inefficient. Based on these reasons, we choose a flexible and customisable JavaScript plug-in "DataTable". DataTables is very simple to use as a jQuery plug-in with a huge range of customisable option, and the functions which we are interested most are sorting(which makes the table sortable by alphabetic order), filtering (which makes the table searchable, we can search one element by typing part of the keyword) and pagination (which limits the number of elements in one page, also it will make the page loading be faster and reduce the pressure of server because of the limitation of elements in the page). What's more, "DataTable" provides more APIs which also make the html table flexible, the developers are easy to get the index or element in the table just by calling the functions, comparing to the static html table, the "DataTable" is more dynamic and interacted for users.

Figure 8: DataTable plug-in example

2.2.3 Improved Account Management

1. Signup

   The new sign up page add two new features:

   - Uploading local image file instead of typing a link
     In the old version web site, the user could only typing an image file url to upload the logo. And at the side of server, it adopts a gem called "fastimage", which could handle of the url and get the image of this url, but one limitation is the image size could not larger than several hundred pixels * several hundred pixels, as a matter of fact, this size is quite small comparing to the normal image file which is nearly several mega bytes. So for this reason, the logo field is nearly useless because of the inconvenient.

     Software requirements In the new website, we used a gem called "carrierwave"(an open sources free project, all the codes are available in their Github repertoire), which permit to upload no matter what format file. Furthermore, here we need

7

Figure 9: old sign up

Figure 10: new sign up

to treat the image file, so at the side of rails application, it requires another gem "mini_magick"(Unlike RMagick, MiniMagick is a much thinner wrapper around ImageMagick, that's why we choose MiniMagick rather than RMagick), at the side of server machine, it requires the software "ImageMagick" (under Linux, type "convert

-v" to check if the "ImageMagick" was installed).

Apply to web site Once all the gems are installed successfully, a new folder "uploaders" will be added into Rails.root(the root path of rails application)/app/uploaders, and the creation of uploading action is similar to create a controller/model in rails application, either by the command "rails generate uploader nameoffile", or by create the file manually. And then in the model, we need to mount the uploader, in our case, we need to add "mount_uploader :logo, MemberlogoUploader" in the file Rails.root/app/models/member.rb. After that, the attribute logo of member would only accept a file uploader object, even at the beginning, we define the type of logo is a string. If we dive deep into the database(either look through by rails console, or by SQLite database browser, the attribute logo is filled by a complicated object where it contains the original file name which means the name of uploaded image file, also the name of full name which means the uploaded image file's absolute file name–path in the rails application + file name and other information). Due to this reason, once an image file was uploaded, we should not change the position of this file, because in the database, the image file was just saved as the path and the name rather than the image self.

What's more, "carrierwave" provided some other customizable convenient options, for instance, we can define our own store directory to put all the image files. Actually by default, all the files accessible for any other user should be put into Rails.root/public. But do not trust the file uploaded via internet, so we should make sure all the files in public directory were inexecutable. From this, "carrierwave" also provide an option to set the extension white list, where we can limit the file format to avoid the virus. In addition, if the rails application was launched by a Linux machine, we can set the permission of the directory public to make sure all the users except the owner(admin) could only read and write, but not execute. One way possible to do so is by using the command "chmod 766 <directory>", then with the command "ls -l", we can check if the directory permission is "drwxrw-rw-".

Also we can define different versions of image files with different sizes based on the same image, which will be applied once the image file was uploaded.

- Simple captcha Another important change for the sign up page is that we added the captcha to filtering human user and avoid basic attacks. The realization of captcha is also simple. We used a gem called "simple_capthca2"(an open sources free project, all the codes are also available from Github repertoire). As controller Based, we just need to do: Add the following line in the file "app/controllers/application.rb"

```
ApplicationController < ActionController::Base
  include SimpleCaptcha::ControllerHelpers
end
```

In the view file within the form tags add this code

```
<%= show_simple_captcha %>
```

and in the controller's action authenticate it as

```
if simple_captcha_valid?
  do this
else
  do that
end
```

- Activate by email In the new web site, also we add the function of sending emails to
  activate the new account when one user register on the site. As we do not have any
  specific email account, we choose Gmail account as our admin email account because
  of several reasons.
  - In order to display the google map in our web site, we register a google account
    to obtain a google map JavaScript API key, so already even we did not choose
    Gmail as our main email, we should at least use the google account for JavaScript
    API.
  - Gmail service is free and easy to use, although Google limits the amount of mail
    a user can send, via its portable SMTP server, and the amount of sending emails
    is limited 99 per day, regarding our web site is focused on the participates, even
    maybe there are some other users, 99 is already enough for the email function. In
    addition, if we want much more capacity, if the budget allows, we could buy the
    service of Gmail SMTP server in the future.
  - Gmail is secure, wide used and compatible with others email service provider.
    And sometimes, Gmail is too secure to use. And the problem we meet at the
    beginning is that we had sent an email by configuring the SMTP server in our
    Rails Application, then Google detected this operation was done by machine
    not a web user and our email was blocked. In order to solve these types of
    problems, we need to configure our Gmail to less secure status by going to this
    page https://www.google.com/settings/security/lesssecureapps and changing to
    less secure.

When the Gmail account was setted correctly, and also we can send emails to other
users, another problem came to our views. How can we send a unique link to the user
and also how can we identify this link is associated with the relative user ? Actually
before create a user, we create a new activate token(which is used to send to the user)
and a activate digest(which is saved into database and used to identify the relative
user). In the file Rails.root/app/models/member.rb

```
before_create :create_activation_digest
def create_activation_digest
  self.activation_token = Member.new_token
  self.activation_digest = Member.digest(activation_token)
end
```

In fact, all the passwords, or more generally, the important information are encrypted
before saving into the database by using the popular gem "Bcrypt". Obviously the
security of encryption depends on the longer of generated token(Here we used 64 bits
to generate a random token, normally it is long enough) and the algorithm(In this case,
it is "Eksblowfish"–"expensive key schedule Blowfish", which based on the algorithm
Blowfish. Cryptotheoretically, this is no stronger than the standard Blowfish key
schedule). Also we need to pass the user's email address as an argument in the url,
then we can identify the token with the member which was saved in the database
found by the email address. In the file Rails.root/app/models/member.rb

```
def Member.digest(string)
    cost = ActiveModel::SecurePassword.min_cost ?
        BCrypt::Engine::MIN_COST : BCrypt::Engine.cost
  BCrypt::Password.create(string, cost: cost)
end

def Member.new_token
    SecureRandom.urlsafe_base64
```

```
              end
```

Then the creation of email sending is just like any other page, firstly create the controller with desired actions, and then create the views which would be sent to the user later.

2. Sign in

   In the new sign in page, we had also added two new features:

   - Remember the User The function of "Remember me" is quite similar to sending the activation link to users. If the user choose to remember in the computer, then the server would write the user id and a remember token into cookies, with the function cookies.signed which provided by rails application, the contents in the cookies would be encrypted. Furthermore, something should be noticed:
     - Cookies exist in different browsers in the same computer, so if the user has more than one browser (for instance: one Firefox and one Chrome), the cookies will be shared between the two browsers.
     - The contents in the cookies contains two values, one is what we want to save into the cookie, and the other one is the expire date which indicates the validate time for cookie. If we use "session", which contains the same contents as cookies, but the expire date is from when the contents were created to when the user close the browser.
   - Reset the Password "Reset the password" has the similar principle as the activation email sending. When a user request to reset its password, it needs to provide its email address as a reference. Then the server generate a token and a digest, send the link which contains the user's email address and a reset token to the user. After, when the user clicked the link, the server verifies the digest with the token, if it is the correct token, then the user could change its information.
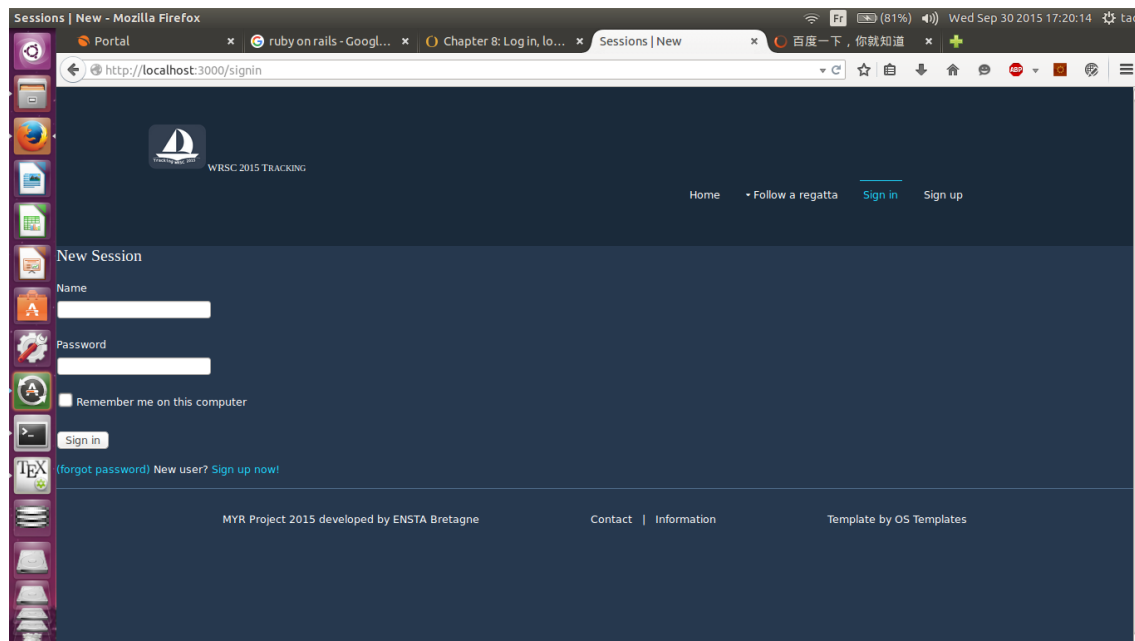

Figure 11: old sign in

Figure 12: new sign in

2.2.4 Admin Markers Creation

2.2.5 Auto Scoring and Ranking

1. Scoring

   The WRSC contains 4 missions: Station Keeping, Area Scanning, Triangular Course and Fleet Race. And I was in charge of the scoring for Station Keeping and Area Scanning. So I will introduce the algorithms I had chosen to calculate the score.

   - Station Keeping
     - Description of mission As the WRSC rules described, the objective of the station keeping contest is to evaluate controlled sailing in a limited region with time constraints. Each boat must start outside a 45 m x 45 m box, marked by four buoys. In the team slot time, the boat must enter the box. The boat shall leave the box at a time as close as possible to 5 minutes after entering the box. Boats fulfilling the entry and exit criteria are awarded points based on the following formula: $P = \max(0, 10 - |T_{enter} + 300 - T_{leave}|/10)$ Where: T_{enter} is the timestamp when the boat first enters the box and T_{leave} is the timestamp when the boat first leaves the box. If the boat enters and leaves the box several times, any leave timestamps within ten seconds from T_{enter} are ignored. All timestamps are given in seconds.

   - Area Scanning
     - Description of mission The course will be divided into 10x10 square sections, each with a width of about 31 m. The sections will be indexed as (i, j), where i goes from 0 to 9 starting at south and going north, and j goes from 0 to 9 starting at west and going east. From the time the boat enters the first section and within a time of 90 minutes, the boat shall pass through as many square sections as possible. The score for the area scanning contest will be calculated as: $P = N_{passed}/10$

12

where Npassed is the number of entered sections. The score will be multiplied with a factor determined from oceanographic measurements provided for each visited section. Depth (m), water temperature (℃), air temperature (℃), water salinity (%), conductivity (S/cm), chlorophyll (ug/l), ammonium (mg/l), nitrate (mg/l), chloride (mg/l) and total dissolved solids (mg/l) can be provided. Inclusion of depth data will add 0.2 to the multiplication factor, the other measurements each add 0.1. The multiplication factor is 1.0 initially, and the maximum that can be achieved is 1.5. Each team shall provide data on the team's performance to the Race Committee within 5 hours after the start of their slot time. The data shall be in XML, specified by the schema below. The data will contain the team name. For each entered square section the section's indices are required along with a UTC timestamp and corresponding coordinates using <gml:pos> property in WGS-84 (EPSG:4326). Optionally the oceanographic data can be added. Each entered section shall be added only once to the XML data.

– algorithm for Area Scanning Actually, for the contest area scanning, the Race Committee decided to do the scoring by themselves, but they ask us to verify the GPS data submitted by the teams with XML format with data gathered by our trackers. Fortunately, Åland University of Applied Sciences(ÅUAS) had a python program which can analyse the submitted XML file and then generate a JSON file(which contains all the useful data and this file is easy to be handled by ruby on rails). And then I based on this JSON file, compare the data with the correspondent tracker's data to check if the gaps of positions were under the tolerance. In order to accomplish this task, several steps were adopted:

(a) The participated team should uploading their XML file to their robot's page (robot#IdOfTheirRobot). And also I added a time stamp to record the time when the team submitted their XML file because the team should upload their XML file in 5 hours since they had finished their attempts.

(b) With python program(provided by Conny Ljunggren, ÅUAS), I need to generate a JSON file manually. Also I had looked into the python program, in fact the python program used a module called "xml.etree.ElementTree", the key point of this program is using this module to analyse XML file. Unfortunately, I did not find any similar module in Ruby to analyse the XML file, but it was also feasible in ruby. It was just like a compilation project, the code source is XML file, and then the target file is JSON file, so then if we define the grammar of XML, define the lexer and Backus Normal Form(BNF), from the Backus Normal Form(BNF), create the parser to analyse the code source, and store all the separate data in the node of Abstract Syntax Tree(AST), then either by using the visitor pattern, or mix erb with visitor pattern, we can generate the target code finally. Since we did not have so much more time, I did not realize this function.

(c) Based on the JSON file, ruby could import the JSON file directly, then I can compare the GPS data to the data in our web site and calculate the difference between the two sets of data. Because all the unities of GPS data are in degree, which are incomprehensible for humans, then I transfer the unity in degree to the unity in meter by using the algorithm proposed by this site: http://www.movable-type.co.uk/scripts/latlong.html
And also I had tried the official contest's buoys positions. In fact, the Race Committee declared officially the distance between two buoys is about 31m [(60.1050,19.9500);(60.1080,19.9500)], and from the formula, I obtain the distance is a little more than 30m, so the gap is less than 1m which is acceptable by the GPS precision.

Figure 13: latitude longitude to meter formula from Internet

(d) After the comparison, in ruby there are three possibilities(JSON/YAML/XML) to display the result, finally I choose XML not only to keep coherent with what they submitted, but also easy to read for users. As a matter of fact, files are also like any other resource in the web site(like views), according to the path in the rails application, we can visit the file by typing the url Rails.root/FilePath.