Ensta Bretagne
2, rue François Verny
29806 Brest cedex
France
Tel +33 (0)2 98 34 88 00
www.ensta-bretagne.fr

Rapport

CI 2016

October 4, 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 developped 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 these reasons, 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 APIs for our electronic cards, which means we had already defined the APIs for gathering GPS data including latitude, longitude, date time, course and speed, but the antennae in the previous version 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 needed to choose appropriate antennae, and we needed to prepare all the materials for the tracking system before the final competition(like batteries, 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 was still unknown, to make sure that we got enough trackers, we booked 10 for each module.

For software part, the previous version of website based on the web framework Ruby On Rails and the use of google map (JavaScript APIs of google map were provided and published on the Internet by Google Company freely.) And the performance of the previous version web site in Galway (Ireland, WRSC 2014) proved that the website was built successfully. However, the old CSS and HTML were obsolete and the functionality of the web site was incomplete. Additionally, the web site had already the capacity to display the GPS data gathered by the tracker on the

google map, but the web site could not handle of these information further until this moment. So far, several domains remain to be improved for the web site:

- 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 Organization

In fact, Sylvain Hunault and I needed to prepare the web site and trackers in two months. Since Sylvain Hunault knows much more about electronic cards, then he was in charge of the hardware part, including all the antennae, batteries and electronic cards, and I was familiar with ruby on rails because of my experience, so I focused on the software part. Since the web site was much more problematic and needed to be enhanced further, after preparing the hardware part, we two both worked into the web site. Normally we worked separately according to the distributed functions which are necessary to be added into the site. And also we worked together when we needed to integrate all the unitary functions, or when we were in front of some problems. We shared our knowledge and information obtained, and discussed together. In addition, all the useful information and our source code were put into the Github (a tool of the distributed revision control and source code management) repertoire.

Actually, the development of website was function-oriented, the branch "master" was the principal branch which would be released to the public, besides every one or two weeks, a new branch was created, which means a new feature was released. Before merging every new branch into the "master" branch, all unitary tests and integrate tests should be passed.

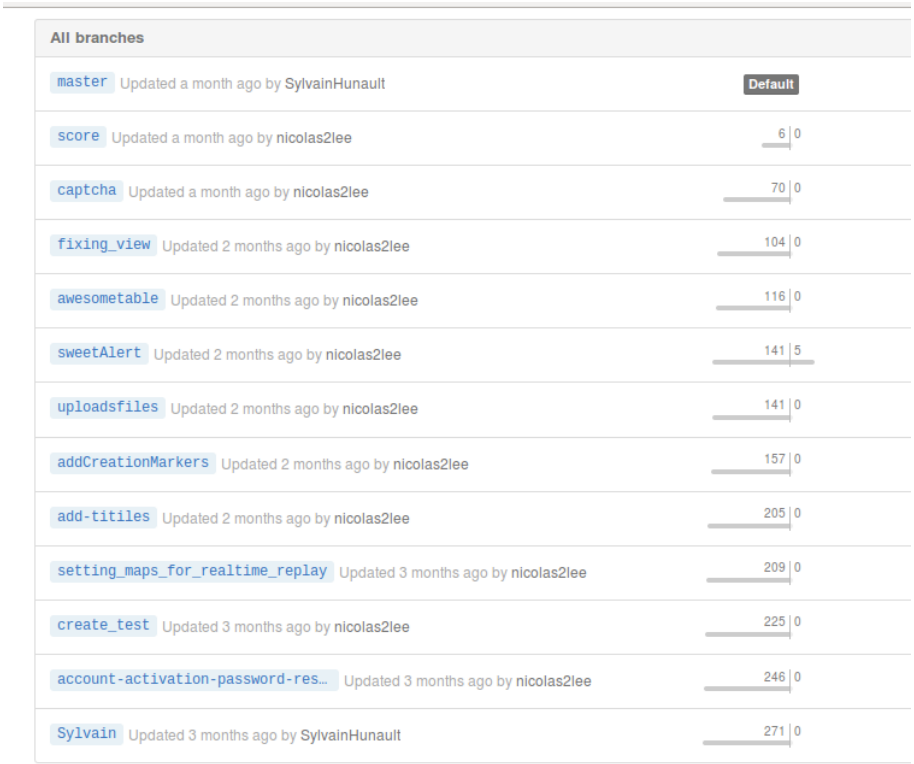


Figure 1: All function-oriented branches

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.

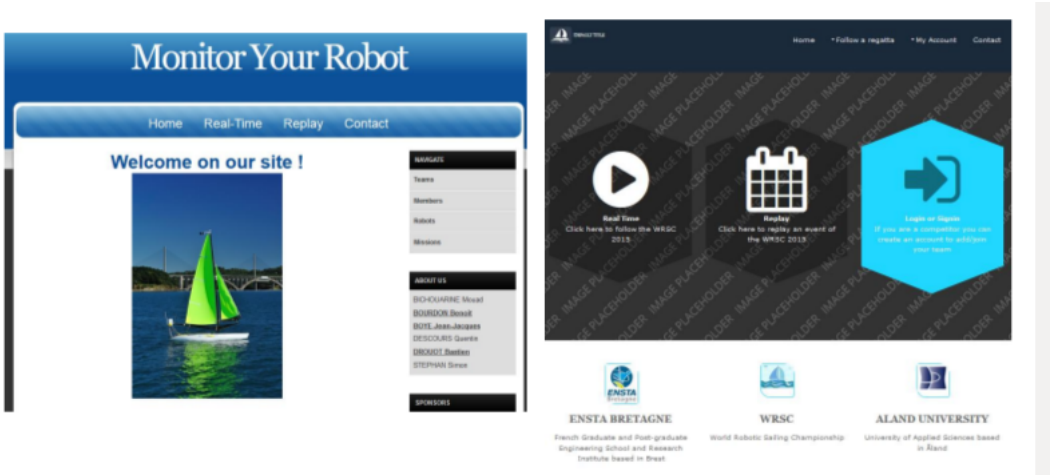


Figure 2: Old home page, WRSC2014(left), MYR2015(right)

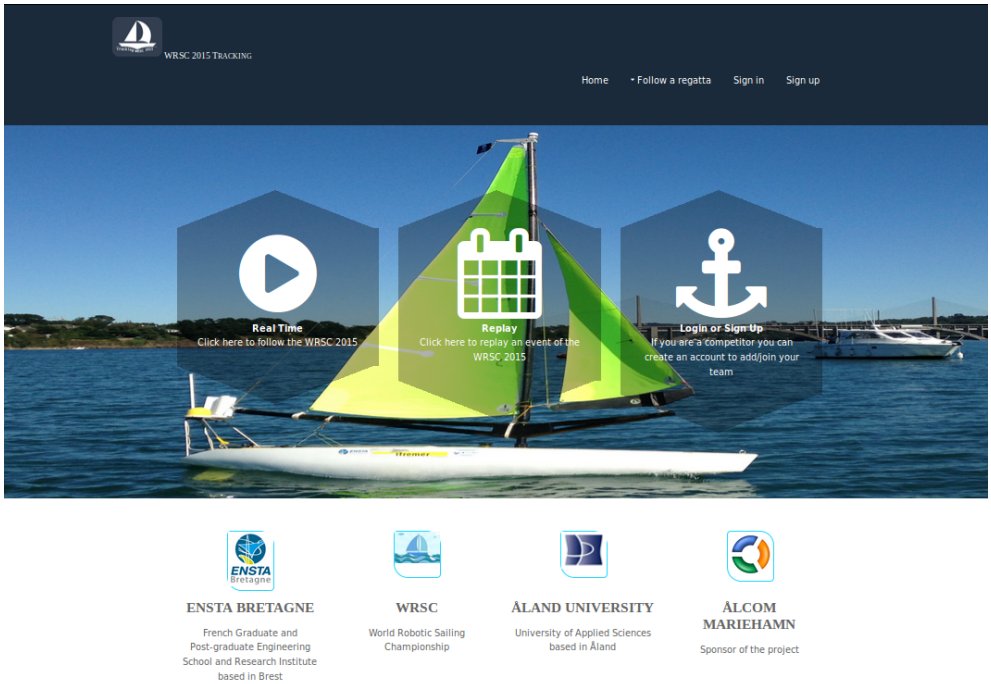


Figure 3: New Home Page

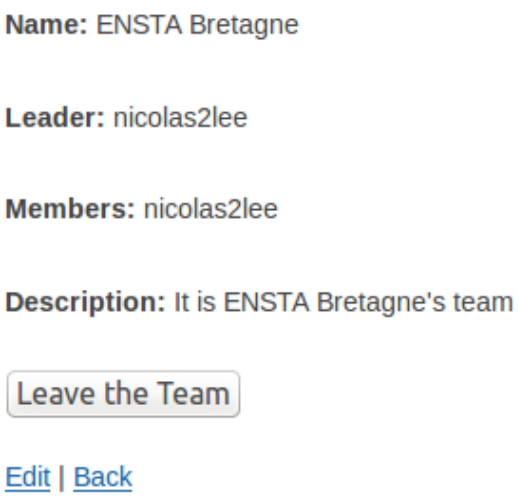


Figure 4: old team show

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 of pages and the use for users, we add two fields to show the team's members and robots by loading with two Ajax(asynchronous JavaScript and XML) requests to permit the easy management for members and robots.

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 users, who are not familiar with our web site. For example, they do not know if they had registered to participate a mission (if they want to know, they need to go to the page attempts, and check if their robot 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 unique attempt. So based on this hierarchical order, it will be better to choose a tree structure chart to show the logic. In order to meet our need, we had chosen google org(organization) charts which are open source and all APIs are provided and published by Google company freely. 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 loads the robot#show page, it will recall an Ajax request and send all the associated information from the ruby part to the JavaScript part, then in the JavaScript part, we could use the org charts' APIs directly.

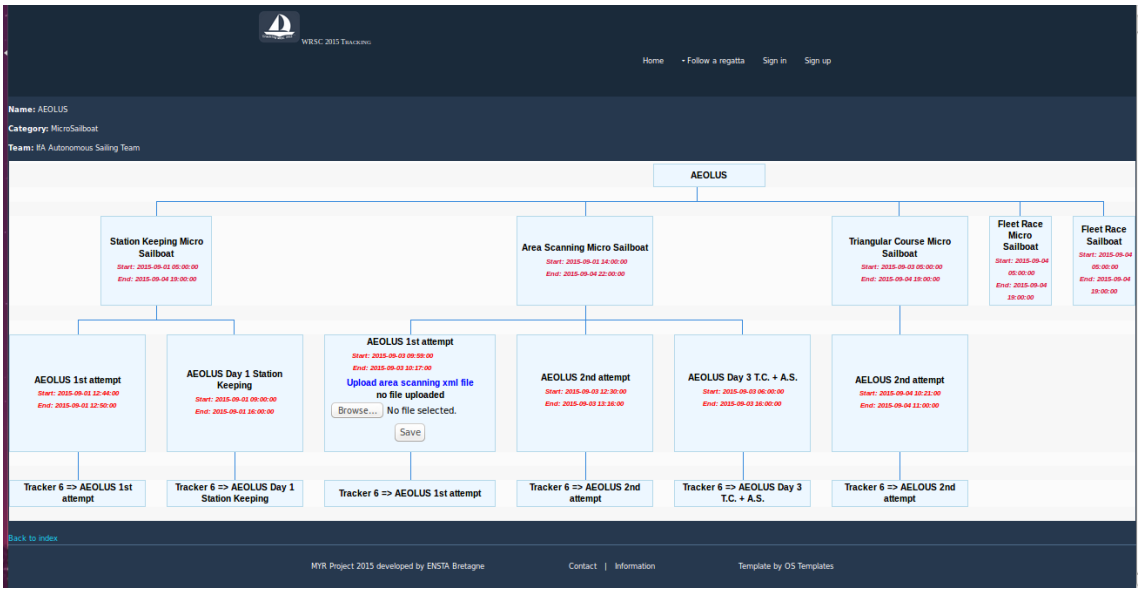


Figure 5: Google org charts example

2.2.2 Sortable & Searchable Table

The previous version web site(WRSC 2014 in Galway) used the form of table to list all the information for teams, robots, missions, members and etc. And all these tables were only sortable by the alphabetic order. The only sortable table works well when there is not enormous data, if we 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 feel uncomfortable and inefficient. Based on these reasons, we choose a flexible and customizable JavaScript plug-in "DataTable". DataTable is very simple to use as a jQuery plug-in with a huge range of customizable options, and the functions which we are interested in 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.

2.2.3 Improved Account Management

1. **Sign UP**

   The new sign up page add three 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 from 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.

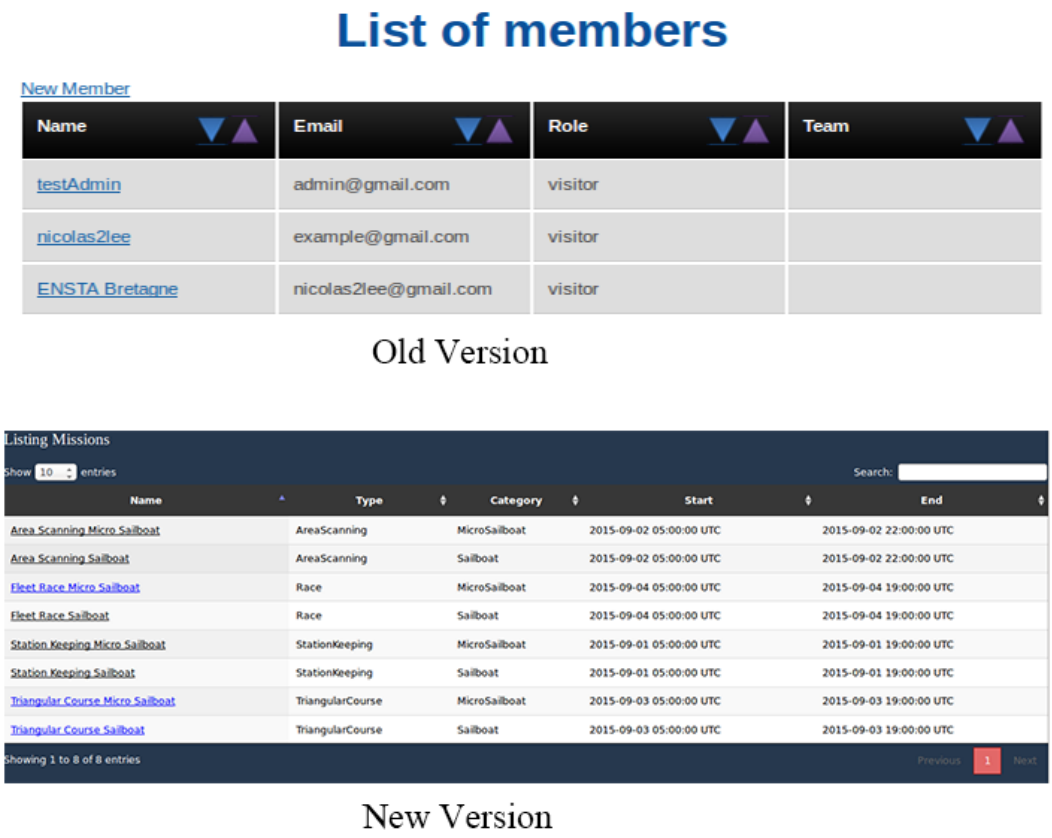


Figure 6: Comparison of two type tables, UP is the previous version, DOWN is the current version

In order to solve this problem, in the new web site, we used a gem called "carrierwave"(an open source free project, all the codes are available in their Github repertoire), which permits to upload whatever format file. Furthermore, here we need 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).

Once all the gems are installed successfully, a new folder "uploaders" will be added into Rails.root(the root path of rails application)/app, 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,

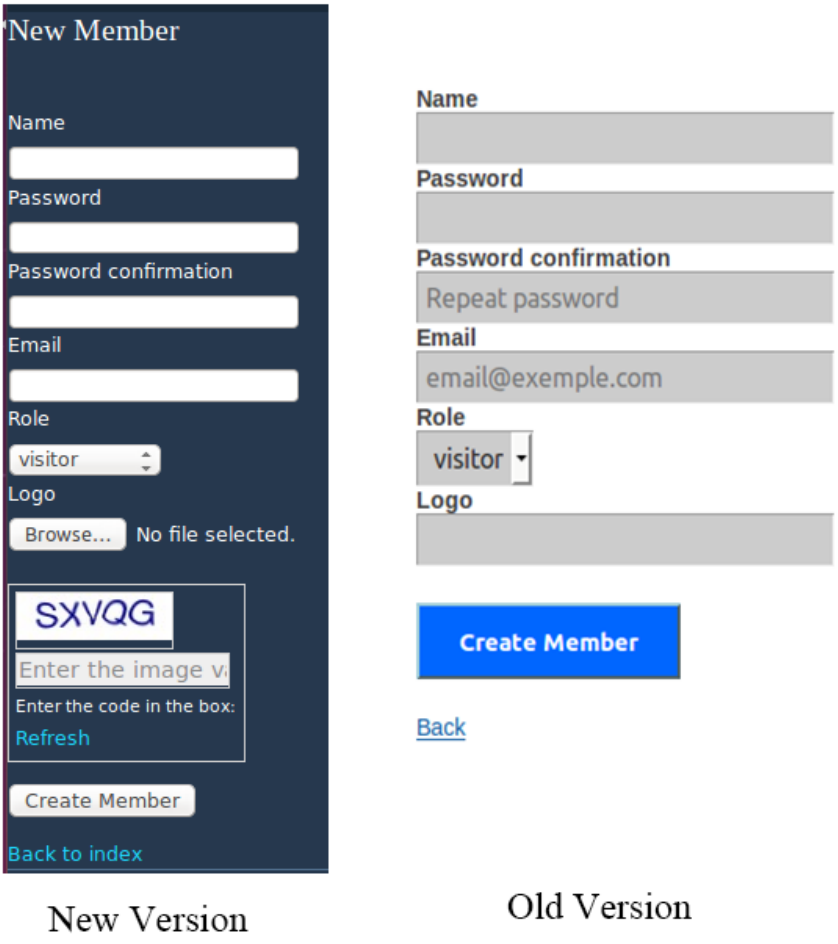


Figure 7: Comparison of page Sign Up

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. Also do not trust the file uploaded via internet, 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 initial image, which will be applied once the image file is 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, we also added the function of sending emails to activate the new account when one user registers on the site. As we did not have any specific email account, we choosed 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 focuses on the participators, 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 creating 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 associated 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:

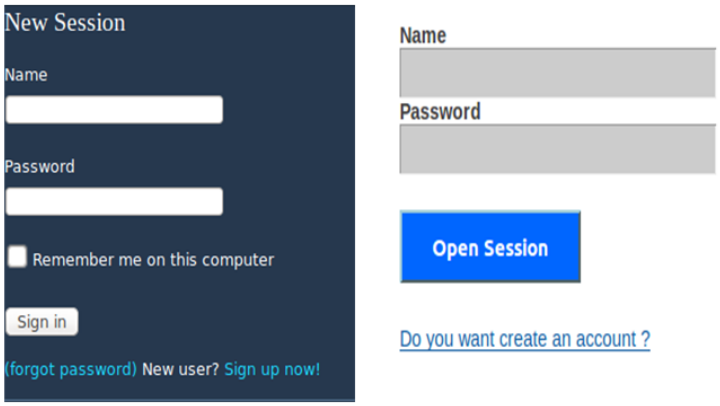


Figure 8: Comparison of page Sign In

  - 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 the 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.

2.2.4 Admin Markers Creation

- Markers

  The pages of real-time and replay worked well last year, but the markers(the buoys) were not added to these pages, also the web site did not display any marker's information, which was inconvenient for the observers following the traces of robots. Inspired by these objectives, we designed to add an administration tool to add markers to a specific mission. Based on the initial designed schema by Bastien DROUOT and Benoit BOURDON, finally we obtained the new function of adding markers for administrators.

  The realization of creation of markers based on the Google map JavaScript APIs provided by Google company(https://developers.google.com/maps/documentation/javascript/markers). Actually, the JavaScript APIs provided the possibilities to add point, line, polygon and circle into the google map. And for each type marker, there are two possibilities to draw on the map: either draw the marker with the given coordinates, so then the marker is fixed on the map and can not be modified later, or draw the marker by clicking on the map directly, then it is up to the developer to add the options to drag or to move the marker or even change the size of marker dynamically. As a matter of fact, the last version web

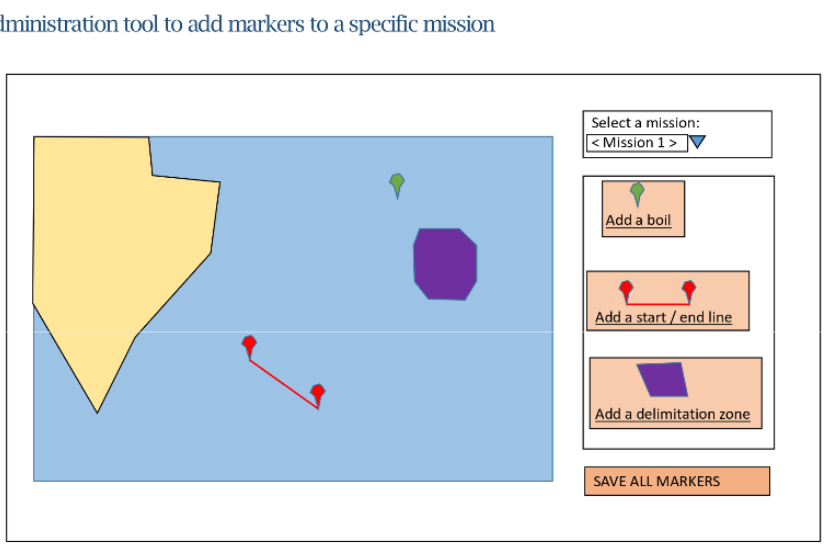


Figure 9: Designed admin markers

site could display some markers in the robots' path by using a gem called "gmaps4rails" which is easy to be applied into rails application, but since this gem also based on the Google map JavaScript APIs, in addition the gem changed and limited some options from Google APIs, so we could not use all the functions provided by Google APIs via the gem. In order to get more freedom, we had abandoned the gem, furthermore, the google APIs provided more customizable options(for example, we can change the icon of marker, add some animations for displaying the marker, the possibility to hide the marker instead of deleting the whole marker and also the adapt zoom for all the markers). Thanks to the project noun, all the icons we used in the admin marker creation were dragged from this project.(https://thenounproject.com/)

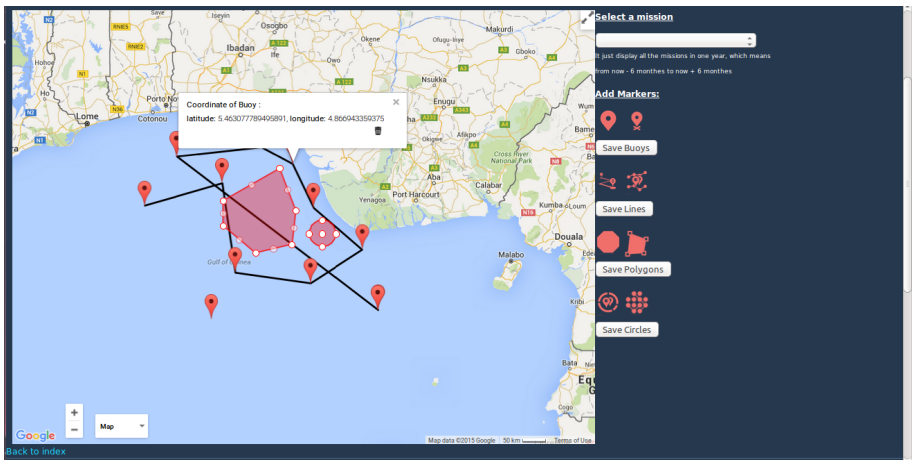


Figure 10: Admin Markers final graphic interface

- **InfoWindow**

Moreover, Google map JavaScript APIs offer the possibilities to add some events to the marker or even some events for the mouse motion. Among all these options, what we are interested in most is the possibility to display the infowindow for every marker to specify the accurate and precise coordinates of the robot's position. In fact, there are two ways to achieve the requirements.

- **Google API**

  As a matter of fact, Google APIs provide the possibility to show the coordinates when someone clicks on the map directly, because google map catches the mouse action which are behavioured on the google map and then print the coordinates directly, but the limitation is that google map could only displays the coordinates of markers, if the user want to know much more information about the team(like: team name, the id of attempt and the tracker id), Google APIs could not meet all these demands. One advantage is that using google APIs does not need to communicate with the server, so the response will be faster and it will reduce the pressure of server.

- **Ajax request to communicate with server**

  Obviously, communicating with server, we could get all the information we want, but the problem is that we need to send the Ajax requests, at least on Ajax request, which will cost much more time, especially in most cases, there are hundreds and thousands data to display, even if the number of markers is 10 times less than the number of coordinates, we could imagine that 1000 positions need to display, then 1/10 of positions(which means 100) need to be shown with the infowindow, also it means 100 Ajax requests, if every Ajax request lasts 30ms, then 100 Ajax requests last 3s, it means one user click a button one time, it will cost 3s to calculate for the server, when there are more than 10 users do the same things at the same time, the server would be broken absolutely.

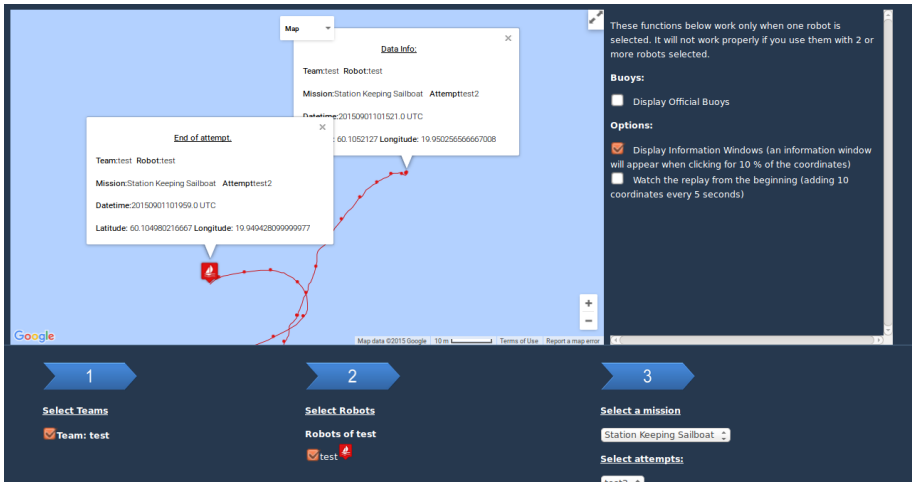


Figure 11: Admin Markers with Ajax request

So actually, there is not any better way to do it, the different methods meet different requirements, finally we decided to use the second way to show the information, because we need to display the team's information, but considering the performance of server, we had displayed 1 markers per 10 coordinates to limit the Ajax request number.

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 - |Tenter + 300 - Tleave|/10)
       Where: Tenter is the timestamp when the boat first enters the box and
           Tleave 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 Tenter are ignored. All timestamps are given in seconds.
     - **algorithm for Station Keeping**
       In general, the aim of station keeping is letting the boat stay 300s exactly in a square(ideally), or more generally stay in a polygon. So the key point of calculating the score is checking whether the position of boat is in the polygon.

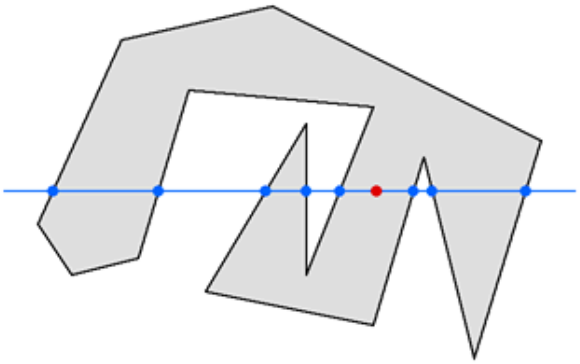


Figure 12: Point in Polygon algorithm example

Based on the Point-in-Polygon Algorithm(http://alienryderflex.com/polygon/), here is an example, the solution is to compare each side of the polygon to the Y (vertical) coordinate of the test point, and compile a list of nodes, where each node is a point where one side crosses the Y threshold of the test point. In this example, eight sides of the polygon cross the Y threshold, while the other six sides do not. Then, if there are an odd number of nodes on each side of the test point, then it is inside the polygon; if there are an even number of nodes on each side of the test point, then it is outside the polygon. In our example, there are five nodes

to the left of the test point, and three nodes to the right. Since five and three are odd numbers, our test point is inside the polygon. Then with more examples:

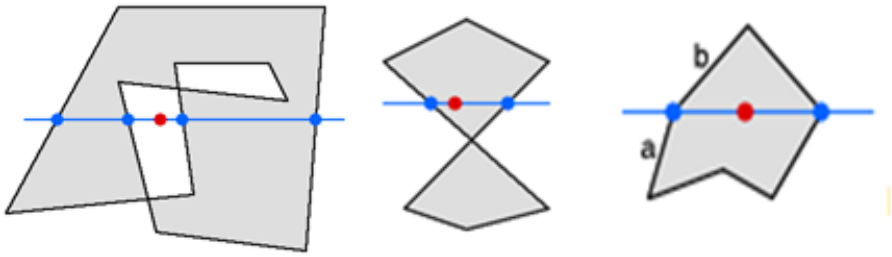


Figure 13: More examples for Point in Polygon algorithm

- **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 = Npassed/10
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 area scanning contest, the Race Committee decided to do the scoring by themselves, but they ask us to verify the submitted GPS data with XML format from participators with our trackers' data. 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). And then departure from this JSON file, compare the JSON data to the correspondent tracker's data to check if the gaps of positions were under certain tolerance. In order to accomplish this task, several steps were adopted:

(a) **Uploading XML file by participators**
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) **Uploading JSON file by admin**
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" to analyse the 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 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 template written by "erb" with visitor pattern, we can generate the target code finally. Since we did not have enough time, we did not realize this function.

(c) **Calculate the distance** Based on the JSON file, ruby could import the

Distance

This uses the '**haversine**' formula to calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface – giving an 'as-the-crow-flies' distance between the points (ignoring any hills they fly over, of course!).

Haversine formula:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$
$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{(1-a)})$$
$$d = R \cdot c$$

where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km);*
note that angles need to be in radians to pass to trig functions!

JavaScript:

```
var R = 6371000; // metres
var φ1 = lat1.toRadians();
var φ2 = lat2.toRadians();
var Δφ = (lat2-lat1).toRadians();
var Δλ = (lon2-lon1).toRadians();

var a = Math.sin(Δφ/2) * Math.sin(Δφ/2) +
        Math.cos(φ1) * Math.cos(φ2) *
        Math.sin(Δλ/2) * Math.sin(Δλ/2);
var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

var d = R * c;
```

Figure 14: Latitude longitude to meter formula from Internet

JSON file directly, then we 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 we transfer the unity in degree to meter by using the algorithm from internet(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.

(d) **Generate the comparison file**

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 the participators' submitted file format, but also it is 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.

2. **Ranking**

- **Improved Model**

  The previous version do not contain the function of scoring and ranking, so the old model do not satisfy the new need. Then we proposed a new model, the principle is every attempt has a unique score, every robot has many attempts, equally every robot has many scores, then we just need to take the best score into account for every robot, at last we compare all the best scores for all the robots. So we add a new table Score into the model and add some new attributes to the table Robot.

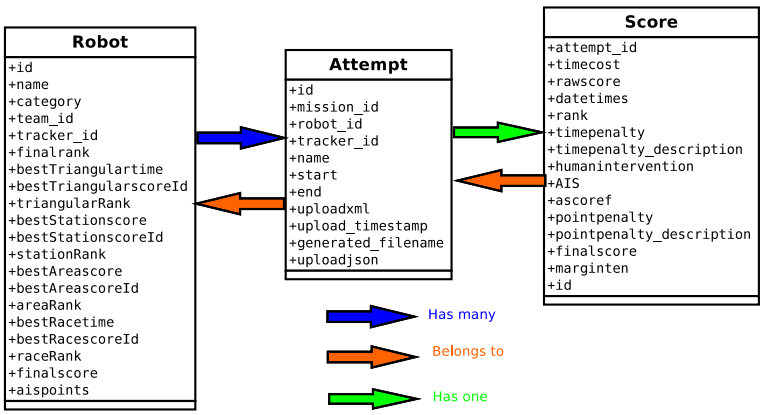


Figure 15: Add new table Score to the database

- **processes of ranking**

  Actually, there are 4 missions for the whole championship, and every mission has different rules and penalties, but the processes of every mission works similarly, when a team finished an attempt, we should create a score for this attempt, from the algorithm we used, we could get the time cost or a raw score differed from missions. Here I take an example of triangular course to illustrate the process to obtain the best score, where Hi means human intervention:

  After obtaining the best score for every team, we could calculate the rank. Here something should be noticed, the AIS points are not taken into account in every single mission, but in the final score, we should add the AIS points.

Based on all the new features, finally we improved our database which is more complete.

2.3 Settings in the virtual machine

After we had finished our development of web site, we need to put our site into the internet. Thanks to Ålcom Mariehamn(A telecommunication company in Mariehamn, Finland) who sponsored a virtual machine to run our server. Initially the server was inaccessible from outside, which means the server do not allow the user from internet to connect to this server. So in order to put our server on the internet, we need to configure the firewall at first. By default, rails

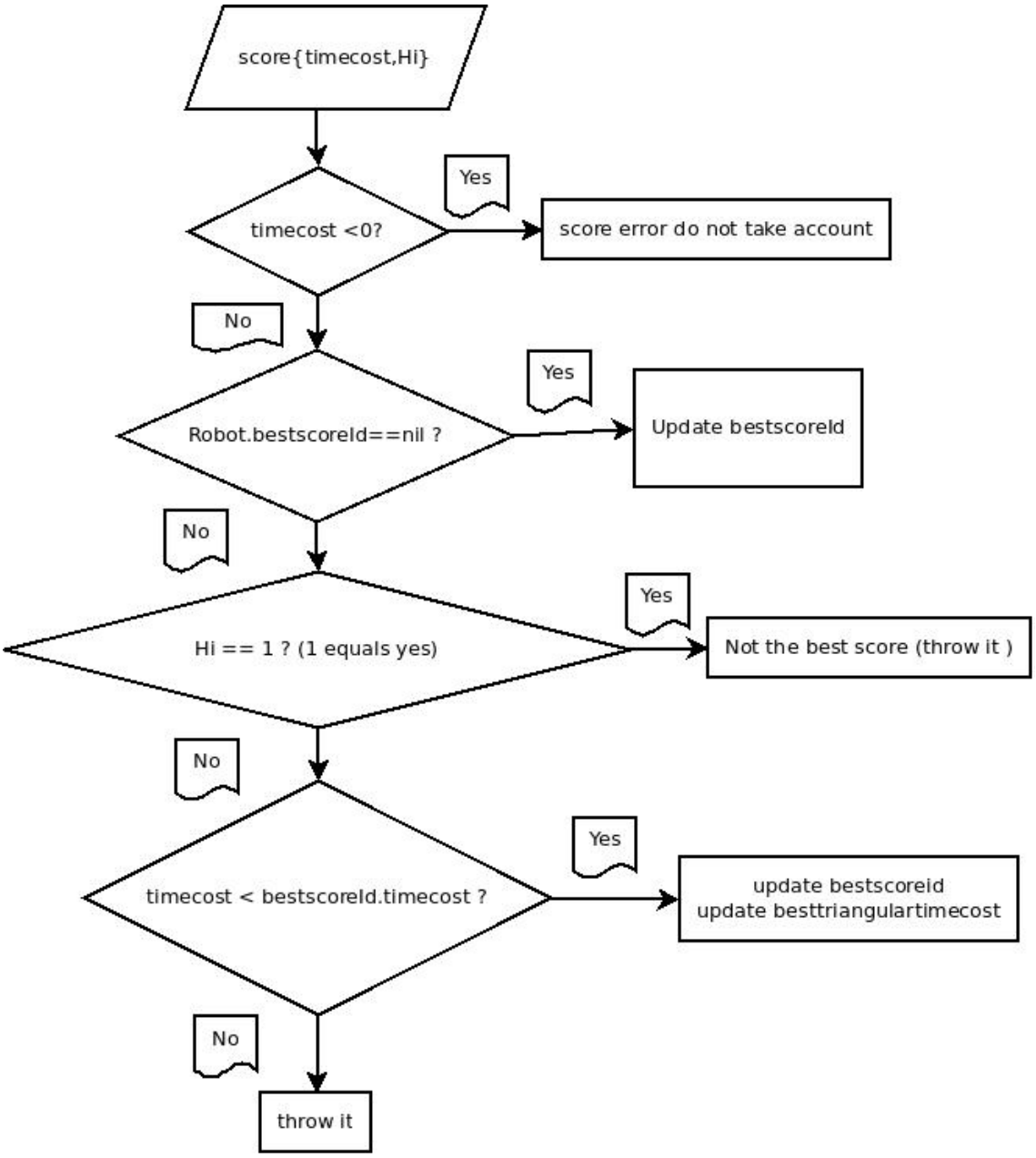


Figure 16: Triangular Course best score chosen process

application uses the port 3000 to run the server, but then the url will be terminated with :3000, which is not the familiar convention for the most users, so we choose the port 80 rather than the port 3000 to offer our service by using "iptables", since it was a Linux machine.

With the command:

```
sudo iptables -I INPUT -p tcp --dport 80 -j ACCEPT
```

we enable the visit outside to the port 80.

After setting the secret_key_base into the environmental variable in the virtual machine, we

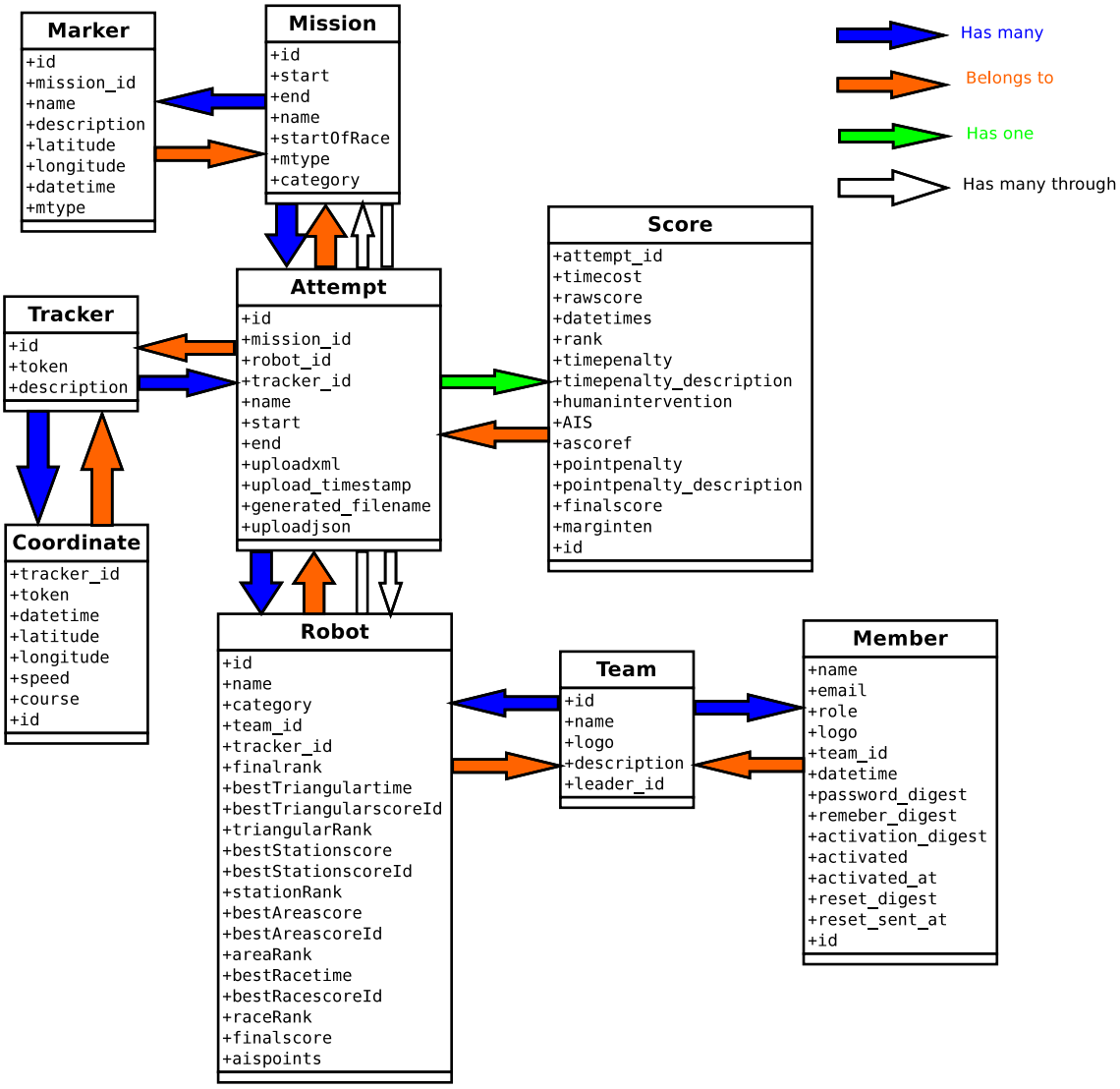


Figure 17: The final Database for WRSC2015

launched the machine with production mode(which is light and safer than development and test mode), our server was well prepared for the service.

2.4 Result for Tracking System and WRSC

Fortunately, our web site works quite well, all the functions we added perform as we expected.

1. **Email**

   Our admin Gmail account works well, it can send the activation emails and reset password emails perfectly, all the users received their emails.

2. **Tracking system**

   Here are some captures of missions: As the figures demonstrated, our web site could indicate the whole trace of all robots clearly, the tracking system works perfectly.

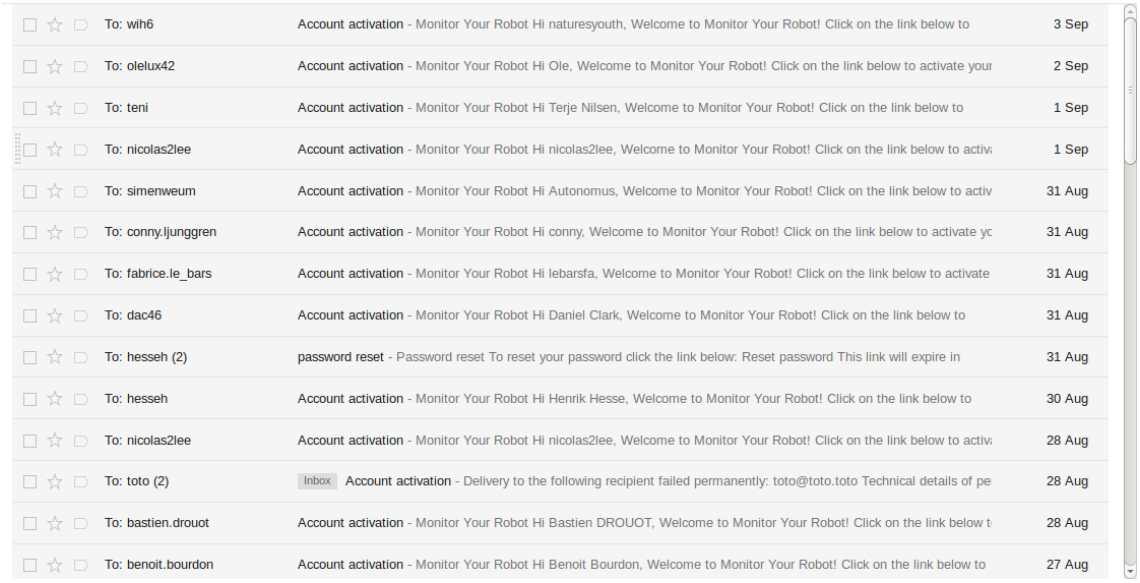


Figure 18: Admin Gmail sending records

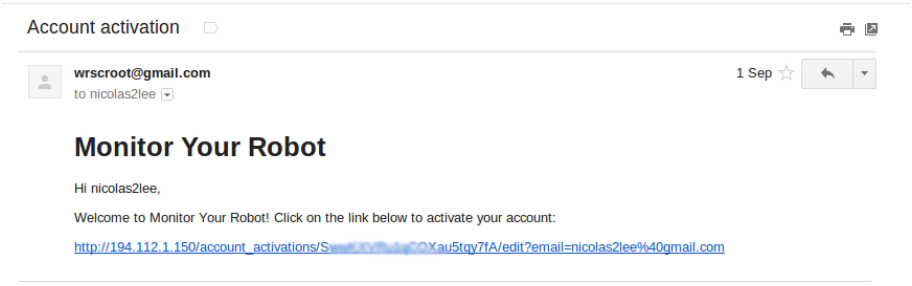
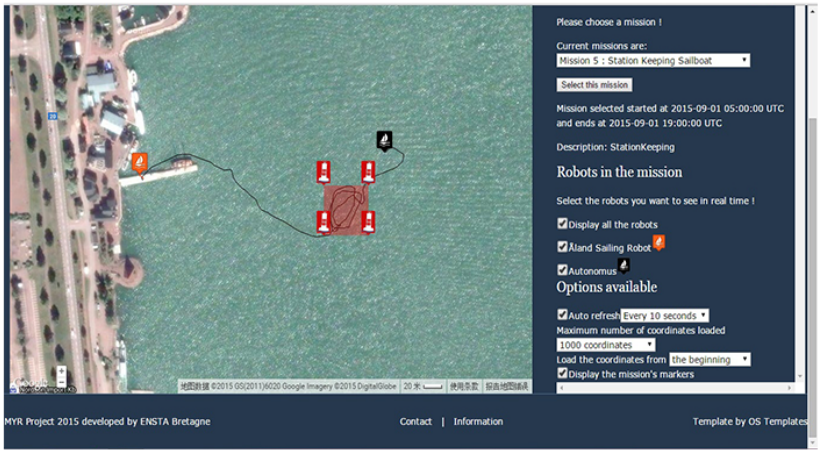


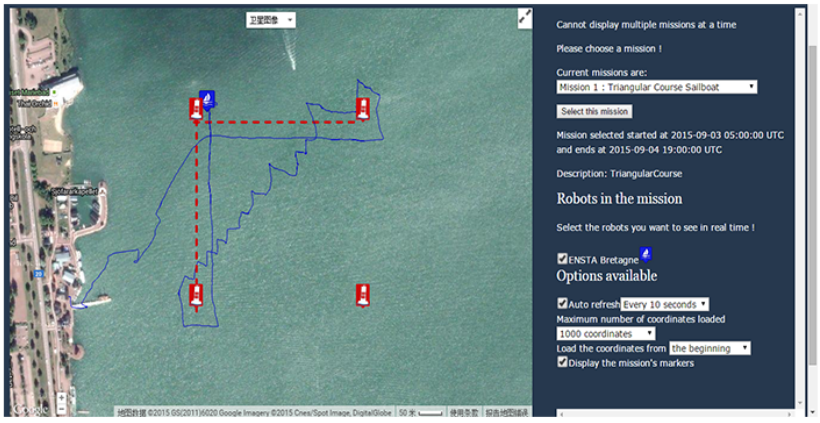
Figure 19: An example of activating email content

3. **Scoring and Ranking**

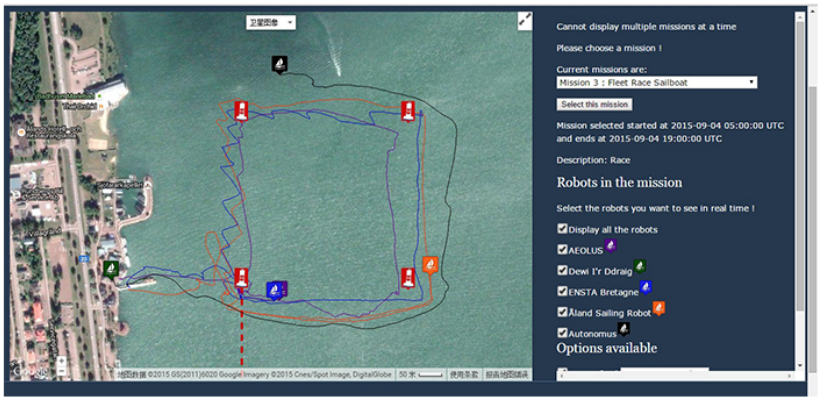
   Before this year, all the scores were given by the Race committee and were measured by humans, but this year, with our tracking system and our algorithms, we can obtain more objective scores, so after discussing with the Race committee, the final result based on our calculations. Here is a quick view of the final standing: Also the uploading file function works well, and based on the submitted data and our trackers' data, here is a hint about the comparison of these two sets of data, finally the maximum difference is about 8 meters which is acceptable.



Station Keeping



Triangular Course



Fleet Race

Figure 20: Captures of WRSC mission in the real-time page

WRSC 2015 FINAL STANDING

Category: Sailboat

Rank	Team	Robot	Station keeping	Area scanning	Triangular course	Fleet race	AIS Points	Final Score
1	Åland Sailingrobots	Åland Sailing Robot	8.10 319	9.68 0	10.00 1488	16.00 2829	0	43.78
2	ENSTA Bretagne	ENSTA Bretagne	9.20 292	10.00 0	9.00 1949	15.00 3587	0	43.2
3	Autonomus	Autonomus	8.20 282	?	?	2.00 99999	0	10.2

Category: MicroSailboat

Rank	Team	Robot	Station keeping	Area scanning	Triangular course	Fleet race	AIS Points	Final Score
1	IfA Autonomous Sailing Team	AEOLUS	9.00 310	9.90 0	10.00 209	16.00 3416	0	44.9
2	Aber Sailbot	Dewi I'r Ddraig	?	?	?	?	0	0.0

Figure 21: Final Standing of WRSC2015

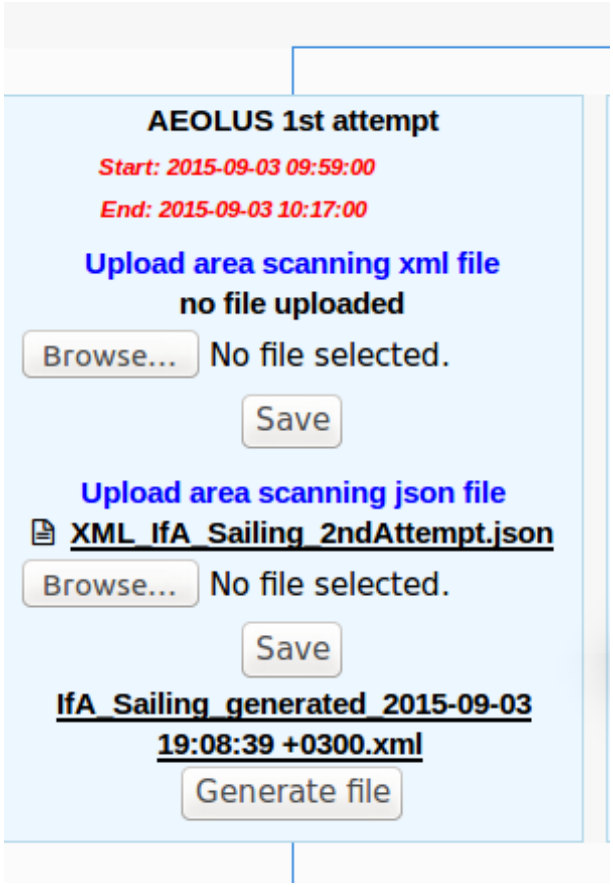


Figure 22: Uploading file example


```
    </datum>
  −<datum>
      <sectioni>9</sectioni>
      <sectionj>8</sectionj>
    −<geo-position-checking type="array">
      −<geo-position-checking>
          <coord-test>20150903150217.0</coord-test>
          <coord-test-id type="integer">186063</coord-test-id>
          <coord-latitude>60.107395083333</coord-latitude>
          <coord-longitude>19.955701366667</coord-longitude>
          <datetime>20150903150217</datetime>
          <difference-from-tracker-to-team type="float">5.493222813555402</difference-from-tracker-to-team>
        </geo-position-checking>
      </geo-position-checking>
      <Section-Max-difference-from-tracker-to-team type="float">5.493222813555402</Section-Max-difference-from-tracker-to-team>
    </datum>
  −<datum>
      <sectioni>9</sectioni>
      <sectionj>9</sectionj>
    −<geo-position-checking type="array">
      −<geo-position-checking>
          <coord-test>20150903150244.0</coord-test>
          <coord-test-id type="integer">186109</coord-test-id>
          <coord-latitude>60.107691383333</coord-latitude>
          <coord-longitude>19.95565535</coord-longitude>
          <datetime>20150903150244</datetime>
          <difference-from-tracker-to-team type="float">2.6533701589466516</difference-from-tracker-to-team>
        </geo-position-checking>
      </geo-position-checking>
      <Section-Max-difference-from-tracker-to-team type="float">2.6533701589466516</Section-Max-difference-from-tracker-to-team>
    </datum>
  </data>
  <Max-distance-from-all-sections type="float">8.626826611257618</Max-distance-from-all-sections>
</hash>
```


Figure 23: old sign in

3 Reflection

3.1 Perspective

Although our web site was improved greatly, but the amelioration is always existing.

1. Security The security is always the most important thing to do. Several actions could be adopted.

    - HTTPS Protocol Foe the users' sign in and sign up pages, they should be applied the HTTPS protocol to provide a white list to avoid attackers.
    - Robots Exclusion Protocol Robots exclusion protocol is used to define what we want to be quoted by the search engine (like google, yahoo). Some admin parts of the web site and also some private data do not hope to be published into the internet, then we can define the contents what we hope not to be visited by the search engine in robot.txt, and put this file into the Rails.root path.

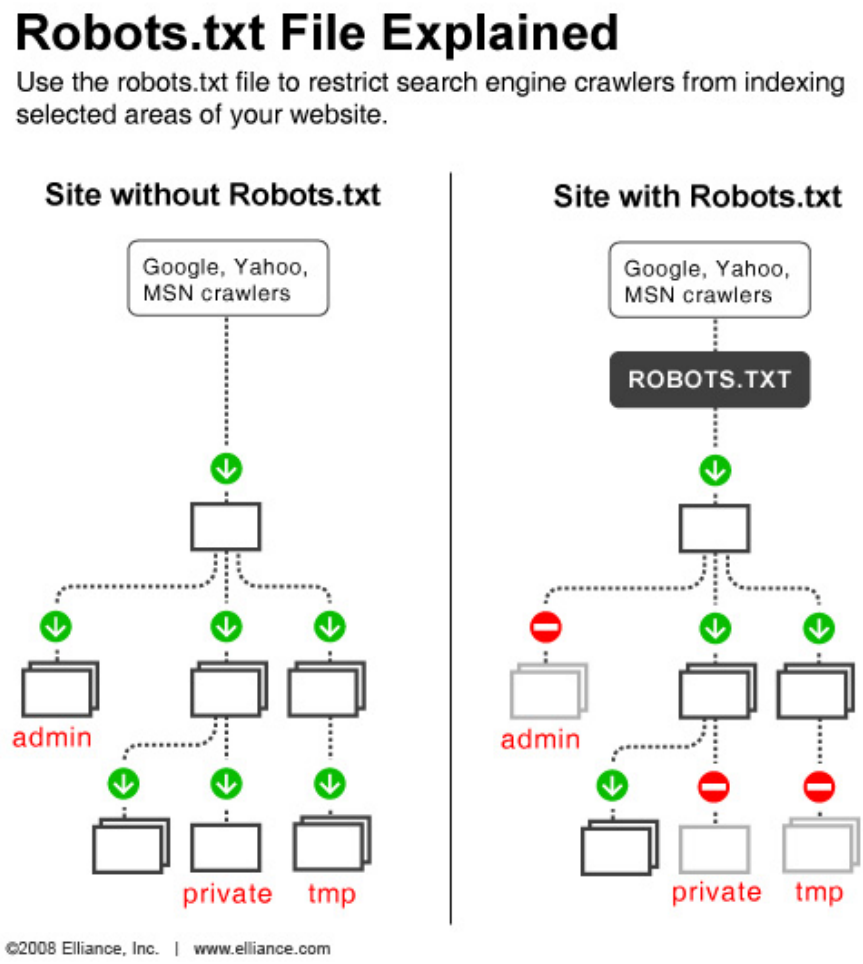


Figure 24: All function-oriented branches

3.2 Industrial analyse

After WRSC, occasionally discussed with some Finnish engineers, we found out that they had developed a similar tracking product in order to follow the trace(http://www.thingsee.com/). Comparing our tracking system with their product, they are nearly the same, except their product contains more sensors, so they can measure more parameters, like: temperature, motion and so on, and the price of product is more than 300 Euros. Although our tracking system do not have so much sensors, but it cost much less than their product, the price of our tracking system is less than 100 Euros. Furthermore, we found another similar project from the internet, in Brittany, France.(https://www.gwenneg.bzh/fr/tifiz) So it was not impossible to develop our own product in the near future.

3.3 Personal analyse

I am glad to have such a good opportunity to take part in a passionate project and also I am glad to be in Aland, Finland for my internship where I discovered a beautiful place with friendly awesome people. This impressive experience not only reinforces what I had learned, but also permit me to discover the world, to learn how real engineers realize their aims and especially work professionally under pressures.

4 Conclusion

Based on a ruby on rails framework, with jQuery libraries, building up a web server is not simple as what we see. In order to make a good product, we need to familiar with font end design technologies(html, CSS). And then JavaScript, Ajax are two indispensable tools for web developers. In addition, script programming, knowledge of database and network, all of them are the basic elements to construct the final building. A full stack engineer is not only an expert in one domain, but also should be familiar with different fields, then capable to suit in different environment and capable to work under pressure. This internship not only allow me to learn much more scientific technologies, but also a good experience to be a responsible engineer.

what is the fuck