

# *Final Report*

# *SWARMFRAME Project*



*BOURDON Benoit*  
*ENSI 2015*  
*Option SLS*

*DROUOT Bastien*  
*ENSI 2015*  
*Option ROB*

## *Abstract*

This project aimed at developing a website generator for the World Robotic Sailing Competition. The generated website will be used to monitor a fleet of sailing robots during the competition. This website is based on an already existing website developed last year. This new version not only improve the previous one but also add new features: creation of a website generator, direct use of Google Map API, proper use of the Rails<sup>1</sup> asset pipeline, adjunction of an administrator tool helping the organization of the competition. This document mostly describes these new features by detailing their inside functioning. It can also be read as a handbook introducing major Ruby on Rails features and explaining recent methods of agile development.

---

<sup>1</sup> “Rails” or “Ruby on Rails” is a powerful web framework based on Ruby programming language. It was our main tool during this project.

# Table of contents

Abstract .....	2
Introduction .....	5
I. Major changes between SWARMON and MYR .....	6
A. Creation of a website generator .....	6
B. Improvement since SWARMON .....	6
1. New CSS/Layout .....	6
2. New “Real-time” page.....	7
3. Administrator tools to add marker .....	7
4. New database .....	8
II. Improvement in coding the Real-time page .....	9
A. Reminders .....	9
1. Reminders about HTML and CSS.....	9
2. Reminders about JavaScript .....	9
B. Where do I put my JavaScript? .....	10
C. Unobtrusive JavaScript .....	10
D. Asset Pipeline.....	11
1. Asset Pipeline features .....	11
2. Asset Pipeline in Monitor your Robot .....	14
III. Ajax in Ruby on Rails .....	16
A. Ajax for data processing .....	16
B. Ajax for dynamic GUI using partial rendering .....	18
IV. Web site generator .....	20
A. What is a web generator .....	20
B. Our web generator.....	20
1. Templates metaprogramming .....	20
2. Ruby scripts .....	22
3. Configuration files.....	24
4. Rake file.....	26
VI. Project Management .....	27
A. Agile Development in Monitor Your Robot .....	27
1. Minimum Viable Product and prototyping .....	27
2. Fail faster and fallback possibilities.....	28
3. Milestones and reporting .....	28
4. Provide needed abilities by iterative learning.....	28

B. Do less but do it better .....	29
C. Future of the project .....	29
Conclusion .....	30
Bibliography .....	31
Annexes .....	33

## *Introduction*

The WRSC<sup>2</sup> is a major robotic event. Once a year, it is the meeting of the world specialists of sailing robots. Two years ago the organizers of the 2015 edition asked ENSTA Bretagne to develop a complete solution allowing the public to remotely follow the competition. A tracker has been developed during a second-year student project and a basic website has been created during our two months internship in Finland. This “Monitor Your Robot” project directly follows this development. It is based on the previous website developed using Ruby on Rails. It adds three new major features will be: a website generator, a new “real-time” page<sup>3</sup> and an administrator interface to edit mission markers directly on map.

The first part sums up the improvement made compared to the “SWARMON” project. The three following parts describe in detail the technologies used to improve the website. The latest two parts focus on the project management and the future of this project.

---

<sup>2</sup> World Robotic Sailing Championship

<sup>3</sup> To follow the race in real-time.

## I. Major changes between SWARMON and MYR

SWARMON system (Tracker and web site) was created for the WRSC 2014; it is only adapted for this event and this year. Therefore we created MYR system, MYR system include a web site generator and improve serval parts of SWARMON system.

### A. Creation of a website generator

Add a web site generator on our system allows regenerating a new web site every year for the WRSC. Every year the organizer of the WRSC can customize his web site without any knowledge about html or Ruby On Rails.

### B. Improvement since SWARMON

#### 1. New CSS/Layout

The web site created for the WRSC 2014 (project SWORMON) contains a no open sources CSS and Layout moreover this CSS is not CSS3 (the last version of CSS). Therefore we created a new Layout and selected a new CSS, an open source CSS3.

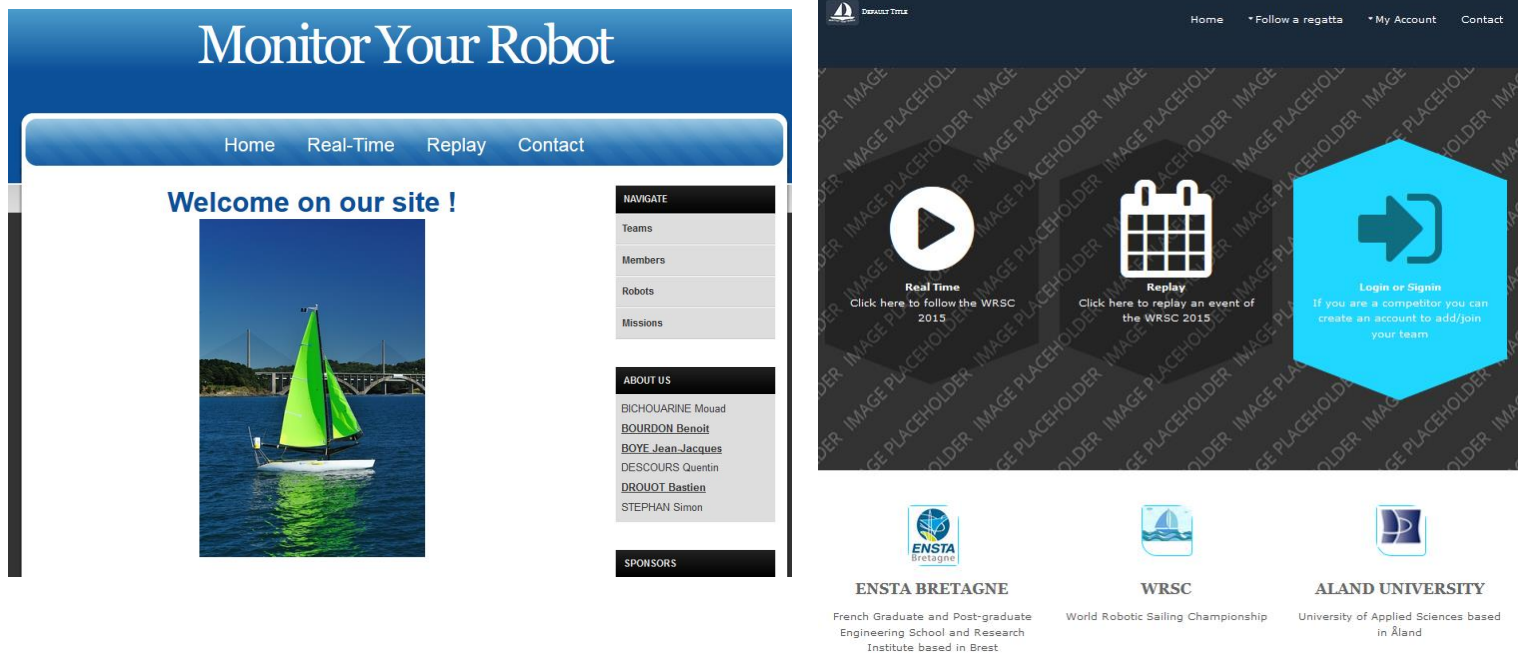


Figure 1: Our home pages with the old layout (left) and the new one (right)

## 2. New “Real-time” page

The major functionality needed by the WRSC is the possibility of following the competition in real-time. The previous website already included this asset but was not totally effective. It was based on the use of a gem which limits the functionalities provided by the Google API and has a very bad impact of performance due to the implementation which has been made (refreshing the map was in fact only re-generating completely the map and adding again all the markers on the map).

The new “real-time” page now directly use the Google Map API and only add the needed markers onto the map. As we will see later in this report its implementation is as clean as possible and respect the Rails principles which enable a large performance boost.

## 3. Administrator tools to add marker

One functionality which was missing in the previous website was the possibility to add “mission markers” on the map to help the public to understand what is the goal of the current mission<sup>4</sup> (for instance: to turn around a boil, to win a race or even avoid an obstacle).

The changes made in the database and the direct use of Google Map API allowed us to develop a prototype of this administration tool. Even though it only the user of one marker at a time (for now no possibility of delimitating an area for instance) it uses all the need technologies needed. Therefore this prototype is a proof that such functionality can be implemented and will added value to our website.

---

<sup>4</sup> The adjunction of this functionality was not even mentioned in the initial subject of this project. It has been added later in the project after the spontaneous proposition of the student working on the Google Map part.

#### 4. New database

We created a new data base, this data bases is generic and adapted for the WRSC. Generic because our data base contains not much links and each table is generic, and adapted because the database is structured for the WRSC. As an example the main activity of the WRSC is the attempt and our main table is the table “Attempt”.

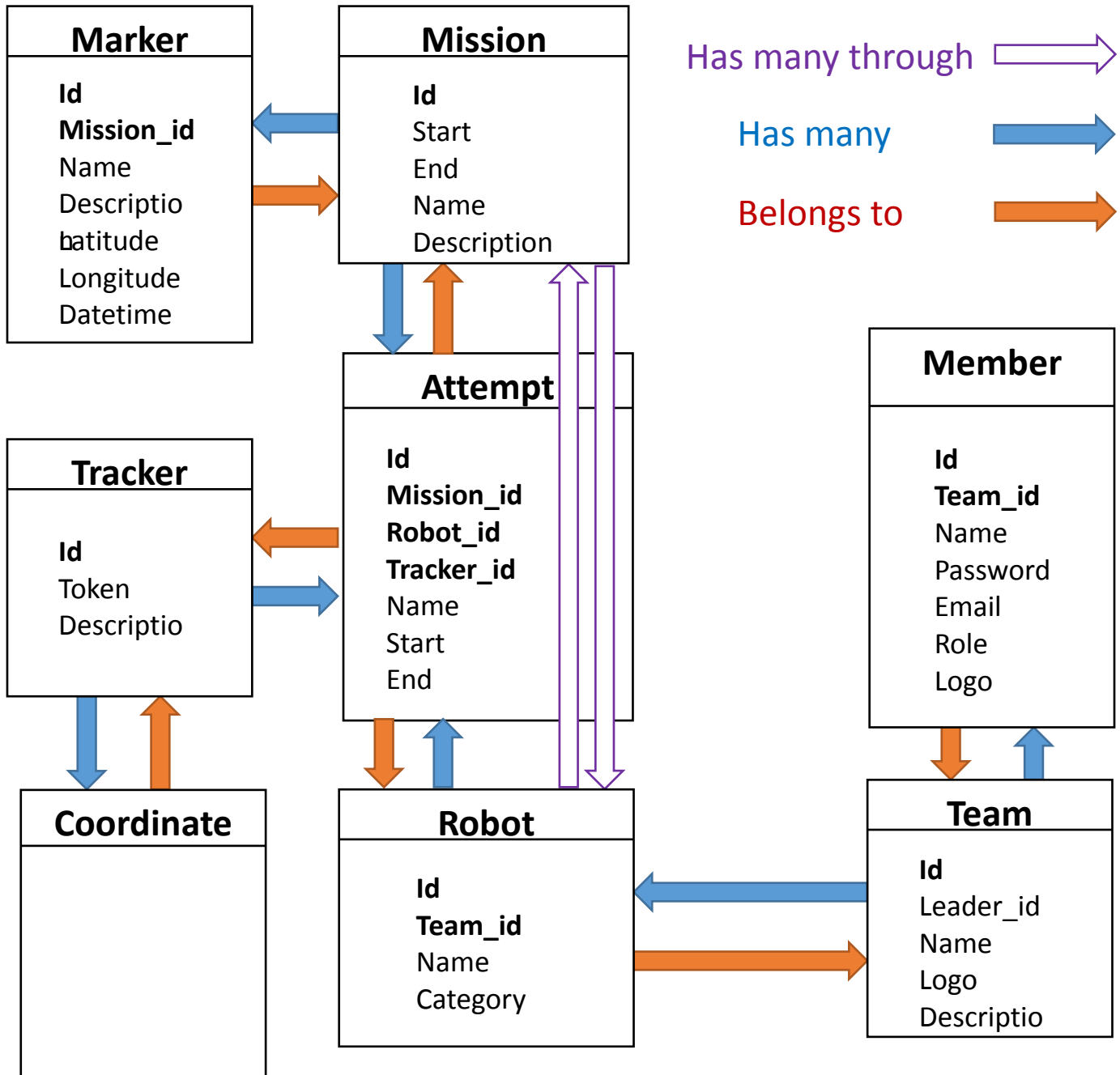


Diagram 01: The new data base



## *II. Improvement in coding the Real-time page*

This part will first put in place some basic reminders about web development languages (reader accustomed to web development can skip these two sections). Then it will depict from the inside the improvement made in JavaScript handling between SWARMON and MYR.

### *A. Reminders*

#### **1. Reminders about HTML and CSS**

In web development the two fundamental ‘programming’ languages are HTML<sup>5</sup> and CSS<sup>6</sup>. HTML is a markup language used to describe the type of data you want to display (text, title, image, content in a footer ...). CSS is the style sheet language which will describe how this data will be display (position, appearance, colors ...).

We could sum up that HTML is “what you say” and CSS is “How you say it”.

#### **2. Reminders about JavaScript**

JavaScript is a programming language used in web development to generate dynamical content on web pages (alter the aspect of the page, allow interaction with the user ...).

All of HTML, CSS and JavaScript are interpreted by the client web browser<sup>7</sup>. That means that the resources will be more limited than on the server. That is why a good practice is to make sure that all complex calculations will be done on the server side.

---

<sup>5</sup> HTML : Hypertext Markup Language

<sup>6</sup> CSS : Cascading Style Sheets

<sup>7</sup> Such as: Google Chrome, Mozilla Firefox, Apple Safari and even Internet Explorer!

## B. Where do I put my JavaScript?

The common “clean”<sup>8</sup> behavior is to place all the JavaScript of a page between the HTML `<head>` `</head>`. It is also possible to put a link to an external file containing the JavaScript. As we will see later in this report, this method is the one to be preferred while using Ruby on Rails.



```
<!DOCTYPE html>
<html lang="...">

<head>
  <script>
    ...
    Some JavaScript
    ...
  </script>
</head>

<body>
  ...
  Body of the page
  ...
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="...">

<head>
  <script type="text/javascript" src="script.js"></script>
</head>

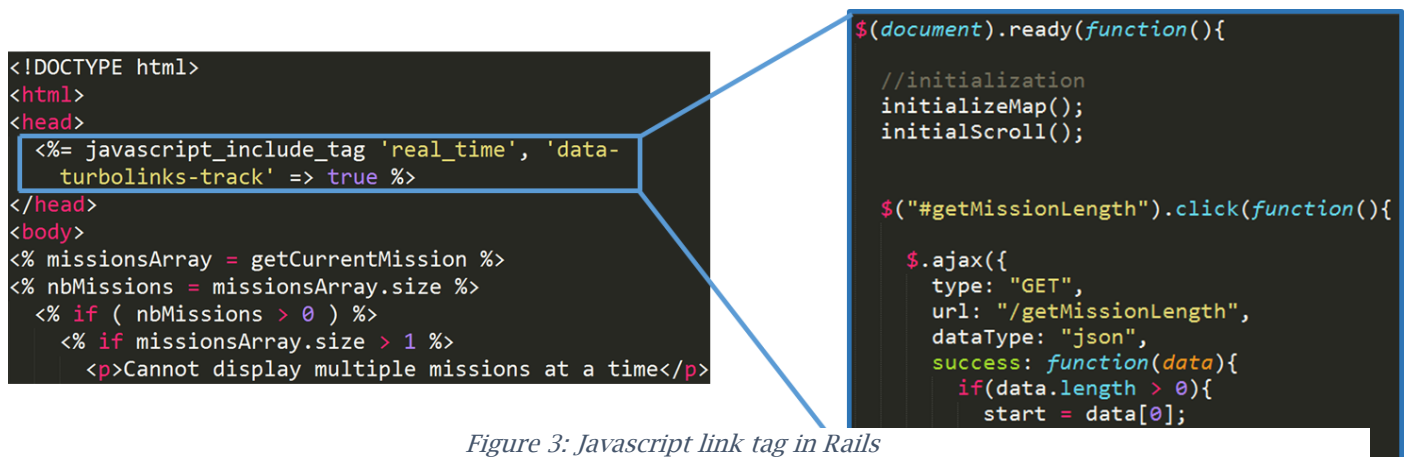
<body>
  ...
  Body of the page
  ...
</body>
</html>
```

Figure 2: Common place to put the JavaScript

## C. Unobtrusive JavaScript

The fact of clearly separating the HTML from the JavaScript is called “unobtrusive JavaScript” and is firmly encouraged by Ruby on Rails.

First benefit: it makes the code easier to read by clearly separating the static part from the dynamic<sup>9</sup> part of the web page. Second benefit: you can store your JavaScript in “the folder of your choice” and therefore take advantage of the performance enhancement proposed by Rails.



```
<!DOCTYPE html>
<html>
<head>
  <%= javascript_include_tag 'real_time', 'data-turbolinks-track' => true %>
</head>
<body>
  <% missionsArray = getCurrentMission %>
  <% nbMissions = missionsArray.size %>
  <% if ( nbMissions > 0 ) %>
    <% if missionsArray.size > 1 %>
      <p>Cannot display multiple missions at a time</p>
    </if>
  </if>
</body>
</html>
```

```
$(document).ready(function(){

  //initialization
  initializeMap();
  initialScroll();

  $("#getMissionLength").click(function(){

    $.ajax({
      type: "GET",
      url: "/getMissionLength",
      dataType: "json",
      success: function(data){
        if(data.length > 0){
          start = data[0];
        }
      }
    });
  });
});
```

Figure 3: Javascript link tag in Rails

<sup>8</sup> I will not mention the possibility of writing JavaScript code directly into the HTML code or the possibility of putting the `<script>` tag in the body.

<sup>9</sup> The « embedded Ruby » of `.html.erb` can alter the content of the page but only on load of the page. Unless it is used with Ajax, it does not provide dynamic content to the web page.

## D. Asset Pipeline

One key feature<sup>10</sup> of Ruby on Rails is its “asset pipeline”. The main features of “asset pipeline” are that it concatenates precompiles and minifies/compresses<sup>11</sup> JavaScript and CSS assets.

To take advantage of these features you “just<sup>12</sup>” need to place all your JavaScript and CSS in the “asset pipeline”<sup>13</sup>.

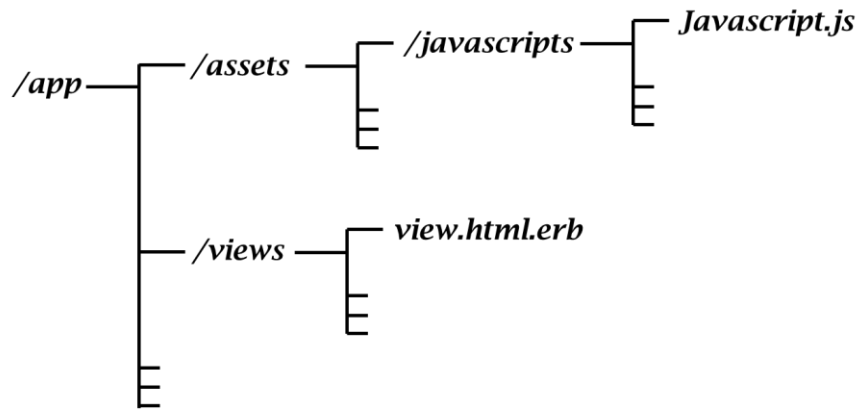


Figure 4: Basic tree view of a Rails application

### 1. Asset Pipeline features

#### a) Minification and compression

These processes come last<sup>14</sup> but are the easier to understand.

The minification (for both CSS and JavaScript files) consists in removing whitespace and comments. The compression is a more specific process which can only be applied on JavaScript files. Compression can consist for instance in shortening variables names and modifying statements in functions (optimize the use of “if” and “case/switch” statement for instance).

---

<sup>10</sup> After the well-know and powerful « scaffolding » which allows anybody to create a template of website (database linked to view and basic CRUD action in controllers) in a few minutes. (note: CRUD means Create Read Update Delete)

<sup>11</sup> The difference between both terms will be detailed later in this report.

<sup>12</sup> For reasons of convenience you do not benefit from all this features while in development phase. The concatenation and is automatic but the compression is time consuming and need to be activated later on at the beginning of deployment phase.

<sup>13</sup> This way of doing is typical of Rails which scatters the code in different folder regarding their usage.

<sup>14</sup> The correct order is: concatenation, precompilation, minification and compression.

## b) Concatenation

The “asset pipeline” can concatenate all JavaScript files into one “master” JavaScript file. The purpose is to reduce the number of requests that the browser need to make to the server to get all the needed JavaScript files. Indeed, GET requests are time consuming for both the server and the client. Moreover current browsers are limited in the number of requests that they can make in parallel (up to 6 for Chrome and Mozilla). Concatenation is therefore a great solution to increase performance by reducing latencies.

By default in Rails, all JavaScript files placed in the “/asset” folder will be concatenate into the file named “application.js”. This “application.js” contains two types of statements allowing the concatenation. They both start with the same typical syntax: “//= require<sup>15</sup>”.

“//= require yourjavascript” permits you to include a JavaScript file (named “yourjavascript.js”) into the only file “application.js” while respecting the order you have specified.

“//= require\_tree” tells the application to add recursively into the “application.js” all the JavaScript files present in the “/asset” folder.

The problem of this statement “//= require\_tree” is that sometimes you do not want a specific JavaScript file to be used in several pages<sup>16</sup>. For this reason I preferred not to use the “//= require\_tree” statement. So that, the files which need to be called from everywhere in the application (e.g. functions to be called), are placed in the “application.js” by using the “//= require something” statement, while other files will just be placed in the “/asset” folder.

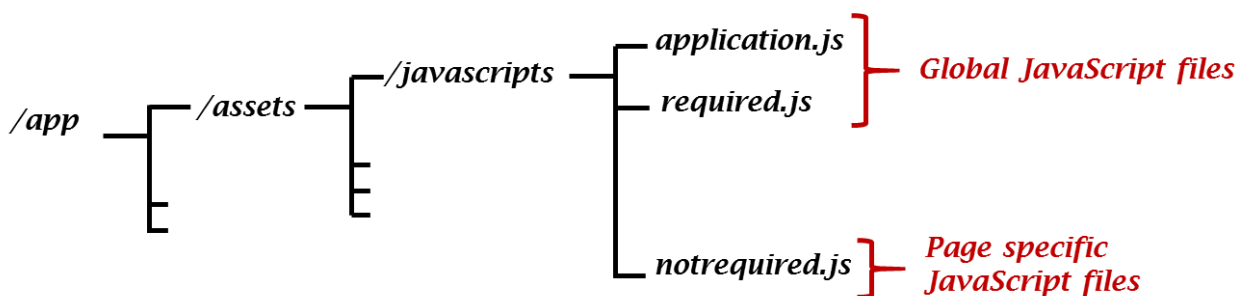


Figure 5: The two different types of JavaScripts

<sup>15</sup> “//= require” can basically be seen as a common “include” statement.

<sup>16</sup> In fact it is common to have some JavaScript files having a specific behaviour for a specific page and some others (functions declaration for instance) which must be accessible from different pages.

### c) Precompilation

Rails needs to precompile all JavaScript files before sending them to the client. Precompiling means, in a given file, to translate all the code written from a higher-level language<sup>17</sup> to only pure lower-level language. In our application “precompilation” means to translate all the JQuery into pure JavaScript.

Placing a JavaScript file in the “/asset” folder without including it in the “application.js”<sup>18</sup> is not enough for Rails to use it properly. In order to tell Rails to precompile a JavaScript file even though it is not called in “application.js” we need to add a specific statement in the “assets.rb” file located in “/config/initializers”.

```
Rails.application.config.assets.precompile += %w( markersCreation.js )
Rails.application.config.assets.precompile += %w( real_time.js )
```

Figure 6:Precompilation of JavaScripts

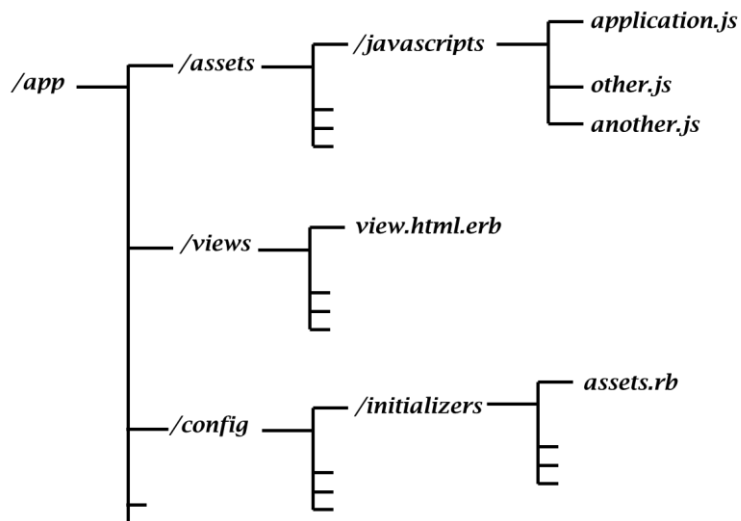


Figure 7:Tree view describing the use of JavaScript in Rails

<sup>17</sup> Regarding JavaScript, it can be CoffeeScript or jQuery (in MYR we use jQuery). JQuery facilitate greatly the writing of specific behaviors such as “on click event” or “Ajax requests”.

<sup>18</sup> i.e. exactly what we did at the end of the section about “concatenation”.

## 2. Asset Pipeline in Monitor your Robot

In “Monitor Your Robot” we have tried to use the “asset pipeline” in the more efficient and the “cleanest manner” possible. This part will sum up what we have seen in the previous section (“asset pipeline features”) by using the MYR application as example. It is advised to read the three following sections at the same time as looking the corresponding figure.

### *a) JavaScript link tag in views*

In “Monitor your Robot” each “views” written in “html.erb” only contains code written in html or erb (no jQuery or JavaScript in the “views”). Web pages called their JavaScript files by using the Rails JavaScript tag. This tag allows to include JavaScript files stored in the “/asset” folder by using relative path (the root is the “/asset/javascripts” folder).

The fact of scattering the code like this provide the advantage of separating each piece of code regarding its concerns (.js in “/asset/javascripts” and .html.erb in “/views”)

### *b) Global JavaScript files*

The jQuery functions which are meant to be used across the application<sup>19</sup> are all written in one file named “functions.js”. This file is called in the “application.js” by using the call “//= require ./functions”<sup>20</sup>. The “application.js” is then called in the layout<sup>21</sup> of the application.

This way of doing provide the possibility of calling these functions from any “view” of the application while taking advantage of the performance enhancement provided by the “asset pipeline”.

---

<sup>19</sup> At least all the functions needed to handle maps because they are use in « /real-time » and « /replay »

<sup>20</sup> The « ./ » can be omitted as seen before. We keep it to show that we have coded and included it by ourselves.

<sup>21</sup> In Rails, the layout is the static “html.erb”-frame in which all generated “html.erb” pages are to be rendered. Therefore the layout is present on each “views” generated by the application.

### c) Page specific JavaScript files

Page specific JavaScript files (e.g. script which wait for a “click-event” or have specific behavior on page loading) are all placed in the “/asset” folder. To be precompiled, a dedicated line of code has been added in “/config/initializers/assets.rb” for each JavaScript file.

These script are really light (only calling external functions) and are only requested by the client when needed.

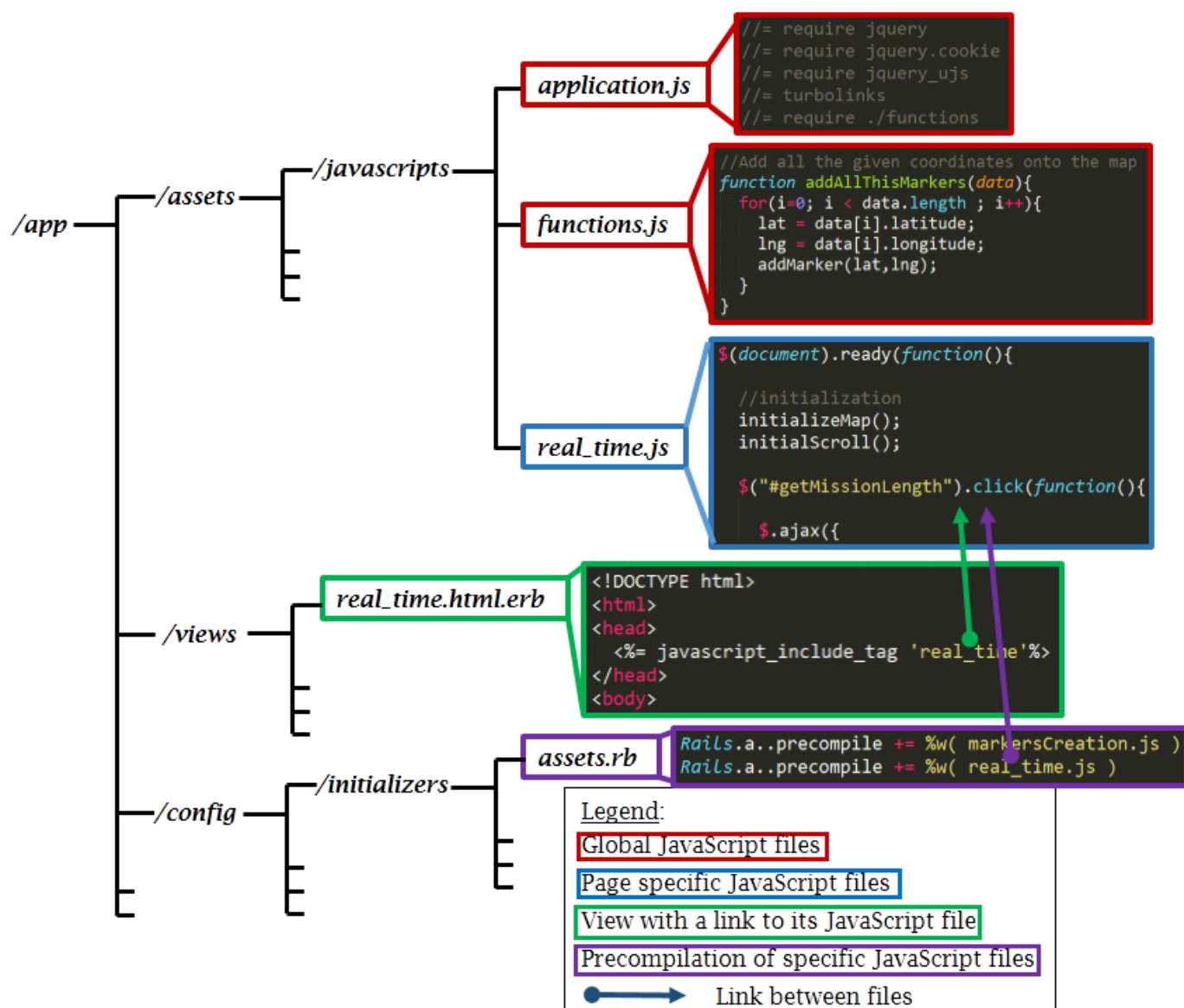


Figure 8: Global tree view pointing out the interactions between files

#### *d) Global benefit of the « asset pipeline »*

By limiting the number of requests made by the client and reducing the size of the .js and .css files, the asset pipeline increase the global performance of the Rails application.

Compared to SWARMON it does not prevent anymore the use of “Turbolinks”<sup>22</sup> which also increases performance.

### *III. Ajax in Ruby on Rails*

#### *A. Ajax for data processing*

The MVC<sup>23</sup> pattern used in Rails does not allow direct data retrieving from the database without completely reloading the web page. The MVC pattern work as a cycle<sup>24</sup>. When you click on link to go on a new page it is always the same process: the routing table calls the controller which retrieves data from the database and pushes the data into the View. Always the same unchangeable order: Routing / Controller / Model / Controller / View.

The first solution which comes to mind is to reload the page. Doing so we will go all around the cycle again and we will have access to the database. Indeed, it works. However it is neither very efficient nor elegant. Each time you want to retrieve data you have to completely load again the web page (even though there is no new data to retrieve ...). It is power consuming and not really pretty for the user to be forced to face a “laggy” and blinkering screen.

In fact the best solution is to use Ajax<sup>25</sup> requests. For the particular application of retrieving data, a dedicated jQuery function exists. This function provides different parameters allowing the user to write easy-to-read Ajax requests.

An Ajax request can be seen as if the jQuery script was asking a question to the controller (this “question” is off course formalized using URL and HTTP parameters). Then the controller answers to the script with a JSON<sup>26</sup>. This JSON is easily “read” and processed by the script which can know modify the web page (alter the html code).

---

<sup>22</sup> Basically, between page changes « Turbolinks » keeps alive an instance of the previous page preventing the browser from compiling again all the .js and .css files when going back to this page. In SWARMON turbolinks must have been disabled in “view” and “replay” to prevent multiple execution of the .js of these pages

<sup>23</sup> MVC : Model View Controller is a classical architectural pattern

<sup>24</sup> See figure in Annexe

<sup>25</sup> AJAX : Asynchronous JavaScript and XML

<sup>26</sup> JSON : JavaScript Object Notation





Figure 9: Ajax for data processing

## B. Ajax for dynamic GUI using partial rendering

When someone what to generate a dynamic GUI<sup>27</sup> he or she faces the problem of changing a part of the web page without reloading the complete page. In fact, as explained before, reloading the page is a really bad solution and would not be acceptable for modern Internet users.

For this problem also Ajax is the solution. In this particular case “Ajax” is neither a method nor function. It is rather a pattern or way of doing it. Ajax implementation may be different in other frameworks (e.g: Django) or languages (e.g: php). However Ajax implementation in Rails just needs few lines of code<sup>28</sup> to create a truly dynamic GUI.

This apparent simplicity comes with a downside: it is hard to understand precisely how it works<sup>29</sup>. To understand this “automagical<sup>30</sup>” process the coder needs first to deeply understand .erb and jQuery script. Indeed, the powerful lines of codes used by Ajax rely on advanced capabilities of both .erb and jQuery. To better explain this point we will just analyze the main line in code present in JavaScript. It is the line which replaces the <div> element from the original html.erb with the new one (dynamic content).

The following code is written in js.erb<sup>31</sup>:

```
$("#carte").html("<%= escape_javascript(render 'mapp') %>")
```

Figure 10: A complex line of code

- > The \$ means that the directly following object will be converted into a jQuery object.
- > The # means that we retrieve the html element having the id “carte” from the original web page (the one wich has triggered the event)
- > \$("#carte”) is now the <div> element having the id “carte” which is now converted to a jQuery object.
- > “.html” means that this element will be replaced by html (html.erb in fact)
- > reminder: <% %> means that what is inside this tag would be preprocessed by the ruby language interpreter (that is why the file is named html.ERB. “erb” for Embedded RuBy)
- > The “=” in <%= %> means that this .erb tag will be directly replaced by the result of the Ruby calculation
- > That is why the “escape\_javascript” is needed in order to generate a ready-to-use html. (No double declaration of the <body> or <header> for instance)
- > At last the “render ‘map’” is a Ruby instruction (remember that we are still in the .erb tag) which will generate the partial view “\_mapp<sup>32</sup>”.

<sup>27</sup> GUI : Graphical User Interface

<sup>28</sup> Three short lines of code to add to be precise. Four if we count the button declaration!

<sup>29</sup> It is still easy to copy paste an example from the Internet but it is a different matter to write the code by oneself by replacing all the names of the variables.

<sup>30</sup> “Automatic, but with an apparent element of magic. Commonly used in computer and other technology fields, referring to complex technical processes hidden from the view of users or operators, resulting in tech that just works.” Definition from “<http://en.wiktionary.org/wiki/automagical>”

<sup>31</sup> In fact it is more something like “jQuery.erb” as we use jQuery as higher-level language library.

<sup>32</sup> After that the routing table has told Rails to execute the method « mapp » of the controller « view ».

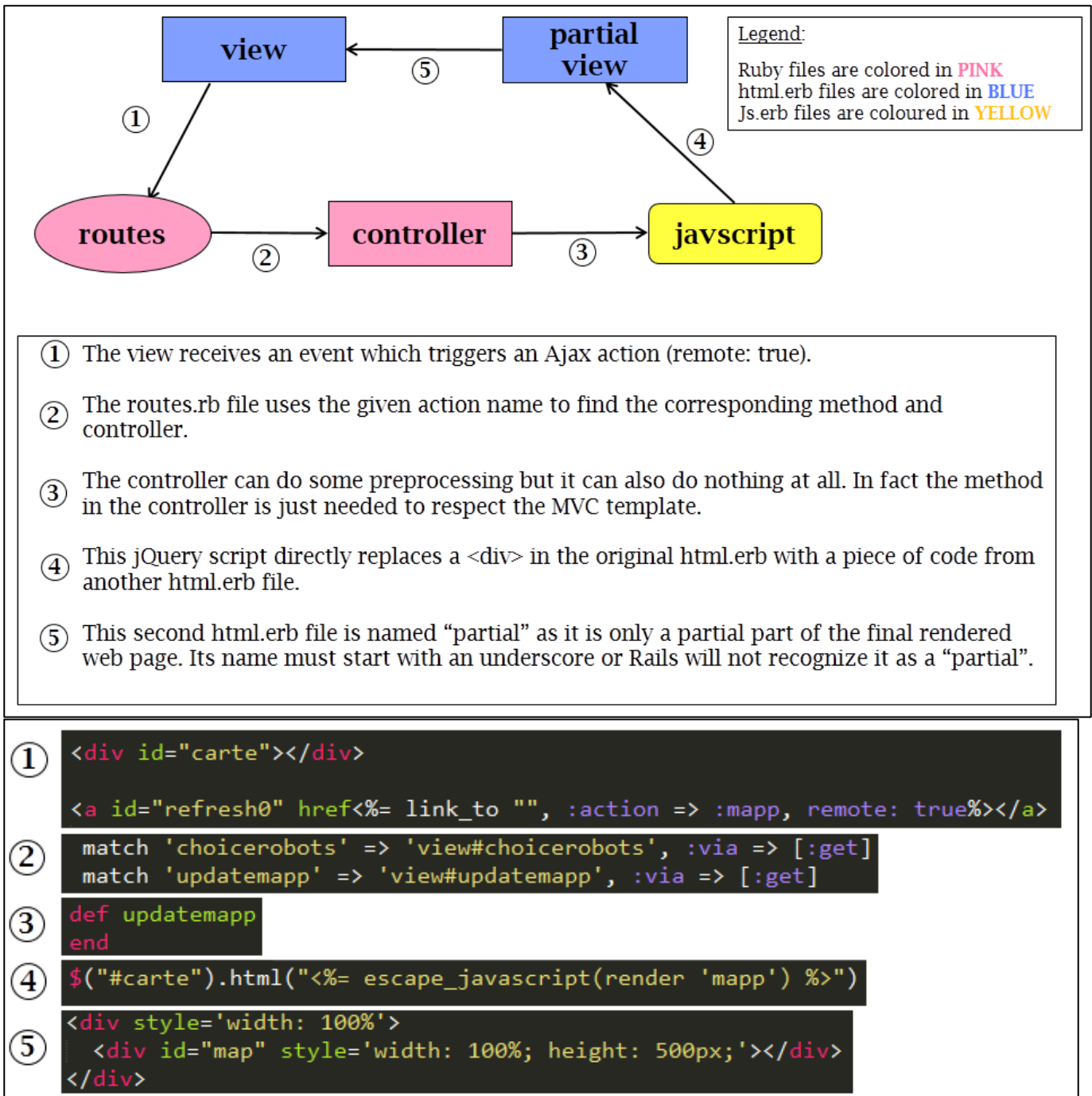


Figure 11: Ajax for dynamic GUI

## IV. Web site generator

### A. What is a web generator

A Web generator is a program which produces a web site on demand. In the status report we present some web generators, all of them are dedicated to generate static web site without data bases. Our web site generator was created to generate static and dynamic web pages (for one specific data bases). This generator was created for the WRSC and specifically to create the future web site of the WRSC in one click (same data bases, different web site ).

### B. Our web generator

Our web generator consists of four parts:

- Templates metaprogramming
- Ruby scripts
- Configuration's files
- Rake file

#### 1. Templates metaprogramming

Template metaprogramming is a metaprogramming technique in which templates are used by a compiler to generate source code. Two type of templates are used: static or dynamics.

##### a) Static templates

Static templates are used to generate the static pages (like the home page), all this pages are in one folder in Ruby On Rails (*/app/view/static\_pages*).

In Ruby On Rails all web pages (static or dynamic) are linked with a controller. Therefore all static pages have a controller, in such cases it is the same controller, for example for all the static pages it is *static\_pages\_controller.rb*.

Therefore our web generator have one template per static pages and generate one view (web page) in *app/view/static\_pages* per templates. Moreover some dynamic pages like the layout are generated like a static page due to the ratio of static line in these pages (more than 90%). In these pages all the dynamic line are used just to create some links (*redirect to, link to, ...*).

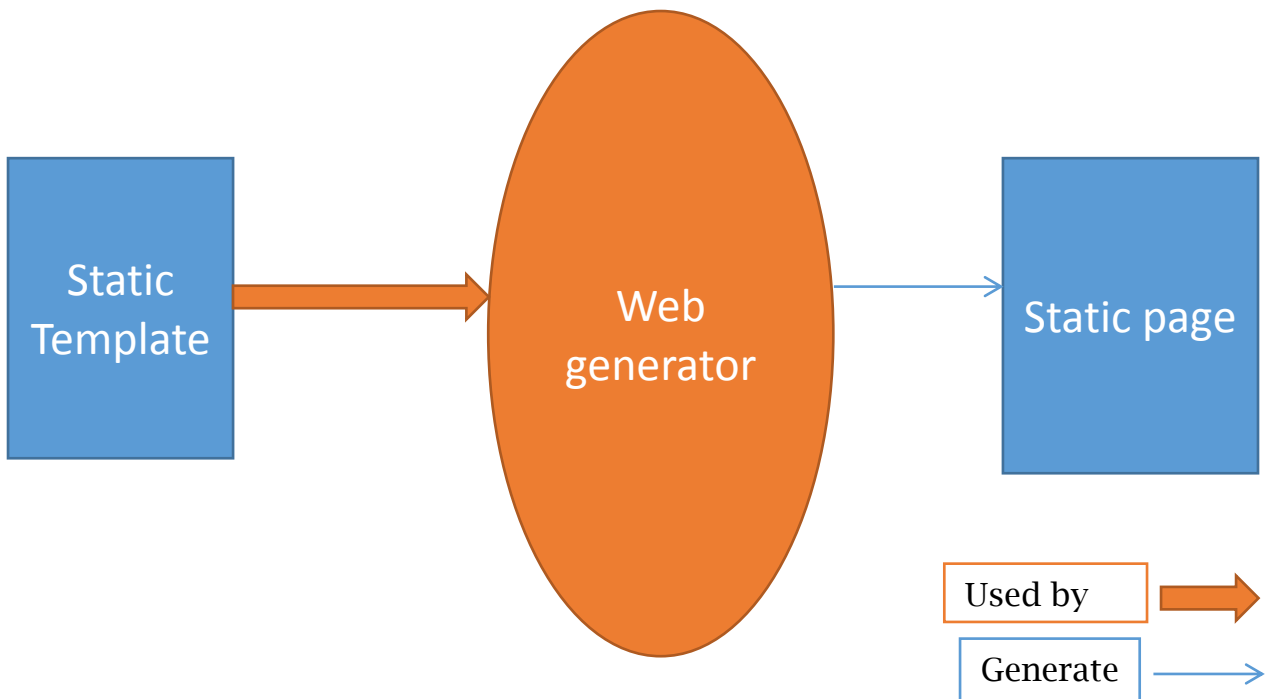


Diagram 02: generation of the static pages

### b) Dynamic templates

Dynamic templates are used to generate dynamic pages. A dynamic page is a page connected with the data base (through a controller). By definition of a template, dynamic templates don't exist. For this reason, to customize dynamics pages we use ruby variable. As an example, in a dynamic page we don't write "`<h1>This is a title</h1>`" but we write "`<%= myTitle %>`" and we create a variable "`myTitle='<h1>This is a title</h1>'`" in an appropriate file (a helper).

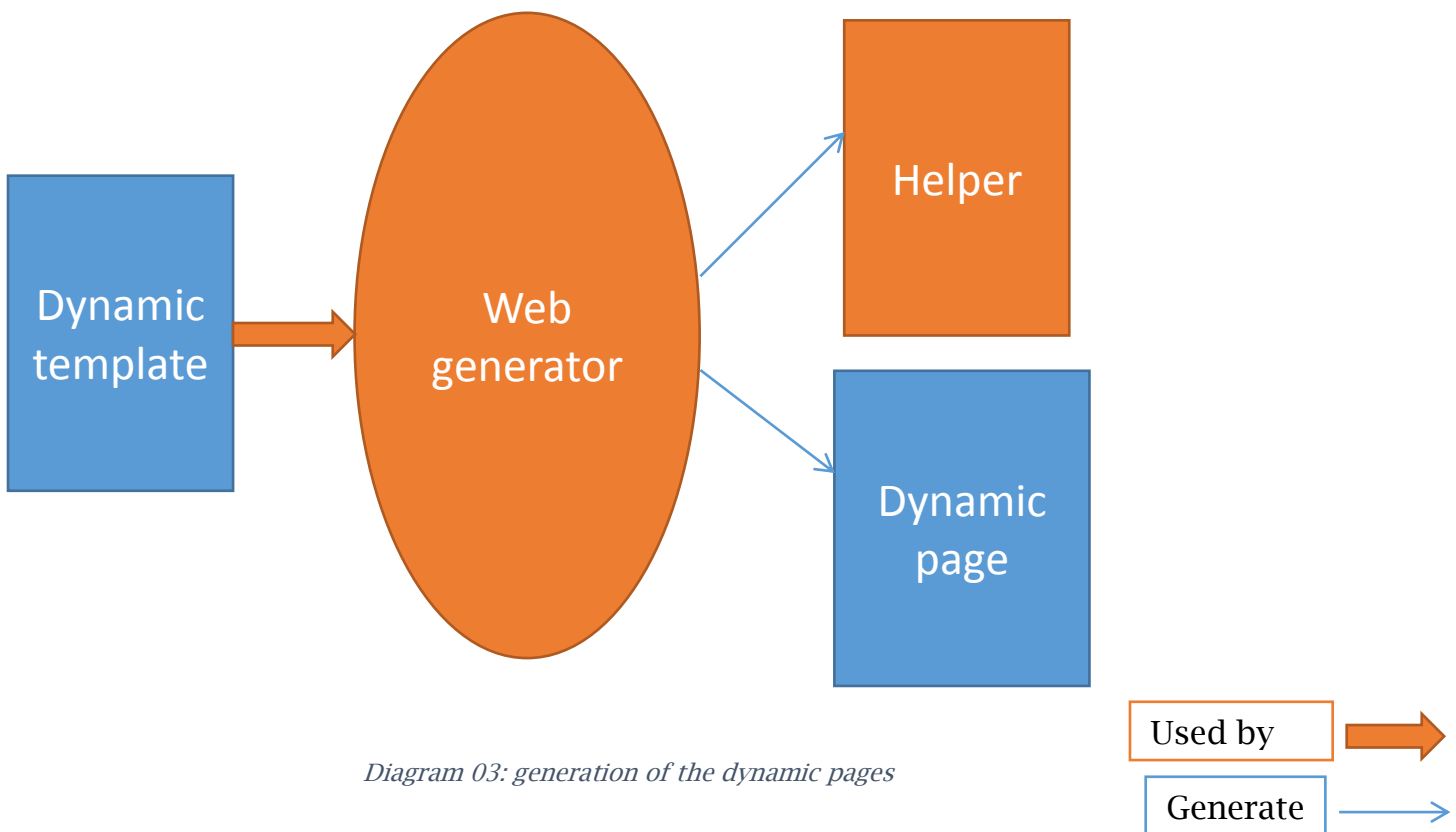


Diagram 03: generation of the dynamic pages

## 2. Ruby scripts

Ruby scripts are a group of script written in Ruby, each script has its own execution sequence. In our generator we use two scripts, one to generate the static pages and one to create all the variables associated to the dynamic pages.

### a) Ruby script of the static pages

This script browses all the static templates of the folder */lib/generator/templates/statics* and solves all the variables. To browse all the static templates our script check each files of the folder *.../templates/statics* two times. The first time is for checking if the file is an Embedded Ruby file (ERB) and the second time is for checking if the template is the template linked to the layout. After these checks all the template are solved (with the data included in an YAML file) and save in a specific folder:

- In *app/views/layouts* for the layouts
- In *app/views/static\_pages* for all the other pages

Moreover our script reads an YAML file (*config\_static.yaml*) and solves all the ERB tags (example : `<%= data["something"] %>`) of each template.

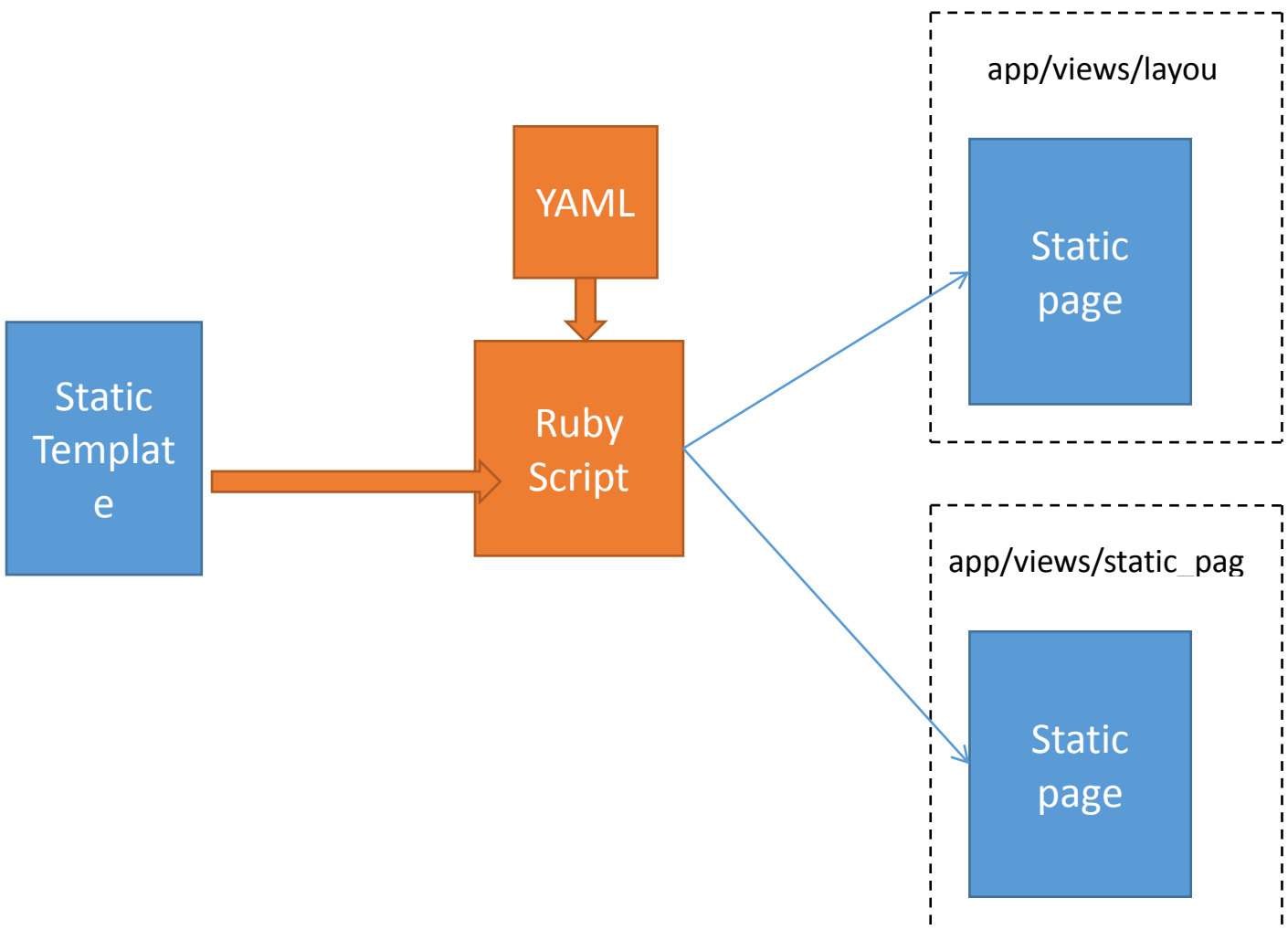


Diagram 04: ruby script for the static pages

## b) Ruby script of the dynamic pages

This script browses all the static templates of the folder */lib/generator/templates/dynamics* and copies each template in the right folder. Moreover our script reads a YAML file (*config\_dynamic.yaml*) and creates several helper in which are initialized all the Ruby variables. Those variables are used in the pages associated to the helper. We use this method due to the ERB files, indeed our templates and the dynamic pages are files with the same extension (*.html.erb*). But we cannot solve just some ERB tags in an ERB file. As an example it is impossible to differentiate `<%= data["title"] %>` , a communication tag, and `<%= f.tex_field :name %>` , a ruby tag which is used to create a text field in a web page.

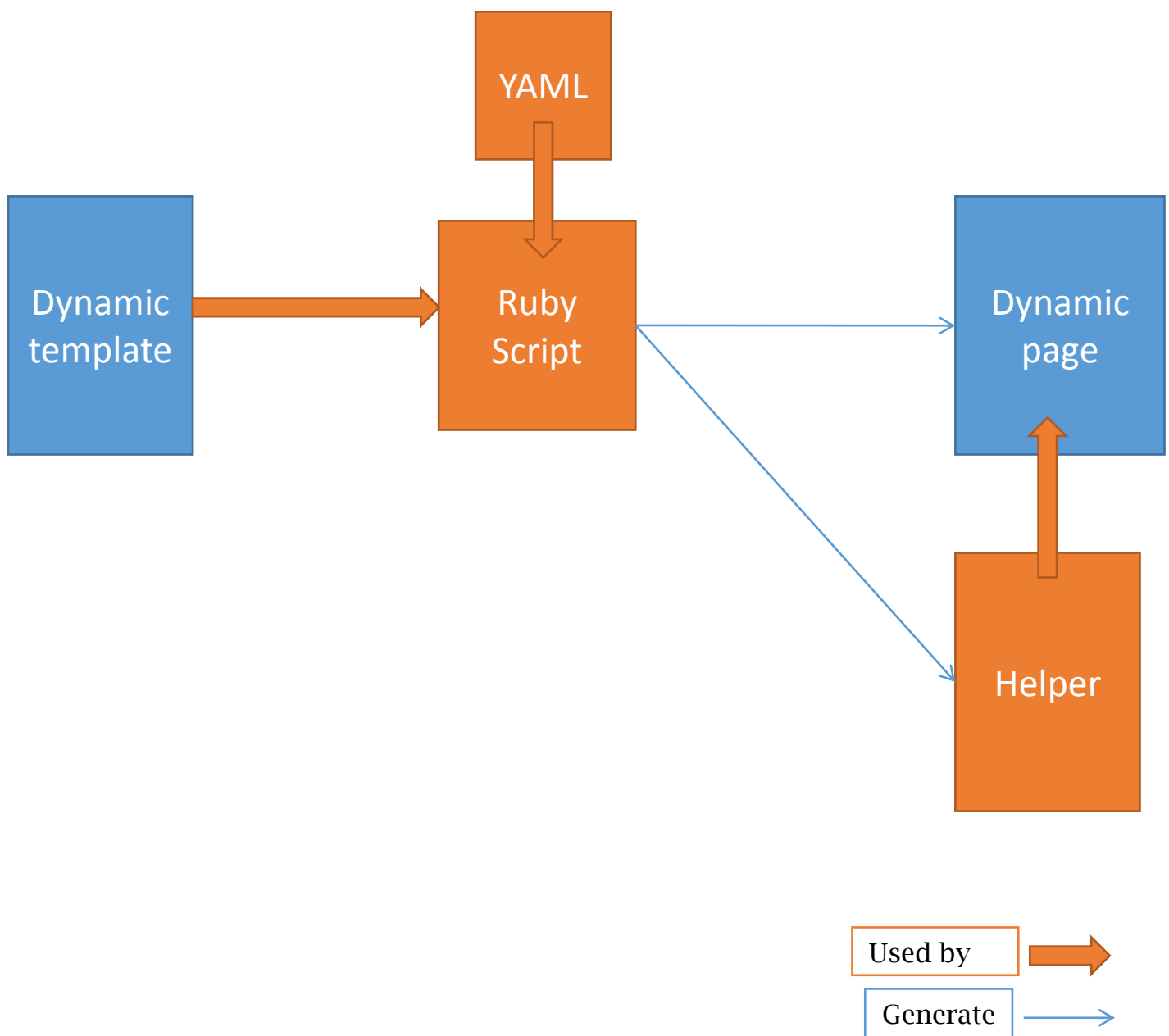


Diagram 05: ruby script for the dynamic pages



### 3. Configuration files

Our generator needs configuration files to initialize all the variables used for the generation of the web pages. Consequently we created two configuration files, one to initialize all the constants used in the static pages and a second one for the dynamic pages. Our configuration files are written in YAML (*config\_static.yaml* and *config\_dynamic.yaml*) , YAML is a human-readable data serialization format that takes concepts from programming languages and ideas from XML.

#### a) Configuration of the static pages

This YAML file (*config\_static.yaml*) is structured in two parts, the first part for the variables used in several static pages (sponsors, ...) and the second for the others variables used in just one static page. Each variable defined in our YAML file is an object, therefore all variables have several components and these components can be objects too.

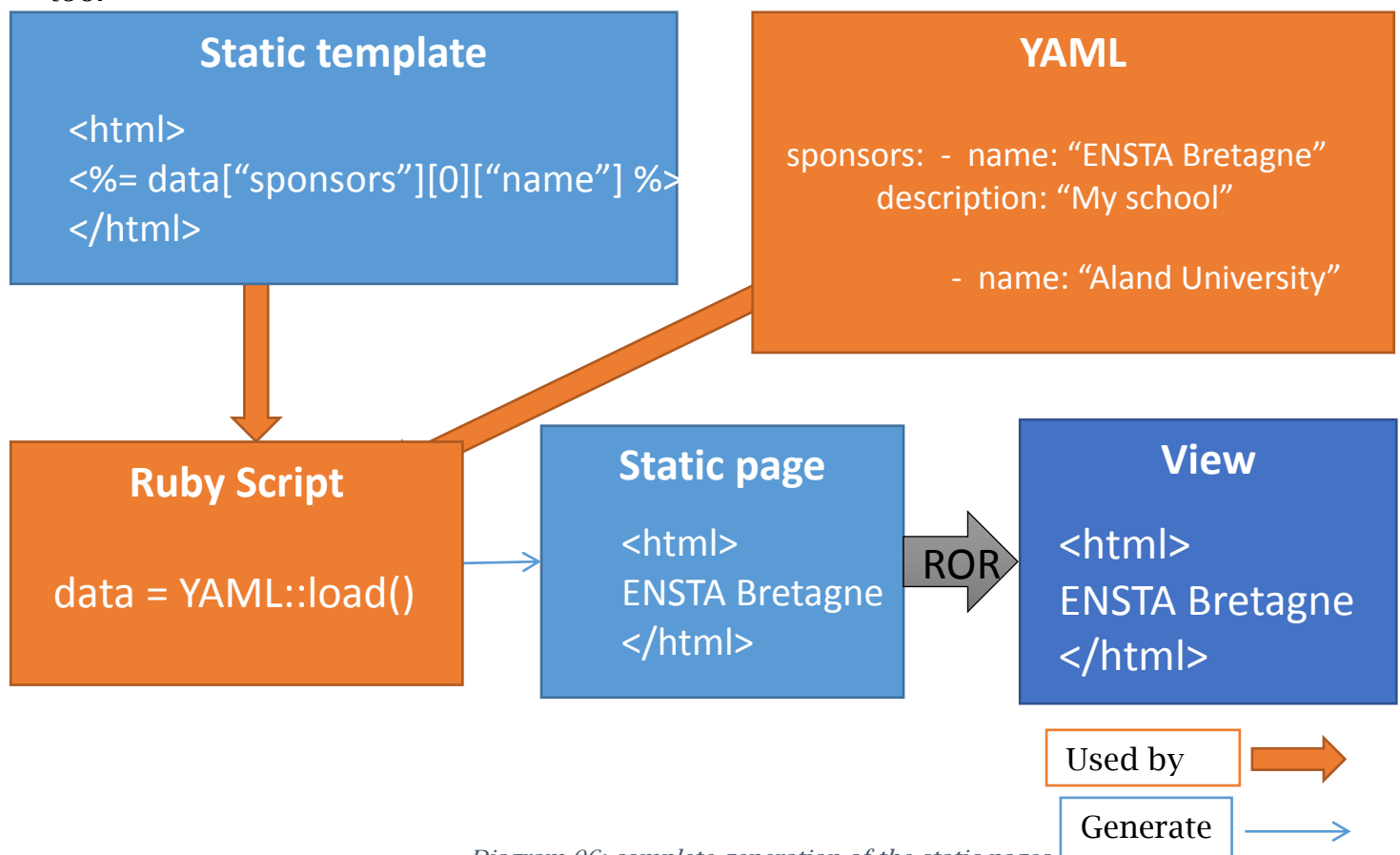


Diagram 06: complete generation of the static pages

In the example we define two sponsors:

- The first is *sponsors[0]*, this sponsor has a name and a description.
- The second is *sponsors[1]*, this sponsor has just a name .

In a static template we use the tag `<%= data["sponsors"][0]["name"] %>` to refer to the name of the sponsor n°0, here "ENSTA Bretagne".

With this system we can customize all the static pages with just one YAML file.



## b) Configuration of the dynamic pages

This YAML file (*config\_dynamic.yaml*) is structured like *config\_static.yaml*, but each dynamic page has an YAML object associated. As an example the page */members* has an YAML object associated "*members:*".

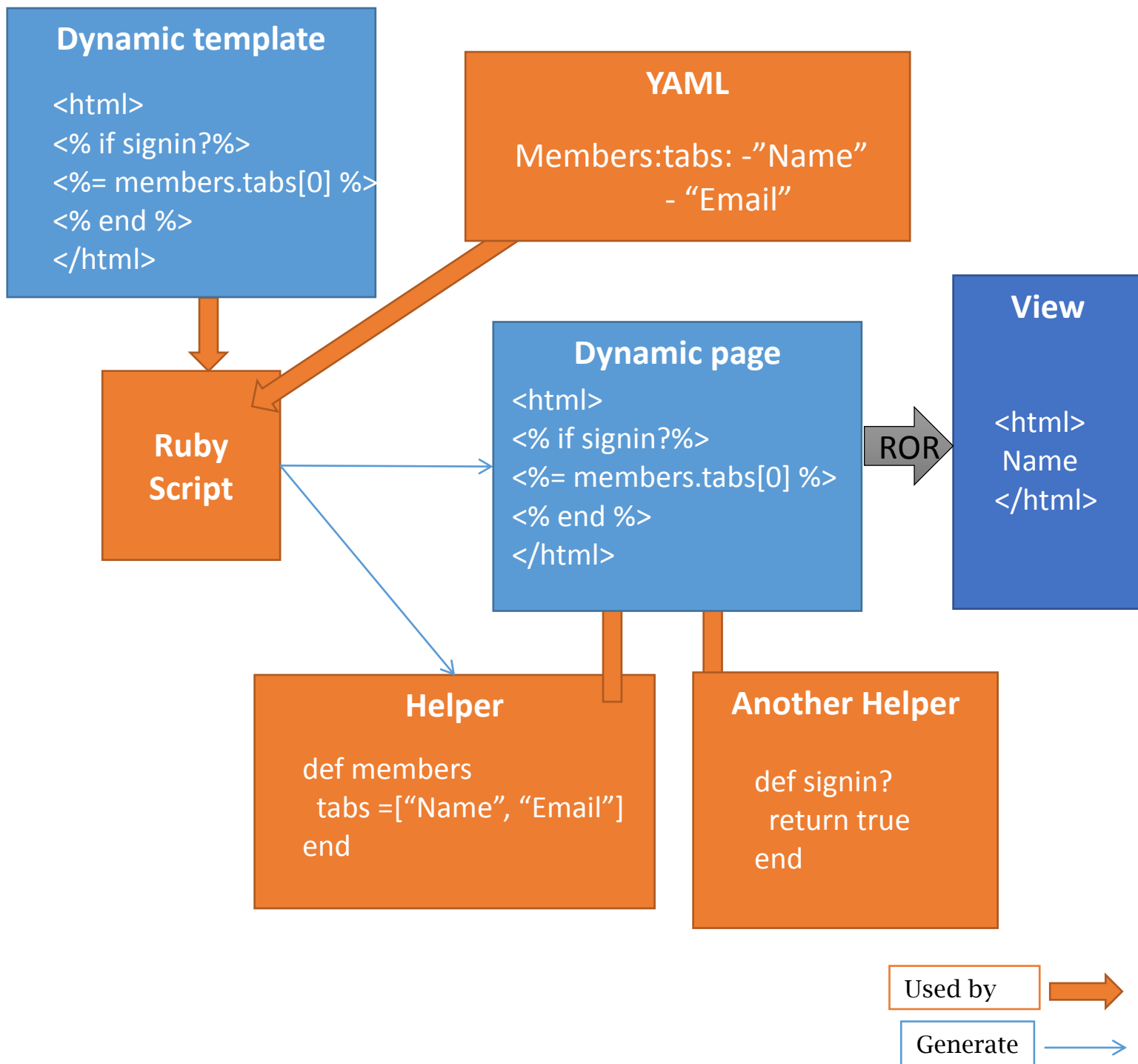


Diagram 07: complete generation of the dynamic pages

In the example we define an YAML object *members*. This YAML file is read in a ruby script. This script creates a ruby object in the helper *Member helper*. Therefore the dynamic pages *member* can use the variable *members.tabs[0]*. Finally Ruby On Rails (ROR) can generate a view.

## 4. Rake file

To start our generator we use an rake file, our rake file (*config.rake*) is in */lib/tasks*

```
1▼ namespace :config do
2  desc "Load config"
3▼  task(:load) do
4    ruby "#{Rails.root}/lib/generator/ruby_script/config_static.rb"
5    #ruby "#{Rails.root}/lib/generator/ruby_script/config_dynamic.rb"
6    puts "Load config completed"
7  end
8end
```

Figure 12: Content of a Rake Script

Currently our rake file executes just one ruby script because the script *config\_dynamic.rb* is not fully functional. One can notice that the namespace of our rake file is "*config*" and the name of the main task is "*load*", therefore to start our rake file we write "*rake config:load*" a Ruby On Rails consol.

## VI. Project Management

### A. Agile Development in Monitor Your Robot

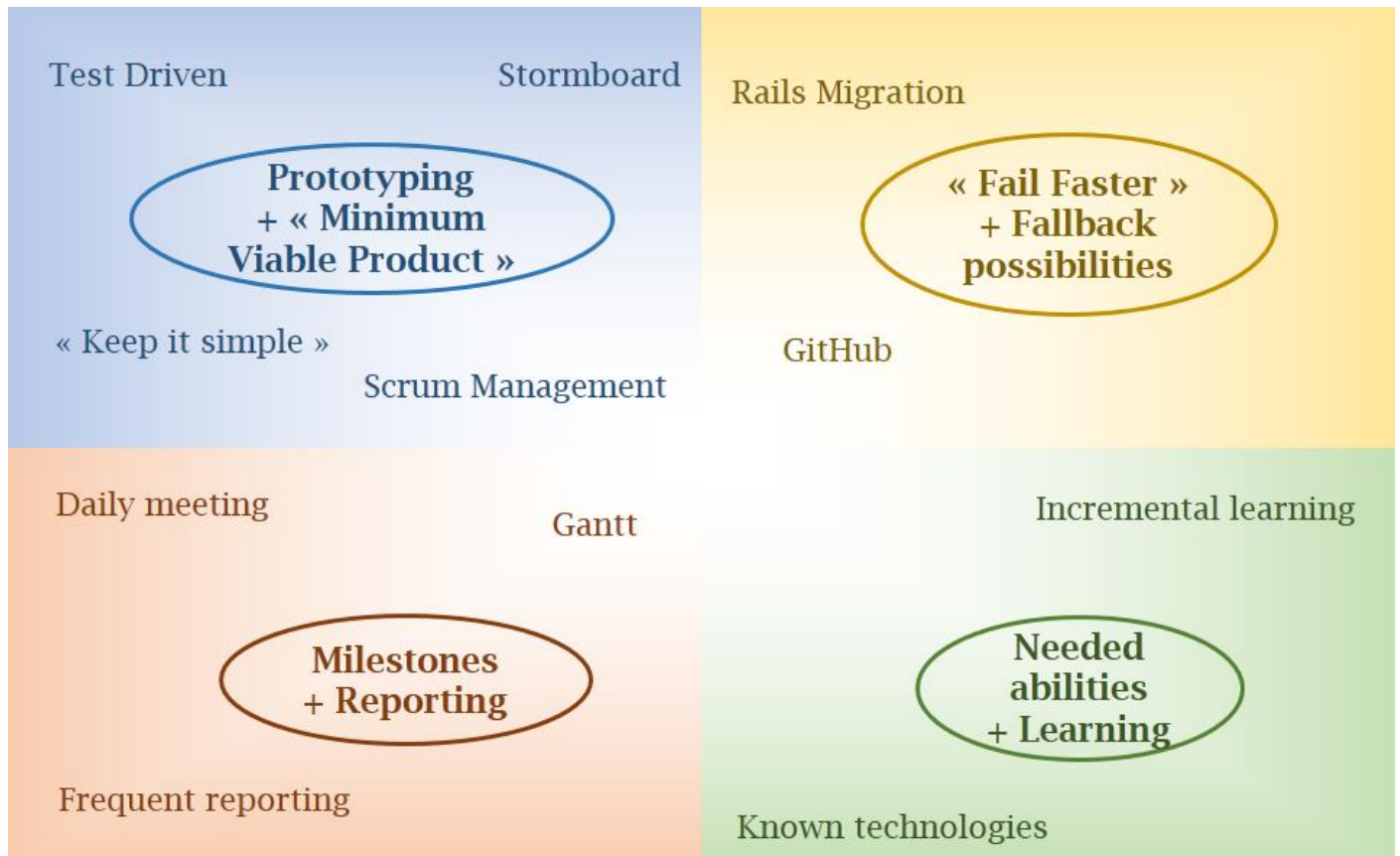


Figure 13: Agile Development in Monitor Your Robot

“Monitor Your Robot” project has been guided by four strong values:

- Minimum Viable Product and Prototyping
- Fail Faster and fallback possibilities
- Milestones and reporting
- Provide needed abilities by iterative learning

#### 1. Minimum Viable Product and prototyping

This principle is: before creating anything, find the simpler version possible of this product and try to make a prototype of it as fast as possible.

By reducing a large amount of work into more simple tasks, the complete project seems simpler.

Early prototypes allow testing as soon as possible (test driven development) and preventing late detection of problems.

## 2. Fail faster and fallback possibilities

Preventing late detection of problem is the key value of the “fail faster” concept. In a common development cycle, the later you find a problem the more expensive it is to fix it. That is why you have to fail faster.

Error is a component of human nature. From “typo” to bad ideas, each developer makes mistake. It is normal, but if the choice is given it is better to make these unavoidable mistakes as soon as possible. Try to fail faster.

## 3. Milestones and reporting

In the direct guideline of the “scrum management” and agile development it is recommend to reduce as much as possible the time need for each iteration<sup>33</sup>. The contact with the client is also deeply encouraged.

For these two reasons we have settled milestones to control the result of each iteration circle (around one week for each but sometimes less) and we have tried to contact our “client<sup>34</sup>” as much as possible to assure the harmony between what he was expecting and what we were developing.

## 4. Provide needed abilities by iterative learning

The former « SWARMON » project has given us reliable basic knowledge about web development<sup>35</sup>. This knowledge have helped us throughout the “MYR” project but was not enough. To assure correct use of the more “recent” and complex technologies<sup>36</sup> we had to learn new methods.

Our process of learning was iterative is the way that we have tried to gather the required knowledge little by little, making sure that the previous step was clearly understood.

---

<sup>33</sup> This iteration are called « sprint ». If the common development cycle is one big iteration following a V-shaped cycle, the agile development cycle is more shaped like a spiral made of several small iteration. See the French article of Wikipedia on this subject for more explicit pictures.

<sup>34</sup> In this case our teacher: Mr. Reynet. But later on it will be the WRSC2015 edition organizers: the Åland University of Applied Sciences.

<sup>35</sup> Knowledge about : Ruby on Rails, Ruby, JavaScript, jQuery, GitHub, HTML, CSS, SSH, SFTP ...

<sup>36</sup> Such as rake file and asset pipeline.

## B. Do less but do it better

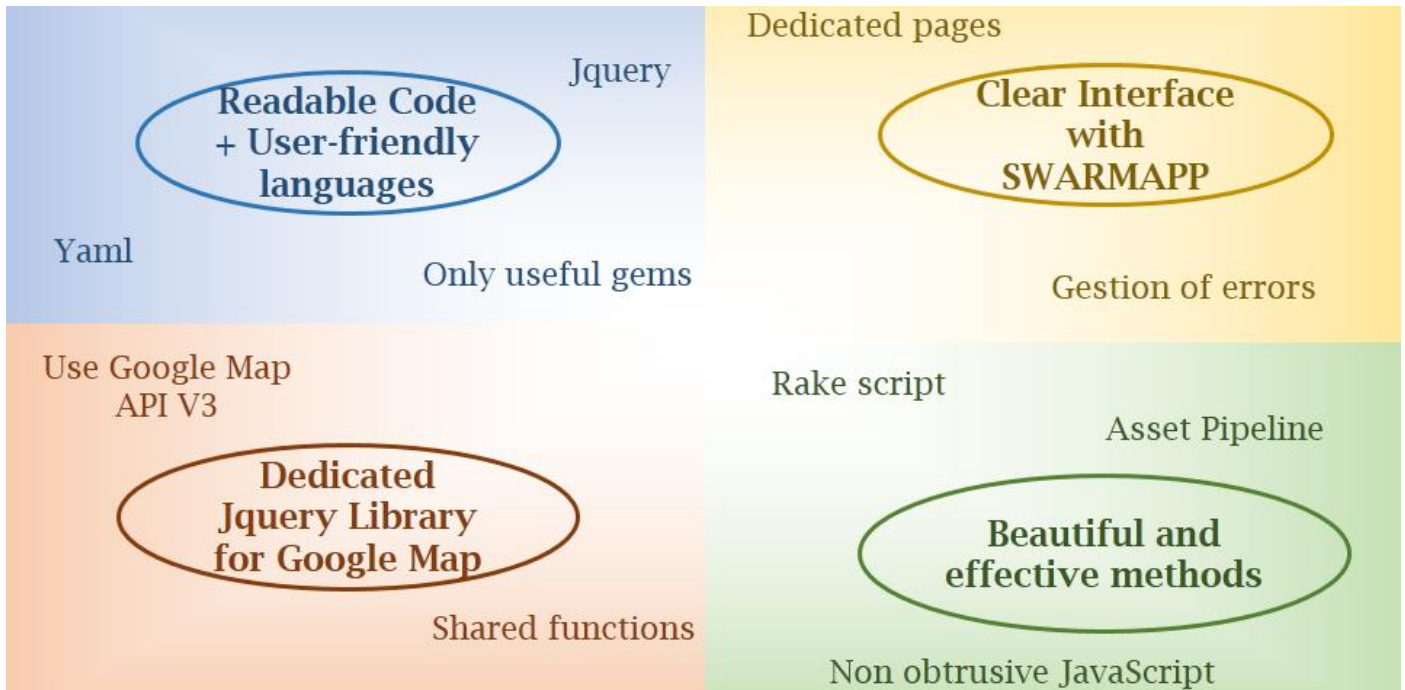


Figure 14: Monitor Your Robot Quality Commitment

Knowing the global lack of time which was impacting us, we preferred to do less but in way permitting later us of what we made.

- The code had been written in easy-to-read and easy-to-handle languages.
- The interface with SWARMAPP is well defined and fully tested
- A library of functions has been created to facilitate the handling of Google Map
- We used the best current-known technologies to assure performance and reliability in time.

## C. Future of the project

There is already an agreement between ENSTA Bretagne and Åland Applied Sciences University. Therefore this website<sup>37</sup> will be user during the WRSC2015. To unsure the fact that its development will be completed in time, a second-year internship is put in place. Based on our work, the chances are high that the website will be finished in time and implement all needed functionalities.

The present project could be a part of the IRSC2015<sup>38</sup>. The required criterion is that the report that we will submit is validated and judged relevant by the committee of the WRSC2015.

<sup>37</sup> And the previously developed tracking device.

<sup>38</sup> IRSC : International Robotic Sailing Conference. An assortment of conferences dealing with sailing robots matters which occurs just before the WRSC.

## *Conclusion*

Despite the lack of time we have succeeded to develop all major features. We provide a fully working static website generator and a new “real-time” web page based on Google Map API. We also provide a prototype of administrator interface which could be used during the competition. Our concern about clarity of coding and wise choice of technologies allows this project to be further develop and fully ready for the WRSC2015.

# *Bibliography*

## *Web development*

"JavaScript." Wikipedia. Wikimedia Foundation, n.d. Web. 15 Mar. 2015.

<http://en.wikipedia.org/wiki/JavaScript>

"Cascading Style Sheets." Wikipedia. Wikimedia Foundation, n.d. Web. 14 Mar. 2015.

[http://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](http://en.wikipedia.org/wiki/Cascading_Style_Sheets)

"HTML." Wikipedia. Wikimedia Foundation, n.d. Web. 14 Mar. 2015.

<http://en.wikipedia.org/wiki/HTML>

## *Map displaying solutions*

"Getting Started: Google Maps JavaScript API v3" Google Developers. N.p., n.d. Web. 14 Mar. 2015.

<https://developers.google.com/maps/documentation/javascript/tutorial>

"Google Maps JavaScript API V3 Reference." Google Developers. N.p., n.d. Web. 14 Mar. 2015.

<https://developers.google.com/maps/documentation/javascript/reference>

## *Agile development*

"Extra Credits: Game Design." YouTube. YouTube, n.d. Web. 14 Mar. 2015.

[https://www.youtube.com/playlist?list=PLhyKYa0YJ\\_5BkTruCmaBBZ8z6cP9KzPiX](https://www.youtube.com/playlist?list=PLhyKYa0YJ_5BkTruCmaBBZ8z6cP9KzPiX)

"Scrum (software Development)." Wikipedia. Wikimedia Foundation, n.d. Web. 13 Mar. 2015.

[http://en.wikipedia.org/wiki/Scrum\\_\(software\\_development\)](http://en.wikipedia.org/wiki/Scrum_(software_development))

"Méthode Agile." Wikipedia. Wikimedia Foundation, n.d. Web. 14 Mar. 2015.

*(Better illustrations than the ones of the English version)*

[http://fr.wikipedia.org/wiki/M%C3%A9thode\\_agile](http://fr.wikipedia.org/wiki/M%C3%A9thode_agile)

## *Ruby on Rails*

"Ruby on Rails Tutorial: Learn Rails by Example." Ruby on Rails Tutorial. N.p., n.d. Web. 15 Mar. 2015.

<http://french.railstutorial.org/chapters/beginning>

"The Asset Pipeline" Guide Ruby on Rails. N.p., n.d. Web. 15 Mar. 2015.

[http://guides.rubyonrails.org/asset\\_pipeline.html](http://guides.rubyonrails.org/asset_pipeline.html)

"Craic Computing Tech Tips." : Rails 3.2 Asset Pipeline. N.p., n.d. Web. 15 Mar. 2015.

<http://craiccomputing.blogspot.fr/2012/10/rails-32-asset-pipeline-requiretree-is.html>

"Rails/turbolinks." GitHub. N.p., n.d. Web. 15 Mar. 2015.

<https://github.com/rails/turbolinks>

## *Coding*

"jQuery API." Documentation. N.p., n.d. Web. 15 Mar. 2015.

<http://api.jquery.com/>

"Stack Overflow." Stack Overflow. N.p., n.d. Web. 15 Mar. 2015.

<http://stackoverflow.com/>

"Ajax (programming)." Wikipedia. Wikimedia Foundation, n.d. Web. 15 Mar. 2015.

[http://en.wikipedia.org/wiki/Ajax\\_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))

"JSON." Wikipedia. Wikimedia Foundation, n.d. Web. 15 Mar. 2015.

<http://en.wikipedia.org/wiki/JSON>



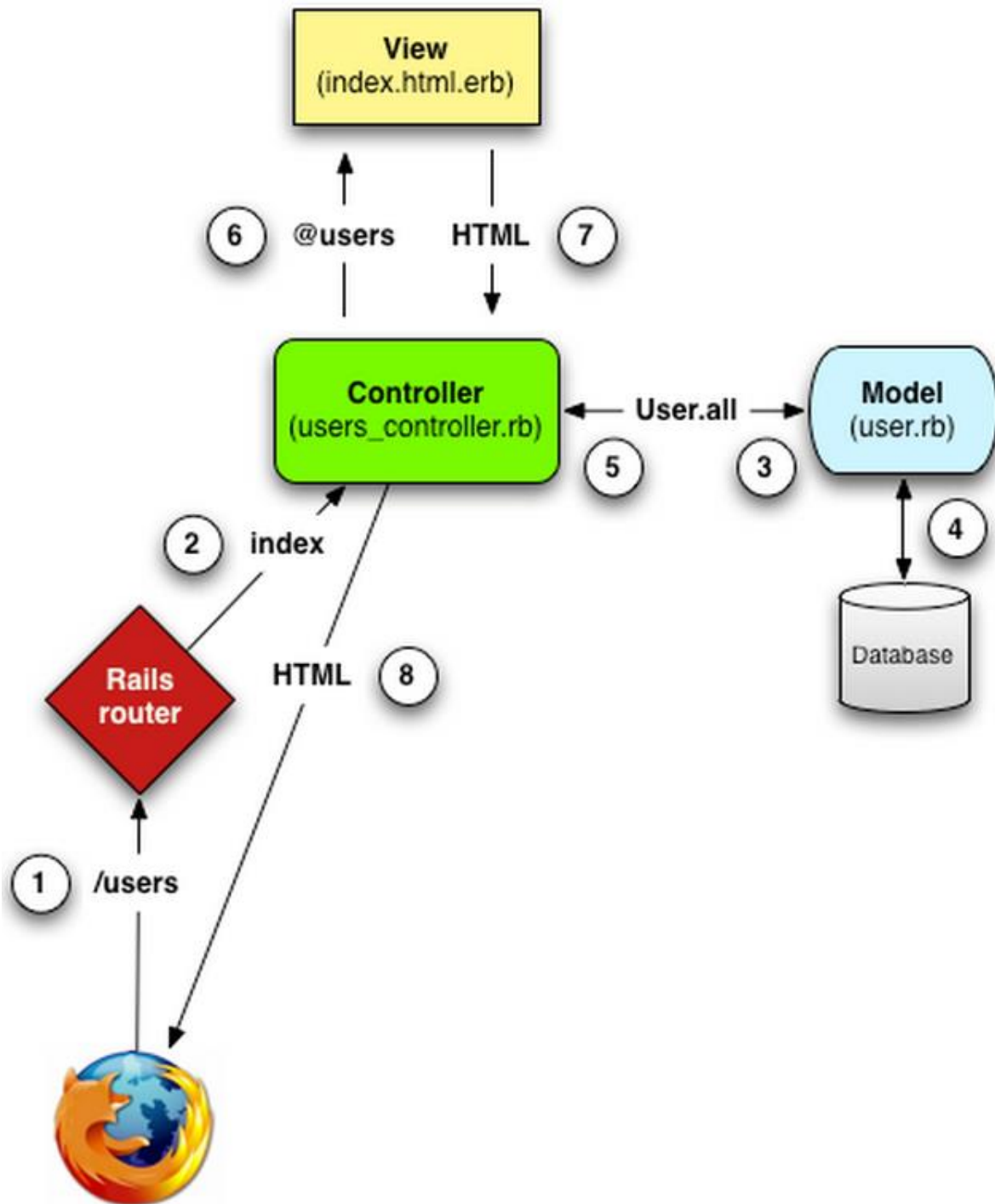
# *Annexes*

**Annex 1:** “Rails MVC in action”. Michael Hartl. Ruby on Rails Tutorial. Web. N.p., n.d. Web. 15 Mar. 2015.

**Annex 2:** State of the Art

**Annex 3:** Status Report on 06/03/2015

**Annex 4:** Status report on 16/03/2015



1. Le navigateur reçoit une requête pour l'URL `/users` ;
2. Rails route (*dirige*) `/users` vers une action **index** dans le contrôleur Users (*Utilisateurs*) ;
3. L'action **index** demande au modèle User de récupérer tous les utilisateurs (**User.all**) ;
4. Le modèle User tire tous les utilisateurs de la base de données ;
5. Le modèle User retourne au contrôleur la liste des utilisateurs ;
6. Le contrôleur place les utilisateurs dans la variable `@users`, variable qui est passée à la vue **index** ;
7. La vue utilise le code Ruby embarqué pour rendre la page au format HTML ;
8. Le contrôleur renvoie le code HTML au navigateur, qui affiche enfin la page.<sup>7</sup>

# *Status Report*

# *SWARMFRAME Project*



*BOURDON Benoit*  
*ENSI 2015*  
*Option SLS*

*DROUOT Bastien*  
*ENSI 2015*  
*Option ROB*

*Grammar and syntax correction: Ms. Northan Andrea*

# Table of contents

Introduction .....	3
I. The SWARMON project.....	3
a) Description of the global system.....	3
b) Description of the web site .....	4
c) The Architecture of the database .....	5
d) Existing gems for Ruby on Rails .....	5
II. Listing of the existing technologies .....	7
a) Website generators.....	7
b) Map displaying solutions.....	8
i. Other possibilities provided by Google Map.....	8
ii. Open Street Map.....	10
Conclusion.....	10
Bibliography .....	11
Web site generators .....	11
Map displaying solutions .....	11

# Introduction

This project aims at developing a website generator for the WRSC<sup>1</sup>. The generated website will be used to monitor a fleet of sailing robots during the competition. To do so, the server will receive localizations in real-time through HTTP requests and display them on a map allowing the users to follow the trajectories.

The website will be developed in Ruby on Rails, based on the last year project 'SWARMON'<sup>2</sup>. The new main feature will be an administrator interface to update or delete the localization data. The user interface will display the trajectories of the robots in real-time or in replay mode.

The report is divided into two parts. The first part focuses on the technologies used for the last-year SWARMON project. The second part details other existing technologies and analyses the prospect of using them for the SWARFRAME project.

## I. The SWARMON project

This part presents the current (and thus final) state of the 'SWARMON' project.

### a) Description of the global system

The system "SWARMON" is composed of two sides, the tracker side (on the robot) and the server side (linked to the Internet). The tracker (placed on the robot) collects the GPS positions and send them to the server. With the current hardware, ten GPS positions are sent each second. In this document we will not present the 'tracker side' because it will not change in 'SWARMFRAME'.

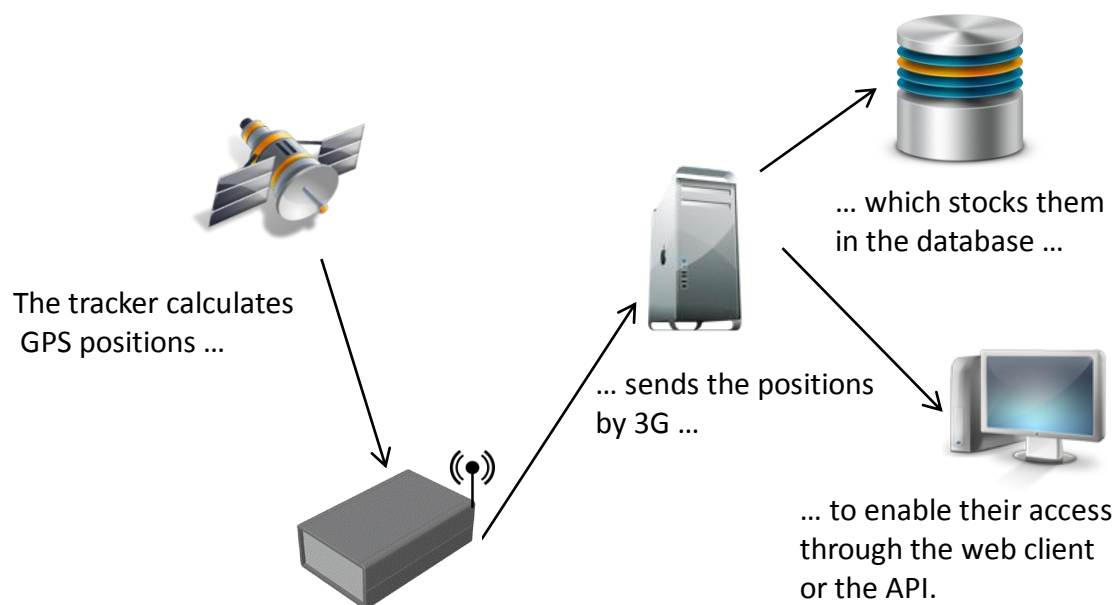


Figure 1: Global system presentation

<sup>1</sup> World Robotic Sailing Championship

<sup>2</sup> "SWARMON" means Swarm Monitoring

## ***b) Description of the web site***

To display the coordinates sent by the tracker a website has been developed (see: <http://haggis.ensta-bretagne.fr:3000/>). Only an internet connection is required to follow the robots using this website.

Two methods have been developed to follow one (or several) robot(s):

- **View:** To follow one (or several) boat in real time.
- **Replay:** To view previous positions of a boat (during a time span).

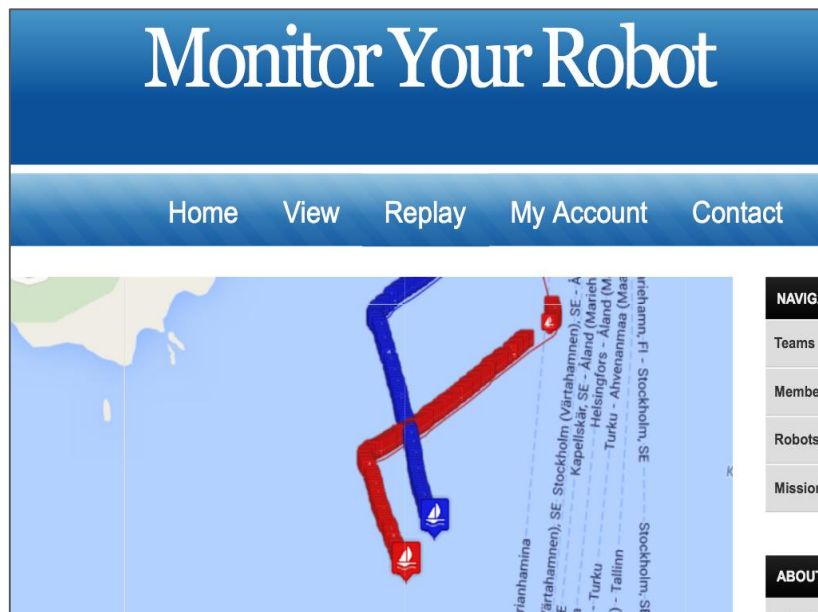


Figure 2: Screenshot taken during a sailing boat race

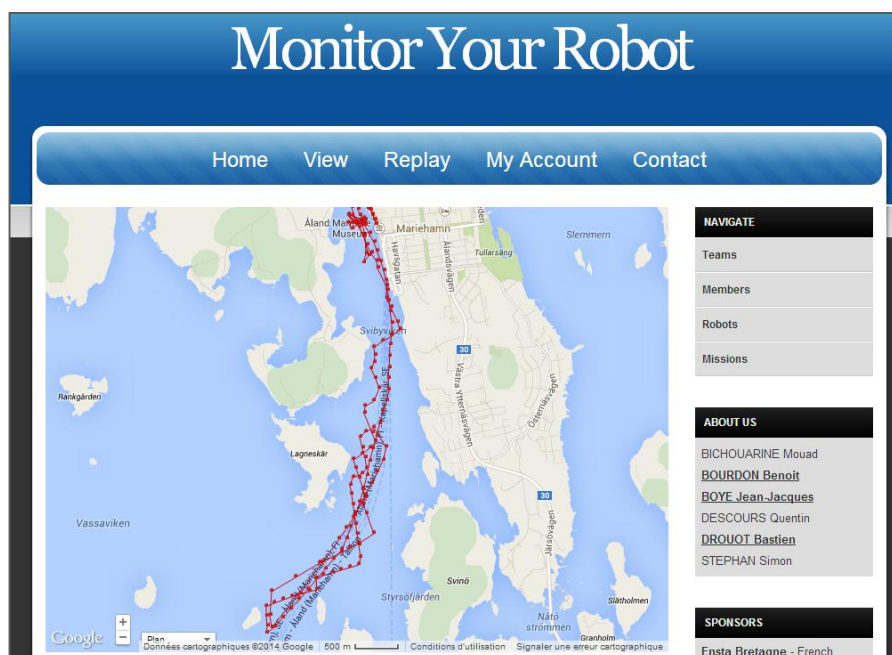


Figure 3: Screenshot of the replay mode

### c) The Architecture of the database

The database's architecture has changed many times during the development. Below is the final version of the database used on the website:

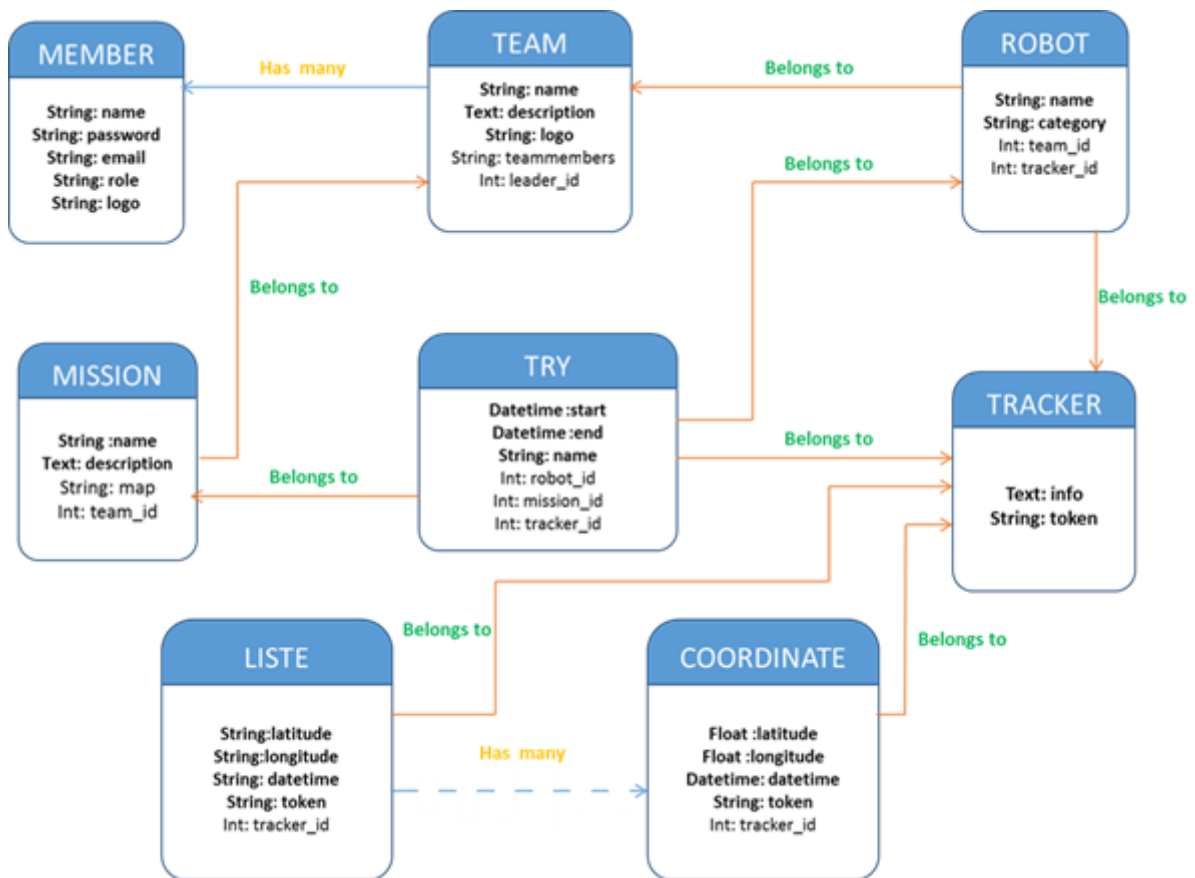


Figure 2: The Website's architecture

In this graph, each square is a class containing its attributes. The attributes in bold belong to their native class whereas the others are the foreign keys.

### d) Existing gems for Ruby on Rails

Several gems<sup>3</sup> have been used to achieve the 'SWARMON' project. In this status report we have decided to analyze in detail all these gems regarding their necessity for the 'SWARMFRAME' project. The result of this analysis points out the fact that a few gems are unnecessary and that some others could be used to improve the new website (they were implemented but not used in the former website). The documentation of the following gems can be found on their Github<sup>4</sup>.

<sup>3</sup> A « gem » is a library in Ruby programming language

<sup>4</sup> Github is a code hosting solution with advanced version management

Class	GEMS	FUNCTION	TO USE ?	NEEDED ?	WHY ?	NOTE	WHY ?	ALTERNATIVES
Core Items	rails	The framework	YES	YES	Nothing without it	5//5	Already known	Python with Django for instance
	sqlite3	Allows you to use a SQLite3 database	MAYBE	NO	Other solutions exist	?//5	Not tested	MongoDB ? No-SQL database ??
	gmaps4rails	Simplify the creation of a Google Map with overlays (markers, infowindows...)	MAYBE	MAYBE	Need for a map to locate things	3 // 5	Not complete	G Map: API/for Work/My Map, Open Street Map
	bcrypt	Secure algorithm for hashing passwords	YES	YES	Secure the passwordhandling	5//5	Really easy to use	MAYBE ?
	fastimage	use fastimage to get the size of external images	YES	YES	For remote profile image	5//5	Easy to use	Yes but to heavy or complex
Performance	thin	Change front end to Thin	MAYBE	NO	Is it really faster ?	4//5	Not tested	Yes, other frontend
	turbolinks	Turbolinks makes following links in your web application faster	MAYBE	NO	Helpful but small Website	3//5	Many bugs with JS	MAYBE
	uglifier	Ruby wrapper for UglifyJS JavaScript compressor.	NO	NO	Not enough Javascript	4//5	Seems efficient	MAYBE
	unicorn	Sets the default server for rack (and rails) to unicorn.	MAYBE	MAYBE	Can be faster	?//5	TO TEST	Rack by default
Facilitate the developpment	capistrano	Allows execution of commands in parallel on multiple remote machines, via SSH	MAYBE	YES	Allow administration and Use at the same time ?	?//5	TO TEST	MAYBE
	debugger	debugger is a fast implementation of the standard Ruby debugger debug.rb.	NO	MAYBE	Can be useful	5//5	By Default	NO
	tzinfo-data	Fix problem preventing from coding on Windows OS	YES	YES	Coders accustomed to Windows	5//5	NEEDED	NO
	faker	Generates fake data	YES	YES	Facilitate and accelerate the development	4//5	Easy to use	
	spring	Spring is a Rails application preloader wich speeds up development.	MAYBE	NO	Does it work on Windows OS ?	5//5	Seems simple	NO
	jbuilder	Jbuilder gives you a simple DSL for declaring JSON structures	NO	MAYBE	For checking the level of authentication easily	2//5	How can it be used ?	Classic Json writing
	sdoc	Generate the API Doc	NO	NO	Available on the Internet	?//5	Not tested	Internet
	sass-rails	Simplifies CSS coding	YES	YES	CSSis verbose	3//5	Others possible ?	MAYBE
Javascript	jquery-rails	Use jquery as the JavaScript library	YES	NO	Simplify the writing of Javascript code	5//5	Powerful, simple	Coffee script (strange syntax ?)
	jquery-cookie-rails	Allows cookies to be used in jquery	YES	YES	Allows information sharing between Views	5//5	The only one	NO
	jquery-turbolinks	jQuery plugin for drop-in fix bound events problem caused by Turbolinks	YES	MAYBE	Fix bug between jquery and turbolinks	4//5	Still some problems	MAYBE
	CoffeeScript	CoffeeScript is a little language that compiles into JavaScript	NO	NO	We prefer to manipulate DOM object with JQuery	3//5	Strange Syntax	Jquery
	therubyracer	Javascript in Ruby and Ruby in Javascript ?	NO	NO	Did not know that it exists. Maybe useful ?	3//5	Strange Syntax	NO
	therubyrhino	Embed the Mozilla Rhino JavaScript interpreter into Ruby	NO	NO	Did not know that it exists. May fix bugs on Mozilla.	?//5	TO TEST	NO

Figure 3: Gem analysis



## II. Listing of the existing technologies

To sort the different technologies we use the following criteria:

- The relevancy to the problem raised by the project
- The difficulty to master the technology
- The time needed to implement

There is a very large number of technologies which can be used in Web development. Confronted with the impossibility of listing them all we prefer to sort them into few generic categories. Then we test each category of technologies to confirm whether or not it matched our needs.

We based our primary analysis on two topics: the website generators and the mapping solutions. We have discussed the relevancy and the efficiency of these technologies but some of them still need to be further tested.

### a) Website generators

Due to the expansion of the Internet, many tools are currently available to generate a website easily. Nowadays everybody, even without any knowledge about coding, can create a good-looking efficient website. For example WIX offers to create a website from a framework and to adapt this pattern to our needs.

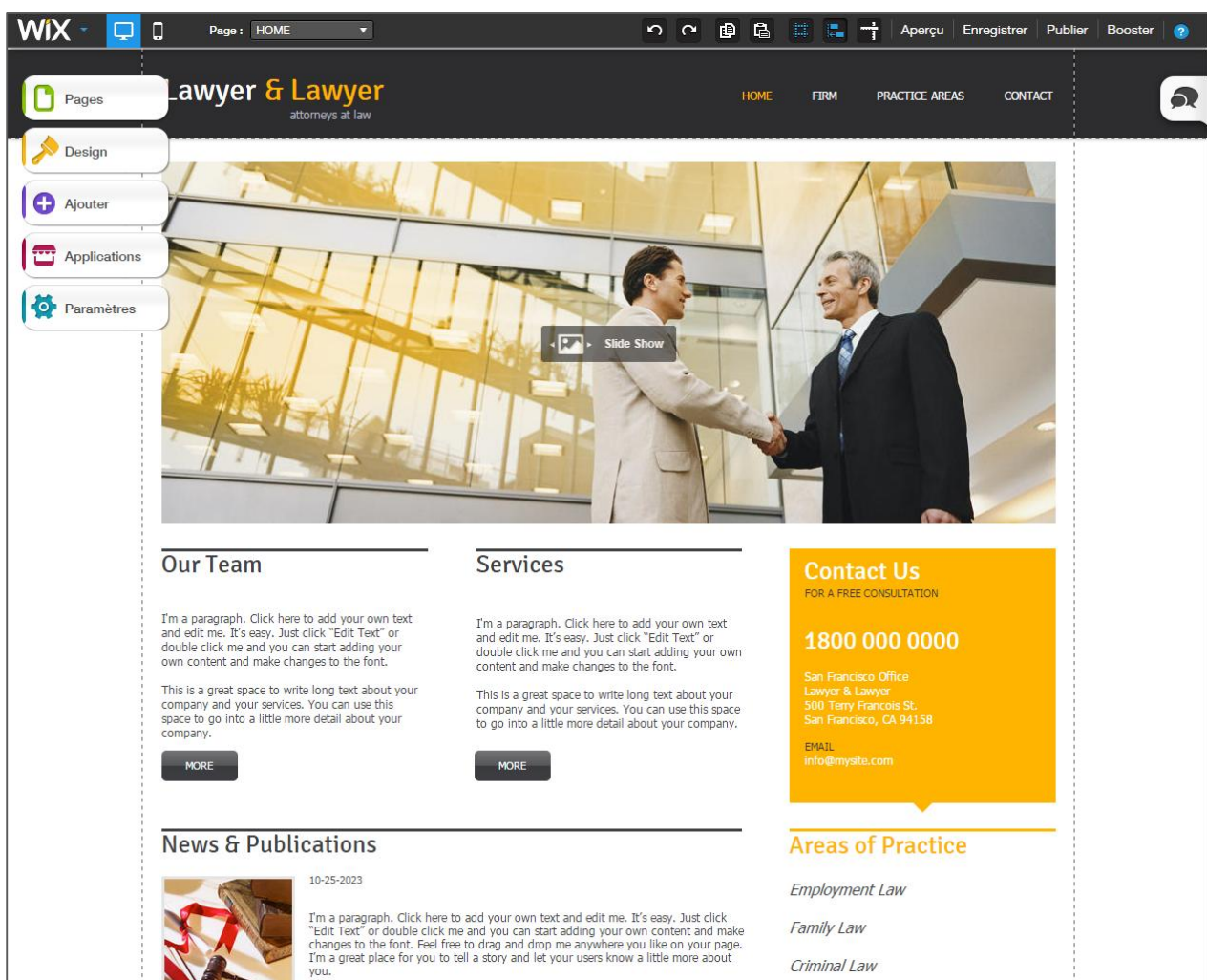


Figure 4: An example of website generator

However, all of these tools (we did not find any exception) only generate static websites based on a set of predefined patterns. There is no possibility to communicate with a database. That means no possibility to store data to create a profile account or implement any kind of [clean] replay-mode. The only 'dynamic' interaction that you can have with your website is to modify the source code of a page (to add a picture or a block of text for instance). That is why our website's project cannot be based just on a website generator, but it may be possible to implement some parts of this generator to improve the Graphical User Interface.

Indeed, one optimal solution could be to use a website generator as a frontend to call the database <sup>5</sup>with basics HTTP requests. However this implementation would take a lot of time due to the fact that we first need to understand in detail how a website generator works from the inside. Moreover the interface between the frontend and the database would be a totally new way of coding for us and would also be against the simplicity brought by Ruby on Rails. For these two reasons and for the overall lack of time we prefer to keep developing with well-known technology: Ruby on Rails.

## ***b) Map displaying solutions***

The 'SWARMON' project was based on a gem (a library in the Ruby world) named 'gmaps4rails'. The main asset of this library was to be a quite effective tool helping the developers to interact with Google Map. It really simplifies the communication between the controllers and the views (Model View Controller pattern of Ruby on Rails) to create a map, add markers or generate complex 'info windows'<sup>6</sup>. However at the end of the 'SWARMON' project, this tool started to show its limits due to the lack of freedom left to the developers. If this tool is perfect to add a restricted numbers of markers on a map it does not seem to be suitable for requests aiming at displaying thousands of points. In such a configuration (a very common and normal situation in fact for SWARMON') the performance drops dramatically due to bad strategy of marker addition. The refreshing of the map was indeed a total reconstruction of the map and of the set of markers and not just a simple modification based on the previous state of the map. Despite the in depth-research made by the 'SWARMON' team on this problem, no solution was found<sup>7</sup>. Maybe deeper inquiries could solve this problem but we truly need to consider other options even if it means to adopt a different and 'not yet mastered' technology.

### **i. Other possibilities provided by Google Map**

The first solution could be to directly use the Google Map API <sup>8</sup>to modify the map and thus having a wider handling of the map modification process. The main benefits of this solution are that the Google Map API is well documented and that it should solve all the problems brought by 'gmaps4rails'. One last pitfall is that the process of adding several markers on a map without reconstructing it totally has never been achieved during the tests accomplished by the 'SWARMON' team. Thus, more tests need to be conducted to ensure the fact that the solution is feasible and will solve the problem.

---

<sup>5</sup> Not generated by the website generator

<sup>6</sup> A kind of small on-demand window providing information about the selected marker

<sup>7</sup> The documentation of this gem is unfortunately some way from being complete

<sup>8</sup> Application Programming Interface

Another solution could be to use ‘Google Map for Work’. At this point we do not have much information about this product (because we need to obtain a license to test it) but the Google website and some tutorials on the Internet have led us to believe that it is a very powerful tool facilitating the use of the Google Map API. Based on our research it is probably a high level interface providing a GUI<sup>9</sup> to the developer. This tool indeed seems to look like the ‘Google My Map’ solution but with more assets offered and the ability to dynamically use it on a website<sup>10</sup>. The biggest obstacle to the use of this tool is that it requires a license and that if although is quite simple to obtain one when living in the United States, a special request has to be made for foreign companies. These two solutions need to be tested further in order to make a reasonable choice between them.

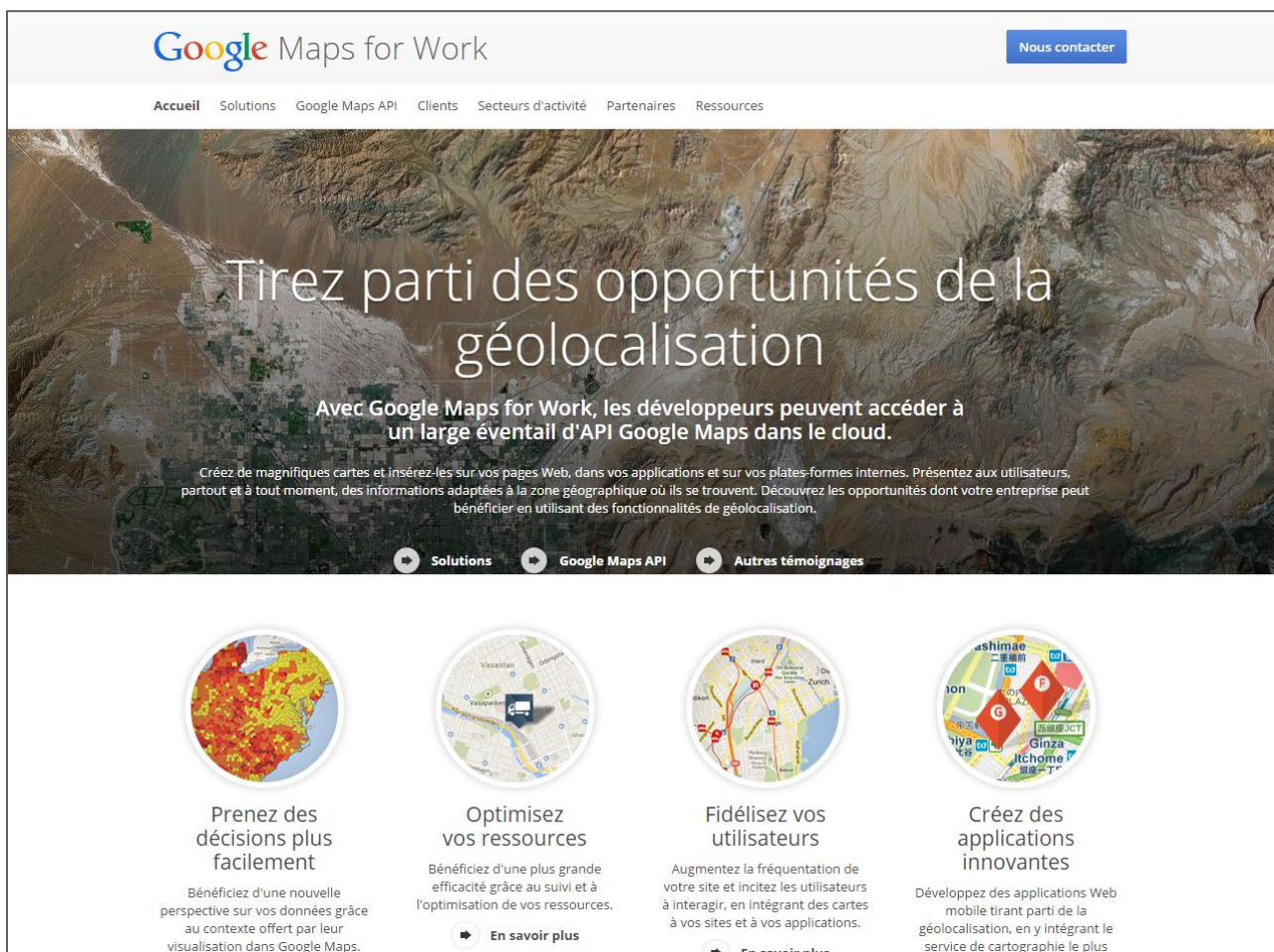


Figure 5: Homepage of Google Maps for Work

<sup>9</sup> Graphical User Interface

<sup>10</sup> It seems not to be possible with ‘Google My Map’ due the fact that the generated map is a link and not a javascript object

## ii. Open Street Map

Another solution could be to use 'Open Street Map' and its API. The solution is open source and is very similar to the one provided by Google. Open Street Map even seems to provide a better render quality (especially for marine area) and a more detailed listing of the Point of Interest (added by the community). This alternative could also be a very good solution but we first need to test the API and to check the possibilities offered by Open Street Map.



*Figure 6: Open Street Map banner*

## Conclusion

This document has shown that the solid foundations of the 'SWARMON' project provides a good starting point for the 'SWARMFRAME' project. The remaining flaws of this project could find their solutions but at this point we can only advise doing more tests in order to take a decision. This time consuming process has not been started yet but is our priority.



# *Bibliography*

## *Web site generators*

"Create YourStunning Website.It's Free." Free Website Builder. N.p., n.d. Web. 25 Feb. 2015.  
<http://www.wix.com/>

"Weebly Is the Easiest Way to Create a Website, Store or Blog." Weebly.com. N.p., n.d. Web. 25 Feb. 2015.  
<http://www.weebly.com/>

"StaticGen." Top Open-Source Static Site Generators -. N.p., n.d. Web. 25 Feb. 2015.  
<https://www.staticgen.com/>

"Website Builder | Web.com." Website Builder | Web.com. N.p., n.d. Web. 25 Feb. 2015.  
<http://web.com/>

"Build a Website." Squarespace. N.p., n.d. Web. 22 Feb. 2015.  
<http://www.squarespace.com/>

## *Map displaying solutions*

"Adding a Google Map to Your Website." Google Developers. N.p., n.d. Web. 25 Feb. 2015.  
<https://developers.google.com/maps/tutorials/fundamentals/adding-a-google-map>

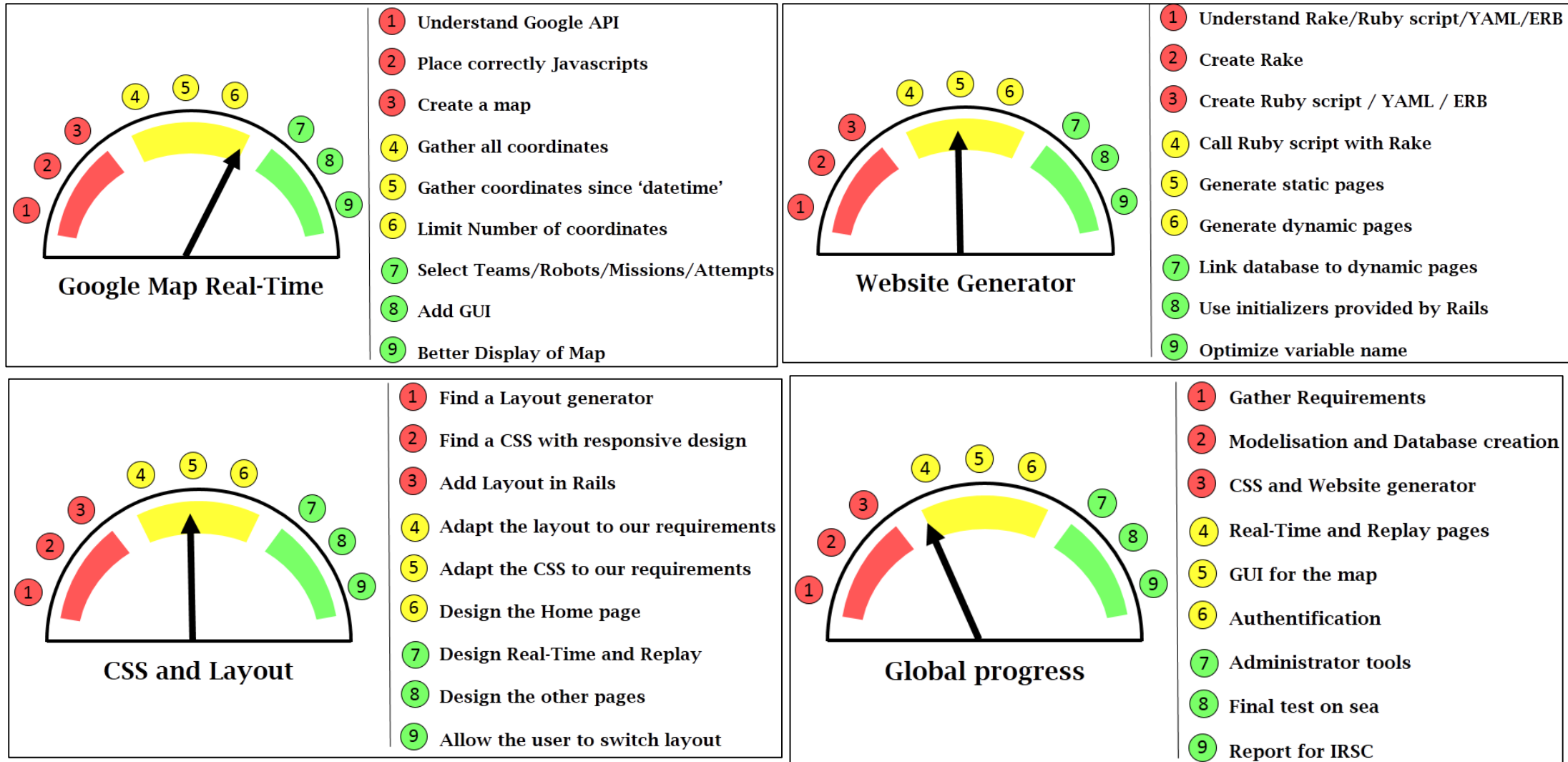
"Maps for Work." Google Maps API for Work. N.p., n.d. Web. 25 Feb. 2015.  
<https://www.google.com/work/mapsearch/products/mapsapi.html>

"My Maps." My Maps. N.p., n.d. Web. 25 Feb. 2015.  
<https://www.google.com/maps/d/>

"OpenStreetMap." OpenStreetMap. N.p., n.d. Web. 25 Feb. 2015.  
<http://www.openstreetmap.org/>

"OpenStreetMap France." OpenStreetMap France. N.p., n.d. Web. 25 Feb. 2015.  
<http://openstreetmap.fr/>

## STATUS REPORT "MONITOR YOUR ROBOT" on 06/03/2015



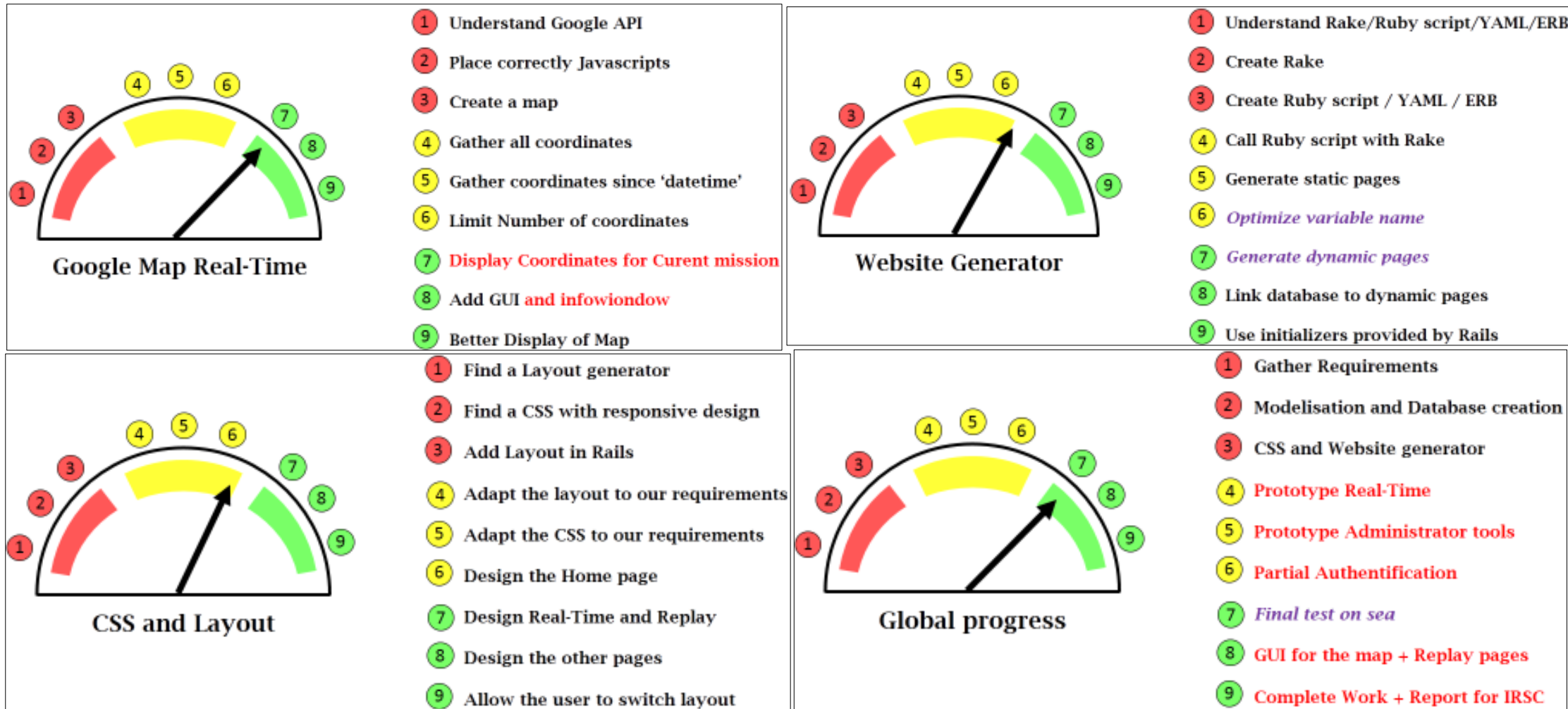
Legend: **RED** means essential features // **YELLOW** means core features // **GREEN** means non-essential features

Despite the initial delay, the project is back on track and globally in time. All major technological deadlock have been overcome.

We can currently provide a generator of static websites including a basic map to display the coordinates.

Next step of the project is to provide a generator of dynamic websites and a Graphical User Interface to add coordinates on the map.

## STATUS REPORT “MONITOR YOUR ROBOT” on 16/03/2015



Legend: **RED** means essential features // **YELLOW** means core features // **GREEN** means non-essential features

**text in red means:** task has been modified // **text in purple means:** order of priority has been modified

Despite the lack of time we have succeeded to implement all core features. Our goal has been to code less functionalities but with a better overall quality. What we developed is clean enough to allow future development.

At the end of the project we provide: - a fully-working and adjustable generator of static websites  
- a map displaying in real-time coordinates of the current mission