CREATING A WEB SPACE FOR COMPANIES WITH SOCIAL IMPACT

Author: Renato Pinto López
Tutor: Mª Victoria Rodellar Biarge

Madrid, January 7th, 2017
A los profesores que entendieron lo que era enseñar.
A los amigos por las noches de mucho estudio y de ninguno.
A los compañeros de trabajo por ayudarme desde este y otros continentes.
A mi familia por el apoyo incondicional durante todos los años de estudio y de trabajo.

A todos, bien trabajado.
ABSTRACT

Nowadays, a digital identity is a basic element for any company. It is necessary to correctly structure and define the website so it is found within the top few results in online searches, and that search engines know what information they have to show. Moreover, it is also necessary that the website presents the information in an intuitive and easy manner, in order not to reduce the audience.

In this project we undertook the creation process of a website from scratch, taking into account interaction with final user, current technology used to browse the website, and then we adapted it to a number of sensorial disability the user may suffer.

Before we started developing this project, a study was carried out on available technologies and relevant regulations. This way we define a structural base level for the development of future modifications. These modifications will have to follow the same process of research and development.

In order to demonstrate that regulations have been followed we have presented results obtained from official entities test tools, as well as further analysis of the performance against search engines.

Keywords: landing-page, website, responsive, cross-device, cross-browser, accessibility, SEO
RESUMEN

A día de hoy, presentar una identidad digital es un elemento básico para cualquier empresa. Es necesario estructurar y definir correctamente la página web para que pueda encontrarse en los primeros puestos en las búsquedas, y que los motores de búsqueda sepan qué información mostrar. Además, también es necesario que en el espacio web se muestre la información de manera fácil e intuitiva, y no limitarla a un público reducido.

Este proyecto desarrolla el proceso de la creación de un espacio web de una empresa desde cero, teniendo en cuenta la interacción con el usuario final, las tecnologías web actuales, y la adaptación a cualquier tipo de minusvalía sensorial que pueda presentar el usuario.

Previo al desarrollo técnico de este proyecto se ha realizado un estudio de las tecnologías disponibles y de las normativas pertinentes, de tal forma que quede definida una base estructural de desarrollo para futuras modificaciones. Estas modificaciones deberán seguir el mismo proceso de estudio y desarrollo.

Para demostrar que se han seguido las normativas hemos presentado los resultados obtenidos a través de pruebas proporcionadas por entidades oficiales, así como un análisis posterior del rendimiento frente a motores de búsqueda.

*Keywords: espacio web, responsive, cross-device, cross-browser, accesibilidad, SEO*
1. Introduction

This project’s objective is to create a web space for social impact companies. But first of all, what is a social impact company?

A social impact company is one which is born with the objective of resolving a social challenge (education, resources, energy, environmental…) for a specific collective while generating revenue.

Following that line of thought, we want our site to be as well-known as possible, and easy to access and use for any person, regardless of their disabilities (such as dyslexia, blindness or low vision, learning disabilities, cognitive limitations…)

As a real life example for the project, we are going to develop the web space for the company “Bridge for Billions, Inc.,” although the techniques used through this process can be apply to any other kind of site.

“Bridge for Billions, Inc.” seeks to democratize access to entrepreneurship, matching mentors looking for fulfilling projects with motivated entrepreneurs looking to fulfil their dreams. This is possible with the company’s web application, which provides the tools necessary to structure, develop, and refine their businesses.

In order to achieve this goal, we are going to design and develop the landing site of the company so it becomes easier to find and understand, and so that new users, mentors, and entrepreneurs, start using the web application.

2. Analysis of Requirements

Through the analysis of the activities that were described in the document “Plan de Trabajo” we are going to break down the requirements on which we will base this project.

These requirements will be presented in “requirement cards” inspired by the book “Mastering the Requirements Process” [1].
However, the cards we will be using will be slightly different, so our cards will resemble those we used in the course “Software Engineering I.”

The meaning of each section is as follows:

- **Requirement #:** identification number for each requirement.
- **Classification:** part of the system the requirement affects.
- **Type:** whether it is functional or not functional.
- **Description:** short sentence that describes the requirement.
- **Rationale:** a justification of the requirement.
- **Originator:** the person who raised the requirement.
- **Priority:** a rating of the value to the customer, with 0 being of highest importance and 3 of least importance.
- **Dependencies:** other requirements on which it depends.
- **Conflicts:** other requirements that cannot be implemented if this one is.
- **References**: points to documents that illustrate and explain this requirement.
- **History**: history of the changes to the requirement (creation, changes, deletion...)

The requirements we must meet are presented on the following cards:

- **Responsive Web Design (RWD)**

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>1</th>
<th>Classification: design</th>
<th>Type: functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should have responsive behaviour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>web traffic will come from any kind of device, so the website should feel and look good from any of those devices (desktop, tablet, mobile...) while working well.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>final client</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>#2, #3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 – creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Cross-device**

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>2</th>
<th>Classification: design</th>
<th>Type: functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should be able to be viewed correctly from any device</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>web traffic will come from any kind of device, so the website should feel and look good from any of those devices (desktop, tablet, mobile...) while working well.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>final client</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>#1, #3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 – creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Cross-browser**

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>3</th>
<th>Classification: design</th>
<th>Type: functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should be able to be viewed correctly from any browser</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>web traffic will come from any browser, so the website should feel and look good from any of those browsers (Chrome, Firefox, Safari...) while working well.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>final client</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>#1, #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 – creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Comply with HTML W3C regulations [2]**

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>4</th>
<th>Classification: structure</th>
<th>Type: non-functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should pass HTML W3C validators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>search engines give points to well-structured websites and move them up in the search rankings.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>developer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>no dependencies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 – creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Comply with CSS W3C regulations [3]

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>5</th>
<th>Classification: structure and design</th>
<th>Type: non-functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should pass CSS W3C validators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>search engines give points to well-structured websites and move them up in the search rankings.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>developer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>no dependencies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 – creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Comply with WCAG 2.0 regulations [4]

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>6</th>
<th>Classification: accessibility</th>
<th>Type: non-functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should pass WCAG 2.0 validators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>we need to keep in mind a wide range of people with disabilities, including blindness and low vision, deafness and hearing loss, learning disabilities, cognitive limitations, limited movement, speech disabilities, photosensitivity and any combinations of these.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>developer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>#4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 – creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Multilingual website

<table>
<thead>
<tr>
<th>Requirement #:</th>
<th>7</th>
<th>Classification: accessibility</th>
<th>Type: functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>the website should be visible in more than one language.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td>the website should offer the possibility of multiple languages to maximize its visibility.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Originator:</td>
<td>final client</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority:</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>no dependencies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflicts:</td>
<td>no conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References:</td>
<td>[point to document in bibliography]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td>v1 - creation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Study of Solutions in Similar Applications

In order to evaluate the competitor’s websites, we will examine if they fulfil the requirements set out for this project.

To protect the privacy and avoid legal inconveniences, we will refer to these competitors as “Competitor” followed by the corresponding capital letter.

3.1. Competitors

After analysing the main competitors, we can assert if they comply the specified requirements:
<table>
<thead>
<tr>
<th>Req. #1</th>
<th>Req. #2</th>
<th>Req. #3</th>
<th>Req. #4</th>
<th>Req. #5</th>
<th>Req. #6</th>
<th>Req. #7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Competitor A</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Competitor B</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Competitor C</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Competitor D</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Competitor E</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Competitor E</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

**Figure 3 - Competitors checklist**

As we can observe, when it comes to complying with regulations, it does not seem to be a priority. This can provoke worse results in search engines (requirements # 4 and # 5), so the website visibility will be lower. The content of the website will note be easily accessible to certain kinds of users, specified in requirement # 6. Moreover, having a monolingual website will reduce the impact of the company in foreign countries (requirement # 7), which also decreases its position in search engine’s rankings.

### 4. Solution Proposal

We are going to use various external tools to validate our solution when possible.

For requirement # 4 and requirement # 5 we have specific official validation services than we can use for free.

Both of them work by writing the desired URL into its only input box, which returns a list of errors and warnings based on the code found inside said URL. For HTML validation we have the W3C – HTML validator [5], and for CSS validation we have the W3C – CSS validator [6].

Moving on to requirement # 6, there is no one official validation tool, but rather many. After analyzing a list of them, it has been decided that we will focus on the results returned by the tools Tawdis [7], Wave [8], and the audit performed by “Accessibility Developer Tools” extension for Google Chrome [9]. This decision is due to their ease of use, their speed, and the reliability of the results obtained.

Requirements # 1, # 2, # 3 and # 7 will be checked manually.

As for maximizing the visibility of the site, we will work with SEO techniques and create a report analysing what has been implemented, what can be improved, and what is yet to be implemented.

Upon the end of this project, we will use SERP checkers to ascertain our position in comparison to the competitors’ positions.

The landing page must resolve the following doubts experienced by the user:

- What do we do?
- How do we do it?
- Where do we come from?

In order to answer this questions, there must be separate sections for each one of them.
The question “**What do we do?**” needs to be answered as soon as the user gets into our website, so this will be clarified in the index page. We need to assure the user that s/he has arrived to the right place, and after that retain her/him there.

The second question, “**How do we do it?**” needs to be explained in a way that the user will quickly and easily understand.

“When we’re creating sites, we act as though people are going to pore over each page, reading our finely crafted text, figuring out how we’ve organized things, and weighing their options before deciding which link to click.

*What they actually do most of the time (if we’re lucky) is glance at each new page, scan some of the text, and click on the first link that catches their interest or vaguely resembles the thing they’re looking for. There are usually large parts of the page that they don’t even look at.*” [10]

![Figure 4 - User behaviour comparison](image)

We will include a design capable of showing a general path of the performance of the web app while at the same time showing specific information for a more interested user that will actually stop and read carefully the information provided.

The third and final question, “**Where do we come from?**” should be treated as a way to give the user confidence in the company. For this, we will create a specific section to talk about the team and the company background, as well as present other users/companies that have trusted in the company’s project.

Those will be the main factors that will influence the content and structure of the site that we are going to develop.

As we are using a real case for this project, we will also include whatever section the company believes to be necessary, following the same line of work for their structure and design.

5. Solution Proposal Development

We have already explained what will be necessary for the design. This section goes into greater detail about the necessities explained above.
As said before, the first question “What do we do?” will be answered in the index page. The second question “How do we do it?” will be separated in two different pages. Because we have two main user types (mentor and entrepreneur), we need to answer the question for both. Thus, we will build the page “how-it-works-entrepreneurs.html” and the page “how-it-works-mentors.html”. The third and final question “Where do we come from?” will also have multiple pages. The page “about-us.html” should give the team and company background, while the page “our-impact.html” will provide the visitor with reviews from previous users.

As mentioned earlier, there is also content requested by the company. In this case, apart from the basic pages we decided to build, we will also create a section specific for big organizations and another for product pricing. In order to increase trust from the visitor, there will also be an FAQ page. Finally, we will need to add a page to show the visitor our terms and conditions.

In this first iteration, the website map should look like in figure 5, having every file indexed to the root directory.

![Figure 5 – Sitemap v1](image)

With the requirements we set out, it is not necessary to develop a highly complex website. However, we will need to have the website correctly structured in order to be scalable for future iterations.

The development of a website can be as easy as writing a couple files with their own HTML, CSS and Javascript code. But this kind of approach is not scalable, nor efficient. In fact, this will come into conflict with one of the principles we should follow when developing software: D.R.Y. (Don’t Repeat Yourself).

Bearing this in mind, the technology we will use to build our website will be Middleman [11]. Middleman is a static website framework that offers the capability to create all the elements needed in this project. There is a great variety of options to easily customize the project, so we can focus in the actual development of the site.

The default Middleman installation consists of a directory structure illustrated below:
While the structure can be slightly modified to achieve our goals, we will first explore the default structure more closely.

5.1. Middleman directory structure

The root directory is called “mymiddlemansite” by default. This should be changed to have a more meaningful name for our project. Inside this root directory we see the following files:

- `.gitignore`: this file specifies intentionally untracked files that Git should ignore. This becomes useful when we work in a team and use Source Code Management (SCM) software. For example, we wouldn’t want to synchronize files inside the directory “build/”, so that directory will be specified in this file.

- `Gemfile`: a gemfile describes the gem dependencies required to execute associated Ruby code. Here we will specify not only which gems we want to use, but also their version.

- `Gemfile.lock`: the `Gemfile.lock` file is where Bundler records the exact versions installed. This way, when the same project is loaded on another machine, running `bundle install` will look at this file and install the exact same versions, rather than just using `Gemfile` and install the most recent versions. We will not edit this file directly because it is automatically updated after installing gems using the previous file.

- `config.rb`: in this file we will be able to specify the configuration needed for the project. For example, if we want to create an optimised production code, we can write the next piece of code:

```
config :build do
  activate :minify_css
  activate :minify_javascript
```

This code will minify our CSS and Javascript code only when we build the production code. This lets our website load faster and have a better punctuation for SEO and SERP. We will go into that in following chapters.

- `source/`: this directory contains our main website source files to be built, including templates, Javascript, CSS and images.

- `images/`: this directory will be used to store our image documents.
- **index.html.erb**: this will be the main file for our source code. Whenever we build a website, the file “index.html” located in the production root directory will be the first page to show up to the user or the user-agents (also known as robots [13]). The suffix “erb” refers to *Embedded Ruby*, a feature of Ruby that enables you to generate any kind of text, in any quantity, from templates. The templates themselves combine plain text with Ruby code for variable substitution and flow control, making them easy to write and maintain.

- **javascripts/**: this directory will contain our Javascript files. The file “all.js” will include a list of the different custom Javascript files so we can create multiple scripts.

- **layouts/**: this folder will contain the layouts used in our site. Layouts allow the common HTML surrounding individual pages to be shared across all our templates. In example, “header” and “footer” includes what we will refer at the top and bottom of every page, so our main file, “layout.erb”, will include both the header and footer, and the wrap the individual page content. Typically, the file “layout.erb” will look like this:

```
<% yield %>
</html>
```

*Figure 8 - layout.erb*

The yield call in line 6 will be where templates will place their contents. Given an index.html.erb template:

```
<h1>Hello World</h1>
```

*Figure 9 - index.html.erb example*

The final output in HTML will be:

```
<html>
<head>
<title>My Site</title>
</head>
<body>
<h1>Hello World</h1>
</body>
</html>
```

*Figure 10 - index.html example*

- **stylesheets/**: this directory will store our Sass [14] files. Notice that we are not talking about CSS files here. The reason for this is that Middleman does not work directly with CSS files, but with Sass.

As we do with the templates using ERB files, the fact that we can use a preprocessor like Sass will facilitate enormously the implementation of our custom styling. Instead of having a huge unreadable “.css” file encompassing everything we need for the site, we will have multiple “.scss” files to separate our styles, for example, in colours, fonts, and the pages we will implement (index, our-impact...).
Like we do in the directory “javascripts/”, all of these files will then be added into a list in all.scss, and at the moment of building the production code, this will create a unified “.css” file into the directory “build/”.

- **build/**: although this directory is not yet integrated into our structure, we should be familiarized with it. What we achieve with the use of Middleman is to speed up the process of development and maintain the development code separated from the production code. Once we finish building our site, we will create the production code in this directory.

Now that we understand the default structure provided by Middleman and how it works, we are going to review our requirements to have a solid base for the project.

- **Requirement #1**: in order to have a responsive website, we will work with the framework Bootstrap [15]. This framework is used for responsive development and mobile-first projects on the web.
- **Requirement #2**: for this requirement, we will also use Bootstrap.
- **Requirement #3**: regarding this third requirement, there are two options to evaluate. We can implement the code having in mind browser compatibilities and adapting our workflow to it, or we can use an external library to detect the browser and adapt our code to each of them.

In the first case, we will consult the website caniuse.com [16] to know which browsers can handle our implementation and determine if our approach to certain issue is correct with regard of this requirement.

For the second option, we will use Modernizr [17], a Javascript library that contains a collection of tests which run as the site loads, detecting which functionalities can be used.

Considering that our project is intended to be simple, light, user-oriented, and accessible, we believe that adding an extra external library will go against our objectives, so we will go with the first option.

- **Requirement #4**: to comply with this requirement we will consult the HTML reference from W3Schools [18]. Following this reference will result in a better development of our HTML code, facilitating our compliance with this requirement. This will not affect our project structure.
- **Requirement #5**: to comply with this requirement we will confer the CSS reference from W3Schools [19]. Following this reference will result in a better development of our CSS code, facilitating our compliance with this requirement. This will not affect our project structure.
- **Requirement #6**: as stated before, we will use external tools to test if we comply with this requirement, and we will correct any detected issue as they appear. This will not affect our project structure.
- **Requirement #7**: another reason for the use of Middleman is its already mentioned capability to be configured with plenty of modules. In respect to this requirement, we will add the following line into the file “config.rb”:

```ruby
activate :i18n, :langs => [:es, :en, :fr], :mount_at_root => :en, :path => "/:locale/"
```

Figure 11 - Localization (i18n)
What we are doing here is activating the “:i18n” extension for internationalization and localization. This way, we can translate the content of our site in as many language as we specify.

By default, this extension will search the “locales” directory in the root of the project for YAML files representing each locale we want to support, so we will have to add this folder to our middleman structure.

The YAML file is a set of key-value pairs for each string that will be needed in our site. Here is an example of how two different YAML files should look like:

```
# locales/en.yml
---
en:
  hello: "Hello"

# locales/es.yml
---
es:
  hello: "Hola"
```

As seen, we first state the language code and then start writing the key-value pairs. Notice that the key must be the same in both files; only the value will be different. We will use the key to identify the string in our templates, and show the value in the final production code.

After activating the extension, we indicate which languages will be included. We do that by inserting the language international code into an array of languages.

Ideally, we should have a directory for each language and one of those directories should be set as default, but that will leave us with an empty root directory, which is bad for search engines. User-agents will start looking for `root/index.html` and will not find anything, so our ranking position will decrease. Again, we will explore the SEO portion in the following chapters.

To avoid that issue, we use the configuration option “:mount_as_root” to designate English as the default language, and finally the option “:path” to identify the path where we will store the content in different languages.

By default, the contents of “source/localizable” will be built in multiple languages, while the rest of our templates will continue to work normally. While we can change the name of this folder to one of our choice, we will need to add another folder to our structure so we can have the content in multiple languages.

Now that we have revised the requirement, we can have a better grasp of how we will need to structure our project. Disregarding the number of images, Javascript files, and Sass files we will have a structure resembling the following:
Once we finish with the development and build the website, all the files will not be indexed into the root directory, but in their corresponding folder. The result of this process is shown below, represented in a graphic web sitemap.

![Figure 13 - Project structure](image)

5.2. Designing a website

Design is the process of collecting ideas, and aesthetically arranging and implementing them, guided by certain principles for a specific purpose.

Web design is a similar process of creation, with the intention of presenting the content in electronic web pages, which the end-users can access through the internet with the help of a web browser.

The elements of web design are similar to other kinds of design, such as:
Layout: in terms of design, this is the way the graphics, ads, and text are arranged.

Colour: the choice of colours depends on the purpose and clientele.

Graphics: this would include logos, photos, icons...

Fonts: most web browsers can only read a select number of fonts, known as web-safe fonts, so it will be safer to work within this group.

Content: written text should always be relevant and useful, so as not to confuse the reader and to give them the information they want so they will remain on the site. Content should be optimized for search engines.

The design itself is out of our professional scope so it will not be part of the purview of this thesis. Nevertheless, based on the knowledge acquired in the course “Human Computer Interaction (HCI)” and research performed, we can establish certain guidelines to be followed. While it is true these guidelines have not been entirely followed throughout the development of this project, they will be taken into account for future iterations.

5.2.1. Layout

One of the main challenges when designing a web is creating a good User Experience (UX). Having a good UX means the user will find any information needed without putting an effort. It also means s/he will not be stressed when using the product. It means that as far as is humanly possible, when the user looks at the web site, using it should be self-explanatory.

![Figure 15 – Bad UX design](image)

Basically, we need to get rid of the questions the user might have when using our web site before s/he has them.
A good technique is to create an effective visual hierarchy in such way that we make clear what elements are important, which are interrelated, and which belong to others. Web sites with a clear visual hierarchy present three characteristics:

- **The more important something is, the more prominent is.** For example, the most important headings are larger, bolder, set off by more white space, or nearer the top page (or a combination of all of the above).

- **Things that are related logically are also related visually.** For instance, the items in a navigation bar are grouped together and have the same style because they belong to the same category.

- **Things are nested visually to show what’s part of what.** For instance, a section heading (“Computer Books”) would appear above the title of a particular book, visually encompassing the whole content area of the page, because the book is part of the section.

Another part that we have to put special care into is the navigation. Whenever we visit a web site, the process is the same:
The purpose of navigation is double: to help the user find what s/he is looking for, and to tell her/him where s/he is.

Done right, navigation also gives more information. It tells the user how to use the site: it tells implicitly what the options are; acting like instructions without explicitly adding them (we will talk about instructions in following sections), and it gives the user confidence in the developers: well-thought-out navigation is one of the best ways to give a great first impression.

A navigation bar is basically a list of links that will take us to different sections/pages of our site. Usually, we find horizontal navigation bars on top of the site. This might be considered as the common practice, but it should not always be that way. We have to take into consideration the number of items we will list, the space this bar will occupy in front of the space needed for the content...

Considering a vertical navigation bar has its advantages, such as fitting all screen sizes. This is ideal for usage on a responsive design layout.

Also, there is no need to choose just one. We have the perfect example for this in the official site of the company Uber [20].

---

Figure 17 - Navigation process
As we can appreciate in figure 18, Uber uses horizontal navigation bar on top of their site, but they do not display all of their content there. Instead of that, what they are showing is just the basic and more important sections of their site. On the upper right corner we observe a “hamburger icon”, typically used to show a menu.

![Figure 18 – Horizontal navigation bar of www.uber.com](image)

In figure 19 the vertical navigation bar is displayed. In order to not lose space for the content, this menu remains hidden until the user clicks/taps on the “hamburger icon”. Once this happens, we get to see the complete navigation menu with the same items as in the horizontal bar and some more. Also, if the user clicks/taps in any of the options with an arrow on the side, it will not take her/him to another section, but show her/him the possible options inside the selected section.

![Figure 19 - Vertical navigation bar of www.uber.com](image)

However, if we check this very same web site from a mobile device, we cannot find the horizontal navigation bar anymore. Instead, we will only see the “hamburger icon” to display the vertical menu.
After tapping said icon, we interact with the vertical menu the same way as in the desktop version, without losing any information like we do with the horizontal navigation bar.

5.2.2. Colour

In every project for a company, we need to follow a specific colour code that conveys the brand identity. In our case, the company colour is green, separated in five types:
This does not mean that we are only allowed to use those, but they are probably the ones that will be the most prominent in the user’s memory, as they represent the company.

In web development, we need to know if the elements we are going to use are safe. In the past, many displays were only capable of representing 256 colours. For this reason, a set of colours were selected as “web safe”, and designers were limited to the usage of only 256 colours, exhibited in figure 23.

![Web safe colours](image.png)

Since 2011, the use of web safe colours have fallen into disuse because actual computers support 24-bits for colours, allowing the use of True Colours.
Today, HTML and CSS allow for 16777216 million colours. Designers can now specify 256 shades of red, 256 shades of green, and 256 shades of blue.

\[256 \times 256 \times 256 = 16777216\]

In web development, colours are used not only for design, but also as indicators (clickable element, visited link...). In the WCAG 2.0, related to requirement #6, we need to refer to Guideline 1.4.

While some guidelines are focused on making information available in a form that can be presented in alternate formats, this one is concerned with making the default presentation as easy to perceive as possible to people with disabilities. This guideline is divided in nine different success criteria:

1.4.1 – Use of Colour
1.4.2 – Audio Control
1.4.3 – Contrast (Minimum)
1.4.4 – Resize Text
1.4.5 – Images of Text
1.4.6 – Contrast (Enhanced)
1.4.7 – Low or No Background Audio
1.4.8 – Visual Presentation
1.4.9 – Images of Text (No Exception)

At this moment we are going to focus on the first part: use of colour.

The success of this criterion ensures all users can access information that is conveyed by colour differences, that is, by the use of colour where each has a meaning assigned to it.

Colour is an important element in the design of web content, enhancing its aesthetical appeal, its usability, and its accessibility. However, people with partial sight may experience limited colour vision, and many older users do not see colour well. In addition, users using text-only, limited-colour, or monochrome displays and browsers will be unable to access information that is presented only in colour.

Per WCAG 2.0, there exist some techniques to meet the requirements of this criterion.

**Situation A:** the colour of particular words, backgrounds, or other content is used to indicate information.

- Ensuring that information conveyed by colour differences is also available in text, in example: a color-coded schedule, a form with required fields... Testing procedure:
  1. Check that the information conveyed is also available in text and that the text is not conditional content.
- Including a text cure for coloured from control labels, in example: required fields in an HTML form. Testing procedure:
  1. Check that the same information is available through text or characters cues.
Ensuring that additional visual cues are available when text colour differences are used to convey information, in example: the default formatting for links on a page includes presenting them both in a different colour than the other text on the page underlining them to make the links identifiable even without colour vision, an article comparing the use of similar elements in different mark-up languages uses coloured text to identify the elements from each language, a news site lists links to the articles appearing on its site… Testing procedure:

1. Locate all instances where the colour of text is used to convey information.
2. Check that any text where colour is used to convey information is also styled or uses a font that makes it visually distinct from other text around it.

Using a contrast ratio of 3:1 with surrounding text and providing additional visual cues on focus for links or controls where colour alone is used to identify then, in example: colours that would provide 3:1 contrast with black words and 4.5:1 contrast with a white background, the hypertext links in a document are medium-light blue (#3366CC) and the regular text is black (#000000). Testing procedure:

1. Locate all instances where colour alone is used to convey information about text.
2. Check that the relative luminance of the colour of the text differs from the relative luminance of the surrounding text by a contrast ratio of at least 3:1 [21].
3. Check that pointing (mouseover) to the link causes a visual enhancement (such as an underline, font change, etc.)
4. Check that moving keyboard focus to the link causes a visual enhancement (such as underline, font change, etc.)

**Situation B:** the colour is used within an image to convey information.

Using colour and pattern, in example: the content includes an interactive game and the game pieces for the 4 players are distinguished from one another using both colour and pattern, a flow chart describing a set of iterative steps to complete a process that uses arrowed lines with a green background to point to the next step when a specified condition passes and arrowed lines with a red background when the condition fails, an online map of a transportation system that displays each route in a different colour… Testing procedure:

1. For each image within the web page that uses colour differences to convey information: check that all information that is conveyed using colour is also conveyed using patterns that do not rely on colour.

Ensuring that information conveyed by colour differences is also available in text, following the same testing procedure as in situation A.

The benefits of taking these guidelines into account are that not only does it includes the impaired or disabled, but also accessible web sites are more likely to be ranked well with the search engines, and by designing them with accessibility for the colour blind in mind, we are also targeting PDAs, phones, and similar technological devices used for web access.

There exist a variety of techniques in order to achieve the colour blind accessibility:

- Convert the design into grayscale [22]. In order for the design to be truly accessible all the elements must clearly stand out from each other.
Introduce some texture to differentiate multiple elements that look similar. This could be utilized to differentiate metrics in a chart, rather than using colour alone.

Colour ADD system [23]. We use five symbols to represent the primary colours (cyan, yellow, and magenta), white, and black. If we want to represent other colours, we would need to mix the symbols.
5.2.3. Fonts

Until recently, if we wanted to get text onto a web page, we were limited to a few font options. These options include Sans serif, Serif, Monospaced, Fantasy, and Script. These are called web safe fonts.

Web safe fonts are fonts that are pre-installed by many operating systems. While not all systems have the same fonts installed, we can use a web safe font stack to choose several fonts that look similar, and are installed on the various systems that we want to support.

If we want to use different fonts, say custom fonts, we are able to load them as images, but the downside to this technique is that the text is locked inside the image, fixed at specific dimensions and resolution. Besides, the text wouldn’t be selectable, searchable, or translatable.

Nowadays, all browsers support the CSS \texttt{@font-face} rule, which allows the website to link to font files like any other asset, so this way we can create custom typeface without relying only on web safe fonts. This will load the fonts directly from the server while the whole website renders, so we will only need to take care of the size of our custom fonts.

As stated in requirement \#6, one of our objectives is to create a web site that everyone can use, regarding any disability, vision problem, cognitive limitation.
When working with fonts, we have addressed dyslexia.

According to the International Dyslexia Association (IDA) [24], dyslexia is a specific learning disability that is neurobiological in origin. It is characterized by difficulties with accurate and/or fluent word recognition and by poor spelling and decoding abilities. These difficulties typically result from a deficit in the phonological component of language that is often unexpected in relation to other cognitive abilities and the provision of effective classroom instruction. Secondary consequences may include problems in reading comprehension and reduced reading experience that can impede growth of vocabulary and background knowledge.

How does this affect the design of our web site? Research [25] shows the choice of the font type has a great impact on the reading performance of people with dyslexia.

Some dyslexia friendly fonts are “Helvetica”, “Courier”, “Arial”, “Verdana”, and “Computer Modern Unicode”. On the contrary, we should avoid “Arial It” because this one reduces readability. Also, Sans serif, roman, and monospaced font types are proved to increase readability, while italic fonts do the opposite.

If we look at an average text and cover the top half of the letters, it becomes quite difficult to read it, but if we do exactly the opposite and cover the bottom half we can still read the text without much effort.

![Figure 28 - Critical visual information is in the top of letters](image)

So we will need a font that carries more of its letter’s identifying features in the top half.

In this project, we have opted for a custom font called Aleo.

Aleo font is a custom font type created by the type designer Łukasz Dziedzic. This font has semi-rounded details and a sleek structure, making it highly readable. It belongs to the same family as “Sans serif”, which means that it is also more easily read by users with dyslexia.
5.2.4. Content

As illustrated in figure 4, the user will not read all of the content but scan it. For this reason we advise reducing the text as much as possible. Just by being there, extra words suggest the user that s/he may need to read them to understand what is going on. Although this might be true in some cases where we go into a deeper explanation of a product, it also makes pages seem more intimidating than they actually are. We face the possibility of scaring user with too much information.

In order to avoid this issue, there is some content that should be avoided in our web site. For example, introductory text that conveys no useful information.

What can be seen in figure 30 is an example of avoidable content. In this case we go into the web site and a dialog box pops up with the introductory text causing also another problem: it is now preventing us from reading the actual information we are looking for.
Other content that we should avoid would be instructions. Everything should be self-explanatory, so unless it is absolutely necessary we want to remove instructions. If we must have them, it is better to have them reduced to a minimum. The perfect example for this is a survey:

Before starting the survey we have a total of 103 words. This is much more than what we really need. We are going to analyse the text to properly understand what is absolutely necessary.

“The following questionnaire is designed to provide us with information that will help us improve the site and make it more relevant to your needs.”

This first sentence is just introductory text that we should get rid of, keeping only the words “help us” so we still indicate the user is going to do us a favour by completing the survey.

“Please select your answers from the drop-down menus and radio buttons below.”

The second sentence is completely unnecessary. Most users know how to complete a survey, and those who don’t will not know what a drop-down menu or a radio button are.

“The questionnaire should only take you 2-3 minutes to complete.”

In this case, the information might be useful, but if we present it after non-informative content the user might not read it at all.

“At the bottom of this form you can choose to leave your name, address, and telephone number. If you leave your name and number, you may be contacted in the future to participate in a survey to help us improve this site.”

This paragraph contains valid information, but the user cannot act on it at this moment. We also have to be careful with the position of the content.
“If you have comments or concerns that require a response please contact Customer Service.”

The fact that the user shouldn’t use this form to contact us is useful, but we also should tell the user how to contact us, or even provide something to do it at the moment.

A better way to write this content, using a lot less space and making the readability easier for the user could be the following:

“There are comments or concerns that require a response. Instead, please contact Customer Service.”

Notice that this time we are only using 41 words (less than a half from before), and we also provide a direct link to contact us.

5.3. Development Environment

There is a broad range of development environments we can use to create a web site, but to edit HTML, CSS, and Javascript, we only need a simple plaintext editor. For this project we will work with the editor Atom.

5.3.1. Atom

Atom is a modern text editor that can be configured in many ways. It works across operating systems (Windows, Linux, or OS X); allows us to search and install packages or even create our own; splits the interface into multiple panes; lets us find, preview, and replace text in a file or across all the projects...

5.3.2. Packages

The packages we will use are the following:

- Pigments: plugin to display pigments colours in the minimap.
- Language-html: HTML language support in Atom.
- Autocomplete-html: HTML element and attribute autocompletions.
- Language-sass: Sass/SCSS language support in Atom.
- Atom-sass: Sass compiler for Atom.
- Autocomplete-sass: CSS property name and value autocompletions for SASS.
- Linter-sass-lint: Atom Linter plugin to lint your Sass/SCSS with pure node sass-lint.
- Language-javascript: Javascript language support in Atom.

5.4. Development

5.4.1. HTML

To explain the process of development we will walk through each and every page of our project, starting with the file layout.

5.4.1.1. layout

As explained in the Middleman chapter, the layout.erb file will be the common structure of our pages.

A typical HTML document can be divided into two parts: head and body. The first one is a container of metadata where we will define the document title, character set, style files, links,
script, etc. The second element contains the actual information exhibited to the user, such as text, hyperlinks, images, tables, etc.

The metadata in the head element will vary depending on the page rendered. We cannot have exactly the same metadata for “index.html” and “organizations.html”.

We will explain this metadata in detail later, but for the moment let’s focus on one of the issue that has been found.

Due to the possibility of creating a multilingual site, we will also have to translate the content of the metadata. However, the content we need to translate will be different on every page, and this file should contain all the possible variations of both content and language. For this issue we will take advantage of the frontmatter.

In books, the frontmatter is defined as all the material preceding the proper text, as the title, dedication, table of content etc. In Middleman (and most web development technologies), frontmatter is the information facilitated at the beginning of a file to indicate that said information is going to affect the layout.

For example, the frontmatter variables in the file “index.html.erb” will be the following:

```
1 ... 
2 title: index-title
3 description: index-meta-description
4 keywords: index-meta-keywords
5 url: url-index
6 title-meta: index-title
7 ...
```

![Figure 32 - index.html.erb frontmatter](image)

But this information is not yet translated, and cannot be translated directly there. Frontmatter is used only to display plain text, so we cannot apply i18n functions there to translate it. If we were to apply these functions, we would end up with something similar to “<%= t('key') %>” as the output of the page after the building process.

In order to avoid this, we will first save the `key-value` pairs into the corresponding file (in our case `en.yml`, `es.yml`, or `fr.yml`).

```
1 url: "https://www.bridgeforbillions.org/
2 index-title: "Bridge for Billions"
3 index-meta-description: "Bridge for Billions is the online incubator that helps early-stage entrepreneurs globally to structure, develop and refine their businesses with a mentor."
4 index-meta-keywords: "Start-up, entrepreneur, incubation, online, program, mentor, mentorship"
```

![Figure 33 - yml file with metadata content](image)

Once this is performed, we can apply the i18n function to translate each key directly in the layout.

Frontmatter variables will be available from the layout using the “current_page.data” hash, e.g.:

```
title: “My title” becomes accessible using “current_page.data.title”.
```

We will get the information and translate it in the same function using “<%= t(current_page.data.key) %>%”. In this manner, we will be able to set different content in the head element and also translate it in as many languages as necessary.
Finally, we will add the line "\(<%= stylesheet_link_tag "all" %>\)" to link the CSS file we will create. It is important to point out that there will exist only one CSS file for the entire website, but we will go into details of this later.

For the second element, the body, we will include two parts. The first one will be the navigation bar (or header), and the second one will be the footer.

In our project, the navigation bar will be horizontal and fixed to the top side of the page. In compliance with requirement #1, we will use components from the Bootstrap library (explained in the next chapter).

Navbars are responsive meta components that serve as navigation headers for our site. They begin collapsed (and are toggleable) in mobile views and become horizontal as the available viewport width increases.

We must use a \(<nav>\) element or, if using a more generic element such as a \(<div>\), add the attribute \(role="navigation"\) to every navbar to explicitly identify it as a landmark region for users of assistive technologies, for example screen readers.

If we see the site from a mobile device, we will show the collapse menu first, and after tapping on the menu icon, the complete navbar will be displayed as a vertical menu. But screen readers also have to detect this kind of no-text element without actually adding text. This can be achieved thanks to the bootstrap class “sr-only” (screen reader only), used to hide an element to all devices except screen readers.
Figure 35 - navbar component

Figure 35 illustrates the final content of our navigation bar. Web crawlers will read the page starting with this navigation bar, so it is critical that the structure is well defined here.

We will also add a language selector that will take us to the corresponding html file inside the selected language folder. To achieve this we will add another frontmatter variable in our files, so that we can indicate the path where we should be redirected.

Any hyperlink element that points to a different site will need to have the attribute target="_blank". By doing that, we are opening that other site in a different tab.

The footer element specified in the final design is not difficult to develop, but it presents a minor technical difference for mobile devices. We do not want to end up with a footer that occupies the whole mobile screen. In order to avoid that, we are going to hide some content when the screen is reduced.

Again, we are going to use yet another component from Bootstrap. For this case, we need to resort to responsive utilities.

The classes “visible-” and “hidden-” are one of the cornerstone of bootstrap. Depending on the size of the device in which we want the content to be shown, we will apply the corresponding class. The following table is an example of how these classes have to be used:
Both header and footer will have to be reviewed while developing the rest of the pages to assure everything looks and feels nice, while working well.

5.4.1.2. index

As explained before, “index.html” will be the front cover of our website. We have to be sure the content here is well structured.

The final design of this page is illustrated in Appendix A. To separate the content, we could just use `<div>`. This tag is used to define a division or a section in an HTML document. But we need to give a better structure to the document, so we have opted for the tag `<section>`. This one is new in HTML5, and defines a division or a section in an HTML document. The difference between these two tags is that `<div>` does not convey any meaning. In fact, `<div>` is recommended to be used for styling purposes or as a convenience for scripting. On the contrary, `<section>` represents a thematic grouping of content, typically with a heading.

In our case, we have divided the template of this page into 9 sections, starting the first one with a jumbotron.

A jumbotron indicates a big box for calling extra attention to some special content or information. This section will work as a call-to-action. Inside the jumbotron we can write any valid HTML code, including Bootstrap elements.

We need to divide the inside of the jumbotron into two containers. The first one will be used for the headings and the button, and the second one to show the icons for entrepreneurs and mentors. The headings with the button will not give us any problem, but for the second we have to take care of the responsiveness.

Bootstrap includes a responsive, mobile first fluid grid system that scales up to 12 columns as the device or viewport size increases.

Grid systems are used for creating page layouts through a series of rows and columns that house the content. Bootstrap grid system work in the following form:

- Rows must be placed within a “.container” or “.container-fluid” for proper alignment and padding.
- Use rows to create horizontal groups of columns.
• Content should be placed within columns, and only columns may be immediate children of rows.
• Predefined grid classes like “.row” and “.col-xs-4” are available for quickly making grid layouts.
• Columns create gaps between column content via padding. That padding (15 pixels) is offset in rows for the first and last column via negative margin on “.row” elements.
• Grid columns are created by specifying the number of the twelve available columns we want to span. I.e.: three equal columns would use three “.col-xs-4”.
• If more than 12 columns are placed within a single row, each group of extra column will wrap onto a new line.

Now that we know how Bootstrap grid system works, we will create a <div> with a row divided in half, having 6 columns for the first icon and the other six columns for the second icon. Using the classes “.col-md-6 .col-sm-6 .col-xs-6” we ensure this layout will not change in any device.

This grid system will save us a lot of time not only in the rest of this page, but also in all other pages.

The following sections can be quickly set up using this grid system, for they also are divided into two and three columns respectively.

For the section “Partners & Clients”, we will change the number of columns used depending on the screen size. The design only provides a few logos, but the real product might contain more of them.

In big screens (desktop view) we have no problem seeing and reading a row with 5 elements.

**Figure 37 - Desktop-view of Partners & Clients logos**

But this will be too much for mobile screens, so we will change it to be shown in a clearer and understandable way:
Again, simply with the use of columns this is easily achievable.

![Figure 39 - HTML code of Partners & Clients logos section](image)

The CSS code applied to this page will be detailed in the next section.

We have to comply with W3C regulations and guidelines, so we will use an online validator to check this file and look for any possible issue.
The result indicates that we are not using the tag `<ul>` in the correct way, because it is not permitted to nest a `<div>` inside a `<ul>`.

Unordered lists, should only have `<li>` elements as children, so we will correct that.

We also got a warning in this file, referring to the use of the tag `<section>` with no headings. Changing this and using a tag `<div>` instead will solve this issue.

On our second try we get the expected message:

**Document checking completed. No errors or warnings to show.**

---

5.4.1.3. how-it-works

We will have two pages with the same structure: `how-it-works-entrepreneurs.html` and `how-it-works-mentors.html`. The final design for both of them is presented on Appendix B and Appendix C.
We start by providing frontmatter variables to the layout so we have the correct metadata and URL to use when switching languages.

The structure itself of these pages are also divided into `<section>` and `<div>` elements. We will use `<section>` for the content with text and `<div>` for the separation lines and graphics.

In “Step 3” we have a list of icons that will change when we hover over them. We will explain how this works in the Javascript chapter. If we have content that cannot be seen without interacting with the website, this content will not be read by robots, so it will not help our page to be better positioned in search engine result pages.

Also, we need to add a modal in this page. A modal is a dialog that is displayed on top of our current page. We need to set a trigger button, and when the button is clicked the current page will darken and the dialog box will appear. There is no need to implement this from scratch because we also have a bootstrap component for this.

The “GET IT NOW” button will begin the process of subscription to the company newsletter. The button and the input field above it are part of a HTML form.

The HTML tag `<form>` defines a form that is used to collect user input. The `<input>` element can be of different types: text, email, radio button, submit, etc.

The most important is submit, because that is the trigger input to activate the form action. In a form, the action attribute defines the action to be performed when the form is submitted. Usually, the form data is sent to a web server.

Forms also have another attribute called “method”. This one specifies the HTTP method, **GET** or **POST**, to be used when submitting data. The method GET is the default method when submitting a form. However, the submitted form data will be visible in the page address field. For this reason, GET must not be used to send sensitive information. It is best suited for short, non-sensitive, amounts of data, because it has size limitations too. The method POST has no size limitations, and does not display the submitted form data in the page address field. Always use POST if the form data contains sensitive or personal information.

The page `how-it-works-entrepreneurs.html` returns the following result when tested with an HTML W3C validator:
The page how-it-works-mentors.html differs from the above explained just in colours and content, so we should have no problem passing W3C tests:

5.4.1.4. our-impact

The page our-impact.html, as seen in Appendix D, will be composed by two `<section>` elements: one with the heading “Entrepreneurs” and the other with the heading “Mentors”. In this page we are showing quotes from previous users. For quoting blocks of content from another source within our document we have to use `<blockquote>` elements.

Inside an `<blockquote>` element we will use the tag `<footer>` to identify the source, and wrap the name of the source in the tag `<cite>`.

In this page we will also have content that will be exhibit only after certain action has taken place, but it will follow the same flow as with the icons in “Step 3” of how-it-works-entrepreneurs.html.

With everything correctly structured, this page will have no problem passing W3C tests:

5.4.1.5. organizations

Along with faq.html, which final design is presented in Appendix G, this page is one of the simplest to structure. As we see in Appendix E we mostly have text separated under four different headings. The key in this case is associate an id to those headings.

If we look at the footer, the sitemap presented there includes the heading in this page, so if the user clicks on any of those headings s/he should be redirected not only to this page, but focus on the heading selected.

With everything correctly structured, this page will have no problem passing W3C tests:

5.4.1.6. pricing

The page illustrated in Appendix F shows a table. When writing the code for the table we have to be very strict in the technicalities. If a user visits our website with a screen reader or other type of assistive technology, we need to be sure that the software used will be able to read and provide the information as it is intended.
An example of a basic table would be the figure 47.

![Figure 47 - example code of a basic table](image)

Using that code, the resulting table would be the next:

<table>
<thead>
<tr>
<th>column 1 heading</th>
<th>column 2 heading</th>
<th>column 3 heading</th>
</tr>
</thead>
<tbody>
<tr>
<td>column 1 row 1 content</td>
<td>column 2 row 1 content</td>
<td>column 3 row 1 content</td>
</tr>
<tr>
<td>column 1 row 2 content</td>
<td>column 2 row 2 content</td>
<td>column 3 row 2 content</td>
</tr>
<tr>
<td>column 1 row 3 content</td>
<td>column 2 row 3 content</td>
<td>column 3 row 3 content</td>
</tr>
</tbody>
</table>

Notice that `<th>` elements, without any styling, are already rendered in a different way than the rest of the cells.

We also have to deal with non-textual content, the check marks. For this issue, we will resort to the bootstrap class for assistive technologies: "sr-only".

![Figure 49 - Desktop-view of pricing table](image)

In this page we run into a different problem. Although we can make our table work in a responsive way, the content inside the table does not adapt so well. Even if it does, the text is not readable anymore. In order to avoid this, we will change the design specifically for small-screen devices, and we will use cards instead of a table.
Once again, we will use bootstrap responsive utilities to hide the table when certain screen width has been achieved, and show the cards in its place. In *figure 50* we can observe that in this manner we do not lose any readability, and still have a mobile adapted template.

![Figure 50 - Mobile-view of pricing card](image)

The rest of the page is plain text and a couple of links styled as buttons that will not give any problem to be implemented.

When trying to validate this page, we get the following three errors:

![Figure 51 - W3C Validator result of pricing-entrepreneurs.html](image)

According to the W3C document for HTML5 guidelines:

> “If there exists a row or column in the table containing only slots that do not have a cell anchored to them, then this is a table model error.” [2]
What we are doing with the attribute “colspan” is to indicate the browser that the specific cell is going to occupy the space of n number of columns. The reason behind it is purely for styling, so if we want to correct these errors we should try a different approach using purely CSS.

**NOTE:** the CSS solution for this has not yet been implemented.

### 5.4.1.7. faq

The page *faq.html*, which designed can be seen in Appendix G, contains two parts: the first one is a list of sections that we can navigate to by clicking on the items of this list, and the second one contains those sections.

In the list, we are working with basic hyperlinks. Using an identifier in its attribute “href” is enough to take us to the proper section, but we also have to add a circle icon next to the item selected. That way we indicate the user where s/he is situated at the moment by using a UX technique called “breadcrumb”. In order to do that, we will use a small Javascript function that will be explained in the next chapter.

The right side of the page, containing the rest of the text, is as simple as writing heading-paragraph pairs for each question.

For the mobile-view of this page we have decide to remove the list at the left side, because it would reduce the space provided for the rest of the text, which in this case has a higher priority than the list.

In our first try with the W3C validator, we get the following warning:

![Figure 52 - W3C Validator first result of faq.html](image)

This warning can be easily solved changing the tag `<section>` by the tag `<div>`. As the `<section>` we created does not contain any inner heading and its used is purely aesthetical, it is recommended using `<div>` instead. With this change the new result of the W3C validator is:

![Figure 53 - W3C Validator second result of faq.html](image)

### 5.4.1.8. about-us

The last page in the design can be seen in Appendix H. This page has its own inner navigation bar, which will be fixed to the top after certain scroll had occurred.

In order to achieve this, we will use the bootstrap affix plugin, adding the attributes “data-spy=affix” and “data-offset-top=200” to the inner navigation bar. With this, the affix plugin will add the class “.affix” to the `<nav>` element when we have scrolled 200 pixels from the top.

The rest of the content is separated in three `<section>` elements. The first, “About Bridge for Billions”, and the last, “Our Manifesto”, carry no issue. The one in the middle, “Meet the team”, will be separated into two parts: image-text, and selector.
The selector will change the content above, picture and text, using some Javascript code that we will explain in the next chapter.

The structural code of the page returns no error in the W3C Validator, as we can see in figure 54:

![Document checking completed. No errors or warnings to show.](image)

*Figure 54 - W3C Validator result of about-us.html*

5.4.1.9. error

The page error.html is not specified in any appendix because there is no design for it. When a user tries to go into an incorrect page of our website, the server will respond with the code “Error 404: Not Found”. In addition, the server will look for a template called error.html, and if it cannot find it, the browser will show something similar to figure 55.

![The webpage cannot be found](image)

*Figure 55 - error 404*

Nowadays, most hosting services also provide an error page by default. But a default error page may reduce the confidence the user has on our website. An example of this can be seen in figure 56.

![Not Found](image)

*Figure 56 - error 404 from www.renatopintolopez.es*

That way, we provide the user some information, but creating a custom error page will not only do this but also affect in search engine optimization, improving our ranking. In this project, the error page is depicted in figure 57.
The error page does not need to have a fancy design, and it is recommended to provide a way to go to our website home page. This is a simple page and it should not give any problem with validations. The result of the W3C Validator tool is seen below:

Figure 58 - W3C Validator result of error.html

5.4.2. Javascript

For this project, we are going to use the library jQuery [26]. This library simplifies DOM manipulation, event handling, animation, and Ajax. It works across a multitude of browsers, so we still comply with requirement #3.

In order to be able to work with jQuery, we need to add the gem “middleman-jquery” into the Gemfile file. Once this is done we can create Javascript files that use the library.

Middleman allows us to have a modular architecture for Javascript files, and still keep it simple in production. Inside the directory “javascripts” we have by default the file “all.js”.

As observed, we have to set the list of files that are going to be used in the project development. For a project like the one we are working on, there is no need to implement a great deal of Javascript code. The interaction with the user is minimum, so we will use Javascript to make some adjustments that are more time consuming to do if we implement them with CSS. Our code will be saved in the file “main.js”.

We have to link the scripts from the page templates, meaning we should have the meta tag <script> in our layout.erb file. We can do this either inside the <head> or after the <footer> in the <body> element, but it is preferable to choose the latest because that way the browser will first load the DOM and when it arrives to the bottom of the page, it will load the necessary scripts. In middleman we add the script using the following line:

```html
<%= javascript_include_tag "all" %>
```

In main.js we will start with the code that should be active as soon as the page loads. Using jQuery’s function ready() we assure that the code will not be active before the page loads.
Inside the function `ready()` we will start by implementing function to control navigation. The main navigation bar on top of our site has to be fixed to the top all the time. In large screens, we start with a transparent navbar, so what we do between lines 9 and 23 is controlling the navbar. First we save the height of the element with id equal “bfb-navbar”, and a starting point set to zero.

If the user scrolls more than `(bfb-navbar).height()`, we will add the CSS class “.navbar-white”. If the user goes back to the top, then the opposite occurs and we remove said CSS class. This is done because if we maintain a transparent navbar and keep it fixed, then we will overlap navbar content and page content, so we will not be able to read properly.

In mobile devices we will not have that problem, because whenever the navbar collapses (small sized screens) we use CSS to apply the class “.navbar-white”.

From line 24 to line 29, we will deal with a different problem. Whenever we redirect the user to a part of the page different than the top, anchor points, we will probably cover part of the content with the navigation bar. With the function `offsetAnchor()` we shift the window 80 pixels down in the Y axis.

Lines 30 and 31 trigger the previous function on the event “hashchange”.

In *figure 62* we are going to analyse a different fragment of our code.
We need to initialize the functions that will be used. The first four functions will be used to handle hover and click events, while `equalizeButtons()` and `equalizeBoxes()` will be used for styling.

```javascript
$(document).ready(function() {
  successPhotos();
  switchMq();
  showTool();
  switchBtn();
  equalizeBoxes();
  equalizeButtons();

  // Sticky navbar header
```

The code presented in figure 63 is used to set the same width for the button elements in `how-it-works-entrepreneurs.html` and `how-it-works-mentors.html`. Those buttons contain the class `.button-1` and `.button-2` respectively, so we will get and compare their width, and then assign the higher value to both elements. This way we will have the same size for those buttons no matter their content.

```javascript
var equalizeButtons = function() {
  var maxWidth;
  var width1 = $('button-1').outerWidth();
  var width2 = $('button-2').outerWidth();

  if (width1 > width2) {
    maxWidth = width1;
  } else {
    maxWidth = width2;
  }

  $('.button-1').css('min-width', maxWidth);
  $('.button-2').css('min-width', maxWidth);
```

Figure 63 - `equalizeButtons()`

Figure 64 - Buttons from `how-it-works-entrepreneurs.html`
The code in figure 65 has a similar use to the one from figure 63. This time what we want to equalize is the containers from one of index.html sections. Once more, this is done to keep style consistency no matter the content. First, we will see an image of the error in figure 65 and figure 66.

![Figure 65 - Boxes are not aligned](image)

![Figure 66 - Boxes not aligned](image)

As seen, in the first case the left container is slightly higher than the other two, while in the second case the middle container is smaller than the rest.

![Figure 67 - equalizeBoxes()](image)

In this function we follow the same process as in equalizeButtons(), but this time we will work with height instead of width.

Concerning the rest of the functions, switchFaq() is implemented to swap the breadcrumb icon in faq.html. In that page we have a list on the left side with two hyperlinks. The redirection
will not be handled with Javascript for two reasons: it is not necessary, and it will decrease our website’s SEO ranking.

Depending on which hyperlink had been clicked, we will swap the breadcrumb icon to “#entrep-link” or “#mentor-link”.

In relation to the functions `successPhotos()`, `switchBio()`, and `showTool()`, we will explain only one of them. These three function have a similar objective: hide some elements and show others. For this, the behaviour and workflow will be the same, but the difference is that they do not share targets.

The function `successPhotos()` applies to the images in the page `our-impact.html`, both first and second sections; `showTool()` applies to the icons and descriptions of “Step 3” in `how-it-works-entrepreneurs.html` and `how-it-works-mentors.html`; and `switchBio()` applies to the pair image-content in “Meet the team” section from `about-us.html`. 
The code above starts by setting an initial target that will be shown to the user, hiding the rest. Then depending on which element is hovered, we will switch it with the current one that is being presented.

To finalize with the code, we have another fragment of the script that has no relation with the user interacting with our website, but the browser interacting with our svg elements.

We are using the same icons many times through the website, but in different colours depending on the information we want to convey. One way of dealing with this would be to have the same icon in different colours, but we risk our website getting heavy, hence taking more time to load.

Moreover, some of these icons will have to rapidly change on hover events, so the idea of changing the media assets every time with this action is far from ideal.

With the code presented in figure 70, we will change svg files to inline svg. By doing this we can now edit the images. The steps we follow in the script are, for every img element that contains the class “.svg”:

1. Get the SVG tag.
2. Add replaced image’s id to the new svg.
3. Add replaced image’s classes to the new svg.
4. Remove any invalid XML tags.
5. Check if the viewport is set. If not, set it.
6. Replace image with new svg.
5.4.3. CSS / Sass

As with Javascript files, we have a main CSS file called “all.css”. Here we will specify what is necessary to the development, in this case just one file: “style.scss”.

This is because we are going to work with Sass, not directly CSS. In this file we will import all necessary sass files in order to have a unique big file ready to be deployed.
First, we will define common elements.

In the file “_colors.scss” we will define the set of colours that will be used in the project. We could directly write the colour code every time we need, but this way we have consistency, and keep the file scalable.

![Figure 72 - _colors.scss](image)

We will do the same for the file “_mixins.scss”. Mixins allow us to define patterns of property-value pairs, which can then be reused in other rulesets. We will only define one mixin, presented in figure 73.

![Figure 73 - _mixins.scss](image)

Our mixin, along with the files “_font.scss” and “_typo.scss”, will be used to declare the fonts we will use and set the general style for basic elements, such as html, body, h1, h2, h3, p, etc. In order to adapt the size of the fonts in accordance with the screen, we will use the following sass function:

![Figure 74 - sass function](image)

The difference between mixins and functions is that mixins returns a CSS rule, while functions return a single value. Notice that this function returns a value in rem units. According to the W3C specification, the definition for one rem unit is equal to the computed value of font-size on the root element. When specified on the font-size property of the root element, the rem units refer to the property’s initial value. This means that 1 rem equals the font-size of the html element. Generally, most browsers have a default value of 16px.

The file “_common.scss” will be used to group our custom sass code shared in all the files. We will create CSS classes to keep consistency, in example:
Instead of writing this rules for specific elements inside each page, we will set them in this file and directly add the correspondent class into the element when we need to. For the buttons we will also write more custom code to have it all together in the same place and make future changes easier to achieve.
As illustrated in the figure above, the common rules for the button are specified at the beginning of the class, from line 317 until line 324. In case the design changes in future iterations, we will only need to add new colours if necessary, but the main style structure will remain the same.

We have to keep in mind that some rules do not work in every browser, so we will have to resort to specific settings. This is the case of CSS3 gradients.

Gradients let us display smooth transitions between two or more specified colours. Before gradients appeared, we were able to do this by using images. With gradients we can reduce download time and bandwidth usage, furthermore, elements using them will look better when zoomed because the gradient is generated by the browser. Although nowadays almost every browser and version support gradients, it is important that we also write code for cases when the browser cannot render the gradient correctly. If we check the site caniuse.com [16], we see that 92.97% of the browsers globally support gradients, but only 82.42% do it unprefixed.

In figure 77 we can observe the prefixes we utilize to comply with requirement #3. The prefix “-webkit-” covers browsers Safari and Google Chrome, “-o-” takes care of Opera, and “-moz-“ satisfy Firefox.

Apart from gradients, other CSS functionalities might need prefixes, for example: shadows, animations, box reflections, filters, flexbox, masks, transforms, etc.

Middleman sets a CSS class in the body opening tag of the layout so that we can address some rules directly to specific pages.

```
<body class="%= page_classes %">
```

For example, in the main navigation bar, the colour of the text is generally white, with a transparent background. In Appendix D and Appendix H we change the colour of the text to black because with a transparent background over a white element, we will not be able to see that text. To avoid this, we will change the styling only on that pages by using the classes “.our-impact” and “.faq”. In addition to this, we can also change exclusively depending on the language. If the template belongs to the root directory the classes will be represented as the name of the file. But if it belongs to another directory, for example, inside /es/, the CSS class that Middleman will create will be “.es_name_of_the_file”.

We are going to make use of this feature to address the CSS code specific to each page in separate files. If we run into some styling that might be shared with two or more pages, we will implement those CSS rules in the file “_common.scss”.

5.5. Responsive Web Design

The changing landscape of web browsers mean that the user’s expectations also change; people expect to be able to browse the web on their mobile devices just as easily as they browse on a desktop computer. In response to this, the web design community started creating mobile
versions of the websites, having a desktop version and a mobile version for the same site. But the spectrum of screen sizes and resolutions is widening every day, and creating different versions of each website to target each individual device is not a good approach.

Responsive Web Design is a collection of techniques and ideas to address this issue. The first key idea behind RWD is the usage of a grid. Fluid grids are designed in terms of size proportions. When a layout is squeezed onto a small mobile device or stretched across a large screen, all the elements in the layout will get resized in relation to one another [27].

Apart from this, RWD has two other core elements: flexible images and media, and media queries. The framework Bootstrap will facilitate us the development of a responsive website.

5.5.1. Bootstrap

Bootstrap is a web framework for developing responsive, mobile first projects. We are going to use Bootstrap because we can easily achieve two of the three RWD core elements using this framework.

As we have explained before, its fluid grid system will help us with the responsiveness of our website. In the following figure we will see how aspects of the Bootstrap grid system work across multiple devices:

<table>
<thead>
<tr>
<th>Extra small devices</th>
<th>Small devices</th>
<th>Medium devices</th>
<th>Large devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phones (&lt;768px)</td>
<td>760px</td>
<td>992px</td>
<td>1200px</td>
</tr>
<tr>
<td>Grid behavior</td>
<td>Horizontal at all times</td>
<td>Collapsed to start, horizontal above breakpoints</td>
<td></td>
</tr>
<tr>
<td>Container width</td>
<td>None (auto)</td>
<td>750px</td>
<td>970px</td>
</tr>
<tr>
<td>Class prefix</td>
<td>.col-xs-</td>
<td>.col-sm-</td>
<td>.col-md-</td>
</tr>
<tr>
<td># of columns</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column width</td>
<td>Auto</td>
<td>-62px</td>
<td>-61px</td>
</tr>
<tr>
<td>Gutter width</td>
<td>30px (15px on each side of a column)</td>
<td>0px</td>
<td>0px</td>
</tr>
<tr>
<td>Nestable</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offsets</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column ordering</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 78 - Bootstrap grid system options

The second core element of RWD is to have flexible images and media. We must make sure the images used will adapt to their container despite the dimensions. Bootstrap provides a class called “.img-responsive” which makes the correspondent image scale into its parent element.

The third component, media queries, are CSS techniques that uses the @media rule to include a block of CSS properties only if a certain condition is true. We need to use media queries in our project to control the navigation bar header. The component we are using relies solely in bootstrap code, but we need to overwrite this so the navigation bar collapses before.
By default, Bootstrap navbar collapses at 768px, but in our case we have much more content and need to change the breakpoint to 1338px. The code to overwrite this is illustrated in figure 79.

5.6. Search Engine Optimization
A web search engine is a software system designed to search information of the World Wide Web. The results of this search, which are usually a mix of images, information, and other type of files, are presented in search engine results pages (SERPs).

SERPs might also contain advertisement, but those are not considered natural, organic results. These results will be part of Search Engine Marketing (SEM), a form of internet marketing that involves the promotion of websites by increasing their visibility in SERPs through paid advertising.
Let’s take the company Trip Advisor as an example [28]. As we see in figure 80, the first result is not an organic result, but one for which Trip Advisor has paid to be positioned at the beginning of SERPs. Following SEM results we spot the real natural results. Those are the results that we will focus on in this project.

We are going to apply SEO best practices in our project so we can achieve the best possible result in SERPs without paying any marketing campaign, because what we are interested in are natural results.

The main page that should be taken into account is “index.html”. This is the one robots will read, and the one that will be shown in SERPs when searching for our website. The tests we will carry out will be based on this page, although the findings can be applicable to the rest.

5.6.1. HTML elements

Regarding HTML elements, the most important ones to take into consideration are the title, the meta description, the images and the hyperlinks.

The title of our pages should not be longer than 70 characters, but no less than 10 characters. Each page should have a unique title for each page and the important keywords should be placed near the beginning. In our project we will use the following:

- **index.html**: "Bridge for Billions | Home"
- **how-it-works-entrepreneurs.html**: "Entrepreneurs | How It Works | Bridge for Billions"
- **how-it-works-mentors.html**: "Mentors | How It Works | Bridge for Billions"
- **our-impact.html**: "Our Impact | Bridge for Billions"
- **organizations.html**: "Organizations | Bridge for Billions"
- **pricing-entrepreneurs.html**: "Pricing | Bridge for Billions"
The meta description is the text that will appear right below the title in the SERP. We point this out in figure 81. The recommendation for this text is to limit it to 155 characters and that it be unique for every page. It is not a problem to have a longer description, but search engines will rank the site lower and the description will not be completely shown.

For the pictures, we will have to be sure to have every image element with its corresponding “alt” attribute. This not only helps achieve a better ranking in search engines, but also improves the accessibility of our site.

The hyperlinks will have to be referenced directly within the HTML element and not with Javascript. This way, when a robot reads the web site it will also read this link and use it to rank the site.

Also, inside our site we will need to define some sort of navigation structure to let the web crawlers know about the structure of our site. This can be achieved by means of the tag `<nav>`, but if we use the method it cannot be assure that spiders will index the site correctly, so an even better way of accomplishing this indexation would be to add the file “sitemap.xml”, that we will describe later on.

![Figure 81 - SERP snippet elements](image)

SEO methodology involves many factors that can be separated in two types [29]:

- **On-Page SEO**: consists of factors that are in our area of control, including our code, meta tags, meta descriptions, heading, title tags, internal links within the site, site maps, page load time, ease of navigation, etc. These are all the elements that allow us to provide an efficient presentation of the content to our website users.
Off-Page SEO: consists of factors that are not dependent on the code or are not under our control. In example: forum postings, social media marketing, blogs, RSS feeds... Off-Page SEO is critical in the long-term ranking of our web site.

5.6.2. Metadata

One of the characteristics of the World Wide Web is that resources do not stand by themselves without any explanation, but there is information about the resource. This information about information is known as Metadata.

Metadata is machine understandable information about web resources. We are talking about information that software agents can use in order to facilitate our use of the internet. Meta elements are typically used to specify page description, keywords, author of the document, etc.

We will use this information to enrich snippets represented in SERPs, and to deploy extra information about our site when this is shared through social media.
With regard to social media, the data will be shown in an object called a graph. The graph will contain all the information and will be presented to the user in a legible, organized, and easy to use manner.

5.6.2.1. General metadata

The tag `<meta>` is the HTML element that represents any metadata information that cannot be represented by one of the other HTML meta-related elements (`<base>`, `<link>`, `<script>`, `<style>`, or `<title>`).

In order to avoid problems with special characters, we will start by specifying the character encoding that we are using on the site. In our case, we have encoded the site with the UTF-8 format.

According to requirement #2, our web site needs to have a cross-device behaviour. With this requirement in mind, we will use the `<meta>` tag with the attribute “name” equal to “viewport”. This gives indications about the size of the initial size of the viewport.

The tag `<title>` will have the information for the title of the current page. As previously mentioned, this information should be unique for each page.

Now, for the information displayed in SERPs snippets, we will have a `<meta>` tag for the description, and another one with a cloud of keywords that will help web crawlers with the indexation. Nowadays, the tag with keywords is becoming obsolete. The most used search engine, Google, does not take it into consideration anymore, but we have decided to include it to gain a better ranking also in other search engines (such as Bing, Yahoo, Yandex, or Baidu).

Apart from the main snippets exhibited in SERPs, we also have to keep in mind that most of the information is shared through social media, specifically Facebook and Twitter.

5.6.2.2. Opengraph (Facebook) metadata

The Open Graph protocol [30] enables any web page to become a rich object in a social graph. This protocol was created by Facebook and is inspired by the Dublin Core Schema, Microformats, and RDFa.

The usage of this protocol consists on adding `<meta>` tags with at least the following properties:

- “Og:title”: the title of the object as it should appear within the graph
- “Og:description”: one or two sentences to describe the object.
- “Og:url”: the canonical URL of the object which will be used as its permanent ID in the graph.
- “Og:image”: an image URL that should represent the object within the graph.
- “Og:site_name”: in cases the object is part of a bigger web site, like in our case, this will be the name of the web site.

The result of using these properties is illustrated in figure 84. In that case, we are sharing the website URL on Facebook, but the result will be the same for other social networks that follow this protocol, such as LinkedIn, Google+, or Pinterest.
5.6.2.3. Twitter Cards Metadata

In the case of Twitter, we can also rely on the Open Graph Protocol, but they have a specific way of showing the graph through this platform that we can take advantage of: Twitter Cards [31].

With Twitter Cards, we can attach photos, videos, and media experience to Tweets that will drive traffic to our website. There are several types of cards:

- **Summary Card**: title, description, thumbnail, and Twitter account attribution.
- **Summary Card with Large Image**: very similar to the previous one, but with a prominently featured image.
- **App Card**: a card to detail a mobile app with direct download.
- **Player Card**: a card to provide video/audio/media.

In this project we are going to use the second type. This twitter card is designed to give the reader a rich photo experience, and clicking on the image it will bring the reader to our website.

As with the Open Graph Protocol, there are some properties that are required:

- **“twitter:card”**: should be set to a value of “summary_large_image”.
- **“twitter:site”**: the Twitter username this card should be attributed to.
- **“twitter:title”**: a concise title for the related content. Depending on the platform the behaviour will be different:
  - **iOS, Web**: truncated to one line in timeline and expanded tweet.
  - **Android**: truncated to two lines in timeline and expanded tweet.
- **“twitter:description”**: a description that concisely summarizes the content as appropriate for the presentation in a tweet. Like in the property above, its behaviour is platform related:
  - **iOS**: truncated to one line in timeline and expanded tweet.
  - **Android**: not displayed.
  - **Web**: truncated to two lines in timeline and expanded tweet.
“twitter:image”: this property is not required, but is preferred that we use it. It contains a URL to an image representing the content of the page. The image must be lighter than 1MB in size.

Applying this properties to our metadata we will create the following graph whenever the URL is included in a tweet.

![Twitter Card of www.bridgeforbillions.org](image-url)

As we can observe in figure 86, all this metadata must be included at the beginning of the HTML code, inside the tag `<head>`.

*Figure 85 - Twitter Card of www.bridgeforbillions.org*
5.6.3. Robots Exclusion Protocol

Robots, also known as spiders, crawlers, or web wanderers, are programs that traverse the web automatically. Search Engines use them to index the web content.

As web site developers, we use the file /robots.txt to give instructions about the site to said robots. This is called the Robots Exclusion Protocol.

Any spider that arrives to our site will first look for the file robots.txt. If it finds it, it reads it; if not, it gets an error 404 from the server, in which case nothing would happen and it would index everything, given that there is no prohibition specified.

It is important to have this file in the root directory of our site, otherwise the robots will not be able to read it.

As illustrated in figure 87, the “User-agent: *” means this sections applies to all robots, while “Disallow: /” states that the robot should not visit any page on the site.

We have to take into consideration two important points when writing this file:

- Robots can ignore our file /robots.txt. Especially malware robots that scan the web for security vulnerabilities, and email address harvesters used by spammers will pay no attention.
- The /robots.txt file is a publicly available file. Anyone can see what sections of our server we don’t want robots to use.
Therefore, it is very important to not use the file /robots.txt as a way to hide information. When studying a competitor’s website, we need to check this file in order to discover if they host any file that should not be seen by robots. We are going to see an example of this fault in security with one of ours competitors.

The file observed in figure 88 belongs to “Competitor F”. As we can see, they hide two files from the web crawlers, but a human can read this public file. What we see in lines 19 and 20 are files that form part of their product and should not be by any means public. Now that we have this information, we have a better understatement of their product, and we can use this information to improve our own product.

Also, this practice should not be limited to the study of competitors. If we want to understand how this protocol is used in the industry, it is recommended that we observe big companies files. In example, figure 89 is part of the robots file of Trip Advisor. As we can see there, not only do we find how they use this file and what they hide from the robots, but we also find a job offer.
5.6.4. XML Sitemap

An XML Sitemap is a text file used to detail URLs on a website. It can include extra information on each URL, such as when they were last updated, their importance, if there is another version of a URL in a different language, etc.

This file is created to help web crawlers index our website more efficiently, allowing any changes to be fed to them directly. It is especially important to have this file in brand new websites, because this will increase the chances of being indexed into search engines.

Once we have this file created, we have to save it in the root directory of our project along with the file “robots.txt”, and also add a line in said file to point at the sitemap. If we check the example provided in figure 89, we can see that Trip Advisor uses multiple sitemaps and points their paths in the lines 14, 15, and 16.

5.6.5. International SEO

What would happen if we want to improve our results depending on the country we are in? And what if we want to extend to a different country?

The ideal situation would be to have a different domain for each country, and host those domains in servers belonging to the corresponding countries.

In our case, as our domain is .org and we have the site translated to different languages, this is not ideal. What we can do in this case is take advantage of the tag “hreflang” and indicate the search engine that the link we have related to this attribute will take us to a version in the specified language. We can do this in the sitemap file or directly in the links, as seen in the following example:
The main URL for this site is the one indicated as canonical. The rest are all alternatives in different languages, and they use the attribute “hreflang” to describe the language of each version.

5.6.6. SEO Analysis

After searching “bridge for billions”, we get the SERP revealed in figure 91. The outcome we obtain analysing this result is that we have correctly inserted meta tags for description and title, and that web crawlers have already indexed our web site into their database.

Of course, the results we have now can always be improved. We are going to use the tool from WooRank [32] to test how we can optimize our site. The test algorithm of WooRank is based on 70 criteria including search engine data, website structure, site performance, and others. A rank lower than 40 means that there are a lot of areas to improve. A rank above 70 is a good mark and means that our website is probably well-optimized.
After developing and deploying our website, we proceed to test it. The resulting report shows that the punctuation is 63/100, and can still be improved. There are four elements that throw an error in this test.

The first elements we have to improve will be the headings. It is recommended to have only one <H1> tag per page. We should use keywords in the headings, with the most important keywords in the top level, and a clear <H1> to <H6> hierarchy. At the moment, we are using two <H1> tags.

Other test criteria in this analysis is the existence of the file “robots.txt” and the file “sitemap.xml”. We will talk about the usage and importance of them further down this document.

Finally, according to the obtained report we do not have an optimized mobile speed for our website. This can be fixed enabling compression for the images and files, minifying the CSS code and the Javascript code.

With these 4 errors fixed, we will surpass the 70 points of this test and we will be able to consider our website a well-optimized website.

The remaining test criteria return a good result. We are going to divide those criteria into five categories:

5.6.6.1. SEO
* Title tag: “Bridge for Billions”, with a length of 19 characters.
* Meta description: “Bridge for Billions is the online incubator that helps early-stage entrepreneurs globally to structure, develop, and refine their businesses with a mentor”, with a length of 154 characters.
* Alt attribute: we are using 34 images, all of them with a valid “alt” attribute that adds a description to the image. Since search crawlers cannot see images, they rely on alternative text attributes to determine relevance to a search query.
* Broken links: no broken links were found on this webpage.
* Robots.txt: as mentioned before, we are missing this file at the moment.
* XML Sitemap: as mentioned before, we are missing this file at the moment.
* Underscores in the URLs: we are not using underscores in our website’s URLs. Google recognizes hyphens as word separators, while underscores aren’t recognized at all.
* Blocking factors: we are not using Flash, nor frames. Web crawlers cannot index correctly the content within frames or Flash content. It is recommended to avoid the usage of both.
* Blog: a blog has been found in our website. The fact that a website has a blog represents a great advantage for SEO. In our case, we are not supporting a blog directly inside our project, but we have a link to a blog website called Medium where the company posts articles.

5.6.6.2. Mobile
* Mobile-friendliness: our website is optimized for mobile visitors
Mobile compatibility: no embedded objects detected (such as Flash, Silverlight, or Java). This means our content can be accessed on all devices without any problem.

Font size legibility: the website’s text is legible on mobile devices.

Mobile viewport: a viewport is present, and the content fits within the specified viewport size.

Mobile speed: as mentioned before, this test throws an error that will be fixed in the next iteration.

5.6.6.3. Usability

- URL: “bridgeforbillions.org”, with a length of 17 characters. A descriptive URL is better recognized by search engines. The user should be able to look at the address bar and make an accurate guess about the content of the page before reaching it.
- Favicon: our website uses a favicon.
- Custom 404 page: our website counts with a customized error page that will be shown to the user whenever the server responds with the HTTP status code 404.
- Load time: 0.57 seconds (498.25 kB/s). The website we have develop is fast, which will enrich the user experience and result in a higher rank in SERP.
- Language: declared language match detected language. Both of them are English.
- Structure Data Markup: our websites presents the tags “og description”, “og image”, “og site_name”, “og title”, and “og url”. Structured Data Markup is used to generate rich snippets in search engine results.

5.6.6.4. Technologies

- Speed tips: our website does not use nested tables; it has few CSS files; it has few Javascript files.
- Analytics: we are using at least one analytics tool in the website (Google Analytics).
- Doctype: HTML5. Declaring a doctype helps web browsers to correctly render the content.
- Encoding: character encoding is specified (utf8). This prevents problems with the rendering of special characters.

5.6.6.5. Social

- Facebook Page: our brand’s Facebook page is linked to our website.
- Twitter Account: our brand’s Twitter account is linked to our website.
6. Budget estimation

For a project like the one we have developed, we have to take into account many variables. We are not using any kind of Content Management System (CMS), and the design has been created from scratch. We need to separate the graphic design, which remains out of our scope as engineers, from user interface design. How we decide to structure the components (files in the project and elements in the pages) in order to make the website easily understandable will affect greatly in the final result. Only this can be valued in around 600€-700€.

We have performed a previous research of guidelines and regulations, which has taken approximately 20 hours, and the analysis of competitors, which in this case has taken around 15 hours.

The complete development of the website has taken approximately 200 hours (5 hours/day).

Although we have developed the website following said guidelines, we have also performed subsequent analysis using external tools to assure the compliance of the requirements, and adding corrections if needed. This can be rounded to about 40 hours.

The total in hours then is 20 + 15 + 200 + 40 = 355 hours. Supposed we charge 10€ per worked hour, and keeping in mind the UX design mentioned before, the total value of this project will be between **3.350€ - 3.450€**.

7. Final Conclusions

The objective of this project is to create a web space for a company, following accessibility regulations and optimizing the performance in search engines to increase its online visibility.

As observed in the initial Gantt diagram in *figure 93*, we were scheduled to finish the project at the beginning of December, but the development of the website according to specific regulations has taken more time than predicted, as well as the development of documentation.

The most important element in a project of this kind is to set a strong base. We will probably want to add more information to the website so, in order to do that, we should follow the same process explained in this report to remain consistency.
We started by defining the requirements, along with the bibliography needed for each one of them. Second, we analysed the competitors and ascertained if their website complied with the requirements we set.

After that, we stated the technology we are going to use in order to create the web site. The final decision being to use Middleman because of the directory structure it provides and its ease of configuration.

Next, we worked along with the design department to create a user-friendly interface. As we established in the report, graphic design is out of our scope as engineers, but User Interfaces and User Experience are not. We have performed a study of good-practices in this field, keeping in mind accessibility related issues and guidelines from regulations WCAG 2.0 and W3C for HTML and CSS.

At this point, with requirements, technology, and design already defined, we started the development of the website while making sure it worked on as many browsers and devices as possible and it followed regulations.

Also, we performed an analysis of search engine optimization. We have specified what information we need to implement in order to take advantage of said engines, always from the technology point of view.

Finally, we have estimated the price of this project in the current market taking into account both research and development.

Upon the completion of this project, we have complied with all the requirements set and explained in section 2.
8. Glossary

**Cross-browser:** ability of a website, web application, HTML construct or client-side script to function in environments that provide its required features and to bow out or degrade gracefully when features are absent or lacking.

**Cross-device:** method to ensure web content is accessible on a range of devices. Compatibility is achieved by making small adjustments to a website to ensure that users can access the site without faults or bugs, regardless of the device being used. Whilst some styling issues may be addressed, the website may have little difference across devices than that of a varying size.

**CSS:** Cascading Style Sheet

**HTML:** HyperText Markup Language

**Relative luminance:** the relative brightness of any point in a colour space, normalized to 0 for darkest black and 1 for lightest white.

**RWD:** Responsive Web Design. Approach to web design aimed at allowing desktop webpages to be viewed in response to the size of the device one is viewing with.

**SEO:** Search Engine Optimization

**SERP:** Search Engine Result Pages

**URL:** Uniform Resource Locator

**UX:** User Experience

**WCAG:** Web Content Accessibility Guidelines

**W3C:** World Wide Web Consortium

9. References


Appendix E

Incubation Software as a Service
Optimize your resources, bring your talent search online, and reach more entrepreneurs than ever before. Our online programs mean you can screen and support more entrepreneurs without increasing operational costs.

Employee Engagement Program
We help you set up an easily monitored mentoring program that allows your employees to polish their entrepreneurial skills, while making a difference where and when it matters the most.

MBAs Mentors Program
Give your students the gift of experience. As mentors for our global community of entrepreneurs, your students will apply classroom lessons within challenging real-world contexts. As the students gain valuable business experience and build their personal networks, instructors and institutions can use our online system to monitor the progress at every stage of the journey.

Investors Access
We offer investors and philanthropists the opportunity to invest in passionate and motivated entrepreneurs from around the world. With monthly reports featuring our top ventures and visual pitches, investing in quality ventures has never been easier.

Want to learn more about how your organization can make an impact?

CONTACT US
Appendix F

Incubation Programs for Entrepreneurs

All of our plans provide you with the tools necessary to rapidly grow your business. Our intuitive Online Workspace offers an organized overview of the entire business development process. Our Business Development Tools offer a structured, step-by-step progressions, ensuring you focus on each stage of development without being overwhelmed by the process. Tracking your success throughout the incubation process is a no hassle. We offer entrepreneurs two distinct incubation plans which vary in their intensity. For those that wish to work at their own pace, our Standard plan offers all the great benefits of our online incubation platform. For a more intense and thorough experience, our Premium plan offers you an experienced online mentor to advise and guide you throughout the incubation process. Gearedline at every stage of development will help you keep you on track, so that you rapidly and efficiently grow your venture.

Interested in our Corporate or Educational Programs?

Copyright © 2016 Bridge for Billions, Inc. All Rights Reserved.
Appendix G
| **Firmante** | CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES |
| **Fecha/Hora** | Tue Jan 10 18:06:50 CET 2017 |
| **Emisor del Certificado** | EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES |
| **Numero de Serie** | 630 |
| ** Metodo** | urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.shal (Adobe Signature) |