ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

SECONDA FACOLTA' DI INGEGNERIA
CON SEDE A CESENA

CORSO DI LAUREA

IN INGEGNERIA AEROSPAZIALE

Sede di Forlì

ELABORATO FINALE DI LAUREA
in Attitude Determination and Control

Development and implementation of a S/W platform to automatically receive and share satellite data

CANDIDATO
Alessandro Romolo

RELATORE
Prof. Paolo Tortora
CORRELATORE
Prof. Fernando Aguado

Anno Accademico [2013/2014]
Sessione seconda

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

La seguente tesi è la relazione del lavoro di creazione e implementazione della piattaforma software che sviluppa l'archivio del progetto SATNET.

I satelliti universitari hanno un tempo di vista della propria Stazione di Terra di pochi minuti al giorno: SATNET risponde all'esigenza di comunicare con un satellite universitario in orbita bassa per più dei pochi minuti al giorno che una singola Stazione di Terra permette. Questo avviene grazie a una rete di Stazioni di Terra Satellitari collegate da specifiche missioni comuni che mettono in condivisione dati ricevuti da uno o più satelliti, aumentando il rendimento dati/giorno di questi e permettendo una migliore fruizione delle Stazioni di Terra stesse. Il network sfrutta Internet come canale di connessione, e prevede la presenza di un archivio nel quale memorizzare i dati ricevuti, per poi renderne possibile la consultazione e il recupero.

Oggetto di questo lavoro di tesi è stato lo sviluppo e l'implementazione di tale archivio: utilizzando un sito web dinamico, il software risponde a tutte le richieste evidenziate nel paragrafo precedente, permettendo a utenti autenticati di inserire dati e ad altri di poterne avere accesso.

Il software è completo e funzionante ma non finito, in quanto manca la formulazione di alcune richieste; per esempio non è stato specificato il tipo di informazioni che è possibile caricare in upload, né il tipo di campi richiesti nel modulo di registrazione dei vari utenti. In questi casi sono stati inseriti campi generici, lasciando all'utente la possibilità di modificarli in seguito.

Il software è stato dunque concepito come facilmente personalizzabile e modificabile anche da utenti inesperti grazie alla sola lettura della tesi, che

rappresenta quindi una vera e propria guida per l'utilizzo, l'installazione, la personalizzazione e la manutenzione della piattaforma software.

La tesi evidenzia gli obiettivi e le richieste, mostrando l'aspetto del sito web e le sue funzionalità, e spiega passo per passo il procedimento per la modifica dell'aspetto delle pagine e di alcuni parametri di configurazione. Inoltre, qualora siano necessarie modifiche sostanziali al progetto, introduce i vari linguaggi di programmazione necessari allo sviluppo e alla programmazione web e aiuta l'utente nella comprensione della struttura del software.

Si conclude con alcuni suggerimenti su eventuali modifiche, attuabili solo a seguito di un lavoro di definizione degli obiettivi e delle specifiche richieste.

In futuro ci si aspetta l'implementazione e la personalizzazione del software, nonché l'integrazione dell'archivio all'interno del progetto SATNET, con l'obiettivo di migliorare e favorire la diffusione e la condivisione di progetti comuni tra diverse Università Europee ed Extra-Europee.

1

INTRODUCTION

In 2013 started the development of SATNET[1], a worldwide network of radio amateur and university Ground Station (GS) to support the operations of university satellites.

A network like SATNET allows the access to a great number of GS and satellites, increasing the data return to many hours per day and do not let the GSs unused.

A single satellite sends data to many GSs along its orbit. At the ground there must be a connection between the GSs that collect the data in order to make them available for successive elaborations and analysis.

This connection must comprehend a line of communication and a database for data storage.

[1] Inserire significato acronico e riferimento alla relazione di preparazione alla tesi.

1.1 The Web Database

The SATNET project requires a database to store the data provided by the missions supported.

Nowadays, one of the best way to exchange data is by internet; internet is available worldwide and create a database online in a web page seems the best solution.

Every GS will receive data from the space segment, probably from different missions. Then it will pass the data through a web page to a main server, which will store the data in a Database Server and will make them available for the clients, distributing every piece of information with his associate mission and blocking with a firewall not authorized requests.

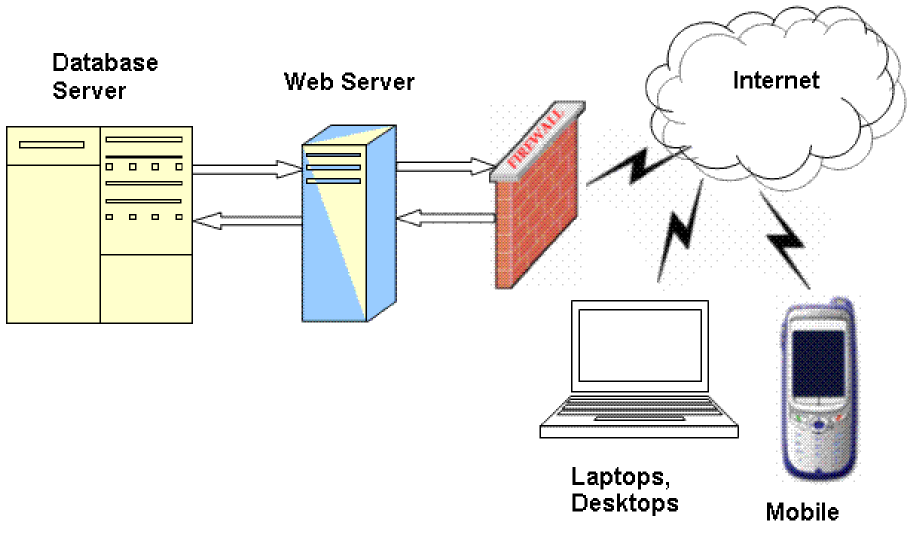


Figure 1-1 Graphic flow between the database and the client devices

A dynamic web application enhances the database and makes it accessible in a protected web page, accessible only by authenticated users.

The main topic of this work focuses on the module of the SATNET, that allows the storage and sharing of information through the web.

1.2 Technical information about the database

The dynamic web application is written in PHP, a server-side scripting language designed for web development but also used as a general-purpose programming language.

In order to speed up the creation of the application and to improve the maintenance we used the framework Symfony2, a free software released under the MIT license. Symfony2 is a PHP framework; it simplifies all the development and improvement process.

The relational database management system is Doctrine and the programming technique is ORM (Object-relational mapping).

ORM is a powerful method for designing and querying database models at the conceptual level, allowing data converting between incompatible type systems in object-oriented programming languages. Relational database management systems (RDBMS) represent data in a tabular format, whereas object-oriented languages, such as Java, represent it as an interconnected graph of objects: ORM allows the connection between the two type systems.

For more information about ORM we suggest [2]

1.3 Thesis outline

The remaining of this thesis is organized as follows.

- Chapter 2 presents a guideline for a preliminary study of the basics of web applications. It involves a series of languages for the programming of static pages (HTML), the settings about look and formatting (CSS), the scripting of basic functions (JavaScript) and, eventually, a look into dynamic web applications with server-side

[2] http://www.orm.net/

programming languages (PHP). It closes with the basics of programming development framework.

- Chapter 3 introduces to Symfony2, the framework used to develop the project presented in this paper. A short introduction helps the user to install PHP, Symfony, Perl and all the useful tools to make the application work. Then follows a quick tour to Symfony, how it works regarding director structure, routing, controllers and bundles. A skilled operator about Symfony and web applications can neglect this and the previous chapter.
- Chapter 4 outlines the project as seen by the users. For every web page we presented a brief description of the principal functions, how to perform them, possible problems an applications.
- Chapter 5 outlines the project as developed by the programmer, with detailed explications about the structure of the bundles, the security system and the third-party repository.
- Chapter 6 is a guide for the installation of the software in a Server, and the customization of settings and templates.
- Chapter 7 presents conclusions and suggestions for future software developments.
- An appendix closes the thesis, showing the most recurring and useful command prompt or terminal (cmd) commands.

2

WEB PROGRAMMING

In order to comprehend and properly use the web application, we need a study of the philosophy of the web and the web application programming. The knowledge of some scripting and programming languages is essential: in this chapter, we try to explain the function of the most important, like HTML, CSS and JavaScript, and to introduce the object-oriented programming and PHP, fundamental for the web programming server side.

2.1 Web Development and Web Programming

The Web development involves the development of every kind of web site or web application for a very generic private network that use Internet Protocol technology to share information.

The Internet Protocol (IP) is the principal communications protocol base of the internetworking and the entire Internet: it establishes every communication on the web.

A conversation on the web starts with a request for a resource. The client (e.g. a browser, a mobile app, etc.) asks for it sending a message, and then waits for the response. The sent message is written in a special format known as HTTP, and must contains everything necessary to identify the requested resource. If the server is not able to find it, answers with an error message.

The server receives the request, analyses it, checks for errors, and creates a new message, the response. The response is written in an HTTP message, which the browser is able to read and display to the user.

The response contains the requested resource, as well as other information about the status of the system, the HTTP response status code, the HTTP header and other metadata.

To reach this aim, there are many languages dedicated to different types of application (web, mobile, JSON API), but the philosophy of the process is standard; every web application is built to understand a request and create and return an appropriate response. This principle, extremely easy and powerful, drives the communication on the web.

The web started with static web pages, written in simply HTML. Write a static web page is a simply matter of compiling an HTML file, without programming logic.

With the growth of the internet, the requirements for the web pages increased, leading to the birth and growth of web programming.

Nowadays every web site presents a big dynamic component, that allows interactivity between the clients and the server and it requires the knowledge of more than only one programming language.

To write a static web site you need to know at least HTML and CSS. To improve the web site you need to start putting a bit of logic inside the HTML code, with possibly JQuery/JavaScript for interactivity, but still applying a client.-side scripting.

In order to create a modern, interactive website, you should know about server-side scripting, like PHP, and SQL for managing data held in a relational database management system (RDBMS).

Languages such as Ruby and Python are useful for different skills, but can be learn at a later stage, and for a starting point, the PHP/MySQL combo is the place to start.

2.1.1 HTML

HTML (HyperText Mark-up Language) is a scripting language intended to write and show the content of a web page.

HTML is HyperText, because it is the principal characteristic defining the structure of the web; it allows reference to other text through hyperlinks or other structures embedded inside a page.

HTML is a mark-up language because the way of annotate the text is syntactically distinguishable from the text. In particular, HTML is a descriptive mark-up, and it uses the tag to determine how the content of the page is shown to the user.

The browser born to read HTML and properly compose the information and presents it to the user. It does not display the tags, and it can be taken as example of no use of the WYSIWYG[3] scripture system; indeed the browser does not display the HTML tags, but uses the tags to determine how the content of the HTML page is shown to the user.

HTML is written in the form of HTML elements consisting of tags enclosed in angle brackets: every tags refers to a different kind of metadata or format.

Here an example of an HTML page:

```
<HTML>
  <head>
```

[3] Acronym for "What You See Is What You Get".

```
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world[4]!</p>
  </body>
</HTML>
```

Basic HTML is a limited approach that does not allow for flexibility or responsiveness. Visitors accessing HTML-only sites see simple pages with no level of customization or dynamic behaviour. That is the motivation for the development of programming languages, in order to integrate it and open the boundary of the web.

The actual version of HTML is the 5.0 released in February, 4th 2014. HTML5 is intended to subsume not only HTML 4, but also XHTML 1[5].

Many books and website widely describe HTML and his tags. We referred to the official documentation of the World Wide Web Consortium (W3C)[6].

2.1.2 CSS

HTML was intended to define the content of a document rather than formatting.

HTML 3.2 specification added tags like <font>, and colour attributes: it improved the visualization of the webpage, but incredibly extended the development of large web sites, where fonts and colour information were added to every single page.

From HTML 4.0, the CSS (Cascading Style Sheet) tried to solve this problem, allowing the storage of formatting in a separate CSS file.

[4] It is by tradition often used to illustrate to beginners the most basic syntax of a programming language with a computer program that outputs “Hello, world” on the display device.
[5] XHTML is an extend versions of the widely used HTML, developed to make HTML more extensible and increase interoperability with other data formats.
[6] http://www.w3.org/MarkUp/

CSS uses keywords to specify the names of various style properties; in this way, the HTML defines the semantic of the text and the CSS the formatting.

```
<style type="text/css">
body {
  color: purple;
  background-color: #d8da3d }
</style>
```

Like for HTML, many books and website describe CSS: the official dispositions and the latest development are in the World Wide Web Consortium website.[7]

2.1.3 JavaScript

HTML allows the development of only static web pages: JavaScript (JS) is used to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed, allowing the development of dynamic web pages.

JavaScript is a prototype-based scripting language with dynamic typing, a syntax influenced by C and first-class functions. JavaScript code can be inserted into any HTML page, and nowadays all web browsers are able to execute it.

The more common way to use JavaScript is the manipulation of the contents of a HTML element, substituting, erasing or creating it.

It allows also the change of the value of HTML attributes, the change of HTML Styles, for input validation and more.

JavaScript born for client-side scripts, but now is also used on server-side for programming, game development and the creation of desktop applications. It finds

[7] http://www.w3.org/Style/CSS/

application also outside the web pages, like in PDF documents, site-specific browsers, and desktop widgets.

2.2 Object-oriented programming

The object-oriented programming (**OOP**) is a programming paradigm, base of many programming languages like Java, Python, C++ and PHP.

In OOP, the “object” refers to a particular concept that can be a combination of variables, functions, and data structures.

The object is defined as **instance of a class**, where a class is an extensible template for creating objects. The class is the abstraction of a concept as implemented in a software: his object is a specific realization of the class.

The OOP includes many others concepts, like encapsulation and inheritance.

The **encapsulation** defines the technique to hide the working principle of a part of a program, restricting access to some of the object's components. It protects the code and allows taking part of the project like a “black box”, without any knowledge of its internal mechanism and using it only in terms of its input, output and transfer characteristics.

The **inheritance** defines a relation between two different objects or classes: it allows an object to inherit another object, using the same implementation. One object can be inherited by more objects; consequently, this procedure leads to a hierarchy, with subclasses and superclasses. Inheritance helps the programmer limiting code redundancy, with overriding and code reuse.

A more detailed description of these concepts goes beyond the scope of this work and it can be found in [8] and [9]. The reader is expected to learn these concepts to better understand the PHP language.

[8] http://docs.oracle.com/javase/tutorial/java/concepts/index.html
[9] http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep#Interface

2.3 PHP

PHP is a recursive acronym for *PHP: Hypertext Preprocessor*, but it originally stood for *Personal Home Page*. The new name fits the improvements and extended capabilities of the last versions of the language.

PHP is a “widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML”.[10]

To say that PHP can be embedded into HTML means that PHP code can be written within the HTML code, and not in a separate file.

PHP is a scripting language, as opposed to a compiled language. This means that PHP is designed to do something only after an event occurs, exactly like JavaScript, which commonly handles events that occur within the Web browser. These two languages can also be described as interpreted, because the code must be run through an executable, such as the PHP module or the browser’s JavaScript component.

The difference between JavaScript and PHP is that PHP is a server-side technology, and its aim is to send information to the Web browser, and not to open new browser window, make pop-up alerts, etc. However, PHP can be used to generate JavaScript, just as it can be used to create HTML.

When a client calls a uniform resource identifier (URI), the server read the request written in HTML, extracts and performs the PHP code: according on these instruction, it creates and sends the appropriate Web page data to the browser in the form of HTML. Shortly, PHP creates an HTML page dynamically based on the request or others parameters (like time, date, etc.).

[10] www.php.net

The client can only see the HTML code, and there is no perceptible difference between *www.homepage.html* or *www.homepage.php* appearance, but how the page's content is compiled is significantly different.

There are alternatives to PHP, but to develop dynamic Web sites, PHP is preferred for the following reasons:

- PHP is much easier to learn and use
- PHP was written specifically for dynamic Web page creation.
- PHP is free and cross-platform.
- PHP is now the most popular tool available for developing dynamic Web sites, covering the 75% of all Web sites, and it is one of the most popular programming language.

PHP is an orient-object programming language (OOP); thus we suggest to study OOP before reading the documentation in the official web page[11], and the dedicated books in the Bibliography.

2.4 Frameworks

A framework is a platform for developing software applications. The exact definition of a framework is quite hard and complex: the idea is to create a real or conceptual structure in order to serve as a support for the programming of some kind of applications.

Technically, the programmer does not need a framework. He may need it because it contains lots of already written and tested functionality that represent a great shortcut when developing applications.

[11] http://www.php.net/docs.php

The basic principle of framework is *"Investing in the task, not in the technology"*. It means that recurring tasks are already written and the developer can fully focus on specific components, with solid sustainable and high quality code.

> *"In computer programming, a software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. A software framework is a universal, reusable software platform to develop software applications, products and solutions. Software frameworks include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or solution."*[12]

The programmer community agrees on this matter: except for very simple web sites, small and isolated, a PHP framework represents a great opportunity for many reasons:

- A PHP framework already has a structured folder and it helps with code and file organization.
- PHP frameworks comes with Libraries and Helpers, and there is plenty of plugins provided by the community.
- A framework facilitates the management of the security system, with tools already tested many times by many programmers.
- After a slowing down due to the learning of the framework, it allows a rapid application development, writing less code.
- The organization of a project in a PHP Framework creates a suitable environment for teamwork.

[12] http://en.wikipedia.org/wiki/Software_framework

- A PHP framework can make available a debugger, facilitating the debug and the maintenance of the project.

Nowadays the majority of complex web projects are written within a framework.

3

THE SYMFONY FRAMEWORK

The purpose of this paper is to provide a practical guideline in order to familiarize with Symfony and start a first basic project. This is not meant to be a User's guide to Symfony.

For a deeper study on this matter, we suggest to read the official documentation integrated by the books and websites in the bibliography.

Symfony2 is widely used in the programmer community, therefore forums, web sites and books can help the neophyte to start programming.

3.1 Why Symfony2

"Symfony2 is a reusable set of standalone, decoupled, and cohesive PHP components that solve common web development

problems. Based on these components, Symfony2 is also a full-stack web framework."[13]

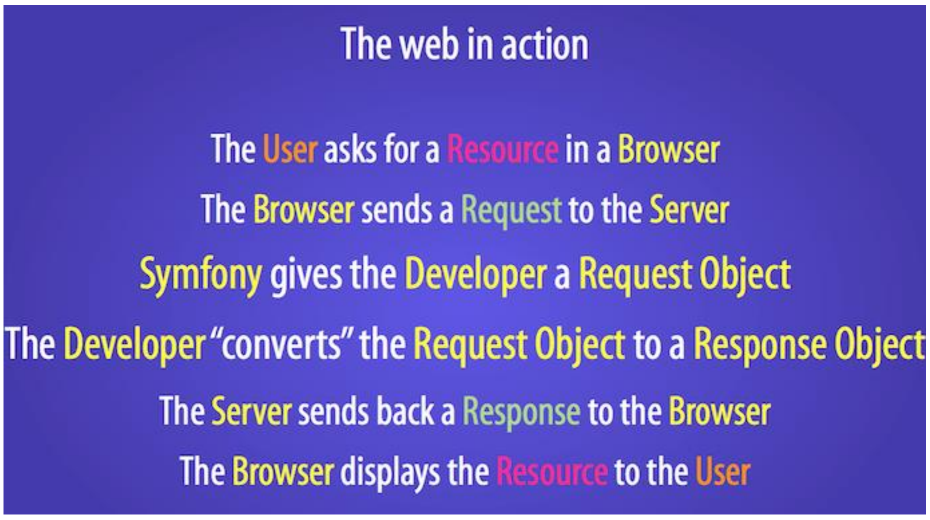
The web offers many PHP frameworks. According to us, Symfony is one of the best for the following reasons:

- The code is rock solid: the majority of Symfony2 components is the result of many years of work and the contributions of many developers.
- Symfony embraces the "don't reinvent the wheel" philosophy, and provides tight integration with many other Open-Source projects (like Monolog, Assetic, Doctrine, Propel ...).
- Symfony enjoys a huge community of users and contributors.
- There are many open source high quality vendors. Almost for any task there is a bundle you can base your stuff on or at least to get an idea how to approach the problem.

As of today, Symfony has twenty-one components and any of them can be used as a standalone library. With these, many tasks are simplified, like validating an object, creating a form, routing or checking the security of the system.

Figure 3-1 shows the workflow of a Symfony request, highlighting Symfony's role in the managing of the Request object.

[13] http://fabien.potencier.org/article/49/what-is-symfony2


The web in action

The User asks for a Resource in a Browser

The Browser sends a Request to the Server

Symfony gives the Developer a Request Object

The Developer "converts" the Request Object to a Response Object

The Server sends back a Response to the Browser

The Browser displays the Resource to the User

Figure 3-1 The Workflow of a Symfony request

3.2 The Bitnami WAMP

To install Symfony we decided to install the Bitnami WAMP, which provides a complete development environment for Windows.

Bitnami comprehends PHP, MySQL and Apache, and bundles many functions among which CURL, PEAR, SQLite and the Symfony framework.

The use of a WAMP gives the opportunity for a fast and easy installation of the useful components for the server-side web programming.

In the Symfony web site[14], there is a useful Quick Start Guide. After the installation of Apache and Symfony2, there is the possibility to start seeing the practical use of Symfony: using and modifying a demo project developed for practicing with Symfony helps understanding the framework and its working principle.

For the sake of completeness, an installation guide for Bitnami WAMP is reported in chapter 6.1.

[14] http://symfony.com/doc/current/quick_tour/the_big_picture.html

3.3 Understanding the Directory Structure

Symfony2 follows and recommends the directory structure:

- ***app/:*** the application configuration;
- ***src/:*** the project's PHP code;
- ***vendor/:*** the third-party dependencies;
- ***web/:*** the web root directory.

The app directory stores the AppKernel class, which is the main entry point of the application configuration. It stores the configurations, including routing and security, and the cache.

The web directory contains public and static files, like images, stylesheets, and JavaScript files. Moreover, it contains the front controller: it creates the Request object and sends the response contents back to the user.

Vendor is the default directory for third-party dependencies, but they can be stored in other directories.

The source directory (src) contains the project’s PHP code, organized into bundles.

3.4 Bundle

A bundle is a directory that has a well-defined structure and can host anything from classes to controllers and web resources. It is also a PHP namespace[15], but a namespace becomes a bundle as soon as you add a bundle class to it.

In other words, a bundle is a part of an application with its own logic, including controllers, views and models in the classical MVC-Paradigm.

MVC paradigm specifies the role that a single object can assume in an application, and the way in which objects communicate to each other.

Figure 3-2 shows the MCV pattern design.

[15] Namespaces are a way of encapsulating items. More information in the next page.

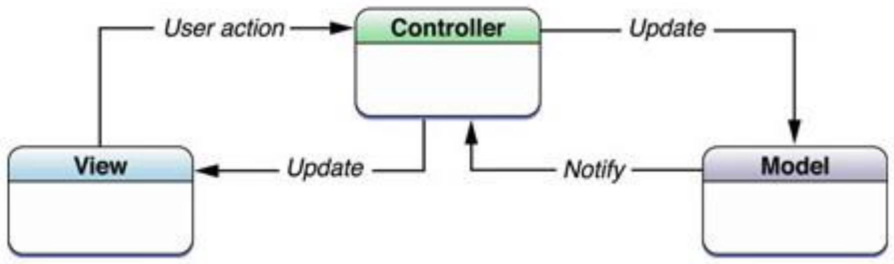


Figure 3-2 MCV Pattern design

The model contains the specific data of an application and defines every procedure for the manipulation of these data.

The view simply displays the content to the client.

The controller is an intermediary between the model and the view.

Symfony is not exactly an MVC software[16], indeed, it provides the tools for the Controller part, the View part, but not the Model part, and it's up to the developer create the model or use other tools. We capitalized the integration for Doctrine, designing the whole application like a MVC.

The bundles give you the flexibility to use pre-built features packaged in third-party bundles or to distribute your own bundles. It makes easy to choose which features to enable in your application and optimize them in your favourite way.

PHP uses namespaces. Namespaces are a way of encapsulating items, designed to solve two problems that authors of libraries and applications encounter when creating re-usable code elements such as classes or functions.

These two problems are the following:

1) Name collisions between code you create, and internal PHP classes/functions/constants or third-party classes/functions/constants.

[16] The author of Symfony wrote few interesting lines about it. You can find it at the web site http://fabien.potencier.org/article/49/what-is-symfony2

2) Ability to alias (or shorten) Extra_Long_Names designed to alleviate the first problem, improving readability of source code.[17]

It provides a way in which to group related classes, interfaces, functions and constants.

In Symfony, the namespaces reflect the bundle structure, so every bundle creates a namespace and every namespace are identified by a bundle, with directories and subdirectories. This structure is briefly shown in the following lines:

- ***Controller/:*** contains the controllers,
- ***DependencyInjection/:*** contains the respective service.
- ***Entity/:*** contains the entities.
- ***Resources/:***
  - ***Config/:*** contains routing and services.
  - ***Views/:*** contains the templates.
  - ***Public/:***
    - ***Css/:*** contains css files.
    - ***Images/:*** contains images.
    - ***Js/:*** contains javascript files.
- ***Tests/:*** contains files for the application testing.
- ***Form/:*** contains the forms used by the application.

3.5 Controllers, Front Controllers and Environments

A Controller is simply a class file that is named in a way that can be associated with an URI.

All requests run through the front controllers: in Symfony are the files *app.php* and *app_dev.php* in the *web/* directory. These are the very first PHP scripts executed when a request is processed.

[17] http://www.php.net/manual/en/language.namespaces.php

The front controller creates an instance of the *AppKernel*, makes it handle the request and return the response to the client. Furthermore, the front controller initializes settings and decorate the kernel with additional features.

The front controller can be chosen by requesting URLs like:

```
http://localhost/app_dev.php/some/path/...
```

In this way, we will call the *app_dev.php* front controller, which opens the application in development environment mode. To open the application in production environment we have to call:

```
http://localhost/app.php/some/path/...
```

In this second situation, the *app.php* string in the URL is hidden automatically, and we can directly request:

```
http://localhost/some/path/...
```

The front controller initializes the bundle controller chosen by the requested routing. The controllers are in the *Controller* folder of every bundle. The aim of the controller is always to produce a *Response object* for the client, rendering a template, showing a form, querying the database and so on: it carries the logic into the static web page, making it dynamic.

3.6 Routing

The request of a client contains an address to the requested resource. This address is the URL, and in Symfony, it identifies the called controller.

A route is the map from a URL path to a controller. In Symfony, the routing system is flexible and allows creating complex routes and generating URLs inside templates and controllers.

The parameters of a single route are:

- Name of the route
- Pattern

- Defaults

We can also optionally define:

- Requirements
- Methods
- Prefix/Suffix

The name of the route is the reference to call it in the project. The pattern is the relative URLs to the page. The defaults are the parameters called automatically by the route: in particular, the controller and the action to call.

The requirement can appear when the route is dynamic and accepts a variable in the routing definition; methods restrict the route for a particular method (i.e. *GET, HEAD, POST, PUT, DELETE*); prefix or suffix are morpheme post before or after the regular pattern.

```
homepage:
    pattern:  /
    defaults: { _controller: AcmeMainBundle:Default:index}
```

In this example, the name of the route is *homepage*, the pattern is just “/”, because it is the homepage and the relative pattern is null, and the controller is defined.

There is an easy way to identify the controller. The first part is the project followed by the name of the Bundle. The second part is the name of the controller. The last part is the name of the Action inside of the controller.

In the previous example, the homepage route calls the *indexAction* of the *DefaultController* into the *MainBundle* of the Acme project.

In the controller, Symfony adds the string *Controller* to the class name (Default=> *DefaultController*) and Action to the method name (index => *indexAction*).

The controller could be called also using its fully-qualified class name and method: *Acme\BlogBundle\Controller\BlogController::showAction*, but in a less flexible way.

Also the templates can be called in a more flexible way, as is shown in the next chapter.

3.7 Template & CSS

A Template is a text file, used to define the pattern of a page and, using some data source, display the data.

The idea of the template meets the idea of **separation of concerns**[18]: in order to develop and deploy applications that are flexible and easily maintainable is important to separate the domain logic from presentation logic.

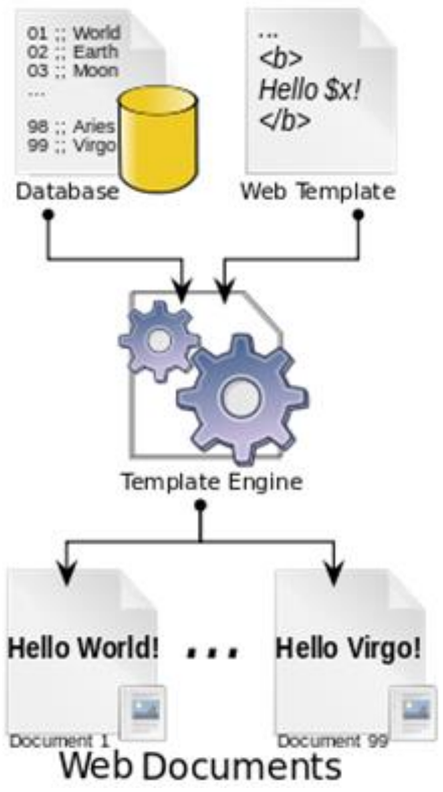


Figure 3-3 The basic process for a server-side web templating system.

[18] The idea that a software system must be decomposed into parts that overlap in functionality as little as possible.

In this way, the logic is in the controllers and the view in the templates. It simplifies the flexibility, but also the reusability of code and allows the content suppliers to focus on content, without the need to know the logic of the application and the programming language.

Figure 3-3 shows the basic process for a server-side web templating system: the template engine collects information from the database, combines it with the template and displays the page. Changing the request, the page pattern is fixed, but the contents change with the requested contents from the database.

Symfony uses a powerful templating language called Twig. Twig allows the user to write concise and readable templates with no PHP code inside, allowing its modification also by unskilled operators.

Twig defines two kinds of delimiters: *{% ... %}* and *{{ ... }}*. The former encloses the programming, the latter prints the result of an expression to the template. Moreover the delimiters *{# #}* enclose the comments into the template.

The Templates are by default in the *app/Resources/views* directory, or into the *path/to/bundle/Resources/views* directory. The first case is used for basic templates (like for layout), the second case for the others: actually, the majority of templates is inside a bundle.

Twig supports inheritance: a child template can extend the basic layout and override any of its blocks. In order to extend a parent template, you should use the command extend:

```
{% extends 'AcmeMainBundle::base.html.twig' %}
```

Like for controllers, there is an easy way to route templates: Symfony uses a *bundle:folder:template* string syntax. In our example, the template extends the "*base.html.twig*" template, into the default template folder inside of the *MainBundle* of the Acme project.

The name of the template, *base.html.twig* is the result of a composition:

- *base :* the name
- *html:* the format

- *twig:* the template engine

Another example of template name may be

```
AcmeMainBundle:Public:index.css.twig
```

It refers to the index template into the *view/Public* directory of the *MainBundle*, and that contains the CSS elaborated with the twig engine.

3.8 Database and Doctrine

Symfony comes with Doctrine, a library that provides powerful tools to easily persist and read information to and from a database.

Doctrine is totally decoupled from Symfony, but their integration is absolute.

Doctrine works with entity classes, placed in the entity folder of the bundles.

> *"The class - often called an "entity", meaning a basic class that holds data - is simple and helps fulfil the business requirement of needing products in your application. This class can't be persisted to a database yet - it's just a simple PHP class.*
>
> *For Doctrine to be able to do this, you just have to create "metadata", or configuration that tells Doctrine exactly how the Product class and its properties should be mapped to the database. This metadata can be specified in a number of different formats including YAML, XML or directly inside the Product class via annotations."*[19]

In the Symfony website,[20] there is a detailed explanation about how Doctrine works.

[19] http://symfony.com/doc/current/book/doctrine.html
[20] As note 19

3.8.1 Annotations

> *"Annotations are meta-meta-object which can be used to describe other meta-object. Meta-object are class, field and method. Asking an object for its meta-object (e.g. anObj.getClass()) is called* **introspection***. The introspection can go further and we can ask a meta-object what are its annotations (e.g. aClass.getAnnotations). Introspection and annotations belong to what is called reflexion and meta-programming."*[21]

An annotation needs to be interpreted in one way or another to be useful. Annotations can be interpreted at development-time by the IDE or the compiler or at run-time by a framework, as Symfony2 does.

Annotation is a powerful mechanism and can be used in many different ways:

- to describe constraints or usage of an element: e.g. @Deprecated, @Override, or @NotNull
- to describe the "nature" of an element, e.g. @Entity, @TestCase, @WebService
- to describe the behaviour of an element: @Statefull, @Transaction
- to describe how to process the element: @Column, @XmlElement

In every case, an annotation is used to describe the element, which is frequently referred as its semantics.

Prior to JDK5[22], the information that is now contained in the annotations needed to be stored somewhere else, and XML[23] files were frequently used.

[21] http://stackoverflow.com/questions/1372876/how-and-where-are-annotations-used-in-java
[22] Java Development Kit.
[23] Extensible Markup Language.

Nevertheless, it is more convenient to use annotations because they will belong to the Java code itself, and are hence much easier to manipulate than XML.

Annotations let you inject behaviour and can promote decoupling. Actually, there are misgivings about the application of annotations in PHP[24-25]. They are configurations, but since PHP only supports them through third party add-ons, they are in comments. A common opinion is that something that is designed for enterprise use should not be using a hack like this. Indeed installing a library, plugin, or module in your application should not oblige you to modify the library code to change a configurable behaviour. This should be done in a centralized configuration location, not chasing down annotations that may be buried in code. Having to pour through library code to chase down a bug or change a configuration is a huge waste of time and resources.

However, in moderation, kept simple and done right, they can make code and configuration simpler and cleaner.

3.8.2 Annotations in Doctrine

One example of a proper use of the annotations would be the Doctrine ORM. Because of the use of annotations, you do not have to inherit from a Doctrine-specific class unlike the Propel ORM. If you did not inherit from a Doctrine class, you would most likely have to use some other metadata specification, like a configuration file, to specify that a particular property is the ID of the record. In that case, it would be too far removed from the syntax that the annotation (metadata) describes.

The Doctrine Common annotations library was born from a need in the Doctrine2 ORM to allow the mapping information to be specified as metadata

[24] http://www.marclewis.com/2013/10/25/php_annotations_are_a_bad_idea/
[25] http://r.je/php-annotations-are-an-abomination.html

embedded in the class files, on properties and methods. The library is independent and can be used in your own libraries to implement doc block annotations.

The documentation process for Doctrine begins with the most basic element of phpDocumentor: a Documentation block or DocBlock[26]. A basic DocBlock looks like this:

```
/**
 *
 */
```

A DocBlock is an extended C++-style PHP comment that begins with "/**" and has an asterisk at the beginning of every line. DocBlocks precede the element they are documenting.

A clear understanding of the inner workings of annotation is important because they are hard to debug. Not having an appropriate awareness of the mechanism can lead us to a huge waste of time.

[26] http://docs.doctrine-project.org/projects/doctrine-common/en/latest/reference/annotations.html

4

THE SOFTWARE – CLIENT SIDE

This chapter describes the S/W from the client’s perspective. It displays the developed web pages, and consequently it helps to better comprehend the operating principle of the application.

The web site is divided into four main areas: the Public area, the Mission area, the Ground Station area and the Management area.

In this chapter and in the following, we use the word *object* to indicate the data uploaded by the GS Operators, including all the fields and the associated file, as explained in Figure 4-7 and in chapter 5.8.1. The reader should not confuse it with the object as defined in chapter 2.2.

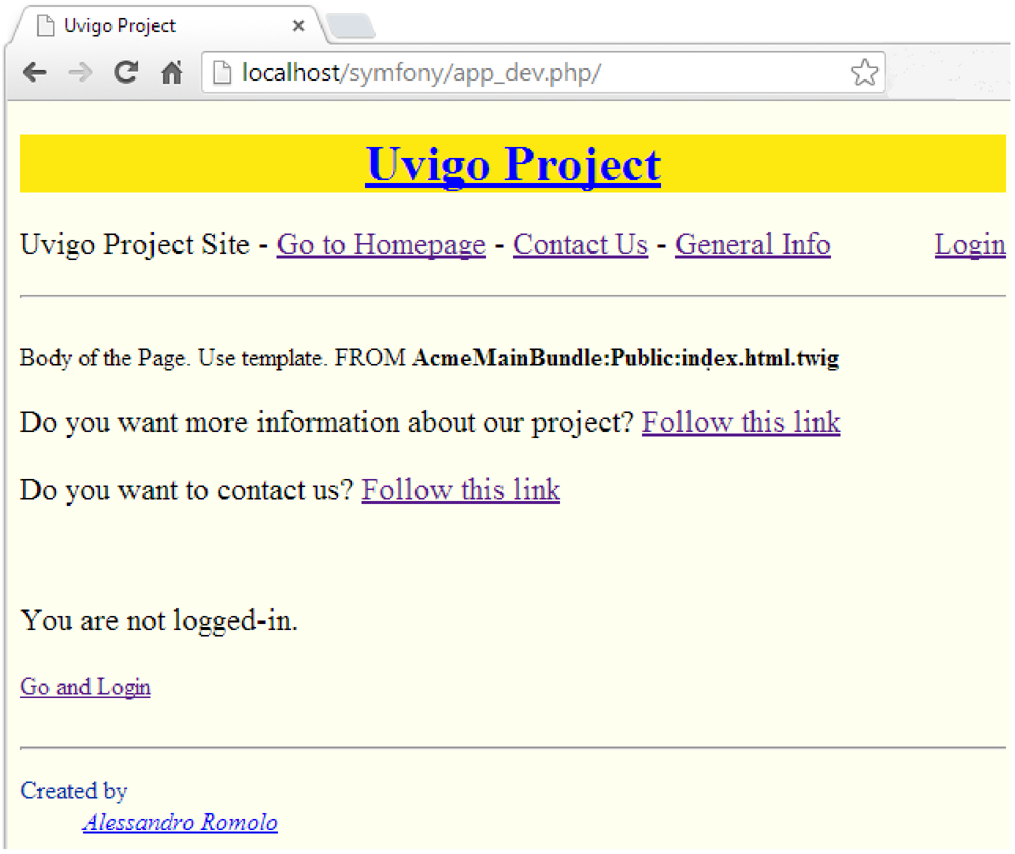
We also use the terms “Ground Station Operator” and “Mission Operator” for the clients, which represent the different kinds of clients.

4.1 The public area

The public area includes a series of pages accessible without authentication.

The pages considered of public interest will be accessible from the bar in the homepage.

The current defined sections can be integrated with more useful pages and information. For example we could add pages for the satellites tracking, with information about satellites modulation, or with pictures and social contents.


Uvigo Project

localhost/symfony/app_dev.php/

Uvigo Project

Uvigo Project Site - Go to Homepage - Contact Us - General Info Login

Body of the Page. Use template. FROM **AcmeMainBundle:Public:index.html.twig**

Do you want more information about our project? Follow this link

Do you want to contact us? Follow this link

You are not logged-in.

Go and Login

Created by
Alessandro Romolo

Figure 4-1 Homepage

The homepage welcomes the user; if the user is not logged in, there is an invitation to login, otherwise it contains the links for the mission area, the ground station area, the management area and the others accessibly pages, depending on the ROLE of the logged user.

Figure 4-1 displays the homepage of the project, when the user is not logged in. The URLs is the one on the local machine, in production environment. To visualize the same page in development environment, the URLs *localhost/symfony/app_dev.php* should be called.

A page will shows information about the project: a presentation, objectives, and the procedure to join and subscribe to the project, as a new ground station operator or as a new mission operator.

Another page, a "contact us" page, contains the contacts to the developers of the project, and the list of the subjects involved in it.

Every page has clearly indicated the relative template, in order to ease the customization.

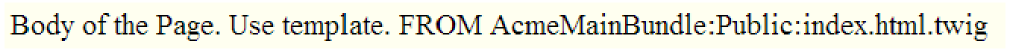

Body of the Page. Use template. FROM AcmeMainBundle:Public:index.html.twig

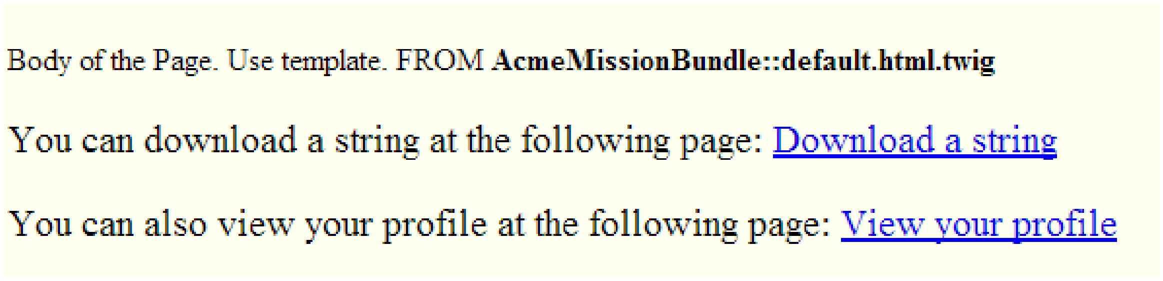
Figure 4-2 Indication of the relative template in the homepage.

The routing system used to indicate the template, is the same of Symfony, as explained in the chapter 3.6; it means that in this case we should modify the template called *"index.HTML.twig"* in the folder *Resources\views\Public* of the *MainBundle*.

4.2 The Mission area

The mission area is dedicated to the operators that need to visualize and download the objects of specific missions.

A *mission homepage*, where the mission operator is redirected after the login, allows the user to reach all the useful pages about his tasks and his own user profile.


Body of the Page. Use template. FROM **AcmeMissionBundle::default.html.twig**

You can download a string at the following page: Download a string

You can also view your profile at the following page: View your profile

Figure 4-3 Mission Operators Homepage's Body

The link *"Download a string"* redirects to the URL *"...mission/download"*: the user has to choose one of the missions associated to him, and to visualize the objects related to that mission. He is authorized to retrieve only the information that concern to him, visualizing in the form only the associated missions.

The associated missions are the missions in which the user is authorized to work. If twenty missions are stored, but the user is authorized to see only products of *Humsat* mission, he is able to query product only from those mission, as shown in Figure 4-4.

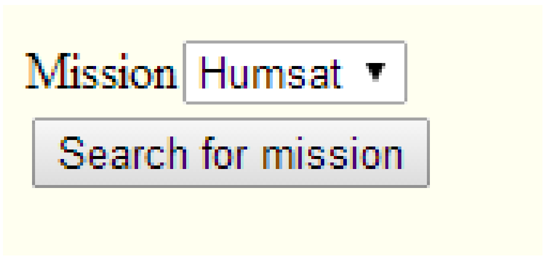
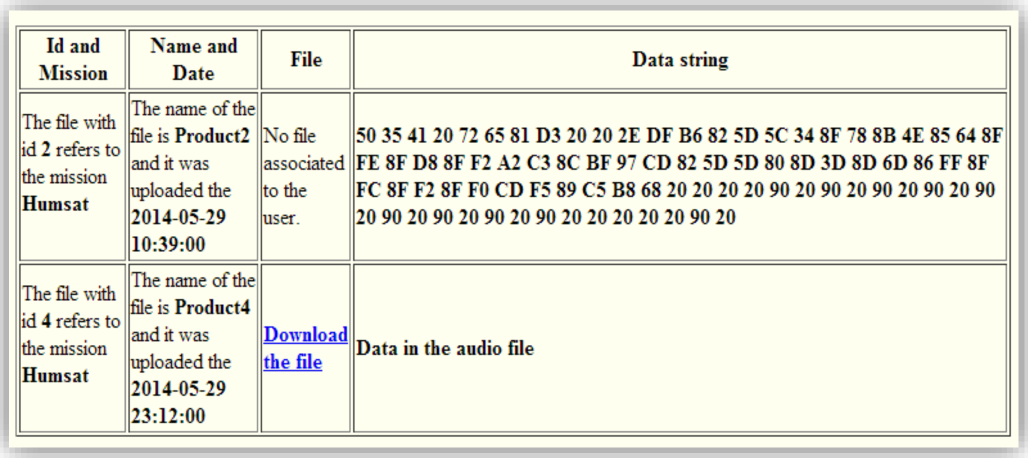


Figure 4-4 Form for the download of the uploaded objects

The visualization also allows the user to download directly, when available, the file associated to the uploaded string.

Figure 4-5 displays how the results of a query are shown to the users.



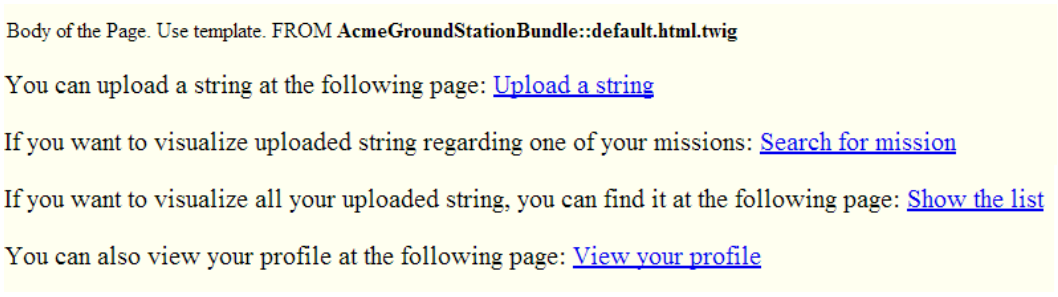
Id and Mission	Name and Date	File	Data string
The file with id **2** refers to the mission **Humsat**	The name of the file is **Product2** and it was uploaded the **2014-05-29 10:39:00**	No file associated to the user.	**50 35 41 20 72 65 81 D3 20 20 2E DF B6 82 5D 5C 34 8F 78 8B 4E 85 64 8F FE 8F D8 8F F2 A2 C3 8C BF 97 CD 82 5D 5D 80 8D 3D 8D 6D 86 FF 8F FC 8F F2 8F F0 CD F5 89 C5 B8 68 20 20 20 20 90 20 90 20 90 20 90 20 90 20 90 20 90 20 90 20 90 20 20 20 20 20 90 20**
The file with id **4** refers to the mission **Humsat**	The name of the file is **Product4** and it was uploaded the **2014-05-29 23:12:00**	Download the file	**Data in the audio file**

Figure 4-5 Screenshot of the visualization of the strings.
Note: Missions' names and data are fictitious

4.3 The Ground Station area

Concurrently, another area is dedicated to the operators of the ground station. The area allows to upload data and to see the uploaded data.

As before, it starts with a *Ground Station homepage*, with useful links and information.



Body of the Page. Use template. FROM **AcmeGroundStationBundle::default.html.twig**

You can upload a string at the following page: Upload a string

If you want to visualize uploaded string regarding one of your missions: Search for mission

If you want to visualize all your uploaded string, you can find it at the following page: Show the list

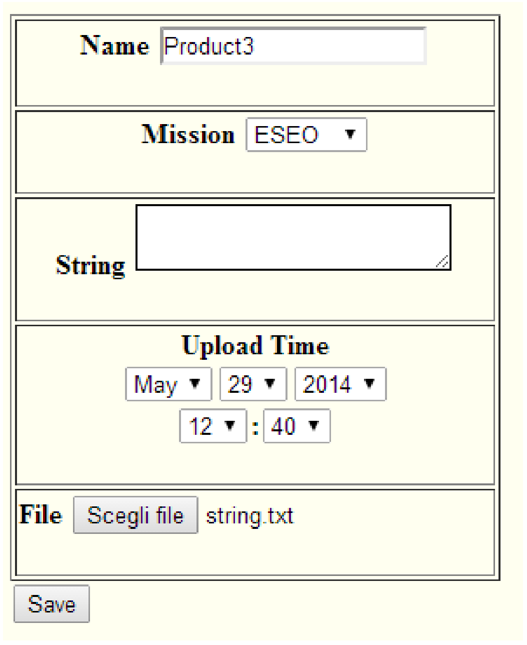
You can also view your profile at the following page: View your profile

Figure 4-6 Screenshot of the Ground Station Homepage's Body

A section is dedicated to download the uploaded string. This section has the same structure of the mission's one; it allows to query and visualize the objects associated to a mission, or to visualize all the objects uploaded by the logged user, independently from the mission.

A different section allows the user to upload the data.

Data uploaded consist in a string, a file audio, and the date. The software automatically integrates the product object, sending information about mission, name and location of the ground station, and other parameters.



Name	Product3
Mission	ESEO ▼
String	
Upload Time	May ▼ 29 ▼ 2014 ▼ 12 ▼ : 40 ▼
File	Scegli file string.txt

Save

Figure 4-7 Form to upload a Product Object

As shown in Figure 4-7, the user has to select a name for the uploaded data, the associated mission, the upload time and a field to choose a file to upload.

Again, the user in the mission field can view and select only missions that he is authorized to work with.

The upload time is automatically set to the current time, and the user can modify it.

4.4 The management area

The management area is not directly referred to the project: it is supposed to manage the users, debug and possibly fit administration needs.

4.4.1 The Admin area

This area is dedicated to the SATNET operators. There is an *Admin homepage*, from which the Admin can visualize all accessible links, news about project, statistics and useful information.

The Admin can upload objects, visualize them and he can save, store and modify the missions.

The upload of the object allows the selection of a user, like anyone of the Ground Station user, as shown in Figure 4-8.

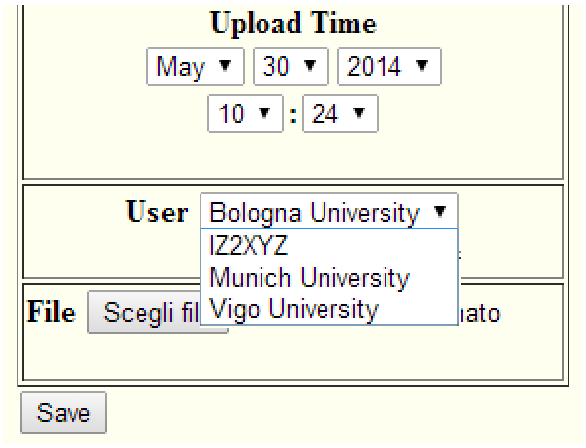
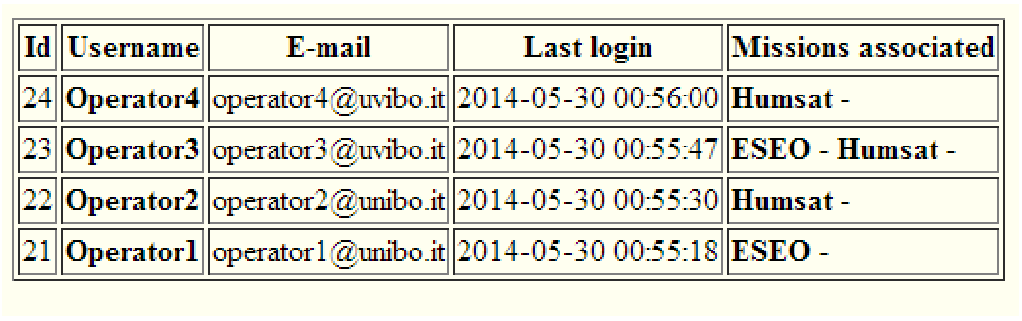


Figure 4-8 Upload object for Admin User

The Admin can visualize the list of the users (Ground Stations and Mission Operators), with information about their associated mission and date of the last login.

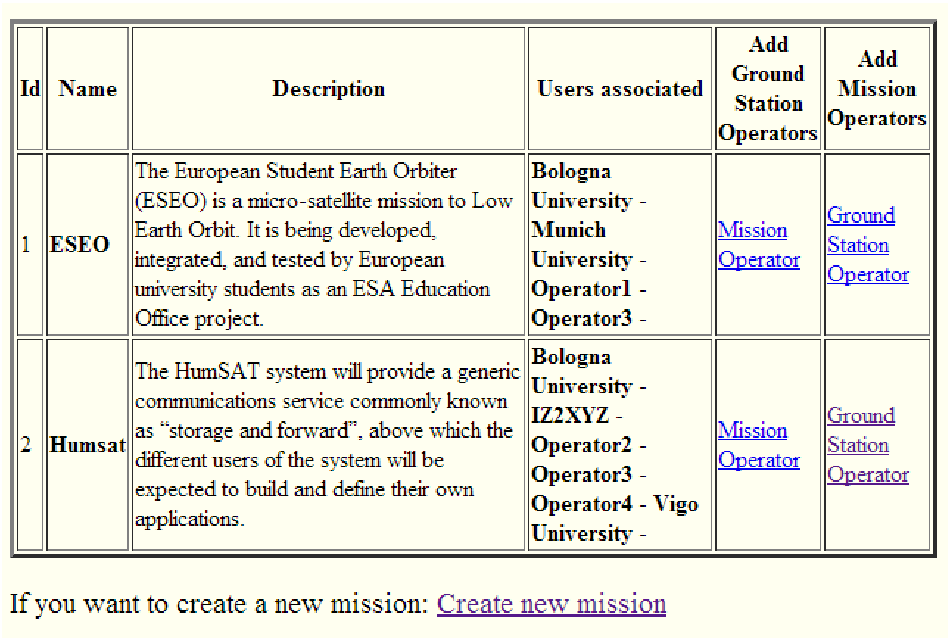


Id	Username	E-mail	Last login	Missions associated
24	**Operator4**	operator4@uvibo.it	2014-05-30 00:56:00	**Humsat** -
23	**Operator3**	operator3@uvibo.it	2014-05-30 00:55:47	**ESEO** - **Humsat** -
22	**Operator2**	operator2@unibo.it	2014-05-30 00:55:30	**Humsat** -
21	**Operator1**	operator1@unibo.it	2014-05-30 00:55:18	**ESEO** -

Figure 4-9 List of the Mission Operators.

The Admin can create a Mission, which is identified by a Name, a Description, and a list of associated users: the only ones who can visualize the objects of a mission, or upload objects related to that mission.

A page displays the list of the missions, associated users and the link to modify it.



Id	Name	Description	Users associated	Add Ground Station Operators	Add Mission Operators
1	**ESEO**	The European Student Earth Orbiter (ESEO) is a micro-satellite mission to Low Earth Orbit. It is being developed, integrated, and tested by European university students as an ESA Education Office project.	**Bologna University** - **Munich University** - **Operator1** - **Operator3** -	Mission Operator	Ground Station Operator
2	**Humsat**	The HumSAT system will provide a generic communications service commonly known as "storage and forward", above which the different users of the system will be expected to build and define their own applications.	**Bologna University** - **IZ2XYZ** - **Operator2** - **Operator3** - **Operator4** - **Vigo University** -	Mission Operator	Ground Station Operator

If you want to create a new mission: Create new mission

Figure 4-10 List of the Stored Missions

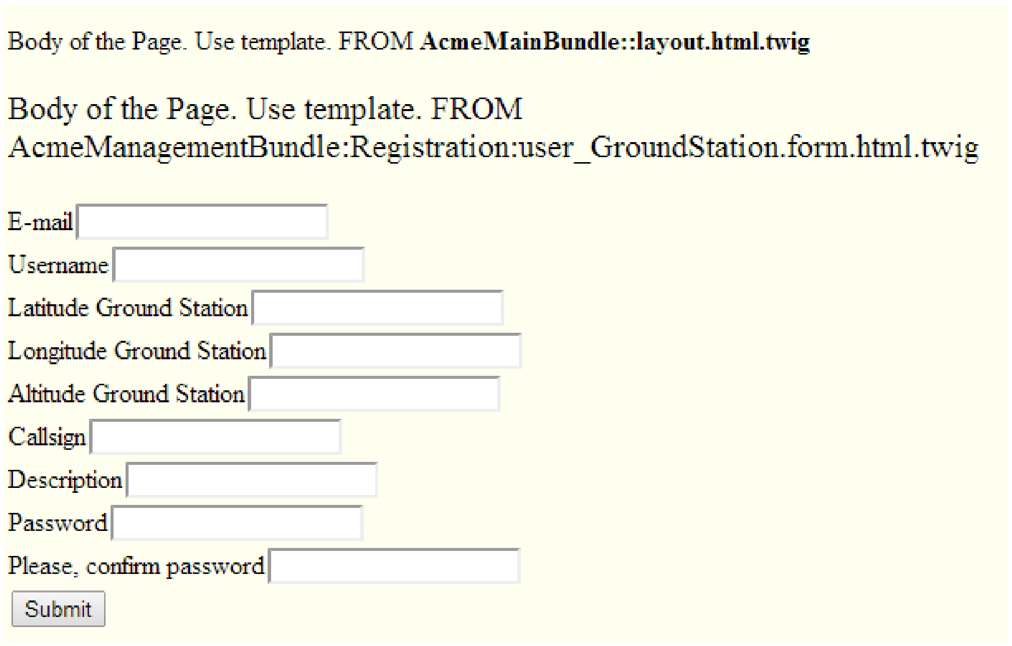
Every mission has two links to be modified: one for the association of Ground Station Operators, and one for Ground Station Operators. The same links allow the customization of the name and the description.

4.4.2 The Users area

There are two user's areas, one dedicated to the *Mission Operators*, and one dedicated to the *Ground Station Operators*.

In this area, the operators can visualize their profile and request adjustments and updates, as change password or delete the user.

In the User Management Area, we include the registration pages. Presently, the registration page is public, accessible by every user. The user can register, but cannot upload, download or visualize anything, until an Admin Operator associates a mission to him. Figure 4-11 shows the registration form for a Ground Station Operator. The form for a Mission Operator is very similar, except for the lack of the Latitude, Longitude and Altitude fields.



Body of the Page. Use template. FROM **AcmeMainBundle::layout.html.twig**

Body of the Page. Use template. FROM AcmeManagementBundle:Registration:user_GroundStation.form.html.twig

E-mail	
Username	
Latitude Ground Station	
Longitude Ground Station	
Altitude Ground Station	
Callsign	
Description	
Password	
Please, confirm password	

Figure 4-11 Ground Station Registration Page

Once registered, the user does not have any associated mission, and cannot access the database: he needs to wait for the Admin approval.

5

THE SOFTWARE – STRUCTURE AND OPERATING PRINCIPLE

The web site developed for this thesis is a dynamic web site, and can be considered like a real software, with its programming logic.

The Symfony2 framework helped the development of the software; therefore, its structure follows the Symfony2 structure, in which every namespace is represented by a bundle that develops a specific task.

Some bundles are open source vendors, specifically downloaded in order to integrate the application. Some others are the results of a scripting and compiling work, which complies with the request of securely storing information.

5.1 Structure of the software (bundles)

The software is divided into four compiled bundles, plus the vendors. The Bundles are called *MainBundle*, *MissionBundle*, *GroundStationBundle* and *ManagementBundle*.

The structure chosen presents advantages and disadvantages: we endorsed the easiness of future modifications, sacrificing the principles of not repeatability of code.

The *MainBundle* contains the public part of the web site, the layout templates, CSS, and the resources accessible without authentication.

The *MissionBundle* is dedicated to the Mission Operator.

The *GroundStationBundle* is dedicated to the Ground Station Operators, and includes the product entity related to the uploaded file.

The *ManagementBundle* is dedicated to all that concern the management of the web site, the Admin area, the mission and users entities and all the resources accessible only by the administrators.

Moreover, we installed some vendors. We downloaded two open source bundles, to implement specific functions: in particular, *FOSUserBundle* and *PUGXMultiUserBundle* allow the management of the users.

Other vendors are automatically integrated with Symfony; the *SecurityBundle*, the *AsseticBundle*, the *FrameworkBundle*, the *DoctrineBundle*, the *TwigBundle* and others.

Table 1 outlines the chosen structure.

Table 1 Bundles Path

Name of the Bundle	***Path***
• **MainBundle**	*[symfony-path]\src\Acme\...]*
o Public part of the web site	
✓ It partially overrides the FosUserBundle	
• **MissionBundle**	*[symfony-path]\src\Acme\...]*
o Dedicated to Mission Operators	
• **GroundStationBundle**	*[symfony-path]\src\Acme\...]*
o Dedicated to Ground Station Operators	
✓ Stores the Product Entity.	
• **ManagementBundle**	*[symfony-path]\src\Acme\...]*
o Dedicated to the management of the application.	
✓ Stores the Mission Entity.	
✓ Stores the User, UserGroundStation and UserOperator Entities.	
• **FosUserBundle**	*[symfony-path]\vendor\friendsofsymfony\...]*
o Adds support for a database-backed user system.	
• **PUGXMultiUserBundle**	*[symfony-path]\vendor\pugx\...]*
o An extension for FOSUserBundle to handle users of different types	
• **DoctrineBundle**	*[symfony-path]\vendor\doctrine\...]*
o Doctrine integration.	
• **TwigBundle**	*[symfony-path]\vendor\twig\...]*
o Twig engine integration.	

…

The file called *routing.yml* in the *config* folder of every bundle stores the routing of the application. These files are imported by a main file into the *app/config* folder, following the scheme in Table 2.

Table 2 Routing scheme of the bundles

Routing Hierarchy

• **app/config/routing.yml**	**Prefix**
○ AcmeIndexBundle	^/
○ AcmeMissionBundle	^/mission/
○ AcmeGrounstationBundle	^/groundstation/
○ AcmeManagement	^/admin/
○ FosUserBundle	^/

5.2 Security

The *SecurityBundle* manages the security of the application. The configuration file, that is the only one that we have to take in account, is in the */app/config/* folder of Symfony, and it is called *security.yml.*

A whole chapter of the Symfony book[27] explains how this configuration file works.

Table 3 shows the *security.yml* file of our project. It uses *FOSUserBundle* for encoders and users provider.

[27] http://symfony.com/doc/current/book/security.html

Table 3 security.yml

```
security:
    encoders:
        FOS\UserBundle\Model\UserInterface: sha512

    role_hierarchy:
        ROLE_ADMIN:       [ROLE_OPERATOR, ROLE_GS ]
        ROLE_SUPER_ADMIN: [ROLE_ADMIN, 
ROLE_ALLOWED_TO_SWITCH]
        ROLE_OPERATOR:
        ROLE_GS:
    providers:
        fos_userbundle:
            id: fos_user.user_provider.username

    firewalls:
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
                csrf_provider: form.csrf_provider
            logout:       true
            anonymous:    true

    access_control:
        - { path: ^/login$, role: 
IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/register, role: 
IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: 
IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/, role: ROLE_ADMIN }
        - { path: ^/mission, roles: ROLE_OPERATOR }
        - { path: ^/groundstation, roles: ROLE_GS}
```

The roles are three: *ROLE_GS*, *ROLE_OPERATOR* and *ROLE_ADMIN*. They are integrated by the *ROLE_SUPER_ADMIN* and by the *ROLE_ALLOWED_TO_SWITCH*.

Eventually, the access control defines the restriction and privileges of every role: the path */mission* is accessible only by Mission Operators, the path

/groundstation by Ground Station Operators, and the path */admin* by the Admins. The login, register and resetting pages are accessible without authentication.

5.2.1 Cross-site scripting

The **XSS**, cross-site scripting, is a vulnerability, specific of dynamic web sites. When a web site is meant to receive data from a client, the client could inject client-side script into the page, and commit illicit actions.

Three different options can solve this vulnerability. In Symfony2, the answer to the problem is *output escaping*, enabled by default using Twig: the site is automatically protected from the unintentional consequences of a XSS attack, until it use twig as template engine. Thus, we do not have to worry about it.

There are also ways to protect the web site against the **Cross-site request forgery** in login form: in few passages, we can activate the protection[28].

5.3 FOSUserBundle

> *"The FOSUserBundle adds support for a database-backed user system in Symfony2. It provides a flexible framework for user management that aims to handle common tasks such as user registration and password retrieval"*[29].

It is related to the User Area described in 4.4.2.

Features include:

- Users can be stored via Doctrine ORM, MongoDB/CouchDB ODM or Propel

[28] http://henrik.bjrnskov.dk/symfony2-cross-site-request-forgery
[29] https://github.com/FriendsOfSymfony/FOSUserBundle

- Registration support, with an optional confirmation via e-mail
- Password reset support
- Unit testing

The whole install process is properly described here on https://github.com[30], and it consists of:

a) Download FOSUserBundle using composer
b) Enable the bundle in the Kernel
c) Create the User class
d) Configure the application's security.yml
e) Configure the FOSUserBundle. config.yml
f) Import FOSUserBundle routing files
g) Update the database schema

We imported the *FOSUserBundle* routing files in *YAML*[31] in the *ManagementBundle* routing file and used ORM[32] as datastore. The class name of the User class it is not important because we modified it at a later stage, importing *PUGXMultiUserBundle*.

The *FOSUserBundle* webpage[33] contains the whole documentation for this Bundle.

In order to adapt the bundle to our use, we changed three templates:

- Resources/views/layout.html.twig
- Resources/views/Security/login.html.twig
- Resources/views/Profile/show_content.html.twig

[30] https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/index.md
[31] YAML is a recursive acronym for "YAML Ain't Markup Language", and it is a human-readable data serialization format.
[32] Object-relational mapping, programming technique for converting data between incompatible type systems.
[33] https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/index.md#next-steps

5.4 Overriding a bundle

As written above, we need to change some settings, parameters and characteristic of the controller of our vendors. Symfony provides an easy way to override things like controllers, templates, and other files in a bundle's Resources/ directory.

Override means that you take an existent function and "shadow" it by redefining elsewhere, where it can be used instead of the original one. This is a common technique for add extra functionalities to a function or to change the function itself.

To override a bundle's template you could simply place a new one in your *app/Resources* folder of the bundle, or directly modify the template already in the folder. To override the layout template located *at Resources/views/layout.html.twig* in the *FOSUserBundle* directory, you should place your new layout template in the *app/* subdirectory, and then at *Resources/FOSUserBundle/views/layout.html.twig.*

Changing directly the code of the vendors can be dangerous for many reasons, we list here the main drawbacks:

- If we need to update the vendors, using the composer erases every modifications we made.
- If more than one bundle in the project need to use the vendor, the custom behaviour makes sense for a bundle but not for the other one. Customizing the behaviour at local bundle level helps to keep logic intact and avoid problems.
- If the bundle is shared with other user, the modification force them to take the customized version of the vendor, without permission of updating it.

The overriding technique takes advantage of inheritance: it is well described in the Symfony website[34].

To customize some functions and templates of the *FOSUserBundle*, we overrode it in the *MainBundle*.

We added the following lines to the main file, the one we use to register the bundle[35], of the *MainBundle*:

```
public function getParent()
{
  return 'FOSUserBundle';
}
```

Then we just move the files that we want to override into the MainBundle, reproducing the "tree-folder-structure" of the original bundle until reaching the class that contains the function to override.

Reading the documentation and visualizing its real implementation into the main bundle makes it clearer to the reader.

5.5 PUGXMultiUserBundle

"PUGXMultiUserBundle came by the need to use different types of users using only one fos_user service. In practice, it is a hack that forces FOSUser bundle through custom UserManager, controllers, and forms handlers. It's a fast way to use for free most of the functionality of FOSUserBundle"[36].

[34] http://symfony.com/doc/current/cookbook/bundles/inheritance.html
[35] *C:\BitNami\wampstack-5.4.23-0\frameworks\symfony\src\Acme\MainBundle\AcmeMainBundle.php*
[36] https://github.com/PUGX/PUGXMultiUserBundle/blob/master/Resources/doc/index.md

The whole install process is properly described in https://github.com[37]. It consists of:

a) Downloading PUGXMultiUserBundle
b) Enabling the Bundle
c) Creating your Entities
d) Configuring the FOSUserBundle (PUGXMultiUserBundle params)
e) Configuring parameters for UserDiscriminator
f) Creating your controllers
g) Using the User Manager

Obviously, as for the previous vendors, we imported the *FOSUserBundle* routing files in YAML in the *ManagementBundle* routing file and we used ORM as datastore. The bundle has been realized as a part of a real application that uses doctrine ORM, thus it only supports the ORM database driver.

5.6 MainBundle

The *MainBundle* contains the public part of the web site, the layout templates, the CSS, and the resources accessible without authentication.

It also partially overrides the *FOSUserBundle*, as explained in 5.4. Due to this overriding, there is the command folder, added in order to register an Admin User, as explained in 6.6.1.

5.6.1 Controllers

The Bundle comprehends two controllers: the *Redirecting Controller* and the *Default Controller*.

[37] https://github.com/PUGX/PUGXMultiUserBundle/blob/master/Resources/doc/index.md

The *Default Controller* simply allows the rendering of the templates of public access.

The *Redirecting Controller* redirects URLs with a trailing slash, as explained in chapter 5.11.

5.6.2 Routing

The pages of the *MainBundle* can be virtually divided into three main sections. The routing follows this same division, with three main sections.

The first section routes the pages of public access, like the homepage, the page with the information about the project and the page to contact the administrators.

The second section allows the Users to register. It comprehends two different route, for registration of a Ground Station Operator or a Mission Operator. Now these parts are public, accessible to everyone, and once registered a user can already enter into his specific section. Actually, there is not a security lack, until the user cannot do anything until authorized by an administrator with an association to a mission: it is like have an inactive user. In the future, the registration form can be moved in a protected area, or an Invitation Model can request[38] the registration.

The last part calls the routing of the *FosUserBundle*, in order to see or edit the profile, change password, etc. You can import more routes, implementing the functions of the *FosUserBundle*: they can be found in the routing folder into the *FosUserBundle* directory[39].

---

[38] https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/adding_invitation_registration.md

[39] In our project it is in *C:\BitNami\wampstack-5.4.23-0\frameworks\symfony\vendor\friendsofsymfony\user-bundle\FOS\UserBundle\Resources\config\routing.yml*

5.6.3 Templates

The *Resource* folder contains the templates of the public part of the website, the template of the login page, the template that confirms the user registration and shows the user profile.

There are also two layout templates. The *base* template is the fundamental template of all the application, with the layout of the website and extended by every other template. The *layout* template is the overriding of the main template of the *FOSUserBundle*, and allows the integration of the style between the templates in the *FosUserBundle* and all the others.

They are organized as shown in Table 4.

Table 4 MainBundle Templates

Folder[40]	***Name***[41]	***Extends***
• /		
	✓ base	no extend
	✓ layout	AcmeManagementBundle::default.HTML.twig
• Security		
	✓ login	FOSUserBundle::layout.HTML.twig
• Public		
	✓ info	AcmeMainBundle::base.HTML.twig
	✓ contact	AcmeMainBundle::base.HTML.twig
	✓ index	AcmeMainBundle::base.HTML.twig
• Profile		
	✓ show_content	trans_default_domain 'FOSUserBundle
• Registration		
	✓ confirmed	FOSUserBundle::layout.HTML.twig

[40] In this and in the following tables, the folder of the bundle is intended as the sub-path from the *[Bundle]\Resources\views* path.
[41] In this and in the following tables, the name of the templates is indicated without the extension .html.twig.

5.7 MissionBundle

The *MissionBundle* develops the part of web site dedicated to the Mission Operators. It requests the *ROLE_OPERATOR* role to be seen.

5.7.1 Controllers

The *MissionBundle* contains two controllers.

The Default Controller includes the *mainAction* that render the Mission Operator Homepage.

The Product Controller includes the *showAction* that queries and shows the products, and the *downloadAction*, which allows the download of the uploaded files associated with the product objects.

5.7.2 Routing

All the routes have the *mission/* prefix, in order to identify and secure this section of the web site.

The bundle has three routes: one is the mission operator homepage, one is the page for the query and the display of a list of objects, and the last one is the route for the download of the uploaded files associated with the product objects.

5.7.3 Templates

There is a template for every page. The template *default.HTML.twig* is the Mission Operator homepage and defines the layout, the header and the footer of the section.

Furthermore, there is a template for the query of the product entities and one for the display. They are organized as shown in Table 5.

Table 5 MissionBundle Templates

Folder	*Name*	*Extends*
• /		
	✓ default	AcmeMainBundle::base.HTML.twig
• Product		
	✓ showlist	AcmeMissionBundle::default.HTML.twig
	✓ querydata	AcmeMissionBundle::default.HTML.twig

5.8 GroundStation Bundle

The *GroundStationBundle* develops the web site part dedicated to the Ground Station Operators, and contains the Product entity.

This Bundle is similar to the *MissionBundle*, but implements more functionalities. Indeed, the Ground Station Operator, as well as the Mission Operator, can visualize the uploaded strings, but can also upload them and manage more functions

5.8.1 The Product Entity

In the *GroundStationBundle*, in the Entity folder, the file *Product.php* represents the product entity.

Product is the object uploaded by the Ground Station Operator, which can be displayed and downloaded by the Mission Operators.

The product entity includes many properties[42]:

• Id	*object identifier*
• Name	*object name*
• User	*associated user*

[42] Class member variables are called "properties". You may also see them referred to using other terms such as "attributes" or "fields". They are defined by using one of the keywords public, protected, or private, followed by a normal variable declaration.

• Data	*object data*
• uploadTime	*uploaded date and time*
• path	*stored file directory*
• file	*stored file name*
• temp	*temporary variable*

Every product object is defined by all these properties: they allow the user to identify univocally the object, and to search it by mission, data or user. Every object also has an associated file, intended to store audio or text information not suitable for the form.

5.8.2 Controllers

The *GroundStationBundle* contains two controllers.

The *DefaultController* includes the *mainAction* that renders the Ground Station Operator Homepage.

The *ProductController*, as in the *MissionBundle*, contains the *showAction* that queries and shows the objects, and the *downloadAction* for the download of the files associated with the objects.

This controller contains two more actions: the *createAction* and the *myproductsAction*. The former allows the user to upload a product object, the latter shows the products uploaded by the user.

5.8.3 Routing

Also he routing system is similar to the one in the *MissionBundle*.

This time, all routes have the *groundstation/* prefix, in order to identify and secure this section of the web site.

We developed five routes: three like in the *MissionBundle*, respectively for the Ground Station Operators homepage, for the query and the display of a list of objects, and for the download of the uploaded files associated with the product objects. They are integrated by two more routes: one for the upload of the products, and one for the display of the products uploaded by the logged user.

5.8.4 Templates

Also the templates are similar to the mission's ones: indeed, in this bundle we find the template default, for the homepage and layout, and the two templates for the query of the product entities and their display.

In addition, two templates refer to the two routing for the upload of the products and for the display of the products uploaded by the user.

They are organized as shown in Table 6.

Table 6 GroundStationBundle Templates

Folder	***Name***	***Extends***
• /		
	✓ default	AcrmeMainBundle::base.HTML.twig
• Product		
	✓ showlist	AcrmeGroundStationBundle::default.HTML.twig
	✓ querydata	AcrmeGroundStationBundle::default.HTML.twig
	✓ formupload	AcrmeGroundStationBundle::default.HTML.twig
	✓ tasksuccess	AcrmeGroundStationBundle::default.HTML.twig

5.9 ManagementBundle

The *ManagementBundle* is dedicated to the management of the web site, the Admin area and the resources accessible only by the administrator.

It contains the missions and users entities, and the forms for the registration of the user and the display of the profile info.

Again, it includes the function already implemented by Mission and *GroundStationBundle*, integrated by many functions prerogative of the Admin.

Consequently, also the structure about routing, controllers and templates, is similar to the *GroundStationBundle*, integrated and completed.

5.9.1 The Mission Entity

The mission entity allows the Admin user to store the missions as objects. It includes the following properties:

• Id	*object identifier*
• Name	*object name*
• Description	*mission description*
• Users	*associated users*
• Products	*associated products*

Every mission has three properties that define the mission, and two properties that put in correlation the mission with the users, and the mission with the associated product objects. The associated products objects are all the uploaded products that refer to that mission. Consequently, we can choose a mission and visualize all object referred to that mission. In the same way, we can check all the users allowed to see or modify the mission’s product objects.

5.9.2 The User Entities

The User Entities are three: one for the Ground Station Operator, one for the Mission Operator, and the last one, extended by the other two, which implements the method and property in common between the two roles.

The *user.php* entity extends the Base user of the *FosUserBundle*, and includes the following properties:

• Id	*object identifier*
• Mission	*associated missions*
• Products	*uploaded products*

We have an Id that identifies the user, and the other two *@ManytoMany* and *@OnetoMany* correlation[43] with missions and products.

The UserOperator does not have new properties, just new actions in order to set the role on the registration.

The UserGroundStation adds, to the already discussed properties, the latitude and the longitude, in order to place the Ground Station on a map. Also this entity sets the role on the registration.

5.9.3 Controllers

The *ManagementBundle* contains four controllers.

As in the previous bundle, there is a *DefaultController*: it renders the homepage, but also allows the Admin to see the list of the users, in two different pages for the two different roles. It also contains the *MissionListAction*, which displays the stored missions.

The Product Controller contains the same Actions as the one in the *GroundStationBundle*, the difference is in the privileges: the Ground Station Operator can see and store only objects related to his associated missions, while the Admin operator does not have this limitation. Therefore, it contains the *showAction*, the *downloadAction*, the *createAction* and the *myproductsAction*, as already described.

The *RegistrationController* contains the two action that, as explained in the 5.5, allow the users to register as Ground Station and Mission Operator, respectively.

[43] We choose the correlation *@OnetoMany* Mission to Object, so every object is associated to a single mission, but every mission has associated more than one object. We also choose the correlation *@ManytoMany* between Users and Missions, and between Users and Objects, with the consequently meaning.

The last one is the *MissionController,* for the management of the mission entity. It allows to create a new mission and to edit it. A supplementary action redirects URLs with a trailing slash, in order to avoid trivial errors in the routing.

5.9.4 Routing

In the Admin section, all the pages have the *admin/* prefix.

Again, the scheme is the same: one route for the homepage, and four for the managing of the product entity.

In addition, there are four routes for the creation and editing of the mission entity, one for the redirect of URLs with trailing slash, and three more routes for the display of the stored missions, and of all the users divided by role.

5.9.5 Templates

The templates are organized like in the *GroundStationBundle*, where we find the same five templates.

What differs is the presence of an index template: for Admin we divided the template containing the layout, header and footer, from the template with the body of the Admin homepage, called *index*.

In addition, a Mission folder contains the templates directly referred to the management of the Mission Entity: to create and edit, to confirm the success of the operations, and to display the stored missions.

Eventually, two templates display the list of the users: one for the Ground Station Operators and one for the Mission Operators.

They are organized as shown in Table 7.

Table 7 ManagementBundle Templates

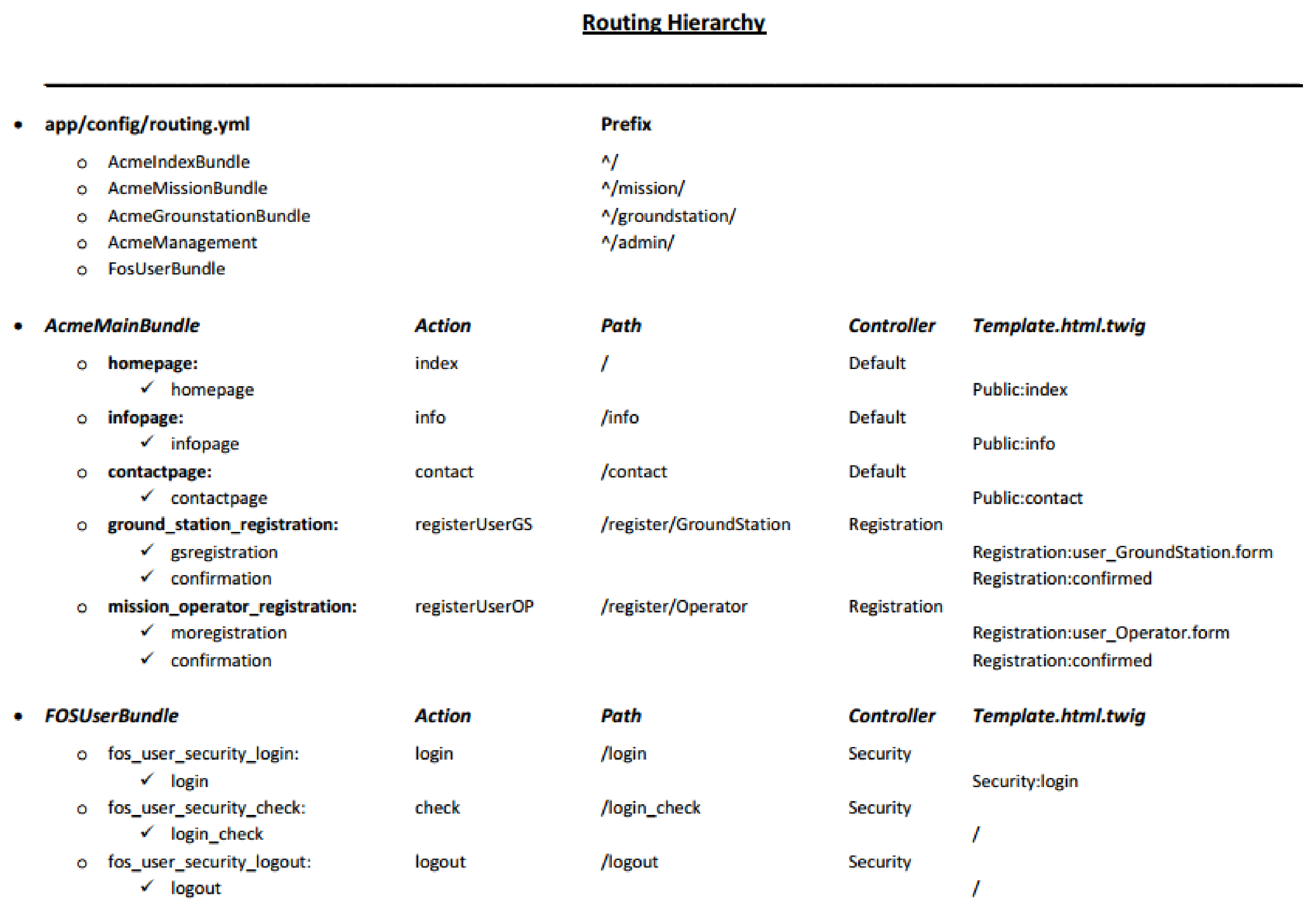
Folder	***Name***	***Extends***
• /		
	✓ default	AcrmeMainBundle::base.HTML.twig

- Admin
  - ✓ index AcmeManagementBundle::default.HTML.twig
- Product
  - ✓ showlist AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ querydata AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ formupload AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ tasksuccess AcmeManagementBundle:Admin:index.HTML.twig
- Registration
  - ✓ User_GroundStation.form FOSUserBundle::layout.HTML.twig
  - ✓ User_Operator.form FOSUserBundle::layout.HTML.twig
- Mission
  - ✓ create AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ tasksuccess AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ edit AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ editsuccess AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ showlist AcmeManagementBundle:Admin:index.HTML.twig
- Users
  - ✓ showOPlist AcmeManagementBundle:Admin:index.HTML.twig
  - ✓ showGSlist AcmeManagementBundle:Admin:index.HTML.twig

5.10 Software Scheme

The following scheme shows the software structure, with routing, controllers, actions, path and templates. The routes are divided by Bundles, and every route shows its associated path, with the relative controller and action. Some paths use only one template; some others use more than one template, for example for the query and the display of query results. In both cases, we indicated the templates, one or two, with their relative page.

Table 8 Software schema – part 1



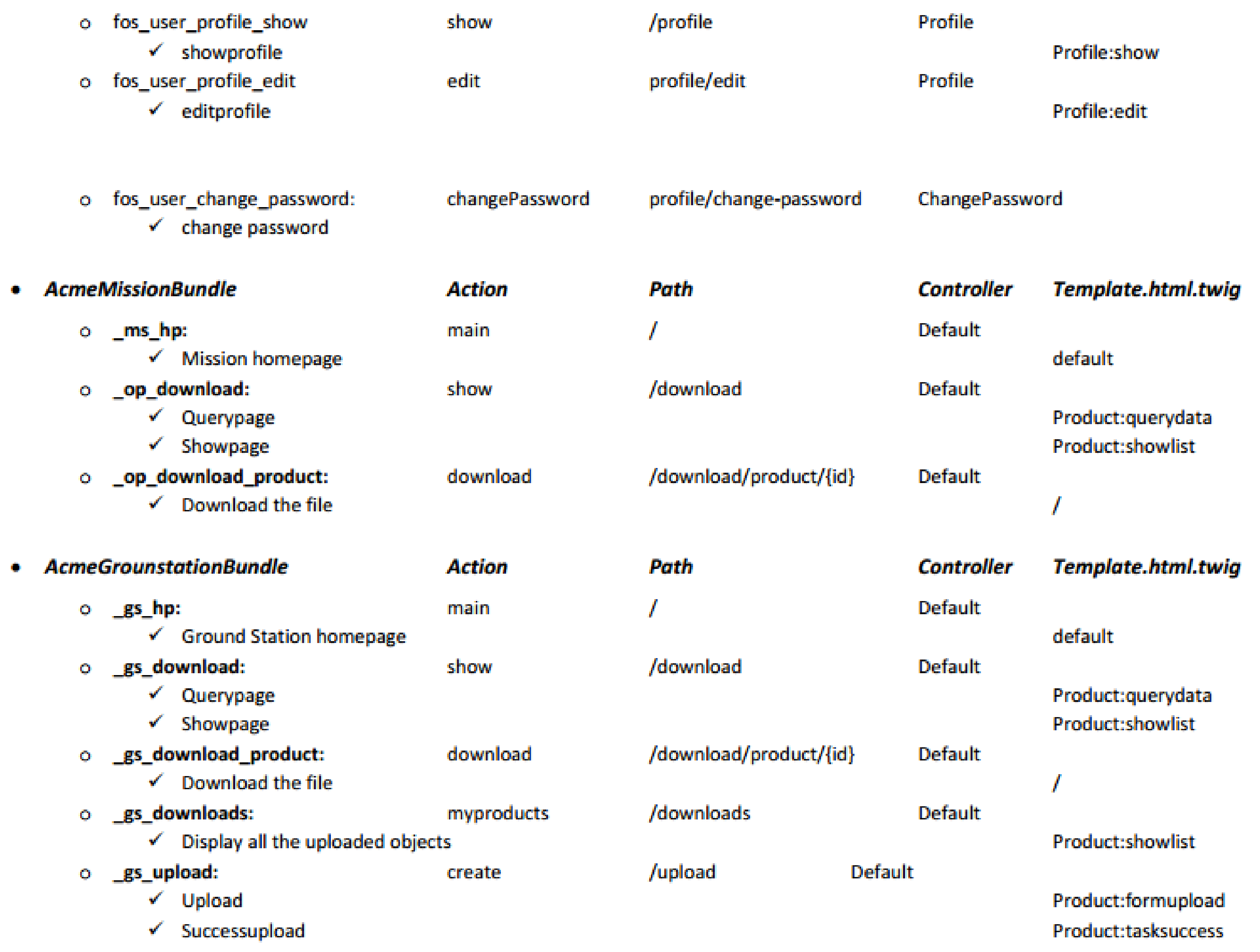
Routing Hierarchy

app/config/routing.yml	**Prefix**
AcmeIndexBundle	^/
AcmeMissionBundle	^/mission/
AcmeGrounstationBundle	^/groundstation/
AcmeManagement	^/admin/
FosUserBundle	

AcmeMainBundle	***Action***	***Path***	***Controller***	***Template.html.twig***
homepage:	index	/	Default	
✓ homepage				Public:index
infopage:	info	/info	Default	
✓ infopage				Public:info
contactpage:	contact	/contact	Default	
✓ contactpage				Public:contact
ground_station_registration:	registerUserGS	/register/GroundStation	Registration	
✓ gsregistration				Registration:user_GroundStation.form
✓ confirmation				Registration:confirmed
mission_operator_registration:	registerUserOP	/register/Operator	Registration	
✓ moregistration				Registration:user_Operator.form
✓ confirmation				Registration:confirmed

FOSUserBundle	***Action***	***Path***	***Controller***	***Template.html.twig***
fos_user_security_login:	login	/login	Security	
✓ login				Security:login
fos_user_security_check:	check	/login_check	Security	
✓ login_check				/
fos_user_security_logout:	logout	/logout	Security	
✓ logout				/

Table 9 Software schema – part 2



	Action	Path	Controller	Template.html.twig
○ fos_user_profile_show	show	/profile	Profile	
✓ showprofile				Profile:show
○ fos_user_profile_edit	edit	profile/edit	Profile	
✓ editprofile				Profile:edit
○ fos_user_change_password:	changePassword	profile/change-password	ChangePassword	
✓ change password				
• ***AcmeMissionBundle***	***Action***	***Path***	***Controller***	***Template.html.twig***
○ **_ms_hp:**	main	/	Default	
✓ Mission homepage				default
○ **_op_download:**	show	/download	Default	
✓ Querypage				Product:querydata
✓ Showpage				Product:showlist
○ **_op_download_product:**	download	/download/product/{id}	Default	
✓ Download the file				/
• ***AcmeGrounstationBundle***	***Action***	***Path***	***Controller***	***Template.html.twig***
○ **_gs_hp:**	main	/	Default	
✓ Ground Station homepage				default
○ **_gs_download:**	show	/download	Default	
✓ Querypage				Product:querydata
✓ Showpage				Product:showlist
○ **_gs_download_product:**	download	/download/product/{id}	Default	
✓ Download the file				/
○ **_gs_downloads:**	myproducts	/downloads	Default	
✓ Display all the uploaded objects				Product:showlist
○ **_gs_upload:**	create	/upload	Default	
✓ Upload				Product:formupload
✓ Successupload				Product:tasksuccess

Table 10 Software schema – part 3

• ***AcmeManagementBundle***	***Action***	***Path***	***Controller***	***Template.html.twig***
○ **_ad_hp**:	main	/	Default	
✓ Admin homepage				Admin:index
○ **_ad_download:**	show	/download	Default	
✓ Querypage				Product:querydata
✓ Showpage				Product:showlist
○ **_ad_download_product:**	download	/download/product/{id}	Default	
✓ Download the file				/
○ **_ad_downloads:**	myproducts	/downloads	Default	
✓ Display all the uploaded objects				Product:showlist
○ **_ad_upload:**	create	/upload	Default	
✓ Upload				Product:formupload
✓ Successupload				Product:tasksuccess
○ **gs_list:**	GSList	/list/groundstation	Default	
✓ GSList				Users:showGSlist
○ **op_list:**	OPList	/list/operator	Default	
✓ OPList				Users:showOplist
○ **ms_list:**	MissionList	/list/mission	Default	
✓ MissionList				Missin:showlist
○ **create_mission:**	Create	/new/mission	Mission	
✓ Create a mission				Mission:create
✓ Task success				Mission:tasksuccess
○ **edit_missions:**	Redirect	/edit/mission	Mission	
✓ redirect to ms_list				/
○ **edit_mission_gs:**	EditGs	/edit/mission/gs/{name}	Mission	
✓ Edit a mission				Mission:edit
✓ Edit success				Mission:editsuccess
○ **edit_mission_op:**	EditGs	/edit/mission/gsop/{name}	Mission	
✓ Edit a mission				Mission:edit
✓ Edit success				Mission:editsuccess

5.11 Redirect URLs with a Trailing Slash

A simple method allows redirecting automatically a user who search for URLs with a trailing slash, avoiding errors and improving the performance of the web site.

It consists adding the affected URLs:

```
remove_trailing_slash:
    path: /{url}
```

And a function in a controller

```
public function removeTrailingSlashAction
```

The procedure is well explained in the Chapter 78 of Symfony Cookbook[44].

[44] http://symfony.com/doc/current/cookbook/routing/redirect_trailing_slash.html

6

INSTALLATION ON THE SERVER AND CUSTOMIZATION

This chapter is meant to be a guide for the installation of the software in a server. It is divided into two principal section; one for installation and one for customization and maintenance.

The first section is divided in turn into a few sections:

- A section about "installation of Bitnami WAMP", to install PHP, MySQL, Apache and Symfony2.
- A section about *"installation of the Bundles"* will help to install and activate the bundles.

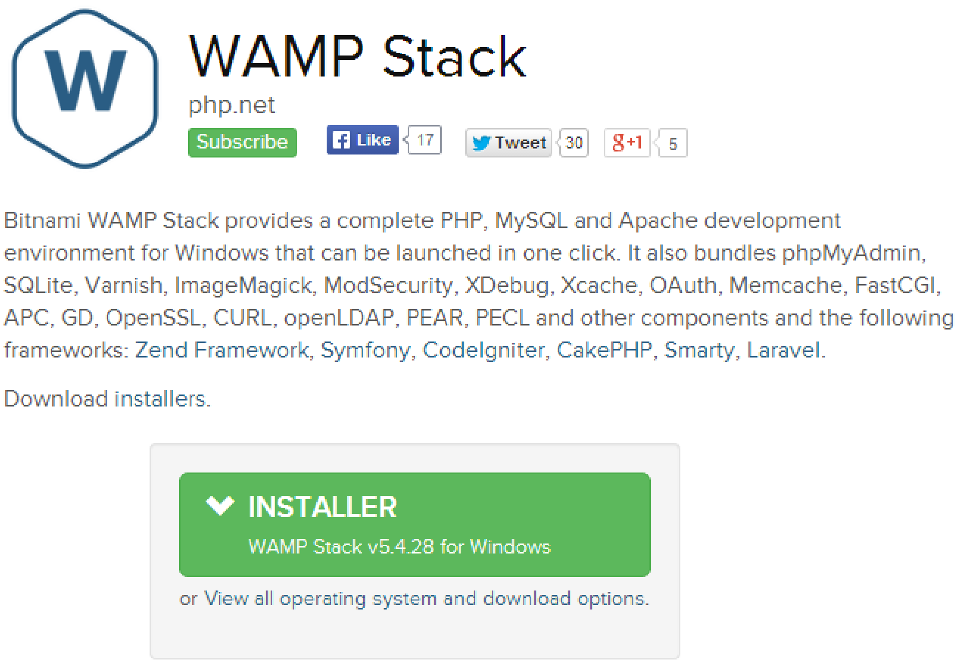
- A section about "installation of the vendors" will help to install and activate the vendors, and to integrate it with the software.
- A section about "Settings, Routing and Configuration files" will help the user to import configuration files in the proper folders.

The second part is in turn divided into:

- A section about the customization of the settings.
- A section that help the user in the customization of the templates.
- A section about testing and maintenance.

6.1 Installation of the Bitnami WAMP with Symfony2

Bitnami comprehends PHP, MySQL and Apache, and bundles many functions among which CURL, PEAR, SQLite and the Symfony framework.


WAMP Stack

php.net

Bitnami WAMP Stack provides a complete PHP, MySQL and Apache development environment for Windows that can be launched in one click. It also bundles phpMyAdmin, SQLite, Varnish, ImageMagick, ModSecurity, XDebug, Xcache, OAuth, Memcache, FastCGI, APC, GD, OpenSSL, CURL, openLDAP, PEAR, PECL and other components and the following frameworks: Zend Framework, Symfony, CodeIgniter, CakePHP, Smarty, Laravel.

Download installers.

INSTALLER
WAMP Stack v5.4.28 for Windows

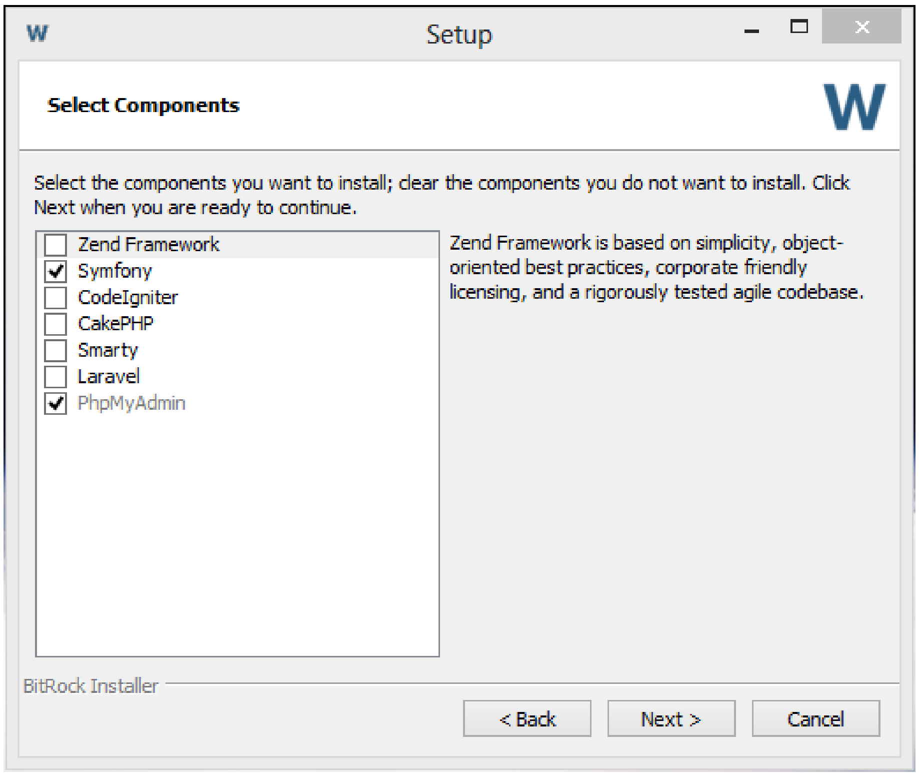
or View all operating system and download options.

Figure 6-1 Installer download page for WAMP Stack (from http://bitnami.com/stack/wamp 2013)

The convenience of using a WAMP is the opportunity of a fast and easy installation of the useful components for the server-side web programming.

- Download and install WAMPStack

To install the WAMP Bitnami application, we have to download the WAMP Stack[45] (see Figure 6-1) for Windows at the website bitnami.com[46]. We installed the version 5.4.23, selecting the Symfony framework (see Figure 6-2) and setting the password for the database, as in Figure 6-3. It can take several minutes.



Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

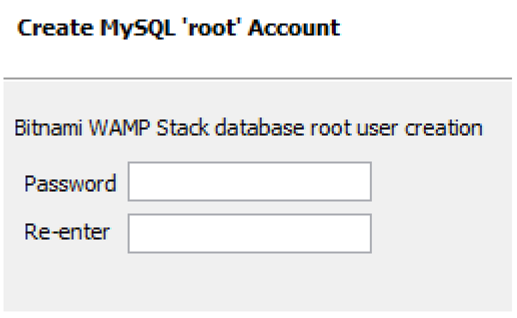
- [ ] Zend Framework
- [x] Symfony
- [ ] CodeIgniter
- [ ] CakePHP
- [ ] Smarty
- [ ] Laravel
- [x] PhpMyAdmin

Zend Framework is based on simplicity, object-oriented best practices, corporate friendly licensing, and a rigorously tested agile codebase.

BitRock Installer

Figure 6-2 Component installing selection

[45] For us bitnami-wampstack-5.4.23-0-windows-installer.exe 92,5 MB
[46] http://bitnami.com/stack/wamp


Create MySQL 'root' Account

Bitnami WAMP Stack database root user creation

Password	
Re-enter	

Figure 6-3 Panel to set database password.

In a different section of the same website,[47] the Stack for LAMP, for Linux, is available.

We installed:

- PHP Version 5.4.23
- HTTP Server, Apache 2.4 Handler Apache Lounge
- MySQL database server
- phpMyAdmin, web application management tool for MySQL database
- Symfony2 framework

- Check the Bitnami installation

When the installation finishes, we can launch Bitnami WAMP Stack: it automatically opens a browser window redirected to the localhost[48] page, which suggests to check the installation[49].

We firstly have to start the servers; the Stacks include a graphical tool to manage the servers easily. You can find the "manager-windows.exe", "manager-osx" or "manager-Linux" tool in your installation directory. Using this tool, you can Start, Stop or Restart the servers and check the log files.

[47] http://bitnami.com/stack/lamp
[48] Localhost is hostname that allows to access the computer's own network services via its loopback network interface. You can reach it at the URL localhost or http://127.0.0.1/ (http://127.0.0.1:100/ if running in port 100)
[49] http://wiki.bitnami.com/Infrastructure_Stacks/BitNami_AMP_Stacks

To check the Bitnami installation you should copy the *installdir/docs/phpinfo.php* file into the *installdir/apache2/htdocs* folder and go to the browser to check the enabled PHP modules by accessing *http://localhost/phpinfo.php*[50].

Installdir is obviously the directory of installation of the wampstack (*C:\BitNami\wampstack-5.4.23-0*, in our case).

Now PHP is installed and operative, we can also check the installation running a file *.php* with the command *phpinfo* (exactly the content of the file *phpinfo.php*).

The operations are well explained in the web site *wiki.bitnami.com.*

We still need some steps, before we can properly use and manage Symfony.

- PEAR installation and upgrade

We have to install PEAR, by manual installation or simply by command windows running the command

```
$ pear install PEAR-1.9.4
```

To update:

```
$ pear upgrade pear
```

The complete procedure can be founded in the *pear.php.net* web site[51].

If you encounter problems, you can install it also by Pyrus[52], with the command

```
php pyrus.phar install pear/PEAR-1.9.4
```

[50] Or http://localhost:100/phpinfo.php if running in port 100.
[51] http://pear.php.net/package/pear/download
[52] http://pear2.php.net/ You first have to download Pyrus,

Another method is requesting *http://pear.php.net/go-pear.phar* in your browser and save the output to a local file *go-pear.phar*. You can then run

```
php go-pear.phar
```

PEAR should be installed and upgraded in your server machine; you can check[53] it simply running *pear* in the installation folder, and a list of all commands appears. Remember to restart the web server every time you change the file *php.ini*.

- PHPUnit Installation

We need to run the Symfony2 test suite to check that everything is working properly. To run the Symfony2 test suite, we first have to install PHPUnit 3.6.4 or later:

```
$ pear config-set auto_discover 1
$ pear install pear.phpunit.de/PHPUnit
```

The system download the files, and installs it. It confirms the installation with

```
install ok: channel://pear.phpunit.de/PHPUnit-4.0.17
```

More information can be founded in the phpunit.de web site[54]

- Installation of the composer

We now have to install the composer. We can install it running the command:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

You can also install the PHAR manually[55] or download and running Composer-Setup.exe[56]. The installation with executable file, asks the location of *Composer-Setup.exe*, which is *in installdir\wampstack-5.4.23-0\php*.

[53] http://pear.php.net/manual/en/installation.checking.php
[54] http://phpunit.de/manual/current/en/installation.html
[55] http://getcomposer.org/download/
[56] https://getcomposer.org/Composer-Setup.exe

More information can be founded in the web-site *getcomposer.org*. We might be asked to restart the computer.

- Vendors' installation.

We already installed composer, then we only have to install vendors by the following command:

```
$ php composer.phar --dev install
```

After installation, we updated the vendors to their latest version with the follow command:

```
$ php composer.phar --dev update
```

These latter two-steps are well exposed in the Symfony webpage.[57]

- Check Symfony installation

We can check Symfony installation[58]. The Symfony framework is installed in the "frameworks" folder in the installation directory. To start a project with Symfony, it is necessary to start the Bitnami Console[59-60].

You can check first the requirements:

```
$ cd installdir/frameworks/symfony/app
$ php check.php
```

A simple way to start learning Symfony is via the Quick Tour accessible via web. To enable it, you should uncomment the following line that you can find in

[57] http://symfony.com/doc/master/contributing/code/tests.HTML
[58] http://wiki.bitnami.com/Components/php_Frameworks/Symfony
[59] Bitnami console is a script to load the Stack environment. This console is useful to run any command included in the Stack: mySQL, PHP, OpenSSL, Ruby, and rake among others. On Windows there is a shortcut in *Start -> BitNami Application Stack -> "Application console"* or *"Use Application Stack"*
[60] http://wiki.bitnami.com/Components/Bitnami_console

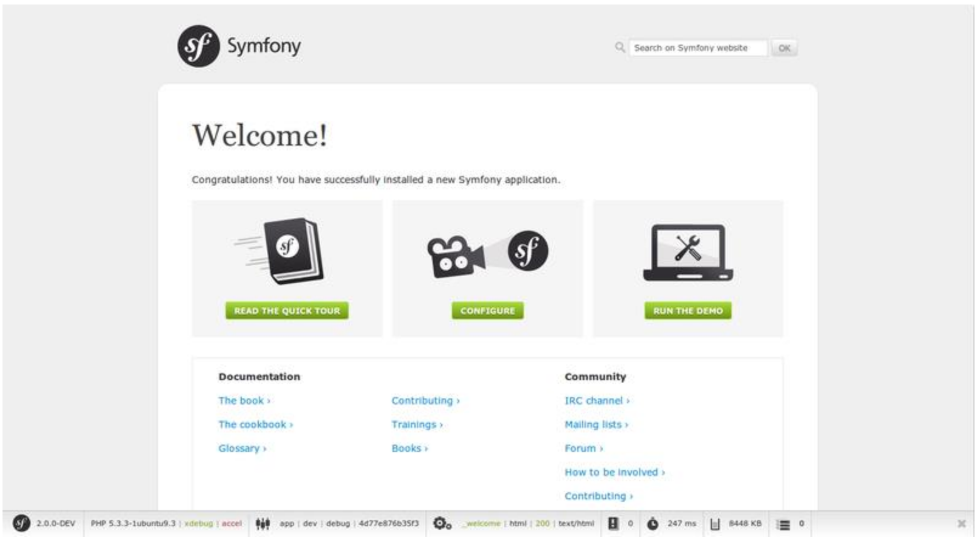
the Apache configuration file *installdir/apache2/conf/bitnami/bitnami-apps-prefix.conf:*

```
Include "installdir/frameworks/symfony/conf/httpd-prefix.conf"
```

And restart the Apache server.

After that, we are finally capable to request our first "real" Symfony2 webpage: point your browser to *http://127.0.0.1/symfony/app_dev.php*[61]. This page is only accessible from the local machine.

Symfony2 welcomes us and congratulate for our hard work so far (see Figure 6-4)!



Congratulations! You have successfully installed a new Symfony application.

Figure 6-4 Symfony localhost.

The Symfony webpage[62] in the local machine explains and deepens this procedure, and provides a first Quick Tour to Symfony.

You can see some examples during the Quick Tour and you can find more information at symfony.com[63].

- Optional

[61] Or http://localhost/symfony/app_dev.php
[62] http://symfony.com/doc/current/quick_tour/the_big_picture.html
[63] http://symfony.com/doc/current/

The project now is ready to be developed, but we might install also a PHP accelerator[64], Packagist[65] and Doctrine[66].

6.2 Installation of the Software

With PHP and Symfony2 working, we can install the web application.

Before installing the software, we suggest to practise with Symfony2 with the Quick Tour and the *DemoBundle*: the installation of the software erases *DemoBundle* and Quick Tour, and prevent from reaching it again.

6.2.1 Installation of the Bundles

To install the bundles you need to copy and paste the entire *Acme* folder into the *src* folder[67] of Symfony, replacing the default folder.

The file *AppKernel.php* in the folder *symfony/app* is the kernel of the application and contains an array with the bundles of the application.

We must add some lines to the *$bundles array*, in the function *registerBundles();*

```
new Acme\MainBundle\AcmeMainBundle(),
new Acme\ManagementBundle\AcmeManagementBundle(),
new Acme\MissionBundle\AcmeMissionBundle(),
new Acme\GroundStationBundle\AcmeGroundStationBundle(),
```

We also must erase the line relative to the demo bundle, a standard bundle of Symfony created to help new users to check the operating principles.

Actually, the modification of the *AppKernel.php* file can be avoided simply importing the relative file available, as explained in the *Settings, Routing and*

[64] Internet offers a wide choice.
[65] https://packagist.org
[66] http://www.doctrine-project.org/projects/dbal.html
[67] C:\Bitnami\wampstack-5.4.23-0\frameworks\symfony\src

Configuration files section of this chapter. This procedure will avoid the same operation with the vendors.

6.2.2 Installation of the Vendors

The Symfony framework installs most of the vendors by default. Online we can also find open source vendors, useful for our aim. We used *FOSUserBundle* and *PUGXMultiUserBundle*.

Chapter 5.3 and 5.5 contain a brief explanation for the installation of these bundles. Thanks to the already compiled files, you can speed the installation. You just should add to the file *composer.json*[68] the following lines:

```
{
    "require": {
        "friendsofsymfony/user-bundle": "2.0.*@dev",
        "pugx/multi-user-bundle": "3.0.*@dev"
    }
}
```

And run in successions the two commands:

```
php composer.phar update friendsofsymfony/user-bundle
```

And

```
php composer.phar update pugx/multi-user-bundle
```

As described in the previous section, the modification of the *AppKernel.php* file, can be avoided importing the relative file already compiled, availed in the installation folder.

Importing also the files *config.yml*, *parameters.yml*, *routing.yml* and *security.yml* as explained in the next section of this chapter, we can skip the passages described in the installation regarding the modification of these files.

[68] It is stored in the Symfony folder.

Symfony could ask us to erase some lines from the file *routing_dev.yml* in the *app/config* folder: you should delete the lines referred to the *demoBundle*, not installed anymore.

6.3 Settings, Routing and Configuration files

The software comprehends the four bundles, the vendors and the configuration files.

The configuration files not allocated in the bundles are in the *app* folder of Symfony. We have to import and replace the files:

- Symfony/app/AppKernel.php
- Symfony/app/config/config.yml
- Symfony/app/config/parameters.yml
- Symfony/app/config/routing.yml
- Symfony/app/config/security.yml

After that, the application should be active and fully functional.

6.4 Customization of the settings

The following chapters explain some modifications available to the user. We focused about the most interesting issues: every other change can be easily done with the knowledge of PHP and Symfony2.

6.4.1 Setting the time zone in Synfony2

Our project is intended to work in every part of the world. In order to synchronize all users, we must define a common time zone.

To set the time zone is possible to set an YML directive and make it available to the app, in order to be able to use it whenever needed.

In */app/config/parameters.yml* we add the line:

```
    default_timezone: "Europe/Paris"
```

And in the controller, when needed, the lines:

```
$tz = $this->container->getParameter('default_timezone');
$now = new \DateTime();
$now->setTimezone(new \DateTimeZone($tz))
```

Another way to set the time zone could be, since the time zone is not really a dynamic part of your application, to stick it inside *.htaccess* file. It is enough to add the following line:

```
PHP_value date.timezone "Europe/London"
```

For the whole list of the functions and the codes, refer to http://www.php.net/[69-70].

6.4.2 Setting the name and folder of the stored file

In the Product entity[71] it is possible to change the name of the stored file and the folder in which the file is stored.

We set it in the following way:

```
$filename = "{$mission}_{$user}_{$newDate}";
```

This name allows identifying immediately the origin of every file, and avoiding misunderstandings. Indeed the filename contains the name of the associated mission, the user that uploaded that mission, and the date and time of the upload.

[69] http://www.php.net/manual/en/book.datetime.php
[70] http://www.php.net/manual/en/timezones.php
[71] C:\BitNami\wampstack-5.4.23-0\frameworks\symfony\src\Acme\GroundStationBundle\Entity\Product.php

The *$newDate* object contains the year, the month, the minute and the hour of the upload.

In order to change the name, add more information in order to simplify the doctrine query or the organization of the folder, it is enough to modify the *public function preUpload()* in the above-mentioned entity.

In the same entity, there is also the function

```
    protected function getUploadRootDir()
    {
        return __DIR__.'/../../../../web/'.$this-
>getUploadDir();
    }
```

We set this to go in the web folder of Symfony[72], where public files are. Actually, it may be better to change the folder and put restrictions; this can be easily done modifying the function.

6.4.3 Upload file validation

Presently, the uploaded file can be everything, inasmuch it does not have a validation process.

The validation can be done with annotation or with a *validation.yml* file in the *Resources\config* folder of the bundle.

The file, in the *GroundStationBundle*, is the following:

```
Acme\GroundStationBundle\Entity\Product:
    properties:
        file:
            - File:
                maxSize: 6000000
```

[72] C:\BitNami\wampstack-5.4.23-0\frameworks\symfony\web

The only limit we set is the maximum size. Of course, for security problems, we must set a stronger validation system. It can be done following the instructions in the Symfony2 web site[73-74].

6.4.4 Customization of the User Entities

In order to customize the users, the files of the entities must be modified.

If the change includes both of the roles, it must be done in the *User.php* file: otherwise, the entities *UserGroundStation* and *UserOperator* in the Entity folder of Management bundle allow editing a characteristic for a single role.

With the entity, we must modify also the forms:

- RegistrationUserOperatorFormType.php
- RegistrationUserGroundStationFormType.php
- ProfileUserOperatorFormType.php
- ProfileUserGroundStationFormType.php

The first two files modify the registration forms, the latter two modify the forms to show or edit the users, respectively for Ground Station Operators and Mission Operators.

These files are in the *Form/Type* directory of the *ManagementBundle*.

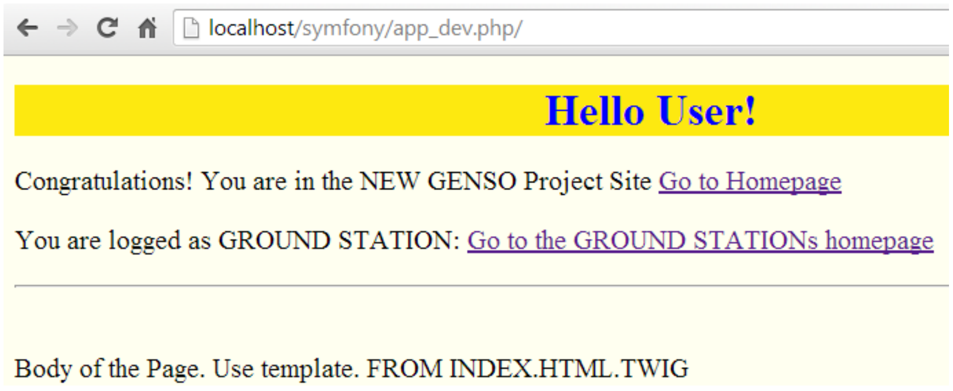
We suggest not to modify directly the file in the *FosUserBundle*, but to override it. To keep a better logic, these files can be moved also to the *MainBundle*, respecting the idea of overriding everything in the same bundle: the controller calls the form, so it can be done changing the destination in the controllers that call the forms, like the *RegistrationController* of the *ManagementBundle*.

[73]http://symfony.com/doc/current/book/validation.html
[74] http://symfony.com/doc/current/reference/constraints/File.html

6.5 Customization of the templates

The project uses the template engine *Twig*[75]: indeed Symfony2 comes with a bundled support for Twig as its default template engine.

All the web pages well show the templates used to display the data (see Figure 6-5). It makes plain to realize which file requires to be modified to have a different layout of the page or add contents. The code for this display is shown in Figure 6-6.


localhost/symfony/app_dev.php/

Hello User!

Congratulations! You are in the NEW GENSO Project Site Go to Homepage

You are logged as GROUND STATION: Go to the GROUND STATIONs homepage

Body of the Page. Use template. FROM INDEX.HTML.TWIG

Figure 6-5 How the template used is shown in every page.

The templates are in the folder *Acme\NameBundle\Resources\views\*, where *NameBundle* is the name of the Bundle that contains the template. The views folder is partitioned, in order to improve the organization.

```
{% block body %}
   <div>
      <p>
         Body of the Page. Use template.
         FROM AcmeManagementBundle:Mission:create.html.twig
      </p>
   </div>
{% endblock %}
```

Figure 6-6 Example of field to modify in order to customize the template.

To modify a template is sufficient the knowledge of HTML and basic programming.

[75] http://twig.sensiolabs.org/

The name of the templates looks like *nameTemplate.html.twig*, in order to be read by the twig engine and preserve the extension of the file (HTML): for a CSS file, the name is *nameTemplate.css.twig*; for a JavaScript, *nameTemplate.js.twig* and so on.

6.5.1 Logic of the templates

The routing of the templates has the same logic as Symfony:

AcmeMainBundle:Index:layout.html.twig locate the template *layout.html.twig* in the *MainBundle* of the Acme Project, in the *\Resources\views\Index* folder.

As for the classes, also the templates support inheritance. To call a parent template add, at the beginning of the file, the following line:

```
{% extends 'AcmeMainBundle:Index:layout.HTML.twig' %}
```

included into the delimiters that enclose the programming sections.

The principal template contains the opening and closing tag, and it is internally divided into different sections (called blocks), whose names explain the function:

- Stylesheets
- Title
- Header
- Body
- Content
- Footer
- JavaScript

Obviously, the block Stylesheets recalls the template with the CSS, the title sets the title of the page and so on. More blocks can be added.

The layout template is filled only with tags and the inheritance logic, leaving the contents to the specific templates.

For a better understanding of the template system, refer to the documentation[76].

6.6 Maintenance

The maintenance of the project consists in continuously updating the logic and managing the connected users and the uploaded objects.

In order to manage the web site, we need Admin users, which registration cannot follow the normal rules explicated for the clients, as they need particular privileges.

6.6.1 Create an Admin User

The official documentation of *FosUserBundle*[77] explains how to create a new user, and promote the role as Admin.

It consist in the use of command lines, and allows to

- Create a user
- Activate a user
- Deactivate a user
- Promote a user
- Demote a user
- Change a user’s password

The command

```
php app/console fos:user:create
```

[76] http://symfony.com/doc/current/book/templating.html
[77] https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/command_line_tools.md

creates an user, asking for username, mail and password. In the project, the Admin User extends the Ground Station User, which has latitude and longitude as mandatory proprieties: during the creation of a new user, the system recognizes the lack of the mandatory properties and prints the error:

```
SQLSTATE[23000]: Integrity constraint violation: 1048 Column
'latitude' cannot be null
```

We solve the problem changing the command. We properly override the cli command of *FosUserBundle* placed into *Command/CreateUserCommand.php* and the *user create* method of *FosUserBundle* placed into *Util*/*UserManupulator.php*. The *MainBundle* does it, in the respective folders.

Now the same command asks for the following properties:

```
C:\BitNami\wampstack-5.4.23-0\frameworks\symfony> php
app/console fos:user:create [--super-admin] [--inactive]
username email password latitude longitude
```

The Admin user can be created as a standard user, then promoted as Admin later.

We can create the user with the command:

```
php app/console fos:user:create testuser email@uvigo.es pswd1
000 000
```

Then promote the user with the command:

```
php app/console fos:user:promote username ROLE_ADMIN
```

7

CONCLUSIONS AND FUTURE DEVELOPMENT

The aim of this thesis was the development of a web-application conceived as the archive system of the SATNET project.

The result has been the realization of the structure of a website that will support the SATNET group work in the development of the platforms.

The software is not completed; it need a full revision and a development.

7.1 Future Development

For the future, we recommend some important developments.

- The public part of the web site needs the composition of the contents. We set the “contact us” page, the “information” page and some others, but now they are almost blank, with no information.

The header and the footer need an improvement, and a lateral navigation bar can help surfing the web site.
The structure and formatting of the website now is plain and simple. The list of missions or users are not formatted in a proper way, and this can be improved with a quick study of HTML applied to the appropriate templates.
Some JavaScript code can help surfing the web site and avoiding errors, for example asking confirmation in the form submissions.

- Almost every page has its dedicated template, even though the contents shown have similar formatting. The similar templates have the same name and are stored in the same folder of different bundles. This structure helps the new developer to understand how to modifications, but should be improved, avoiding the repetition of code.

- As for the templates, the controllers with the same name in different bundles have similar Actions, and their relation can be improved, avoiding repetitions of code. This task is more critical than with the controller, because the differences between the Actions determine the privileges of the user and a wrong implementation can lead to a lack of security.

- In order to endorse the clearness, we sacrificed the shortness of the routes. The system can be improved, properly organizing the routing structure.

- The Bundle system follows the routing system: there is a Bundle for every section of the web site. We choose it as the best structure to start, but many different choice can be developed: for example, we could divide the bundles into a *UserBundle*, a *ProductBundle* and a *MissionBundle*. The first one would extend the *FOSUserBundle* and contain the Users Entities, the second one for the download and upload of the Product Entities, and the last one for the management of Mission Entities. This choice would help the templating system, but would need more work on the routing system. The new developer can chose a new structure, also following the suggestion in the official documentation[78], in order to improve the maintainability and efficiency.

- The entities are as simple as possible: they can be changed and updated according to the needs in the applications. For example, in the User Entity, a "reputation number property", and a consequently classification, can drive the users to upload more strings, in order to increase their ranking.

- Some pages are public, but may be put behind a firewall, and vice versa. The routing system allows changing it in a easy way. We refer particularly to the pages that shows the list of users and the list of missions.

- Now the error pages are the standard pages of Symfony: an appropriate setting of the layout and the content will improve the navigation of the page.

[78] http://symfony.com/doc/master/cookbook/bundles/best_practices.html

- An appropriate testing, and the consequent upkeep, can be done only with the continued use of the software.

7.2 Conclusions

This thesis is realized as a guidebook for the use, modification, and improvement of the web application.

The website has only essential functionalities. We expect a future improvement in the number and quality of these functionalities, thanks to the growth of the project and the number of stakeholders. The contents of the website are only sketched; indeed the aim of the thesis was the development of the structure, while the contents can be easily modified and integrated.

Considering this future integration, we conceived the entire project as easily adjustable as possible, with clear reference between the webpages and the associated codes, and a template view that a user without acknowledge in PHP can quickly modify.

➢Appendix

USEFUL CMD COMMANDS

This appendix provides a quick reference to the commonly used cmd commands. Every command is followed by a brief explication: for a deeper analysis, we suggest the reading of the official documentation[79].

- How to use the cmd commands.

All the commands should be used in the WAMP Stack Environment. It can be reached from the file *use_wampstack.bat* in the *C:\BitNami\wampstack-5.4.23-0* folder. Once opened the new window, you should submit all the command in the */symfony* folder. You can reach it by the following command:

```
cd frameworks/symfony
```

- Generate a bundle.

The *SensioGeneratorBundle*, installed by default with Symfony2, extends the default Symfony2 command line interface by providing new interactive and

[79] http://symfony.com/doc/current/components/console/introduction.html

intuitive commands for generating code skeletons like bundles, form classes or CRUD controllers based on a Doctrine 2 schema.

The command to generate a bundle is:

```
php app/console generate:bundle
```

The command asks questions to determine the bundle name, location, configuration format and default structure.

After the options submission the command creates the bundle and the relative namespace, and modifies the files *app/AppKernel.php* and *app/config/routing.yml* in order to include the new bundle.

- Visualize all the routes.

A very useful command for the debugging and maintenance is the following:

```
php app/console router:debug
```

This command shows on the screen a list of all the routes, with the associated method, scheme, host and the relative path.

- Clear the cache.

After modifying the project, especially after a configuration modification, the clear of the cache is mandatory for the proper visualization of the new project. The command is:

```
php app/console cache:clear
```

By default, console commands run in the dev environment, so the command will clear and warm the cache for the specified environment only. To clear and warm the prod cache you need to run:

```
php app/console cache:clear --env=prod
```

We suggest also running the command in “no-debug” mode, avoiding the performance hit of collecting debug data, with the command:

```
php app/console cache:clear --env=prod --no-debug
```

- Open the shell.

If you need to run several commands, you can avoid specifying *php app/console* each time, with an interactive shell provided by Symfony. To enter the shell run:

```
php app/console –shell
```

- View Doctrine command list.

The Doctrine2 ORM integration offers several console commands under the doctrine namespace. To view the command list you can use the list command:

```
php app/console list doctrine
```

All Doctrine commands are well deepen in the *Chapter 8, Databases and Doctrine*, of the Symfony2 book [80].

- Create and drop the database.

To create a new database, you should run the command:

```
php app/console doctrine:database:create
```

If the command does not work, can appear the following message:

```
SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost'
                  (using password: YES)
```

You should open the *parameter.yml* file, in the *app/config* folder, and modify the line with the password database, writing the password you set during the Bitnami Stack installation.

80 http://symfony.com/doc/current/book/doctrine.html

```
    database_password: password_set
```

To drop the database, erasing all the data stored, run:

```
php app/console doctrine:database:drop --force
```

- Create the getters and setters with doctrine.

Even though Doctrine knows how to persist a Product object to the database, you need to create getter and setter methods. You can do it with the command:

```
php app/console doctrine:generate:entities
Acme/StoreBundle/Entity/Product
```

The command automatically creates all the methods for the Product class. It is a safe command: it does not replace existing methods, so it can be run any times without any risk of code corruption. The command creates simple setters and getters, and should be adjusted to your own needs

The same command also generate all known entities of a bundle or an entire namespace:

```
php app/console doctrine:generate:entities AcmeStoreBundle
php app/console doctrine:generate:entities Acme
```

- Creating the Database Tables/Schema.

Doctrine automatically create all the database tables needed for every known entity in your application just running the command:

```
php app/console doctrine:schema:update –force
```

It is a very powerful command: indeed update the database according to the mapping information of the entities and generates the relative SQL statements.

- Visualize all the entities.

As for the routes, there is a command that display all the entities that Doctrine is aware of and whether or not there are any basic errors with the mapping. The command is:

```
php app/console doctrine:mapping:info
```

- Execute SQL queries.

The Console component also allows executing SQL queries directly from the command line with the command:

```
php app/console doctrine:query:sql
```

Chapter 22, How to create a Console Command, of the Symfony cookbook [81] explain how to create new Console commands.

[81] http://symfony.com/doc/master/cookbook/console/console_command.html

Bibliography and Webography

[1] Francois Zaninotto; Fabien Potencier . 2007. *The definitive guide to Symfony.* Berkeley: Apress.

[2]Holzner, Steven. 2006. *Spring into PHP.* Prentice Hall Ptr.

[3]2013. *http://bitnami.com/stack/wamp.* http://bitnami.com/stack/wamp.

[4]2013. *http://www.genso.org/.* December. http://www.genso.org/.

[5]Ullman, Larry. 2013. *PHP Advanced and Object-Oriented Programmin.* Berkeley: Peachpit Press: Visual Quickstart Guides.

[6]—. 2012. *PHP and MySQL for Dynamic Web Sites.* Berkeley: Peachpit Press: Visual Quickstart Guides.

[7]—. 2011. *PHP for the Web.* 4th Edition. Berkeley: Peachpit Press: Visual Quickstart Guides.

[8] http://symfony.com/

http://symfony.com/doc/current/cookbook/index.html

http://symfony.com/doc/current/book/index.html

http://symfony.com/doc/current/components/index.html

[9] http://wiki.bitnami.com/

[10] http://www.w3.org/

[11] http://getcomposer.org/

[12] http://pear.php.net/

[13] http://pear2.php.net/

[14] https://github.com/

[15] http://www.wampserver.com/en/

[16] http://www.apache.org/

[17] http://www.php.net/

[18] http://www.mysql.com/

[19] http://www.apress.com/

[20] http://www.doctrine-project.org

[21] http://php.about.com/

[22] http://docs.oracle.com/

[23] http://propelorm.org/

[24] http://www.phpframeworks.com/

[25] http://www.levinecentral.com

[26] http://www.celestrak.com

[27] http://www.stoff.pl/

[28] http://www.esa.int/Education

[29] http://www.humsat.org

[30] http://www.klofas.com

[31] http://vbn.aau.dk/ws/files/13411830/report.pdf

[32] http://www.orm.net/

[33] http://www.phonesat.org/

[34] http://www.nasa.gov/

[35] http://www.codeproject.com

[36] http://fabien.potencier.org

[37] http://henrik.bjrnskov.dk

[38] http://en.wikipedia.org/

[39] https://packagist.org

[40] http://twig.sensiolabs.org/

[41] http://www.phptherightway.com/

Useful communities:

[41] http://stackoverflow.com/

[42] https://moodle.org

[43] http://forums.phpfreaks.com/

[44] http://www.codingforums.com/

[45] http://forum.symfony-project.org/