Titulo: e-CRF Manager

Volumen: 1  
Alumno: MARCELLO ORRICO

Director/Ponente: JOSÉ ANTONIO GONZÁLEZ ALAISTRUE  
Departamento: Estadística.  
Fecha: lunes 7 de mayo de 2012
### Datos del proyecto

<table>
<thead>
<tr>
<th>Estudiante</th>
<th>MARCELLO ORRICO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="mailto:marcello.orrico@est.fib.upc.edu">marcello.orrico@est.fib.upc.edu</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Director/a</th>
<th>JOSÉ ANTONIO GONZÁLEZ ALAÍSTRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="mailto:jose.a.gonzalez@upc.edu">jose.a.gonzalez@upc.edu</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tribunal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Presidente/a</td>
<td>ERIK COBO VALERI</td>
</tr>
<tr>
<td>Vocal</td>
<td>ALBERT OLIVERAS LLUNELL</td>
</tr>
<tr>
<td>Secretario/a</td>
<td>JOSÉ ANTONIO GONZÁLEZ ALAÍSTRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fecha de inscripción</th>
<th>lunes 7 de mayo de 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estado</td>
<td>Matrículado</td>
</tr>
<tr>
<td>Titulación</td>
<td>Ingeniería Técnica en Informática de Gestión</td>
</tr>
<tr>
<td>Créditos</td>
<td>22.5</td>
</tr>
<tr>
<td>Fecha de lectura</td>
<td>jueves 5 de julio de 2012 - 16.00</td>
</tr>
<tr>
<td>Lugar de lectura</td>
<td>Sala de Reuniones (Edificio C5), Campus Nord</td>
</tr>
</tbody>
</table>
INDEX

1. INTRODUCTION .................................................................................................................. 7
2. OBJECTIVES ......................................................................................................................... 8
3. REQUIREMENTS ANALYSIS ............................................................................................... 9
   3.1 FUNCTIONAL REQUIREMENTS ......................................................................................... 9
      3.1.1 EXCEL sheet definition (Define.xls) ........................................................................ 9
      3.1.2 SYSTEM FEATURES ............................................................................................... 10
      3.1.3 ADMIN FEATURES ................................................................................................ 10
      3.1.4 INVESTIGATOR FEATURES .................................................................................... 10
   3.2 NON-FUNCTIONAL REQUIREMENTS ............................................................................... 11
      3.2.1 SECURITY ............................................................................................................... 11
      3.2.2 SCALABILITY ........................................................................................................ 11
      3.2.3 EFFICIENCY .......................................................................................................... 11
      3.2.4 PORTABILITY ......................................................................................................... 11
4. PROPOSED SOLUTION ......................................................................................................... 12
   4.1 DEVELOPMENT .............................................................................................................. 12
      4.1.1 SYSTEM ARCHITECTURE AND PLATFORM .......................................................... 12
      4.1.2 INITIAL CONSIDERATIONS ................................................................................... 12
      4.1.3 PROGRAMMING LANGUAGES AND LIBRARIES ................................................. 13
      4.1.4 DATABASE SERVER MySQL .................................................................................. 15
      4.1.5 FRAMEWORK PHP CAKEPHP ............................................................................. 16
   4.2 HARDWARE AND OPERATIVE SYSTEM ARCHITECTURES .......................................... 17
   4.3 SOFTWARE ARCHITECTURE .......................................................................................... 18
   4.4 USERS INTERFACE ........................................................................................................ 20
   4.5 CONCEPTUAL MODEL .................................................................................................... 21
      4.5.1 MODEL .................................................................................................................. 21
      4.5.2 CONTROLLER ........................................................................................................ 22
      4.5.3 VIEW ...................................................................................................................... 22
      4.5.4 DATABASE ............................................................................................................ 22
   4.6 SYSTEM MODEL ............................................................................................................. 23
      4.6.1 USE CASE MODEL ............................................................................................... 23

INDEX

IMAGES INDEX .......................................................................................................................... 6

1. INTRODUCTION .................................................................................................................. 7
2. OBJECTIVES ......................................................................................................................... 8
3. REQUIREMENTS ANALYSIS ............................................................................................... 9
   3.1 FUNCTIONAL REQUIREMENTS ......................................................................................... 9
      3.1.1 EXCEL sheet definition (Define.xls) ....................................................................... 9
      3.1.2 SYSTEM FEATURES .............................................................................................. 10
      3.1.3 ADMIN FEATURES ............................................................................................... 10
      3.1.4 INVESTIGATOR FEATURES .................................................................................... 10
   3.2 NON-FUNCTIONAL REQUIREMENTS ............................................................................... 11
      3.2.1 SECURITY ............................................................................................................... 11
      3.2.2 SCALABILITY ........................................................................................................ 11
      3.2.3 EFFICIENCY .......................................................................................................... 11
      3.2.4 PORTABILITY ......................................................................................................... 11
4. PROPOSED SOLUTION ......................................................................................................... 12
   4.1 DEVELOPMENT .............................................................................................................. 12
      4.1.1 SYSTEM ARCHITECTURE AND PLATFORM .......................................................... 12
      4.1.2 INITIAL CONSIDERATIONS ................................................................................... 12
      4.1.3 PROGRAMMING LANGUAGES AND LIBRARIES ................................................. 13
      4.1.4 DATABASE SERVER MySQL .................................................................................. 15
      4.1.5 FRAMEWORK PHP CAKEPHP ............................................................................. 16
   4.2 HARDWARE AND OPERATIVE SYSTEM ARCHITECTURES .......................................... 17
   4.3 SOFTWARE ARCHITECTURE .......................................................................................... 18
   4.4 USERS INTERFACE ........................................................................................................ 20
   4.5 CONCEPTUAL MODEL .................................................................................................... 21
      4.5.1 MODEL .................................................................................................................. 21
      4.5.2 CONTROLLER ........................................................................................................ 22
      4.5.3 VIEW ...................................................................................................................... 22
      4.5.4 DATABASE ............................................................................................................ 22
   4.6 SYSTEM MODEL ............................................................................................................. 23
      4.6.1 USE CASE MODEL ............................................................................................... 23
# IMAGES INDEX

<table>
<thead>
<tr>
<th>Image</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 1</td>
<td>CRF Example</td>
<td>7</td>
</tr>
<tr>
<td>Image 2</td>
<td>CakePHP request</td>
<td>19</td>
</tr>
<tr>
<td>Image 3</td>
<td>Use Case Model</td>
<td>23</td>
</tr>
<tr>
<td>Image 4</td>
<td>Class Diagram: Models</td>
<td>29</td>
</tr>
<tr>
<td>Image 5</td>
<td>Class Diagram: Controllers</td>
<td>30</td>
</tr>
<tr>
<td>Image 6</td>
<td>Sequence Diagram: Login</td>
<td>31</td>
</tr>
<tr>
<td>Image 7</td>
<td>Sequence Diagram: Logout</td>
<td>31</td>
</tr>
<tr>
<td>Image 8</td>
<td>Sequence Diagram: Create User</td>
<td>32</td>
</tr>
<tr>
<td>Image 9</td>
<td>Sequence Diagram: Users list</td>
<td>32</td>
</tr>
<tr>
<td>Image 10</td>
<td>Sequence Diagram: Edit User</td>
<td>33</td>
</tr>
<tr>
<td>Image 11</td>
<td>Sequence Diagram: Delete User</td>
<td>33</td>
</tr>
<tr>
<td>Image 12</td>
<td>Sequence Diagram: Create study</td>
<td>34</td>
</tr>
<tr>
<td>Image 13</td>
<td>Sequence Diagram: Studies List</td>
<td>34</td>
</tr>
<tr>
<td>Image 14</td>
<td>Sequence Diagram: Edit Study</td>
<td>35</td>
</tr>
<tr>
<td>Image 15</td>
<td>Sequence Diagram: Delete Study</td>
<td>35</td>
</tr>
<tr>
<td>Image 16</td>
<td>Sequence Diagram: Create Patient</td>
<td>36</td>
</tr>
<tr>
<td>Image 17</td>
<td>Sequence Diagram: Patients List</td>
<td>36</td>
</tr>
<tr>
<td>Image 18</td>
<td>Database Model</td>
<td>38</td>
</tr>
<tr>
<td>Image 19</td>
<td>Excel Definition: Line</td>
<td>47</td>
</tr>
<tr>
<td>Image 20</td>
<td>Excel Definition: Table</td>
<td>48</td>
</tr>
<tr>
<td>Image 21</td>
<td>Excel Definition: Seq</td>
<td>48</td>
</tr>
<tr>
<td>Image 22</td>
<td>Excel Definition: Text</td>
<td>49</td>
</tr>
<tr>
<td>Image 23</td>
<td>Excel Definition: Type &amp; Format</td>
<td>50</td>
</tr>
<tr>
<td>Image 24</td>
<td>Screen Mockup: Login</td>
<td>53</td>
</tr>
<tr>
<td>Image 25</td>
<td>Screen Mockup: Home Admin</td>
<td>53</td>
</tr>
<tr>
<td>Image 26</td>
<td>Screen Mockup: Home Investigator</td>
<td>54</td>
</tr>
<tr>
<td>Image 27</td>
<td>Screen Mockup: Home CRF</td>
<td>54</td>
</tr>
<tr>
<td>Image 28</td>
<td>Screen Mockup: Home Users</td>
<td>55</td>
</tr>
<tr>
<td>Image 29</td>
<td>Screen Mockup: Create CRF</td>
<td>55</td>
</tr>
<tr>
<td>Image 30</td>
<td>Screen Mockup: Edit CRF</td>
<td>56</td>
</tr>
<tr>
<td>Image 31</td>
<td>Screen Mockup: Define CRF</td>
<td>56</td>
</tr>
<tr>
<td>Image 32</td>
<td>Software cost</td>
<td>57</td>
</tr>
<tr>
<td>Image 33</td>
<td>Hardware cost</td>
<td>57</td>
</tr>
<tr>
<td>Image 34</td>
<td>Staff cost</td>
<td>58</td>
</tr>
<tr>
<td>Image 35</td>
<td>Design cost</td>
<td>58</td>
</tr>
<tr>
<td>Image 36</td>
<td>Integration cost</td>
<td>58</td>
</tr>
<tr>
<td>Image 37</td>
<td>Demo cost</td>
<td>59</td>
</tr>
<tr>
<td>Image 38</td>
<td>Cost per profiles</td>
<td>59</td>
</tr>
<tr>
<td>Image 39</td>
<td>Total cost</td>
<td>59</td>
</tr>
<tr>
<td>Image 40</td>
<td>Total space occupied</td>
<td>59</td>
</tr>
<tr>
<td>Image 41</td>
<td>Improvement 5.3.4</td>
<td>60</td>
</tr>
<tr>
<td>Image 42</td>
<td>Improvement 5.3.5</td>
<td>61</td>
</tr>
</tbody>
</table>

---

Image 41 - Improvement 5.3.4 ................................................................. 60
Image 42 - Improvement 5.3.5 ................................................................. 61
1. INTRODUCTION

The overall project objective is to develop a Web portal that will create a Case Report Form (CRF). A Case Report Form (CRF) is the document that the investigator uses to record all the information regarding a patient during a medical study or clinical trial. Common information to record in a clinical trial is demographic data, previous clinical history, concomitant medication and adverse events. Depending on the objective of the study, the CRF will incorporate specific medical assessments that may be repeated a long time, organized on visits, in order to assess if an intervention produces a change in the health of a patient.

In our application, the CRF can be defined by means of either a wizard or an excel spreadsheet. This document explains how to generate CRFs using the excel spreadsheet, referred as Define.xls.

The information of a CRF is organized in Define.xls on different levels, by pages, sections, structures and variables, each of this nested in the former. Thus, a CRF is composed by one or more pages, each page contains one or more sections, each section contains one or more structures and each structure is composed by one or more variables.

As an example, the following Figure represents a common CRF page. This page is composed by 3 sections: Demographic Data, Smoking Habits and Alcohol Consumption. The section Demographic Data is composed by 5 different structures, where each structure in this case only contains one variable. Other structures can be defined in order to organize more complex data layouts like for instance a table with variables organized by rows and columns.
2. OBJECTIVES

The overall project objective is the creation of a CRF (Case Report Form) in an electronic form. A CRF is a document where researcher can record a series of data of the patients (demographics, medical history, medication and adverse events) during a study or a clinical trial. Each CRF can be repeated over time by scheduling several visits to assess with the data obtained whether an intervention produces an improvement or a worsening of the patient.

1. Create and manage an application in a web environment through a framework that enabling easier development:
   1.1. Create a study from zero using a GUI\(^1\) that allows the realization of the form also to advanced users;
   1.2. Export CRF structure to Excel spreadsheet (Define.xls);
   1.3. Organize and define spreadsheet’s structure.
2. Make an application that is independent of the used system and more portable as possible.
3. Know and comply with the rules and procedures, in all matters relating to security measures implemented in the data processing systems of patients.
4. Develop and deploy the application using free software (open source).

\(^1\) Graphic User Interface
3. REQUIREMENTS ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

The project is based on a set of functional requirements, which allow the system to function properly and according to specifications. We’ll don’t spent a lot of time on the visual aspect of the system, because it isn’t one of our main goals, but we’ll devote much time and effort to the functional part of the application to make it more robust and efficient as possible.

3.1.1 EXCEL sheet definition (Define.xls)

The Define.xls is organized into four spreadsheets:

1. General Info (GI)
2. Sections distribution (SDIS)
3. Sections definition (DEF)
4. Formats (FM)

General Info (GI)
This sheet define the customer data, client code, protocol, name and also include a short name for the study.

Sections distribution (SDIS)
In this sheet will define how to distribute the sections of the CRF on different pages.
- The sections should be identified by an alias.
- The sections may be repeated throughout the study with different page numbers.
- The sections should contain a title.
- Each section must necessarily be associated with a name to visit (numeric or text).
- In each section compulsorily should indicate in what page is presented.

Sections definition (DEF)
In this sheet will define the content of the sections.
- Each section should be defined only once.
- A section should be formed by structures and variables.
- Section Name must contain the same section for variables that form the section.
- It must necessarily define four different structures: line, panel board section.
- The variable names should have the first two letters of the table to which they belong and not exceed 8 characters in length. This field should be mandatory.

For complete information variables must define the columns with the names of table in the database mandatory. Another column of question text to be displayed in the web form on the left side of the field, this field may be optional and another column with the text to display on the right side of the field this field can be optional. Two columns of field type and format.

Formats sheet (FM)
In this sheet should be defined values and labels of the formats.
3.1.2 SYSTEM FEATURES
The system must provide a variety of features depending on the type of user connected. We have distinguished two kinds of users: administrator and investigator. The web system is composed of two elements, the site accessible by investigators and the control site where administrators can access to manage the Web.

3.1.3 ADMIN FEATURES
The administrator can manage users and study informations using the application. The basic features are:

- Manage users:
  - Create new User
  - View Users
  - Edit User
  - Delete User

- Manage studies:
  - Create new Study using Define.xls
  - Create new Study using the wizard
  - View Study
  - Edit Study
  - Delete Study
  - Export Study’s structure to Define.xls

3.1.4 INVESTIGATOR FEATURES
The investigator will have access to the system where can manage patients and their data. The basic functions are:

- Manage Patients:
  - Create New Patient
  - Edit Patient
  - View Patient
  - Delete Patient

- Manage CRF:
  - Preview Study
  - Enter patient data
  - View patient data
  - Edit patient data
3.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements define the qualities or properties that the system needs, like security, scalability, efficiency and portability.

Regarding the legal and licensing we have checked whether there are tools used to require license validation or payment of this. We ensure compliance with all legal aspects of entire software, technologies and libraries used to develop the application.

3.2.1 SECURITY

The system security ensuring that unauthorized persons have access to the application: the identification and recognition of the session are required to access the system. Administrators have access to manage users and privileges to ensure the data confidentiality and they can see only the usernames (passwords are encrypted using the MD5 algorithm). This way ensure that no one can perform functions for which haven’t privileges.

3.2.2 SCALABILITY

The project has potential improvements in visual appearance, but is done so that it can be modified easily. Is based on the MVC pattern, allowing easy modification and restructuring. Virtually all web content is stored in the database change to another language would be made easily.

We used a Model-View-Controller to separate the user interface, controllers and business logic modules that seamlessly connect to the database. This will facilitate the analysis of new features, changes in the future and testing.

3.2.3 EFFICIENCY

The system ensures that users can use the platform at optimum performance without excessive response times. Ensures that the administrator and the investigators can work with a flexible and reliable. We have not found many problems of efficiency in implementing our code. Good design provides the most common mistakes when we are programming.

3.2.4 PORTABILITY

The bests things of the system are its versatility and portability to other platforms. The web was made in the test environment on a Windows Operating System and was no problem to change the platform offered by the University that was Linux. The only requirement was that the OS had installed a web server and a database server.
4. PROPOSED SOLUTION

4.1 DEVELOPMENT

4.1.1 SYSTEM ARCHITECTURE AND PLATFORM

We used Windows in pre-production and during the development. For the final system it’s possible to use Windows, Linux and Mac OS X.

4.1.2 INITIAL CONSIDERATIONS

First initial considerations were trying to choose programming language would realize the web application. The most common programming languages are Active Server Pages (ASP) and PHP Hypertext Preprocessor (PHP). With the two languages can develop the same applications, properties and mode of operation are similar. The two scripts are run through which will be interpreted by the server, provide access and communication with databases. The fact is that ASP differential to be free software, involves the development of modules or plug-ins specific by the community to provide any function or feature for free. Instead ASP is supported by libraries or modules that must be purchased from private companies that created them. ASP language it is easier to interpret, but the programming and implementation is more complicated, especially compared to PHP.

Another consideration for the performance of the web application was to see the possibilities in the world of data base management, systems designed to interface between the database, users and applications. One of the features that we will consider in the election, is that it is a free software. Managers are more traditional Open Source PostgreSQL, MySQL and SQLite.

Another important consideration was to decide whether we would use a framework to help us in developing the website, without the need for great programming, and to create a solution from scratch can be very expensive.

The Frameworks are born from the need to create projects organized and provide development tools that streamline this process. A Framework already provides many pre-built software modules, easily attachable to your project if necessary. All they enable designers and programmers to focus as much as possible in identifying requirements instead of discussing problems of low level. One drawback, we can find that there Frameworks introducing excessive unnecessary lines of code that make the code harder to read for a programmer unfamiliar with the environment.

Among the most popular frameworks are CakePHP, Symfony and CodeIgniter, Django and Python Frameworks. Between these we have valued Framework that best fits our needs and chose the use of PHP Frameworks, CakePHP or Symphony.

To decide for one of two chosen were proposed a vote simply by selecting a number of features that help us to decide for one and whenever possible was voted the best of both, assigning a + to the best and - the worst. The more positive points that I had won.
## 4.1.3 PROGRAMMING LANGUAGES AND LIBRARIES

### HTML

HyperText Markup Language (HTML) is the main markup language for displaying web pages and other information that can be displayed in a web browser. HTML is written in the form of HTML elements consisting of tags enclosed in angle brackets (like `<html>`), within the web page content. HTML tags most commonly come in pairs like `<h1>` and `</h1>`, although some tags, known as empty elements, are unpaired, for example `<img>`. The first tag in a pair is the start tag, the second tag is the end tag (they are also called opening tags and closing tags). In between these tags web designers can add text, tags, comments and other types of text-based content.

The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

---

1. **Execution speed**: 2 (+) vs -
2. **Easy to install and configure**: 1 (+) vs -
3. **Validation rules in JS (client)**: 2 (+) vs -
4. **Validation rules in PHP (server)**: 2 (+) vs -
5. **Availability on hosting**: 1 (−) vs +
6. **Simple to use**: 1 (−) vs +
7. **Learning curve**: 1 (−) vs +
8. **Object-Oriented**: 2 (+) vs -
9. **Stability**: 2 (+) vs -
10. **Readable code**: 2 (+) vs -
11. **Easy to extend**: 2 (−) vs +
12. **Testing and debugging**: 2 (+) vs -
13. **Using templates**: 2 (+) vs -
14. **Active community of users**: 2 (−) vs +
15. **Assistant Views**: 2 (−) vs +
16. **Cache Management**: 2 (+) vs -
17. **Authentication**: 2 (+) vs -
18. **Smart URLs**: 2 (−) vs +
19. **PHP compatibility**: 2 (−) vs +
20. **Construction of views**: 2 (+) vs -
21. **Creating Forms**: 2 (+) vs -
22. **Excels libraries**: 2 (−) vs +

**TOTAL**: 25 - 15

---

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts in languages such as JavaScript which affect the behavior of HTML webpages.

**PHP**

PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. It is one of the first developed server-side scripting languages to be embedded into an HTML source document rather than calling an external file to process data. The code is interpreted by a Web server with a PHP processor module which generates the resulting Web page. It also has evolved to include a command-line interface capability and can be used in standalone graphical applications. PHP can be deployed on most Web servers and also as a standalone shell on almost every operating system and platform free of charge. A competitor to Microsoft’s Active Server Pages (ASP) server-side script engine and similar languages, PHP is installed on more than 20 million Web sites and 1 million Web servers. Software that uses PHP includes Joomla, Wordpress, MyBB, and Drupal.

PHP was originally created by Rasmus Lerdorf in 1995. The main implementation of PHP is now produced by The PHP Group and serves as the formal reference to the PHP language. PHP is free software released under the PHP License, which is incompatible with the GNU General Public License (GPL) due to restrictions on the usage of the term PHP. While PHP originally stood for "Personal Home Page", it is now said to stand for "PHP: Hypertext Preprocessor", a recursive acronym.

**PHPEXCEL**

Project providing a set of classes for the PHP programming language, which allow you to write to and read from different spreadsheet file formats, like Excel (BIFF) .xls, Excel 2007 (OfficeOpenXML) .xlsx, CSV, Libre/OpenOffice Calc .ods, Gnumeric, PDF, HTML, ...

This project is built around Microsoft's OpenXML standard and PHP.

Checkout the features this class set provides, such as setting spreadsheet meta data (author, title, description, ...), multiple worksheets, different fonts and font styles, cell borders, fills, gradients, adding images to your spreadsheet, calculating formulas, converting between file types and much, much more.

**JAVASCRIPT (jQuery)**

jQuery is a cross-browser JavaScript library designed to simplify the client-side scripting of HTML.[1] It was released in January 2006 at BarCamp NYC by John Resig. Used by over 55% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today.

jQuery is free, open source software, dual-licensed under the MIT License or the GNU General Public License, Version 2. jQuery’s syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced

---

effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the
creation of powerful dynamic web pages and web applications.
jQuery includes the following features:

- DOM element selections using the cross-browser open source selector engine Sizzle, a
  spin-off out of the jQuery project
- DOM traversal and modification (including support for CSS 1-3)
- DOM manipulation based on CSS selectors that uses node elements name and node
  elements attributes (id and class) as criteria to build selectors
- Events
- Effects and animations
- Ajax
- Extensibility through plug-ins
- Utilities - such as user agent information, feature detection
- Compatibility methods that are natively available in modern browsers but need fallbacks
  for older ones - For example the inArray() and each() functions.
- Cross-browser support

4.1.3 WEB SERVER APACHE\(^5\)
The Apache HTTP Server, commonly referred to as Apache, is web server software
notable for playing a key role in the initial growth of the World Wide Web. In 2009 it
became the first web server software to surpass the 100 million website milestone.
Apache was the first viable alternative to the Netscape Communications Corporation web server
(currently named Oracle iPlanet Web Server), and since has evolved to dominate other web
servers in terms of functionality and performance. Typically Apache is run on a Unix-like operating
system.
Apache is developed and maintained by an open community of developers under the auspices of
the Apache Software Foundation. The application is available for a wide variety of operating
systems, including Unix, FreeBSD, Linux, Solaris, Novell NetWare, Mac OS X, Microsoft Windows,
OS/2, TPF, and eComStation. Released under the Apache License, Apache is open-source software.
Apache was originally based on NCSA HTTPd code. The NCSA code has since been removed from
Apache, due to a rewrite.
Since April 1996 Apache has been the most popular HTTP server software in use. As of March 2012
Apache was estimated to serve 57.46% of all active websites and 65.24% of the top servers across
all domains.

4.1.4 DATABASE SERVER MySQL\(^6\)
MySQL is the world's most used open source relational database management system
(RDBMS)[6] that runs as a server providing multi-user access to a number of databases.
It is named after co-founder Michael Widenius' daughter, My. The SQL phrase stands
for Structured Query Language.
The MySQL development project has made its source code available under the terms of the GNU
General Public License, as well as under a variety of proprietary agreements. MySQL was owned

\(^6\) [http://en.wikipedia.org/wiki/Mysql](http://en.wikipedia.org/wiki/Mysql)
and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

Free-software-open source projects that require a full-featured database management system often use MySQL. For commercial use, several paid editions are available, and offer additional functionality. Applications which use MySQL databases include: TYPO3, Joomla, WordPress, phpBB, MyBB, Drupal and other software built on the LAMP software stack. MySQL is also used in many high-profile, large-scale World Wide Web products, including Wikipedia, Google (though not for searches), Facebook and Twitter.

4.1.5 FRAMEWORK PHP CAKEPHP

CakePHP is a rapid development framework for PHP that provides an extensible architecture for developing, maintaining, and deploying applications. Using commonly known design patterns like MVC and ORM within the convention over configuration paradigm, CakePHP reduces development costs and helps developers write less code.

**Hot Features**

- **No Configuration** - Set-up the database and let the magic begin
- **Extremely Simple** - Just look at the name...It's Cake
- **Active, Friendly Community** - Join us #cakephp on IRC. We’d love to help you get started.
- **Flexible License** - Distributed under the MIT License
- **Clean IP** - Every line of code was written by the CakePHP development team
- **Best Practices** - covering security, authentication, and session handling, among the many other features.
- **OO** - Whether you are a seasoned object-oriented programmer or a beginner, you'll feel comfortable

**Basic Features**

- MVC\(^8\) Architecture
- Application Scaffolding
- Code generation via Bake
- Helpers for HTML, Forms, Pagination, AJAX, Javascript, XML, RSS and more
- Access Control Lists and Authentication
- Simple yet extensive validation of model data
- Router for mapping urls and handling extensions
- Security, Session, and RequestHandler Components
- Utility classes for working with Files, Folders, Arrays and more

**More Features**

- Internationalization and Localization with static translations in gettext style or dynamic translations of model data.
- Full Console environment with support for multiple tasks. Included shells: bake, schema, acl, i18 extractor, api
- CSRF protection via Security Component
- HTTP Authentication via Security Component

---


\(^8\) Model - View - Controller
- Flexible Caching: use memcache, apc, xcache, the file system, or a model to speed up your application
- Configure class to provide dynamic handling of configuration settings and App class to handle importing required classes

### 4.2 HARDWARE AND OPERATIVE SYSTEM ARCHITECTURES

The computer used for the development is a Sony VAIO VGN-NS11Z with the following features:

- CPU: Intel (R) Core (TM) 2 Duo T5800 @ 2.00 GHz
- RAM: 4.00 GB
- Operating System: Windows 7 32-bit

The installation was done through the WAMPP software.

> **WAMPP** is an acronym formed from the initials of the operating system Microsoft Windows and the principal components of the package: Apache, MySQL and one of PHP, Perl or Python.\(^9\)

After carrying out the installation, configure some parameters to finish preparing the team. Define the web server root directory, where we store the files that make up the page. This will modify the "httpd.conf" file. The manager phpMyAdmin configure a user account with sufficient privileges and permissions to access the database.

> **phpMyAdmin** is a free and open source tool written in PHP intended to handle the administration of MySQL with the use of a Web browser. It can perform various tasks such as creating, modifying or deleting databases, tables, fields or rows; executing SQL statements; or managing users and permissions.\(^10\)

We can say that this will be the pre-production environment and we have a server to host the web, provided by the university.

Holmes is a KVM virtual machine resident on a Linux server and has the following features:

- CPU: (2.13GHz, 4M Cache, 4.86 GT / s QPI), 800MHz Max Memory
- Memory: 12GB
- Operating System: Linux

---


4.3 SOFTWARE ARCHITECTURE

The application structure is dictated by the design pattern Model-View-Controller (MVC) of the CakePHP Framework.

"CakePHP follows the MVC software design pattern. Programming using MVC separates your application into three main parts:

**The Model layer**
The Model layer represents the part of your application that implements the business logic. This means that it is responsible for retrieving data, converting it into meaningful concepts to your application, as well as processing, validating, associating and any other task relative to handling this data.

At a first glance, Model objects can be looked at as the first layer of interaction with any database you might be using for your application. But in general they stand for the major concepts around which you implement your application.

In the case of a social network, the Model layer would take care of tasks such as saving the user data, saving friends associations, storing and retrieving user photos, finding new friends for suggestions, etc. While the model objects can be thought as “Friend”, “User”, “Comment”, “Photo”

**The View layer**
The View renders a presentation of modeled data. Being separated from the Model objects, it is responsible for using the information it has available to produce any presentational interface your application might need.

For example, as the Model layer returns a set of data, the view would use it to render a HTML page containing it. Or a XML formatted result for others to consume.

The View layer is not only limited to HTML or text representation of the data, it can be used to deliver a wide variety of formats depending on your needs, such as videos, music, documents and any other format you can think of.

**The Controller layer**
The Controller layer handles requests from users. It’s responsible for rendering back a response with the aid of both the Model and the View Layer.

Controllers can be seen as managers taking care that all needed resources for completing a task are delegated to the correct workers. It waits for petitions from clients, checks their validity according to authentication or authorization rules, delegates data fetching or processing to the model, and selects the correct type of presentational data that the client is accepting, to finally delegate this rendering process to the View layer.
The typical CakePHP request cycle starts with a user requesting a page or resource in your application. This request is first processed by a dispatcher which will select the correct controller object to handle it.

Once the request arrives at the controller, it will communicate with the Model layer to process any data fetching or saving operation that might be needed. After this communication is over, the controller will proceed at delegating to the correct view object the task of generating an output resulting from the data provided by the model.

Finally, when this output is generated, it is immediately rendered to the user.\(^{11}\)

4.4 USERS INTERFACE

Graphic design of application will look like standard design of CakePHP Framework. This design contains the presentation code from view.

Design files are into the folder “/app/View/Layout/” of our application. The default layout offered by CakePHP can be replaced overwriting the file “/app/View/Layout/default.ctp”. You can create as many designs as you wish: just place them in “app/View/”.

CakePHP has a built style sheet located into “app/webroot/css”. The style sheet is very comprehensive, with various styles for fields: input, select, etc.

Will also discuss how to make the link, this link would function, followed by the name you want to appear in the design and then the folder name, for example the name of the table followed by the file want to link (index, edit, add, view).

As we mentioned before is not a priority of this project for this reason we have used the default.ctp.
4.5 CONCEPTUAL MODEL

4.5.1 MODEL

A “model” is represented into the database as a table. CakePHP reads this information from the table structure automatically from the database and used as the basis of the model. CakePHP models does not include any information about the fields it contains. All informations are automatically set to retrieving data from database tables.

CakePHP has many conventions for tables:

- The names of the tables are defined in the plural and lowercase letters, while the model class is defined in the singular. For example, for the User class we must define the table Users.
- All tables that define a model class, you need the primary key attribute id. This key must be the property of being an auto-incremental field.

If a table has the attributes created and modified, its value is automatically updated by CakePHP when you create or modify a “model-object” with framework’s APIs.

Each table should have a model class of the same name as the table, but in the singular also kept on file with the same name in the table, but in the singular, with “.php” extension. This file should be stored in the folder “app/Model/”.

Data Validation

CakePHP has a feature to validate form data validation rules that defines the CakePHP avoid problems and are essential to ensure data integrity:

- **alphaNumericos**: The field can only contain letters and numbers.
- **boolean**: The field to be a value booleano. Los Valid values are true or false or 1 or 0.
- **date**: Validates that the field is a proper date. It has a parameter to indicate the date format.
- **decimal**: Validates that the field has a decimal value. It has a parameter to request the exact number of decimals.
- **inList**: The rule ensures that the field value is a value from a list that is passed as parameter.
- **numeric**: The value to be a valid numeric value.
- **notEmpty**: Basic rule to ensure that the field is not empty.

Besides each validation rule to indicate whether the field is:

- **required**: The field must be present.
- **allowEmpty**: The field allows null values or empty regardless of the validation rule.
4.5.2 CONTROLLER
The driver class name is formed by placing the table name in the plural, followed by 'Controller'. This class should be stored in a file inside the folder “app/Controller/”.

4.5.3 VIEW
To create the views of the tables in the database, create a folder for each table in the folder “app/View/”. The file must have the same name of the function specified in the controller (for example “login.ctp” to render the login function of UsersController).

4.5.4 DATABASE
- The ID field must be PRIMARY KEY and must have the auto_increment attribute.
- Text fields should use the collate utf8_unicode, this indicates that the text is UTF-8 and that the searches on this field are not case-sensitive (case insensitive).
- Related fields are indexed.
- The engine must be InnoDB.
4.6 SYSTEM MODEL

4.6.1 USE CASE MODEL
Unlike the conceptual model, the use case model identifies the features that the system must be associated with each actor and the tasks to be performed.

<table>
<thead>
<tr>
<th>ADMIN</th>
<th>INVESTIGATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Manage users:</td>
<td>• Manage Patients:</td>
</tr>
<tr>
<td>o Create new User</td>
<td>o Create New Patient</td>
</tr>
<tr>
<td>o View Users</td>
<td>o Edit Patient</td>
</tr>
<tr>
<td>o Edit User</td>
<td>o View Patient</td>
</tr>
<tr>
<td>o Delete User</td>
<td>o Delete Patient</td>
</tr>
<tr>
<td>• Manage studies:</td>
<td>• Manage CRF:</td>
</tr>
<tr>
<td>o Create new Study using Define.xls</td>
<td>o Preview Study</td>
</tr>
<tr>
<td>o Create new Study using the wizard</td>
<td>o Enter patient data</td>
</tr>
<tr>
<td>o View Studies</td>
<td>o View patient data</td>
</tr>
<tr>
<td>o Edit Study</td>
<td>o Edit patient data</td>
</tr>
<tr>
<td>o Delete Study</td>
<td></td>
</tr>
<tr>
<td>o Export Study’s structure to Define.xls</td>
<td></td>
</tr>
</tbody>
</table>

The use case diagram describes how the actors interact with the system, describing each use case in the sequence of events. In our specification of a use case we have this properties:

• Use Case Name
• Actor
• Events Description
## **ADMIN: MANAGE USERS**

<table>
<thead>
<tr>
<th>Use Case: Create User</th>
<th>Actor: Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td><strong>System</strong></td>
</tr>
<tr>
<td>1. The administrator fill the form with user’s data and click on Add button.</td>
<td>2. The system check if exist an user with the same username. If it exist return an error, else create the new user and return a confirm message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case: View Users</th>
<th>Actor: Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td><strong>System</strong></td>
</tr>
<tr>
<td>1. The administrator request users list.</td>
<td>2. The system retrieves information from the database and lists the users.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case: Edit User</th>
<th>Actor: Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td><strong>System</strong></td>
</tr>
<tr>
<td>1. The administrator fill the form with user’s data and click on Modify button.</td>
<td>2. The system check if exist an user with the same username. If it exist return an error, else modify user’s data and return a confirm message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case: Delete User</th>
<th>Actor: Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td><strong>System</strong></td>
</tr>
<tr>
<td>1. The administrator click on Delete button in the users list.</td>
<td>2. The system ask the user to confirm delete operation.</td>
</tr>
<tr>
<td>3. The administrator click on OK or Cancel.</td>
<td>4. if the user has clicked on OK, the system deletes the user.</td>
</tr>
</tbody>
</table>
## ADMIN: MANAGE STUDIES

### Use Case: Create new Study using Define.xls
**Actor:** Admin

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The administrator select the file Define.xls and clicks on Import button</td>
<td>2. The system check if exist a study with the same name. If exist return an error, else create the new study and return a confirm message.</td>
</tr>
</tbody>
</table>

### Use Case: Create new Study using the wizard
**Actor:** Admin

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The administrator fill the form with study’s data and click on Add button.</td>
<td>2. The system check if exist a study with the same name. If exist return an error, else create the new study and return a confirm message.</td>
</tr>
</tbody>
</table>

### Use Case: View Studies
**Actor:** Admin

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The administrator request studies list.</td>
<td>2. The system retrieves information from the database and lists the studies.</td>
</tr>
</tbody>
</table>

### Use Case: Edit Study
**Actor:** Admin

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The administrator fill the form with study’s data and click on Modify button.</td>
<td>2. The system check if exist a study with the same name. If it exist return an error, else create the new study and return a confirm message.</td>
</tr>
</tbody>
</table>
### Use Case: Delete Study
**Actor:** Admin

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The administrator click on Delete button in the studies list.</td>
<td>2. The system ask the user to confirm delete operation.</td>
</tr>
<tr>
<td>3. The administrator click on OK or Cancel.</td>
<td>4. If the user has clicked on OK, the system deletes the study.</td>
</tr>
</tbody>
</table>

### Use Case: Export structure to Define.xls
**Actor:** Admin

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The administrator click on Export button in the studies list.</td>
<td>2. The system returns the file Define.xls with study’s structure.</td>
</tr>
</tbody>
</table>
**INVESTIGATOR: MANAGE PATIENTS**

### Use Case: Create Patient
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Investigator fill the form with patient’s data and click on Add button.</td>
<td>2. The system check if exist a patient with the same username. If it exist return an error, else create the new patient and return a confirm message.</td>
</tr>
</tbody>
</table>

### Use Case: View Patients
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Investigator request users list.</td>
<td>2. The system retrieves information from the database and lists the patients.</td>
</tr>
</tbody>
</table>

### Use Case: Edit Patient
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The investigator fill the form with patient’s data and click on Modify button.</td>
<td>2. The system check if exist a patient with the same username. If it exist return an error, else create the new patient and return a confirm message.</td>
</tr>
</tbody>
</table>

### Use Case: Delete Patient
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The investigator click on Delete button in the patient list.</td>
<td>2. The system ask the user to confirm delete operation.</td>
</tr>
<tr>
<td>3. The administrator click on OK or Cancel.</td>
<td>4. If the user has clicked on OK, the system deletes the patient.</td>
</tr>
</tbody>
</table>
## INVESTIGATOR: MANAGE CRF

### Use Case: Preview Study
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The investigator click on Preview button in the studies list.</td>
<td>2. The system shows a preview of CRF.</td>
</tr>
</tbody>
</table>

### Use Case: Enter Patient Data
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Investigator fill the CRF with patient’s data and click on Save button.</td>
<td>2. The system save the patient’s data relative to that CRF and return a confirm message.</td>
</tr>
</tbody>
</table>

### Use Case: Edit Patient Data
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Investigator fill the CRF with patient’s data and click on Save button.</td>
<td>2. The system save the patient’s data relative to that CRF and return a confirm message.</td>
</tr>
</tbody>
</table>

### Use Case: Delete Patient Data
**Actor:** Investigator

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Investigator click on Delete Data button in the patient list.</td>
<td>2. The system ask the user to confirm delete operation. 4. if the user has clicked on OK, the system deletes the patient’s data relative to him CRF.</td>
</tr>
</tbody>
</table>
4.6.2 CLASS DIAGRAM

MODELS

Image 4 - Class Diagram: Models
CONTROLLERS

Image 5 - Class Diagram: Controllers
### 4.6.3 SEQUENCE DIAGRAM

The sequence diagrams will allow us to model how the operation of system operations will be done and give us a high level diagram of how each operation should be implemented. Sequence diagrams are created for all operations of our system, referring to our classes.

**Login**

![Image 6 - Sequence Diagram: Login](image6.png)

**Logout**

![Image 7 - Sequence Diagram: Logout](image7.png)
Create user

![Sequence Diagram: Create User](image8)

Users list

![Sequence Diagram: Users list](image9)
**Edit user**

![Sequence Diagram: Edit User]

**Delete user**

![Sequence Diagram: Delete User]
Create study

Studies list
**Edit study**

![Sequence Diagram: Edit Study]

**Delete study**

![Sequence Diagram: Delete Study]
Create Patient

Image 16 - Sequence Diagram: Create Patient

Patients list

Image 17 - Sequence Diagram: Patients List
**Edit patient**

![Edit patient diagram]

**Delete patient**

![Delete patient diagram]
4.6.4 DATABASE MODEL

Below we have the relational schema of the database
TABLES CREATION SCRIPTS

TABLE STRUCTURE `generalinfos`

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>No</td>
</tr>
<tr>
<td>client</td>
<td>varchar(45)</td>
<td>No</td>
</tr>
<tr>
<td>code</td>
<td>varchar(32)</td>
<td>No</td>
</tr>
<tr>
<td>protocol</td>
<td>varchar(32)</td>
<td>No</td>
</tr>
<tr>
<td>name</td>
<td>varchar(50)</td>
<td>No</td>
</tr>
<tr>
<td>shortname</td>
<td>varchar(32)</td>
<td>No</td>
</tr>
<tr>
<td>created</td>
<td>datetime</td>
<td>No</td>
</tr>
<tr>
<td>modified</td>
<td>datetime</td>
<td>No</td>
</tr>
</tbody>
</table>

CREATE TABLE IF NOT EXISTS `generalinfos` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `client` varchar(45) NOT NULL,
  `code` varchar(32) NOT NULL,
  `protocol` varchar(32) NOT NULL,
  `name` varchar(50) NOT NULL,
  `shortname` varchar(32) NOT NULL,
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
TABLE STRUCTURE `sectiondefinitions`

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>generalinfo_id</td>
<td>int(11)</td>
<td>No</td>
<td>sectiondistributions -&gt; generalinfo_id</td>
</tr>
<tr>
<td>section</td>
<td>varchar(32)</td>
<td>No</td>
<td>sectiondistributions -&gt; section</td>
</tr>
<tr>
<td>structure</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>left_label</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>right_label</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>varchar(20)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>validation</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>other_parameters</td>
<td>varchar(200)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>error_message</td>
<td>varchar(50)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>help_message</td>
<td>varchar(50)</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

CREATE TABLE IF NOT EXISTS `sectiondefinitions` (  
`id` int(11) NOT NULL AUTO_INCREMENT,  
`generalinfo_id` int(11) NOT NULL,  
`section` varchar(32) NOT NULL,  
`structure` varchar(32) NOT NULL,  
`variable` varchar(32) NOT NULL,  
`table` varchar(32) NOT NULL,  
`left_label` varchar(32) NOT NULL,  
`right_label` varchar(32) NOT NULL,  
`type` varchar(32) NOT NULL,  
`format` varchar(20) NOT NULL,  
`validation` varchar(32) NOT NULL,  
`other_parameters` varchar(200) NOT NULL,  
`error_message` varchar(50) NOT NULL,  
`help_message` varchar(50) NOT NULL,  
PRIMARY KEY (`id`),  
KEY `section` (`section`),  
KEY `generalinfo_id` (`generalinfo_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
### TABLE STRUCTURE `sectiondistributions`:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>generalinfo_id</td>
<td>int(11)</td>
<td>No</td>
<td>generalinfos -&gt; id</td>
</tr>
<tr>
<td>section</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>visit</td>
<td>varchar(12)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>page</td>
<td>varchar(12)</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

```sql
CREATE TABLE IF NOT EXISTS `sectiondistributions` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `generalinfo_id` int(11) NOT NULL,
  `section` varchar(32) NOT NULL,
  `label` varchar(32) NOT NULL,
  `visit` varchar(20) NOT NULL,
  `page` varchar(12) NOT NULL,
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`),
  KEY `generalinfo_id` (`generalinfo_id`),
  KEY `section` (`section`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=16 ;
```
### TABLE STRUCTURE `formats`

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>generalinfo_id</td>
<td>int(11)</td>
<td>No</td>
<td>sectiondefinitions -&gt; generalinfo_id</td>
</tr>
<tr>
<td>format</td>
<td>varchar(32)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>values</td>
<td>varchar(12)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>varchar(12)</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

```sql
CREATE TABLE IF NOT EXISTS `formats` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `generalinfo_id` int(11) NOT NULL,  
    `format` varchar(32) NOT NULL,  
    `values` varchar(12) NOT NULL,  
    `label` varchar(12) NOT NULL,  
    PRIMARY KEY (`id`),  
    KEY `generalinfo_id` (`generalinfo_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

-- --------------------------------------------------------
TABLE STRUCTURE `users`:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(10)</td>
<td>No</td>
</tr>
<tr>
<td>username</td>
<td>varchar(50)</td>
<td>No</td>
</tr>
<tr>
<td>password</td>
<td>varchar(50)</td>
<td>No</td>
</tr>
<tr>
<td>role</td>
<td>varchar(20)</td>
<td>No</td>
</tr>
<tr>
<td>created</td>
<td>datetime</td>
<td>No</td>
</tr>
<tr>
<td>modified</td>
<td>datetime</td>
<td>No</td>
</tr>
</tbody>
</table>

CREATE TABLE IF NOT EXISTS `users` (  
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,  
`username` varchar(50) DEFAULT NULL,  
`password` varchar(50) DEFAULT NULL,  
`role` varchar(15) DEFAULT NULL,  
`created` datetime DEFAULT NULL,  
`modified` datetime DEFAULT NULL,  
PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=8 ;
TABLE STRUCTURE `patients`

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>generalinfo_id</td>
<td>int(11)</td>
<td>No</td>
<td>generalinfos -&gt; id</td>
</tr>
<tr>
<td>initials</td>
<td>varchar(11)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>created</td>
<td>datetime</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>modified</td>
<td>datetime</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

CREATE TABLE IF NOT EXISTS `patients` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `generalinfo_id` int(11) NOT NULL,
  `initials` varchar(11) NOT NULL,
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`),
  KEY `generalinfo_id` (`generalinfo_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

-- ---------------------------------------------------
TABLE LIMITS `formats`

ALTER TABLE `formats`
  ADD CONSTRAINT `formats_ibfk_1` FOREIGN KEY (`generalinfo_id`) REFERENCES `sectiondefinitions` (`generalinfo_id`) ON DELETE CASCADE ON UPDATE NO ACTION;

-- ---------------------------------------------------------------

TABLE LIMITS `patients`

ALTER TABLE `patients`
  ADD CONSTRAINT `patients_ibfk_1` FOREIGN KEY (`generalinfo_id`) REFERENCES `generalinfos` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

-- ---------------------------------------------------------------

TABLE LIMITS `sectiondefinitions`

ALTER TABLE `sectiondefinitions`
  ADD CONSTRAINT `sectiondefinitions_ibfk_1` FOREIGN KEY (`section`) REFERENCES `sectiondistributions` (`section`) ON DELETE CASCADE ON UPDATE NO ACTION,
  ADD CONSTRAINT `sectiondefinitions_ibfk_2` FOREIGN KEY (`generalinfo_id`) REFERENCES `sectiondistributions` (`generalinfo_id`) ON DELETE CASCADE ON UPDATE NO ACTION;

-- ---------------------------------------------------------------

TABLE LIMITS `sectiondistributions`

ALTER TABLE `sectiondistributions`
  ADD CONSTRAINT `sectiondistributions_ibfk_1` FOREIGN KEY (`generalinfo_id`) REFERENCES `generalinfos` (`id`) ON DELETE CASCADE;
4.6.5 EXCEL DEFINITION

Sheet - Sections distribution

In this spreadsheet is defined how the sections are distributed along the CRF on different pages. Each page can contain one or more sections. The sections can be repeated along the study using different page numbers. For example, it is pretended to measure the blood pressure of the patients on different visits; for that, a section called Blood Sample is defined and repeated on different pages (visits) using one row per each page.

Four columns must be defined:

Section: Abbreviated name to identify the section. These names will linked to the spreadsheet Sections Definition where the contents of each section will be defined.

Label: Title of the section which will be showed as a header in the web form. If this field is omitted no header will be printed.

Visit: Each section must be related with a visit. The visit can be numeric or text. This field is mandatory.

Page: Indicates on which pages will be presented each section. If one section is repeated on different pages, a new row with the same section name and different page number must be added. This field for now is “experimental”.

**Sheet - Sections definition**

In this spreadsheet is defined the content of the *sections*. Each *section* must be defined only once. As introduced before, a *section* is formed by structures and variables. Since the variable is the smaller information unit that can be defined, each row on the excel will define one single variable. Then, by means of different properties given in the excel columns, the variables will be gathered constituting *sections* and *structures*\(^\text{12}\).

The following columns are available:

**Section:** Name of *section*, previously assigned to a CRF page and visit in Sections Distribution. Must contain the same *section* name for all the variables that form the *section*. This field is mandatory.

**Structure:** This field is mandatory. Four different structures are defined: LINE, SEQ and TABLE.

**LINE**
The LINE *structure* is the most simple type of structure. It only contains one variable which is displayed on the WEB as simple line with the specified variable properties in other columns, for example,

```
Height: [ ] m.
```

Image 19 - Excel Definition: Line

**TABLE**
The TABLE *structure* gathers information using a table layout. The first variable defined as TABLE will correspond to the row identifier. This must have a format defined by the user where each value will correspond to one row. The order of the rows will be given by the format value, although the labels will be printed in each row.

The next variables will define the columns of the table. To identify unequivocally this data in the database, a new primary key will be added corresponding to the variable defined for the rows.

The headers of each column in the table are given in the column *left_label*.

To generate a table like below for Vital Signs, the first variable with TABLE *structure* must be Post_dose. This variable is associated with a user defined format which contains the levels: 1="1 hour", 2="2 hours" and 3="3 hours". See column Format for more information about user defined formats. The following variables defined as TABLE will be Date, Time, Systolic and Diastolic Blood Pressure.

---

47
SEQ
This structure organizes the variables in a similar way to the TABLE. In this case, however, the first variable will contain an automatically generated sequence (row number) instead of the values from a user defined format. In this way, there is no limitation on the number of rows. The following variables in the SEQ structure will define the columns of the table.

### CONCOMITANT MEDICATIONS

<table>
<thead>
<tr>
<th>Num.</th>
<th>Medication</th>
<th>Start Date (MM/DD/YYYY)</th>
<th>Stop Date (MM/DD/YYYY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: section END OF STUDY contains four LINE structures with checkbox variables. Besides, different titles are added to group appropriately the variables and a note at the end of the section is added.

TEXT, TITLE1, TITLE2

These structures do not affect the behavior of the variables, are just used for adding texts inside sections.

The text format can be changed using the column other_parameters, see: text_font, text_size, text_bold, text_italic, text_underline, text_just, text_color.

TITLE1 by default uses text_size = 5 and text_bold = TRUE.

TITLE2 by default uses text_size = 4 and text_bold = TRUE, text_italic=TRUE.
END OF STUDY

End of Study Reason:  
(click only one reason)  

☐ Completed study  
☐ AE/SAE (complete AE CRF & SAE form, if applicable)

Other Reasons:  
(click only one reason)  

☐ Non-compliant participant  
☐ Concomitant medication _

Once the END OF STUDY page is completed, please freeze the patient if this is ready for review.  

**Variable:** name of the variable; it is recommended to use the first two letters of the table which the variable belongs and not exceed 8 characters long. This field is mandatory.

The following columns are used to complete the information of each variable:

**Table:** this column indicates the name of the table in the database that will store this variable. It is recommended to use 2 letters to define the name of the table that will be always included in the name of the variable. This field is mandatory.

**Left_label:** text of the question that will be showed in the web form on the left side of the field to introduce the answer. See the column other_parameters for further options on how to modify the appearance of this label. This field is optional.

**Right_label:** text to show on the right side of the answer field; commonly used indicate the unit or format of the answer, e.g, meters or date format as dd-mm-yyyy. This field is optional.

**Type and Format:**
The columns type and format work together to specify how to retrieve and present the information.
<table>
<thead>
<tr>
<th>Type</th>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM</td>
<td>n/n.d</td>
<td>Numeric variables where n represent number of digits for the integer part and d the number of decimals.</td>
<td>Height: 1.65 m.</td>
</tr>
<tr>
<td>TEXT</td>
<td>n</td>
<td>Text variables, n indicates the number of character that are allowed.</td>
<td>Sex: Female</td>
</tr>
<tr>
<td>DATE</td>
<td>ddmmyyyyy ddmmmyy ddmmmyy ddmmmyy</td>
<td>Date format. Two options for the month: mm for numeric format (01-12), or mmm for text format (JAN-DEC). Two options for the year presentation: yyyy for the full year, e.g. 2012, and yy for only the last two digits, e.g., 12.</td>
<td>ddmmyyyyy 01-JAN-2012 ddmmyy 01-01-12</td>
</tr>
<tr>
<td>PDATE</td>
<td>Idem DATE</td>
<td>Like date but in this case partial dates are allowed, e.g, if ddmmyy is specified in format column, users can also provide month-year or only year. See Date for more details on data formats.</td>
<td>ddmmyyyyy 01-JAN-2012 JAN-2012 2012</td>
</tr>
<tr>
<td>TIME</td>
<td>hh:mm</td>
<td>Time 24 hours format.</td>
<td>Gender: Male/Female</td>
</tr>
<tr>
<td>LIST</td>
<td>User_format</td>
<td>Drop down list, allows one of multiple choices given by a user defined format, see spreadsheet Formats.</td>
<td></td>
</tr>
<tr>
<td>RBUTTON</td>
<td>User_format</td>
<td>Radio button, allows one of multiple choices given by a user defined format, see spreadsheet Formats.</td>
<td></td>
</tr>
<tr>
<td>CHECKBOX</td>
<td>User_format</td>
<td>Checkbox, allows one or multiple choices among the values given by a user defined format, see spreadsheet Formats.</td>
<td>Treatment A/B</td>
</tr>
<tr>
<td>DERIVED</td>
<td>n/n.d</td>
<td>This type of input indicates that the content will be automatically calculated given information from other fields. The format must specify how to present the result, see numeric format.</td>
<td></td>
</tr>
</tbody>
</table>

**Validation:**
If an answer is given, one of the following functions can be specified to perform a validation:

\[\text{in}(\text{value1, value2, ...})\]: the value must be contained among the given list of values, e.g. score must be 1, 2, 3 or 99, \(\text{in}(1,2,3,99)\).

\[\text{gt(value)}\], greater than a value, e.g. must be heavier than 40 kg, \(\text{gt}(40)\).

\[\text{ge(value)}\], greater equal to a value, e.g. must be at least 18 years old, \(\text{gt}(18)\).

\[\text{lt(value)}\], less than a value, e.g. must be younger than 18 years old, \(\text{lt}(18)\).

\[\text{le(value)}\], less or equal to a value, e.g. must be smaller or equal to 220 cm, \(\text{le}(220)\).
**bw(min,max):** between min and max values, both included; e.g. age must be between 0 and 120, bw(0,120).

**Comments:**
By default, all fields defined as dates are automatically validated. To avoid this, `validate_date = FALSE` must be specified in the column `Other_parameters`.

**Other_parameters:**
Several parameters may be together, separated by semicolons “;”

**Formula:** `formula=expression`, being `expression` a mathematical formula comprehensible on Javascript for calculating the value for a Derived type variable.

**left_input:** if TRUE the input field is presented before the label; the default is FALSE.

**open_if:** `logical condition`, if true, the field will be open and will be possible to introduce information; if false, the field will be locked.

**show_if:** `logical condition`, if true, the question will appear; hidden otherwise.

**close_if:** `logical condition`, if true, the field will be locked; open otherwise.

**col_size:** percentage of table

**input_size:** specify the width of an input in number of characters.

**required:** if TRUE, the field must be answered before saving the data of that section. (?)

**Text options:** the following options are applied to the text specified `left_label` (SDEF).

**text_font:** one of the following: verdana, arial, courier,... html available fonts?

**text_bold:** TRUE for bold text, otherwise FALSE (default).

**text_italic:** TRUE for italic text, otherwise FALSE (default).

**text_underline:** true for underlined text, otherwise FALSE (default)

**text_size:** text size, from 1 (smallest) to 7 (biggest).

**text_just:** justification for TEXT structures. One of the followings: left (default), right, center.

**text_color:** color for TEXT structures. One among the following: black (default), green, red, blue, yellow, grey, ???

**validate_date:** FALSE to omit the automatic validation of the dates.
**Error and Help messages:**
Variables with HELP messages will be shown with a “?” sign aside, so that the message appears when the user moves the mouse over it.

Variables with ERROR messages need to be linked to a validation expression. When it does not hold, the error message raises up. These messages can override default messages, if implemented.

For instance: validation expression in() could have a default message stating something like “The input must be one of … (the list between parentheses)”.

---

**Sheet – Formats**
In case any variable with type LIST, RADIO_BUTTON or CHECKBOX in included in the CRF, it is necessary to include in this spreadsheet the associated formats.

Three columns must be completed:

**Format:** name for the format.

**Values:** numeric value to order the possible levels of the format.

**Labels:** levels that will be showed as the different option in the LIST, RADIO_BUTTON or CHECKBOX.
4.6.6 SCREEN MOCKUP

Login

Image 24 - Screen Mockup: Login

Home Admin

Image 25 - Screen Mockup: Home Admin
Home Investigator

Image 26 - Screen Mockup: Home Investigator

Home CRF

Image 27 - Screen Mockup: Home CRF
**Home Users**

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
<th>Role</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>josa</td>
<td>55</td>
<td>admin</td>
<td>Delete</td>
</tr>
<tr>
<td>julia</td>
<td>443</td>
<td>admin</td>
<td>Modify</td>
</tr>
<tr>
<td>marc</td>
<td>443</td>
<td>investigator</td>
<td>Delete</td>
</tr>
</tbody>
</table>

<< Previous Next >>

Image 28 - Screen Mockup: Home Users

**Create CRF**

Create CRF

**Edit CRF informations**

Name

EFFICACY AND SAFETY PHASE III CLINICAL TRIAL OF OR

Shortname

MIGRATIX PHASE III

Client

Migrain Pharmaceuticals

Protocol

MIG-35831

Code

MIG001

Save

Image 29 - Screen Mockup: Create CRF
**Edit CRF informations**

- **Name**: EFFICACY AND SAFETY PHASE III CLINICAL TRIAL OF OR
- **Shortname**: MIGRATIX PHASE III
- **Client**: Migrain Pharmaceuticals
- **Protocol**: MIG-35831
- **Code**: MIG001

Image 30 - Screen Mockup: Edit CRF

**Edit CRF sections**

<table>
<thead>
<tr>
<th>Section</th>
<th>SCREENING</th>
<th>BLABLABLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPAGE (TITLE PAGE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIATES (VISIT DATE)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Add section]  [SAVE]

Image 31 - Screen Mockup: Define CRF
4.6.7 COST ANALYSIS

In this section we will make the economic valuation of the cost for the project taking into account human resources, costs of software and hardware needed to carry out the project.

To reflect the cost of this project we rely on the entire application infrastructure. We can group our project:

- Assessment of the cost of software: We quantified the value of programs and the platform on which the system is maintained.
- Hardware Cost: Cost of computers and peripherals that we installed on our system.
- Staff costs: Here we assess the number of hours / profile person we believe is necessary to develop the project.
- Physical Cost: Here we assess the physical disk space occupied by the application.

The cost of all applications and tools used is zero.

<table>
<thead>
<tr>
<th>SOFTWARE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WAMP</td>
<td>Gratis</td>
</tr>
<tr>
<td>Notepad++</td>
<td>Gratis</td>
</tr>
<tr>
<td>MySQL Workbench Visual Database Designer 5.1</td>
<td>Gratis</td>
</tr>
<tr>
<td>OS &amp; Office Applications</td>
<td>Gratis</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>0€</strong></td>
</tr>
</tbody>
</table>

Regarding the project, the hardware has been donated by the University. They wanted to reflect this cost. The hardware cost is included within the budget of total project costs.

<table>
<thead>
<tr>
<th>HARDWARE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Server UPC</td>
<td>2.000€</td>
</tr>
<tr>
<td>SAI</td>
<td>400€</td>
</tr>
<tr>
<td>ROUTER(20Mbits/s)</td>
<td>700€</td>
</tr>
<tr>
<td>PC</td>
<td>1200€</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4.300€</strong></td>
</tr>
</tbody>
</table>
We had to learn to use the tools used in the project, this has meant higher consumption in hours spent on training.

Here is an approximation to the number of people who should carry out the project. Divide the work involved in two, the analyst and programmer who also perform the tests. Prices hours for the professional categories are:

<table>
<thead>
<tr>
<th>Resource</th>
<th>€/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager/Analyst</td>
<td>80</td>
</tr>
<tr>
<td>Programmer</td>
<td>40</td>
</tr>
</tbody>
</table>

The analyst make all the functions of requirements analysis and conceptual model development, also has to do the design of use case diagrams, and layout of the database. You must make the graphical interface of different users. The programmer will perform the interfaces or application screens and should test software generated. It will also make the development of the interface and platform installation and tuning of servers.
To calculate the total actual cost of the project, we have relied on previously developed budgets, which provide for all costs of resources used. We have the result that the value of this project represents a final cost of:

$$\text{FINAL COST} = \text{Hardware} + \text{Software} + \text{Staff} = € 6,552$$

In this section we define the physical cost, ie the physical disk space that occupies the entire application. You want to assess the physical space occupied by all components (images, pages, data, etc., ...). Once this is accomplished, we can make the assessment of how this information is transmitted to users. Quantified as the bandwidth we have, how many concurrent users can connect to the Web application.

<table>
<thead>
<tr>
<th>SPACE REQUESTED</th>
<th>MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server space</td>
<td>20 MB</td>
</tr>
<tr>
<td>Hard Disk space</td>
<td>31MB</td>
</tr>
<tr>
<td>Total</td>
<td>51 MB</td>
</tr>
</tbody>
</table>
5. FUTURE IMPROVEMENTS

Here are some improvements that could be incorporated and considered that can add value to the functionality of the application. There are many possible improvements and we can represent them in a list:

5.1 MULTILANGUAGE

Translation of the program to other languages. CakePHP has a Helper for the Multilanguage that facilitates the programmer to write multilingual views.

5.2 GRAPHICAL IMPROVEMENTS

The application has been developed with a simple graphic. Although this is very simple to intervene on views to improve the look and feel without affecting the data processing part.

5.3 FUNCIONAL REQUIREMENTS

5.3.4 ADD DOCUMENTATION TO CRF

Provide the user the ability to add various documentation files (for example, Protocol, informed consent, etc..), Additional information on the study (type of CRF) and decorative elements (logo). Below we find the mock-up of a possible solution:

![Image 41 - Improvement 5.3.4](Image 41 - Improvement 5.3.4)
5.3.5 ADD INTERMEDIATE SECTIONS AND VISITS

At present you can not add a new section in an intermediate position, you can only append. For this reason we might think to add a button in each section (or visit) to add an intermediate section (or visit).

Below we find the mock-up of a possible solution:

![Image 42 - Improvement 5.3.5]
6. CONCLUSIONS

This project was very ambitious from the beginning but we have designed all parts of the project from the definition of the Excel spreadsheet “Define.xls” until the creation of a Case Report Form (CRF) and its visualization.

We have spent a lot of time training in the CakePHP Framework. In understanding and performance, the more difficult points were the topics of planning and execution time of different tasks. Things are not always gone like we wanted, because of coding errors or bad interpretations of requirements and functionality. We wanted to minimize these errors and therefore, we used the tools to correct these errors as much as possible into design step.

We chose to use a spiral model for software construction. While developing the application has suffered several changes in requirements and design. These changes affect most of the modules, both as regards the application logic that the graphics, as we have a high level of cohesion between some of them. Since the structure is strictly modular, the spiral model facilitates these changes, re-run without having to go through all the phases of the application.

In the physical part of the application, we found major problems in the model database, which often do not meet all the functional requirements and sometimes we had to start all over again. The database has its own update system and maintain system consistency with an action specification about deleting table rows in cascade.

As for the graphics, we have tried to create a stylish and very functional interface although it is very “spartan” and certainly can be improved, perhaps with the addition of graphical effects and advanced features that can enhance the usability of the product. It’s been very hard to combine the graphics with the functional requirements so that everything works together. The end result was achieved thanks to the support and suggestions from all persons involved in the project.

To conclude we can say we are happy with the hours spent working on this project, as already said very ambitious, to get your results. Despite missing some key parts for the release of a final version will not be difficult to upgrade the project to make it competitive in the market, thanks mainly to the design performed carefully to ensure easy maintenance in future versions.
7. PERSONAL EXPERIENCE

As personal experience, technically, I really enjoyed the experience gained using new programming languages, frameworks, etc. and the improvement obtained in the skills that I already had.

But the more experience I have obtained is about the human profile. Working on a similar project during the Erasmus program for me has been as difficult as rewarding. Working on a similar project in their own country is a challenge, just imagine in a country where I had to learn two new languages, new cultures, go out every night (otherwise the friends did not stop ringing the intercom), cooking, washing clothes, clean the house, going to class at hours unproposable, ...

But mostly I had the pleasure of working with wonderful people like Jose Antonio, Juanvi and Luisa that always helped me, despite my obvious problems with the language and the basic differences between the UPC with my Italian university.

Thanks to all my friends of Barcelona and Menorca that, while I was working on the project, they sent me photos of all the beaches where they went every day.
8. BIBLIOGRAPHY AND RESOURCES

I. Sommerville - Software Engineering (6th Ed.) - Addison-Weisley

B. Bruegge & A.H. Dutoit - Object-Oriented Software Engineering - Prentice Hall

jQuery – Official jQuery Documentation
http://docs.jquery.com/Main_Page

CakePHP Cookbook v2.x – Official CakePHP 2.x Documentation

PHP Manual – Official PHP Documentation

PHPExcel Manual – Official PHPExcel Library Documentation
http://phpexcel.codeplex.com/documentation

CakePHP – Framework PHP
http://www.cakephp.org/

Symfony – Framework PHP
http://www.symfony-project.com/

CakePHP vs Symfony - Comparative between the two frameworks
http://stackoverflow.com/questions/1242060/symfony-vs-cakephp

SoftIcons – Images and icons repository
http://www.softicons.com