University of Mendoza
Faculty of Engineering
Bachelor's Degree in
Information Systems

"MVC CONTENT MANAGEMENT SYSTEM"

CREATING AN MVC CMS WITH COMPONENTS AND DRAG-AND-DROP MODULES AND A CACHING SYSTEM

Author: Borja Abad López
Student ID: 5544
Course professor: Eng. Osvaldo Marianetti
Year: 2013

ACKNOWLEDGEMENTS

The University of Mendoza was the third university I attended. On the previous two occasions, personal circumstances prevented me from completing my education. At 25, I decided to try once again, and, as the saying goes, third time lucky.

Now aged 30 and about to graduate, I want to encourage everyone who has been unable to complete their studies because of personal circumstances to do so: it is worth the effort.

My thanks to Dr Eng. Cristina Párraga, who encouraged me to enrol on the first day I visited the Faculty of Engineering's administrative office to ask for information.

I owe a special debt of gratitude to my parents for all their support and for teaching me the importance of education. Thanks also to everyone who has supported me: the lecturers, fellow students and friends who made this possible. Thank you.

TABLE OF CONTENTS

INTRODUCTION 7

ABSTRACT 9
Methodology 10
Progress to date 10

CHAPTER I - CONTEXT 11
PROBLEM STATEMENT 11
Questions 13
Objectives 14
Scope and limitations 15
Advantages 15
Feasibility 16
Facts and figures 17

CHAPTER II - THEORETICAL FRAMEWORK 21
BACKGROUND 21
THEORETICAL FOUNDATIONS 23
Basic functionality 23
Classifications 24
Frameworks and CMSs 24
Persistence 25
Caching 25
Templates and themes 26

CHAPTER III - APPLIED METHODOLOGY 27
Hypothesis 27
Type of study 27

CHAPTER IV - CREATING THE CMS 28
The application's foundations 28
Two applications 31
BACK END 33
The back end and its components 33
API - Creating a component 37
Components and controllers 39
Components and models 41
Components and views 43

Media 44
Messaging system 45
Creating the news component 47
Notes 51
WYSIWYG 51
Standardisation 51
Managing components in the back end 52
FRONT END 53
The concept of a page 55
The concept of a module 57
Implementing the front end 58
Direct access to a module 60
Module metadata passed to the page 62
Static modules 63
Dynamic modules 63
Custom modules 63
index.php 64
The caching system 65
On-demand caching 65

CHAPTER V - LESSONS LEARNT 68
ANALYSIS AND INTERPRETATION OF RESULTS 68
CONCLUSIONS 69
REMAINING WORK AND RECOMMENDATIONS 69

BIBLIOGRAPHY 71
OTHER RESOURCES 72
GLOSSARY 74
APPENDICES 78
External comparison of Joomla and WordPress 78
Putting dynamic content into perspective 79
Internal CMS response-time study 82
Internet searches 84
Questionnaires 86

INTRODUCTION

It all began with an idea: the MVC pattern. I have explored dozens of open-source codebases, and have always been interested in the way their files are structured, their methods are named and their variables are organised. What I always disliked was starting a new project with unfamiliar code, and the arduous process of unravelling its mysteries until I understood how it worked.

When I discovered the Model-View-Controller pattern, I realised that any project, however different it might be, could remain easy to understand if it followed this pattern, regardless of the programming language used.

The choice of topic for my final-year project is not accidental: it has a history. For years, I have worked for a company whose main product is a content management system, Editmaker, which it has been developing for 15 years.

This has helped me understand the problems and needs involved in developing software over the long term: software that evolves, grows and becomes more complex.

At university, I also learnt the importance of standardising and managing the communication involved in team software development. The unified software development process inspired by Microsoft's organisation, agile methodologies focused on productivity, communication and teamwork, and the need for patterns, abstraction techniques and other architectures that help structure or standardise the creation of a particular software product have all encouraged me to bring these ideas together, or at least attempt to do so, in this project.

Creating a CMS involves considerable complexity. These are substantial, fairly complex applications. Running dozens of instances, or websites, on a single server will almost certainly put a heavy strain on its performance.

The need for a lightweight framework led me to rule out the existing CMSs, which have evolved over many years and are very comprehensive, and to try building one from the ground up, with lightness as a central concern from the outset.

The existing systems have several things in common: they are written in PHP, use MVC and are mostly component-oriented.

I benefited greatly from following the courses of Jaisiel Delance, a Dominican online technology educator with a YouTube channel, Twitter account, Facebook profile and his own website, dlancedu.com. An expert in open-source CMSs, he has distilled their basic shared features into a genuinely lightweight and powerful foundation, which I value highly and have adopted as my starting point.

I believe a project of this kind should involve a team, as well as months of planning and development. Everything depends on the initiative's ambition. Although I am aware of this limitation, in this thesis I would at least like to establish the foundations of a sound architecture for a system that meets the stated requirements: scalable, lightweight and ready for teamwork.

ABSTRACT

This final-year project will be a record of the experiences, approaches, lessons and conclusions arising from the creation of a CMS's core architecture.

The aim is to develop an application with a back end for managing its front end. Its component-oriented structure will support team development.

The back end must be robust, reliable and scalable; its weight is less important. Component orientation means that, once the core has been established, any developer can add components through a clearly defined interface.

The front end, by contrast, must be simple and, above all, very lightweight, because it bears the greatest server load. It will also be module-oriented: pages are formed by combining modules.

The concept of a page will be explored in depth. A drag-and-drop system will allow pages to be created in the back end by placing dynamic, static or custom modules onto them.

The web server's output buffers will be used to implement an on-demand caching system based on the file system, storing rendered output as flat files. The aim is to demonstrate that this kind of system can improve performance without substantially increasing complexity or harming the user experience.

Methodology

Working alone rather than in a team has allowed me to save considerable time on communication through diagrams, system information and shared reviews. In large organisations, modelling tools such as UML, or production based on RUP, become essential: in practice, programmers sometimes have no idea what the software they are writing is for.

The PHP5 programming language, supported by a MySQL database, has been used to address the functionality and objectives set out in this thesis. This has been my main tool. Although my approach has been closer to Extreme Programming than to RUP, I have included several analysis and design diagrams.

The development has been mirrored on GitHub as its central version-control repository.

Progress to date

The starting point is Jaisiel Delance's basic MVC pattern. The project is defined as two applications: the front end and the back end.

I begin with the back end, adding a layer of abstraction through components. A few basic components are defined, together with an interface for creating them.

In the front end, I introduce the concept of a page and, therefore, of a module. A drag-and-drop library is used to create pages in the back end. It is adapted to the system and extended to support dynamic modules. Finally, the page concept provides the basis for an on-demand caching system.

CHAPTER I - CONTEXT

PROBLEM STATEMENT

Most small and medium-sized software producers develop small- and medium-scale systems and projects.

There are software solutions for almost everything on the market. In content management, some are so widespread that many of the websites we regularly visit run on a CMS. Some specialise in publishing, others in audiovisual content, while others serve bloggers or classified advertising.

CMSs are clearly a growing trend. Open-source systems become more modular with each release: functionality can be added or removed, and communities form around them, contributing or reusing pieces of code to provide solutions for millions of businesses and individuals.

Today, every small or medium-sized business knows that it needs to organise its information using reliable computer systems. As the web grows, businesses also want to manage that information online.

I recognise that the existing CMSs are giants in terms of the amount of code developed, person-hours invested, support communities and, above all, the number of live websites that use them.

These CMSs are highly customisable and can be adapted to almost anything. They also provide a well-defined interface for adding bespoke modules, components or plugins to meet particular needs.

From a developer's perspective, using these tools to build web-based systems is undoubtedly positive and advantageous.

From a software producer's business perspective, however, the picture changes. Adopting these tools as the foundation for medium- and long-term development can become a burden, with many potential risks and problems:

1. CMSs have evolved into very comprehensive, and therefore complex, products. They may fit a particular website project well, but larger systems may require changes to their behaviour. Modifying software of this scale could demand considerable resources.

2. The core of the resulting application would depend on third parties, a severely limiting inconsistency for a software product. Its producer would have to update systems according to another organisation's timetable, and any obsolescence imposed by that third party would affect the producer using the software. In short, its release cycle would be dependent on someone else, or it would remain tied to an obsolete version.

3. These applications consume substantial resources on their host servers. Deploying many instances would therefore produce high levels of

resource consumption, even when their business logic is simple or the number of requests is low.

4. They attract cyberattacks. Because their code is open, anyone can study and understand it. Individuals and organisations with different aims systematically compromise these systems, forcing their producers to release security patches and new versions. Releases do not always add improvements: some simply fix bugs.

Questions

What software should a developer of web-based systems use as the foundation for their projects?

Is it appropriate to build a bespoke system rather than use one already available on the market?

Can one system provide the foundation for almost all of a software producer's web projects?

None of these questions has a definitive answer. Everything depends on who answers, and on their experience and needs. The appendices include responses from several members of the development community.

Objectives

In this project, I will try to establish whether bespoke software built from the ground up can offer architectural advantages comparable to those of the leading CMSs on the market.

Main objective:

Present a broadly applicable structure for creating a software producer's own content management foundation. Develop a visual page-building interface in the back end that differs from existing products and aims to be more natural and intuitive: configuring a page and the content it can contain, rather than specifying which content appears on which page.

Additional objectives:

Implement a simple on-demand caching system, rather than a time-expiry cache.

Use components in the back end to encapsulate functionality.

Use modules in the front end to render pages.

Retain the MVC structure throughout the application.

Define interfaces for creating modules and components.

Scope and limitations

The application's functionality will be very limited. Only a small number of components, modules and application functions will be implemented.

Resource constraints mean that the software resulting from this study will not be particularly complex. With continued work, however, it could become a software foundation or framework for small and medium-sized software producers.

Advantages

When I compare the advantages of software developed in-house with those of the leading market products, I mean their main architectural advantages:

1. They are usually component-oriented, encapsulating functionality and allowing components to be added or removed as required. This also supports teamwork.

2. They include caching systems, all based on time expiry.

3. They are highly scalable and configurable.

4. Their design implements the MVC pattern.

5. They manage styling through themes or templates.

Feasibility

If the principal architectural advantages shared by existing CMSs can be combined in an in-house product that supports almost all the projects a small or medium-sized software producer might undertake, the result could be a powerful, adaptable tool for growing the organisation.

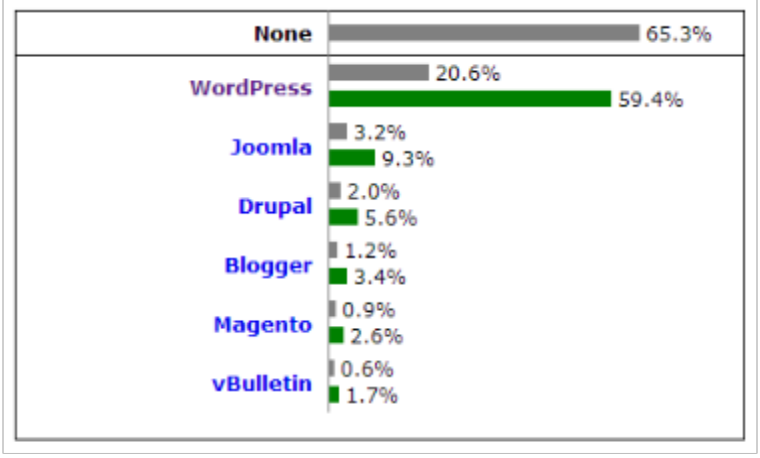
Facts and figures

CMS market penetration

According to W3Techs[1], 65.3% of the ten million websites analysed do not use a content management system.

If CMSs account for around 35% of all websites worldwide, the importance of these tools is immediately apparent.

The following chart shows WordPress, Joomla and Drupal leading the market. WordPress alone has a remarkable worldwide presence. Green indicates each system's share among CMS-based websites, whereas grey indicates its share of all websites. To say that WordPress powers more than 20% of the world's websites is therefore a significant statement.

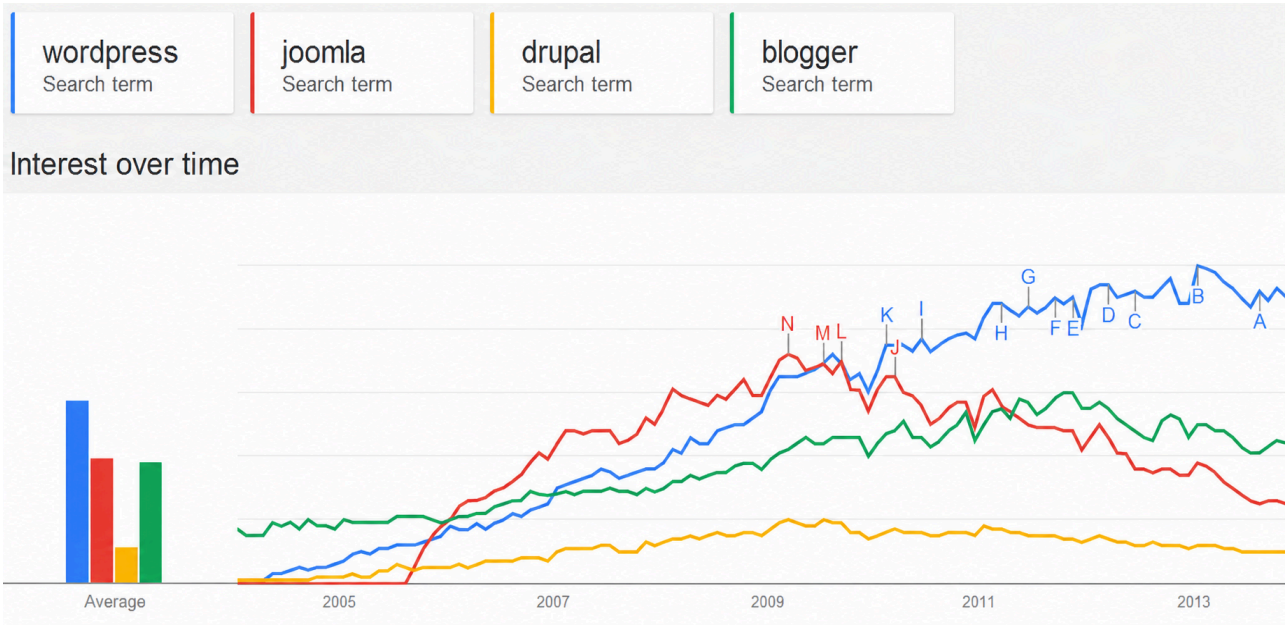


Fuente: http://w3techs.com/technologies/overview/content_management/all

1 The website w3techs.com provides information about the use of different web technologies. Although it is a private company, like most organisations driving the internet, it promises accurate information and states that it has no affiliation with the technology suppliers in its reports. It examines website technologies rather than individual pages. This study includes the ten million most-visited websites. Unlike Alexa, acquired by Amazon and a leading provider of global web traffic and usage statistics, W3Techs treats subdomains as part of the same website rather than as separate sites.

Trends in interest in CMSs

Other data further illustrate the scale of these applications and their place in society. Trends in searches for relevant keywords provide another perspective. Google Trends[2] is used to draw the conclusions shown in the following chart.

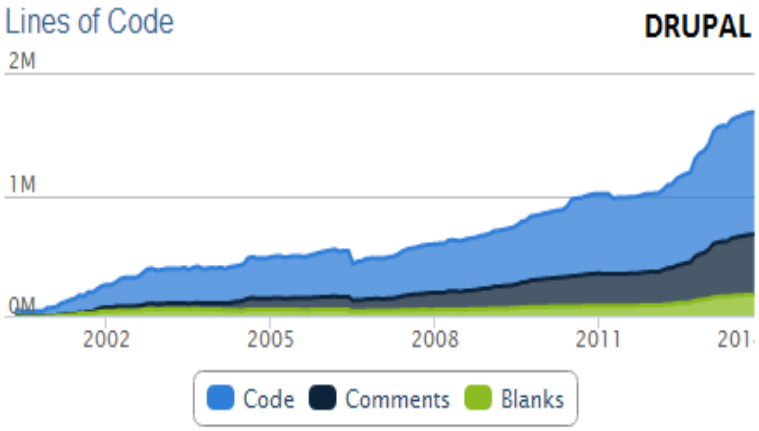


Although Joomla shows a downward trend, all the systems have grown from their values at the chart's starting point in 2005. CMSs are a pillar of the modern web.

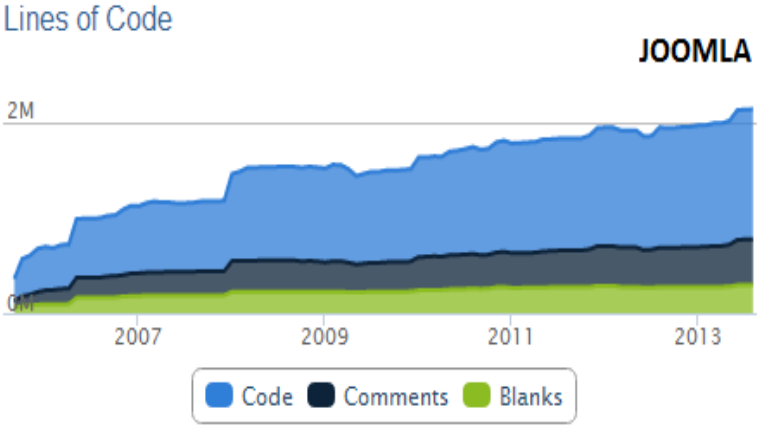
2 Google Trends is a Google Inc. tool that shows search-term trends and how they change over time, using information gathered by its search engine. Google is treated as the reference search engine for this study.

CMS performance

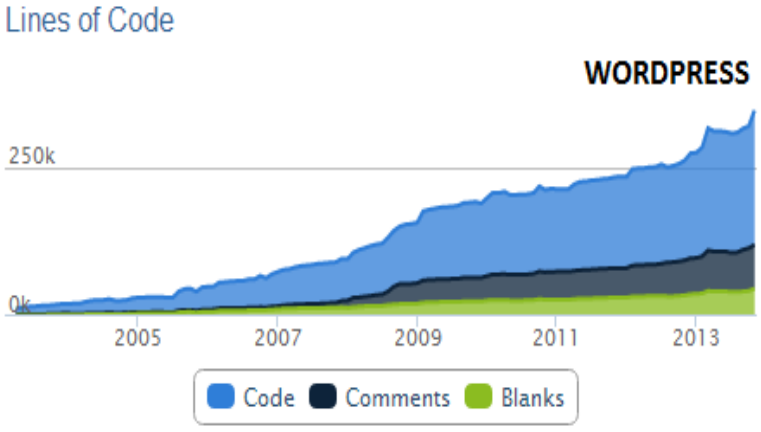
CMS codebases have undoubtedly grown enormously. For some projects, they can become unwieldy.



The tool provided by Ohloh.net[3] allows us to draw some interesting conclusions.



All the CMSs have substantially increased their line counts in recent years, further demonstrating that they are becoming more comprehensive and therefore heavier applications.



These charts do not include contributions from the wider community: they account only for the company developing each CMS's core.

3 Ohloh.net is a website for comparing open-source projects. It uses trackers to examine the public repositories hosting those projects and obtain information such as contribution counts and lines of code.

CMS security issues

Searching google.es for “hack joomla” produces more than three million results. Hundreds of posts describe security vulnerabilities and explain how to reproduce them.

Searching for the Spanish phrase “parche seguridad joomla” (“Joomla security patch”) produces almost 30,000 results. Hundreds of links lead to security patches for different versions.

Caching

The main problem I have identified in market-leading CMSs is that they all use time-based caching. On news websites, where information changes rapidly and articles are often edited live, caching can cause visitors to see outdated content.

CMSs do provide a way to clear their caches, but this is a manual process that depends on user training and therefore often causes problems.

CHAPTER II - THEORETICAL FRAMEWORK

BACKGROUND

A content management system, or CMS, is software for storing, editing and publishing information. CMSs are usually web applications, although they can also be used on an intranet.

They allow large numbers of people to access and communicate through the internet without technical knowledge such as programming skills.

CMSs emerged in the late 1990s as the internet became popular, meeting the new needs it created. Major print publishers began launching web editions and soon realised that they needed systems for updating content continuously.

The boom arrived with Php Nuke[4]. More people gained internet access, low-cost hosting plans appeared, and everyone wanted to communicate and establish an online presence.

Although newspaper and magazine publishers pioneered the first CMSs, their use has spread across every sector. These tools are now a pillar of the internet.

They are used worldwide by all kinds of companies, individuals, universities, public bodies and NGOs.

4 Php Nuke was the most popular early CMS and the first to build a large community. Others followed, including Mambo, from which Joomla emerged.



Logos de CMS´s populares.

There are hundreds of varieties. Many, including the most popular, are free, while others are proprietary. They include sector-specific, standardised and bespoke CMSs, using different technologies for different purposes.

THEORETICAL FOUNDATIONS

Basic functionality
Following the framework proposed by James Robertson[5], any CMS should be divisible into four basic areas of functionality:

1. Creation: all the forms for entering information and the processes behind them, whether adding a news article, video, gallery or other content.

2. Management: all created information must be manageable: it must be possible to edit, delete or modify it. Some structural elements, text and pages appearing on the website must also be editable.

3. Publication: items must be publishable and unpublishable. CMSs may manage this by dates, editions or other criteria. Unpublished information is archived and remains available when needed.

4. Presentation: the website's framework and structure in which content appears. For example, a CMS may provide separate mobile and desktop versions.

5 James Robertson is the founder and Managing Director of Step Two Designs, an Australian company specialising in an intranet CMS. He describes the company as a content management specialist, is a founding member of the Content Management Professionals Association, and has helped many organisations select a CMS suited to their needs.
http://www.steptwo.com.au

Classifications

By programming language, most are written in PHP, although many use ASP, Java, .NET, Ruby, Perl or Python.

By licence, they can broadly be divided into open-source and proprietary products.

By purpose, there are CMSs for blogs, forums, publications, shops, social networks and other uses.

Frameworks and CMSs

A framework provides a foundation on which to build an application. Like a mechanic's toolbox, it helps developers create applications by providing libraries, functions and other solutions within the development environment.

Frameworks provide ready-made functionality, but do not themselves provide a website. Some have evolved towards capabilities close to those of a CMS, although the concepts remain very different.

A CMS is more than a framework and is likely built on a bespoke one. A CMS could be described as an applied framework.

Some CMSs can themselves be used as frameworks because their well-defined interfaces allow programmers to build applications as extensions of the CMS.

Persistence

As the internet has developed, site builders have appeared that make it quick and easy to create websites from styled templates for static content. In some cases, these could be considered CMSs, even if they do not use databases and offer only very simple content management.

CMSs have traditionally been associated with databases for persisting information. In addition to classifying data, they must allow it to be accessed freely. Many database engines are available, and the choice depends on the CMS using them.

Configuration files may sometimes be used to persist or share particular settings. Images and other visual resources are normally stored in the web directory.

Caching

Caching primarily saves server processing. Each page request triggers many operations: executing server scripts and their variables, functions and calls; accessing physical file-system resources; querying the database; allocating memory; and so on.

The principle is simple. A cache takes the result of the operations required to handle a request and stores it as a plain-text file in the file system. When the same request arrives again,

it reuses the previous result, avoiding all that processing.

Templates and themes

One feature all CMSs share is the use of templates or themes.

A template is the container in which content is rendered. Together, the template and content form the interface or response generated for a user's request.

A template can accommodate content produced by the CMS.

Templates are associated with structure. Changing a template can therefore change the visual arrangement of content as well as its styling.

Themes are more limited: they affect styling without changing structure. With a theme, the visual organisation of content remains constant.

CHAPTER III - APPLIED METHODOLOGY

Hypothesis

A developer of web-based systems should be able to build bespoke software instead of selecting an existing market product: one web system that underpins almost all their projects, visually organises content, uses a cache without update errors, and provides a component system that supports teamwork.

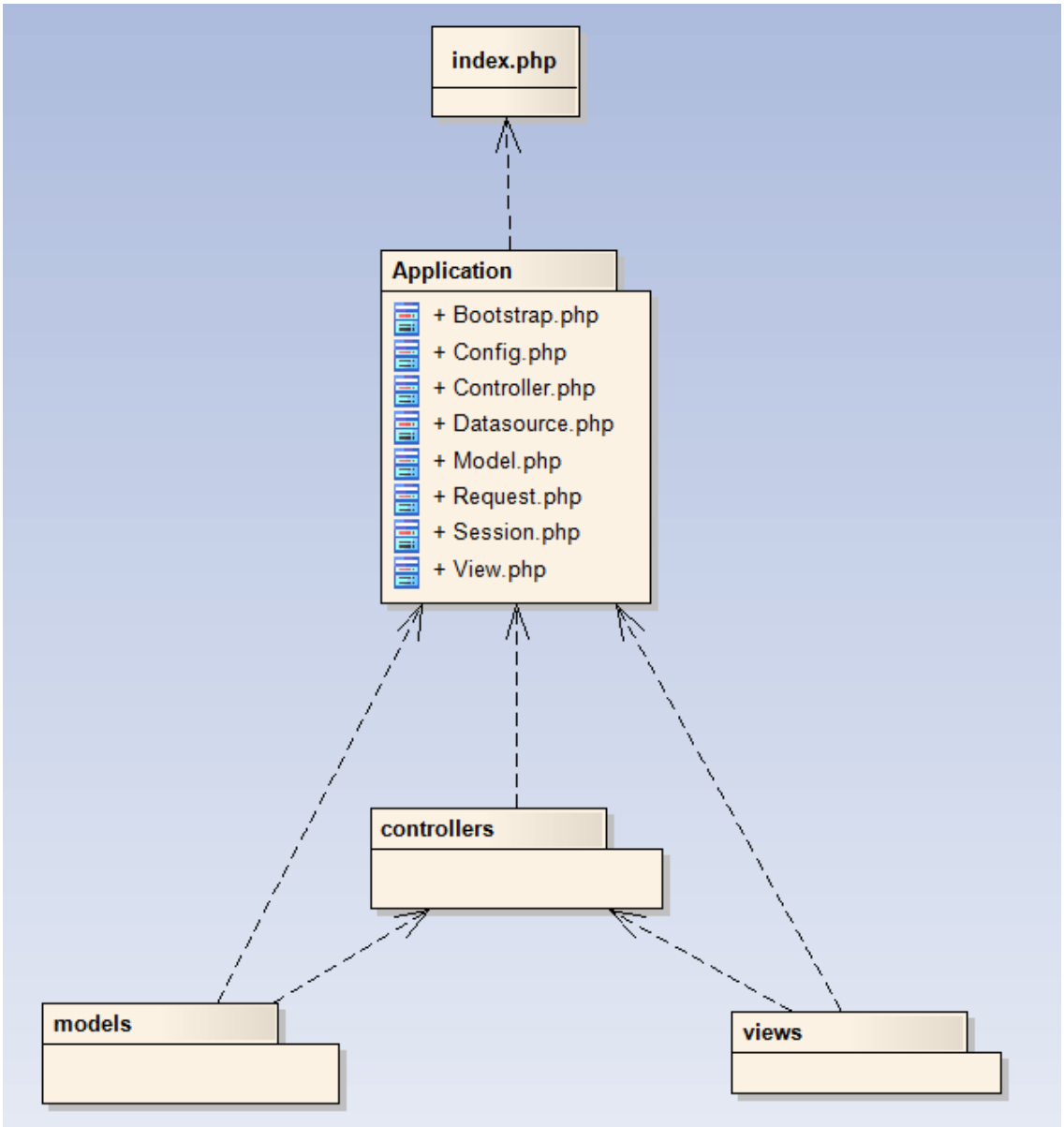
Type of study

This is a proposal-based study. I will formulate a concrete response to the stated problem by creating the CMS's basic architecture.

The study is presented in detail in CHAPTER IV.

CHAPTER IV - CREATING THE CMS

The application's foundations
As noted earlier, I will begin with Jaisiel Delance's PHP implementation of the MVC pattern, whose structure is summarised in the following diagram:



The implementation of the Model-View-Controller pattern is clearly visible.

Let us examine the website's index.php file:

```
<?php
	try{
		define('DS', DIRECTORY_SEPARATOR);
		define('ROOT', realpath(dirname(__FILE__)) .
DS);
		define('APP_PATH', ROOT . 'application' .
DS);



	require_once APP_PATH . 'Config.php';
			require_once APP_PATH .
	'Request.php';
			require_once APP_PATH .
	'Bootstrap.php';
			require_once APP_PATH .
	'Controller.php';
			require_once APP_PATH .
	'Model.php';
			require_once APP_PATH .
	'View.php';
			require_once APP_PATH .
	'Session.php';


Bootstrap::run(new Request());
	}
	catch (Exception $e){
		$e->getMessage();
	}




?>
```

First, it defines constants that are accessible throughout the application. It then adds further constants from config.php. The require_once statements load the application so that its code is available.

Bootstrap does the work of launching the application. It receives the Request and uses it to instantiate the specified controller, invoking the requested method with the supplied arguments.

URL format in the basic MVC pattern

```
BASE_URL/controller/method/arg/s
```

If no method is supplied, index is used. If no controller is supplied, indexController is used.

Bootstrap's power comes from the following function, whose parameters are provided by the Request:

```
call_user_func_array(array($controller,$metodo), $args);
```

When a page is requested, Request assigns its controller, method and arguments, and Bootstrap instantiates the controller.

Following MVC, the controller then accesses the models and passes their data to the views, which produce the Response for the user.

Another difference from more traditional systems is that the entire website always passes through index.php. It acts as a funnel, channelling all the code through a hierarchy. Everything goes through index.

This approach is very practical. Two applications with completely different business models can retain similar file structures, organisation and functions.

Any programmer who understands the pattern will know how the application is organised. The familiar structure greatly reduces the time needed to search the codebase and understand it.

Moreover, a very lightweight foundation of barely 100 lines of code can support reasonably scalable websites.

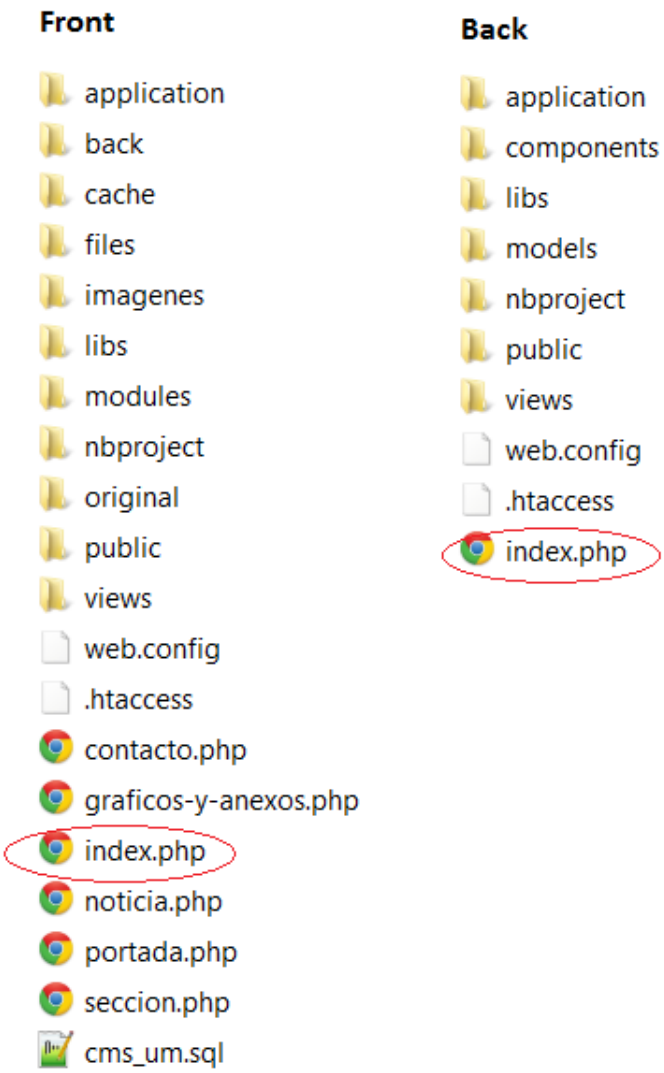
Taking this concept as the essence of a web application, I will try to extend it to achieve the stated objectives.

Two applications

As we know, CMSs are highly complex structures.

Breaking a large problem into smaller problems is, I believe, fundamental to computing. Abstracting and encapsulating functionality allows information systems to evolve, building on what has already been established.

I therefore consider it sensible to isolate the front end from the back end. Applying the proposed PHP MVC pattern to an application with both a public website and an administration area for editing its information would otherwise create a problem.



Front-end and back-end controllers would be mixed together, as would their models and views. Shared models, for example, would share all their methods. A front-end request for a news view would therefore load not only the method that supplies that view's data, but also methods for CRUD operations: creating, reading, updating and deleting records, or other operations specific to the back end.

That makes little sense. It would reduce website performance and make the code more confusing.

I will adapt the proposed MVC pattern separately for the back end and the front end. Two index.php files will therefore load two different applications.

BACK END

The back end and its components

The back end will be the more substantial part of the system. It does not matter if it loads many libraries or files: content changes far less often than it is accessed. See the appendix, “Putting dynamic content into perspective”.

The main organisational abstraction proposed for the back end is the component. A component retains the essence of MVC, like the initial example, but within a framework with additional layers. It has its own views and models and, when necessary, can access other components’ models, shared libraries and the loaded application.

Organising code into components is very useful when building complex, scalable systems.

Additional abstraction layers bring advantages, but may also require more code, files and general framework “bureaucracy” before work can begin. The more clearly the architecture is defined, the more rules and interfaces developers must follow.

Even so, in this context the advantages clearly outweigh the drawbacks:

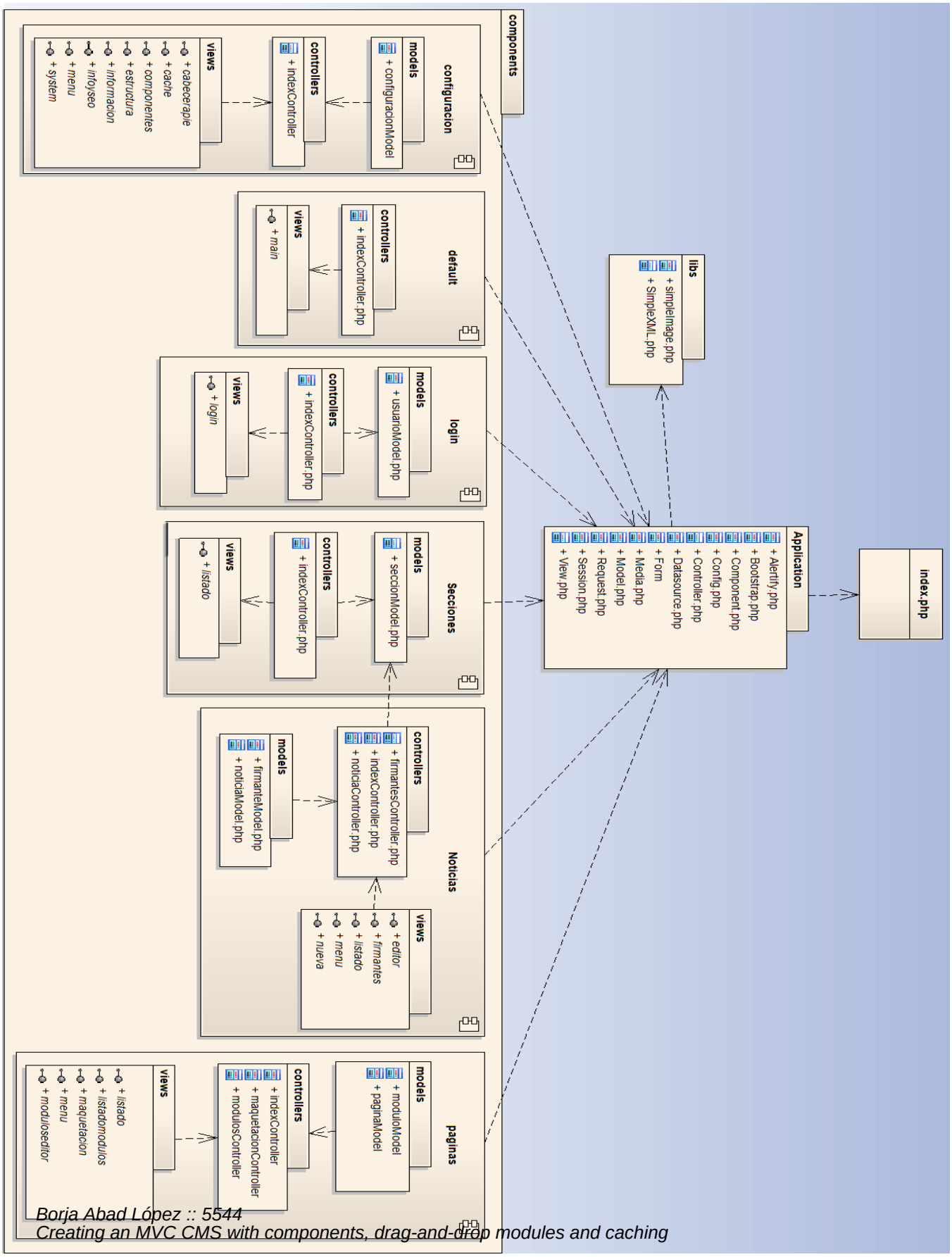
- Each component behaves like a small application in its own right, following the environment’s rules while encapsulating its own functionality.

• Teamwork is supported because different team members can develop different components independently, extending the system in parallel.

• Components can have independent release cycles. The front end can be updated without affecting the back end, and individual components can be updated without destabilising the rest of the application.

• Focus is retained. If the business model is to develop a particular kind of software, customer requests need not turn the application into a succession of patches. Bespoke components can be developed, then enabled or disabled as required.

• Components can be reused. A bespoke system can be assembled by enabling or disabling components. Two customers' needs may seem very different, but their technological and software solutions will share many features. The aim is to reuse comparable functionality that is not part of either customer's core business model.


index.php
Application
+ Alertify.php
+ Bootstrap.php
+ Component.php
+ Config.php
+ Controller.php
+ Datasource.php
+ Form
+ Media.php
+ Model.php
+ Request.php
+ Session.php
+ View.php
libs
+ simpleImage.php
+ SimpleXML.php
components
configuracion
models
+ configuracionModel
controllers
+ indexController
views
+ cabeceraple
+ cache
+ componentes
+ estructura
+ informacion
+ infoyseo
+ menu
+ system
default
controllers
+ indexController.php
views
+ main
login
models
+ usuarioModel.php
controllers
+ indexController.php
views
+ login
Secciones
models
+ seccionModel.php
controllers
+ indexController.php
views
+ listado
Noticias
models
+ firmanteModel.php
+ noticiaModel.php
controllers
+ firmantesController.php
+ indexController.php
+ noticiaController.php
views
+ editor
+ firmantes
+ listado
+ menu
+ nueva
paginas
models
+ moduloModel
+ paginaModel
controllers
+ indexController
+ maquetacionController
+ modulosController
views
+ listado
+ listadomodulos
+ maquetacion
+ menu
+ modulosedittor

The preceding diagram summarises the proposed back-end architecture.

Compared with the initial MVC diagram, Application loads more functionality, libraries have been added, and components are organised as small MVC applications sharing an environment. They can interact: for example, the news component imports the sections model from another component. This approach offers the advantages already discussed.

API - Creating a component

Interfaces for implementing components

New component

Add the component to the list in components.xml, located at the root of the components directory:

https://github.com/borjaabad/cms.um

The global $components variable makes this file's contents available anywhere in the code.

Constants defined by the framework's Component.php class when a component is loaded, and available throughout the application:

NAME_COM	name defined in components.xml (required)
ROOT_COM	Physical path to the component's root directory
DISPLAYNAME_COM	displayName defined in components.xml
MENU_COM	menu defined in components.xml
SQLPREFIX_COM	sqlprefix defined in components.xml
ENABLED_COM	enabled defined in components.xml

Each component has the following default directory structure:

ROOT.DS.'components'.DS.'component'.DS.'controllers'

ROOT.DS.'components'.DS.'component'.DS.'models'

ROOT.DS.'components'.DS.'component'.DS.'views'

ROOT.DS.'components'.DS.'component'.DS.'public'

To protect a component using sessions:

Session::control();

Call this in the constructors of the component’s controllers.

If the user is not logged in, return to BASE_URL.

Components and Controllers

The complete default URL structure is as follows:

```
BASE_URL/component/controller/method/arg/s
```

And the paths:

```
/back/components/controllers/controller.php

/back/components/models/model.php

/back/components/views/view.php
```

Example request: https://borjaabad.com/back/login

```
/back/login/controllers/indexController.php

/back/login/models/usuarioModel.php

/back/login/views/index/login.phhtml
```

Note: in the previous example, the view is nested one directory deeper, in a directory named after the controller that invokes it.
If the component controller does not exist, the method is looked up in indexController:

/back/component/controller/method/arg/s

This allows:

/back/login/index/validar/arg/s

To become:

/back/login/validar/arg/s

Components and Models

To load a model belonging to our component from its controller:

```
$modelo = $this-> loadModel('modelo');

ROOT_COM.DS.'models' *

    (*) ROOT_COM refers to the component's path.
```

To load a model belonging to another component from the controller:

```
$modelo = $this-> loadModelFromOtherComponent('modelo','componente');
```

The database connection is not loaded by default, since not every model needs data access. To give a model access to the database, put the following code in its constructor:

```
public function__construct() {
parent::__construct();
$this->cargaBD();
        }
```

The database could be loaded in any other controller method; it need not be in the constructor. It all depends on the requirements.

Example of a method using PDO:

```
public function getSecciones(){
            try {
                $secciones = $this->_db->query("SELECT * FROM
secciones")


        return $secciones->fetchAll();
                }
                catch (PDOException $e) {
                    echo
$e->getMessage();
                    return false;
                }
            }
```

Components and Views

Each component's views can access various framework features, such as the messaging system and the Media class.

To call a view from our component's controller:

```
$this->_view->renderizar('vista');


$this->_view->renderizar('vista',true);
      La bandera true agrega el header y footer por defecto de la
aplicación:


Rendering a view:
ROOT .DS . views .DS . layout .DS . default .DS . header.php
ROOT .DS . components .DS . component .DS . views .DS . view . DS . vista.phtml
ROOT .DS . views .DS . layout .DS . default .DS . footer.php




To send information to views, assign variables in the controller as follows:
$this->_view->_variable = 'foo';


To receive information in views, read the variables as follows:
echo $this->_variable;
```

Media

The Media.php class, loaded in the application core, was created to distribute and include public files in our views.

```
<?php Media::css(); ?> // Loads the framework's base CSS files

<?php Media::js(); ?> // Loads the framework's base JS files

<?php Media::addCss('estilos.css'); ?> // Adds view-specific CSS

<?php Media::addJs('main.js'); ?> // Adds view-specific JS

Default path:

  * components/component/views/public/css/archivo.css

  * components/component/views/public/js/archivo.js

The framework's base files should preferably be loaded before the </head> tag.
```

JS and CSS can also be included through the components.xml configuration file.

Messaging system

Using the alertify.js and alertify.css library to send messages from code to our views, the Alertify.php class was created in the application directory, the framework core. It manages messages from any component's views.

To display messages in a view:

```
<?php Alertify::mostrarMensajes();
?>
```

Use this code in view templates; the only requirement is that it appears before the closing </body> tag.

This code is already implemented in the default footer.php.

To display messages in a view:

```
<?php Alertify::add($mensaje,$tipo);
?>
```

Use this code in view templates; the only requirement is that it appears before the closing </body> tag.

The possible message types are:

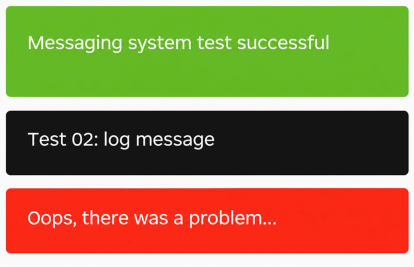
'success' - 'log' - 'error'

```
Alertify::add('Messaging system test successful','success');

Alertify::add(('Test 02: log message','log');

Alertify::add(('Oops, there was a problem...','error');

Alertify::add(('login');
```



Creating the news component

This new component will be built on the framework's basic structure, which already includes login, messaging (alertify), media management (JS and CSS), form helpers, menu management and so on. It will allow news articles to be added, edited and deleted in the system.

First, add the component to components.xml so that the framework has the information it needs about it:

```
<component>
		<name>noticias</name>
		<displayName>News</displayName>
		<menu>true</menu>
		<menus>
			<menuItem>
			<name>main</name>
			<path>noticias/views/common/menu_main.phtml</path>
			<default>true</default>
			</menuItem>
		</menus>
		<sqlprefix>not_</sqlprefix>
		<enabled>true</enabled>
	</component>
```

The name is used to refer to it in code and as a unique identifier. The menu option controls whether it appears in the application's main menu.

The menuItem option lets us specify the menus our component will use. They can then be selected from the controller: simply set the MENU constant to the name of the menu to load.

Basic files:

Controllers	Models	Views	Tables
index	noticia	listadonoticias	noticias
firmante	seccion (external)	editor	firmantes
	firmante	listadofirmantes	
		menu_componente	

Let us examine the developing contents of our news component's main controller:

```
<?php

class indexController extends Controller{

	public function __construct(){
		parent::__construct();
		Session::control();
		}



	//noticias/
		public function index(){
		$this->_view->renderizar('listado',true);
		}



	//noticias/nueva
		public function nueva(){
		$firmante =
$this->loadModel('firmante');
```

```
$seccion = $this->loadModelFromOtherComponent('seccion','secciones');


$this->_view->_secciones = $seccion->getSecciones();
        $this->_view->_firmantes =
$firmante->getFirmantes();
        $this->_view->renderizar('nueva',true);
        }
}
?>
```

The constructor checks the session. If the check succeeds, the default method is index(), which calls the listado view.

The nueva() method is more interesting: it loads authors through a model belonging to this component, and can also load models from other components, as with the section component. This demonstrates a clear relationship between components.

Here is an example of firmanteModel.php:

```
class firmanteModel extends Model{
        public $id;
        public $nombre;
        public $email;



        public function __construct(){
                parent::__construct();
                $this->cargaBD();
        }



        public function getFirmantes() {
                try {
                    $firmantes = $this->_db->query("SELECT * FROM firmantes") or die(mysql_error().mysql_errno());
                        return $firmantes->fetchAll();
                }
                catch (PDOException $e) {
                        echo $e->getMessage();
                        return false;
                }
        }
```

```
}
```

The resulting view is called, for example nueva:

```
<form method="post" action="<?php echo BASE_URL.'noticias/nueva'; ?>">
<table>
	<tr>
	<td>
	<?php Form::select($this->_firmantes,'id','nombre','Authors'); ?>
	</td>
	<td>
	<?php Form::select($this->_secciones,'id','seccion','Sections'); ?>
	</td>
	<td>Order
	<input type="text" name="orden" size="5" maxlength="9"/>
	</td>
	<td></td>
	<td>
	<?php Form::button('Save article','positive','add'); ?>
	<?php Form::button('Expand article','regular','blue'); ?>
	<?php Form::button('Delete article','negative','delete'); ?>
	</td>
	</tr>
	</table>
	<h2>Title</h2>
	<textarea rows="1" name="texto"></textarea>
	<h2>Subtitle</h2>
	<textarea rows="1" name="subtitulo"></textarea>
	<h2>Lead</h2>
	<textarea rows="4" name="entradilla"></textarea>
	<h2>Body</h2>
	<textarea rows="15" name="texto"></textarea>
</form>
```

This illustrates another framework feature: the Form class helps render form elements linked to data sources, such as select lists, as well as elements without a data source, such as buttons, to which it simply adds images and styling.

NOTES

WYSIWYG
Combining the power of major CMS platforms with that of major HTML editors is certainly a considerable challenge. WYSIWYG, short for 'what you see is what you get', is more than an acronym: it represents an important goal for the user experience. It enables users to become familiar with the system intuitively and confidently.

There are WYSIWYG programs for creating websites, including well-known examples such as Adobe Dreamweaver. There are also modular editors that developers can incorporate to improve user data entry.

In this project, I use WYSIWYG in a comprehensive sense, referring both to individual user inputs (modular editors) and to the layout of dynamic, static and structural content. A drag-and-drop facility will make it possible to arrange web pages quickly and intuitively.

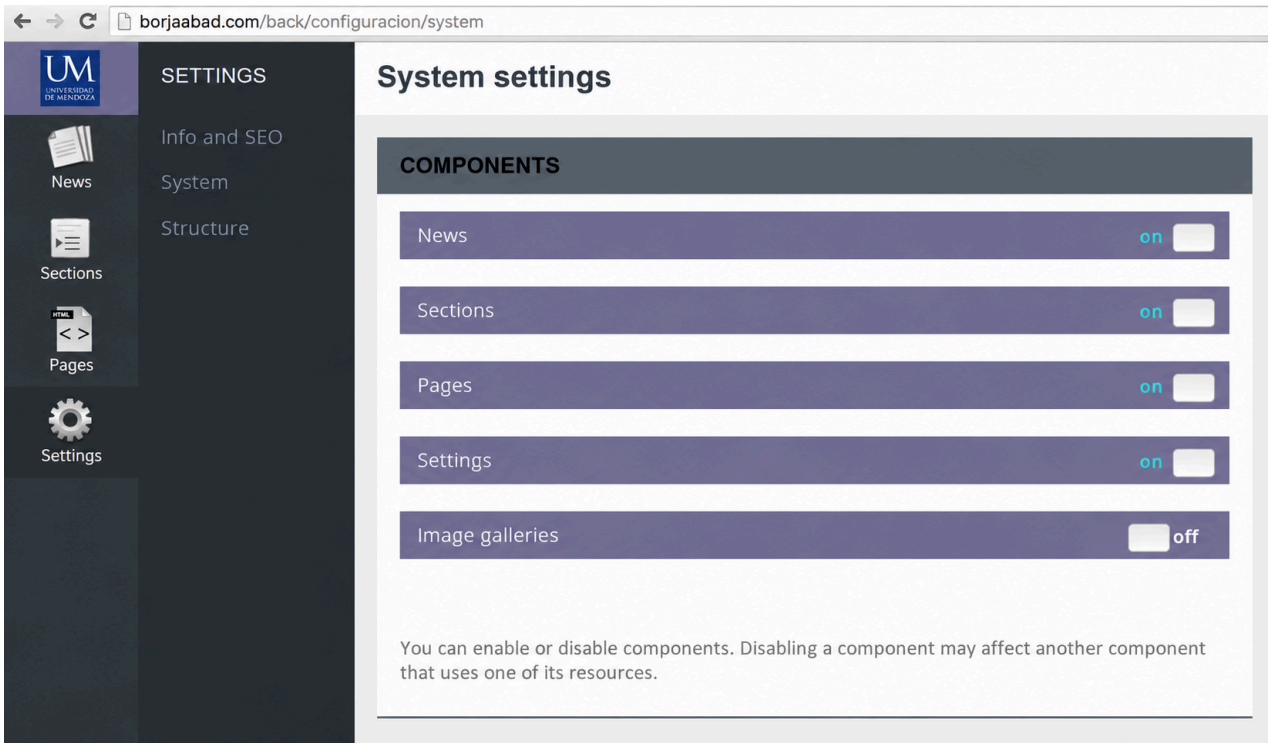
From a social perspective, I believe that continuing along these lines could make these tools more accessible to people.

Standardisation
The system can be deployed on Unix, MacOS and Windows servers. It can run on Apache, IIS and any other web server capable of interpreting PHP.

URL rewriting must be taken into account, because friendly URLs depend on functionality provided by the web server handling the request. Apache requires a .htaccess configuration file, while IIS requires web.config.

The application includes both configurations, since they are compatible and can coexist.

Managing components in the back end



Components can be added or disabled as required. A disabled component will not be loaded into the system.
This also makes it possible to implement user access control lists (ACLs) and configure access by component. An administrator, for example, would have access to all components, while a journalist would only have access to news and sections.

FRONT END

The CMS comes with news, sections, pages, default, login and configuration components installed in the back end.

The front end uses modules to present units of information, whether associated with a component or not, and pages act as their containers.

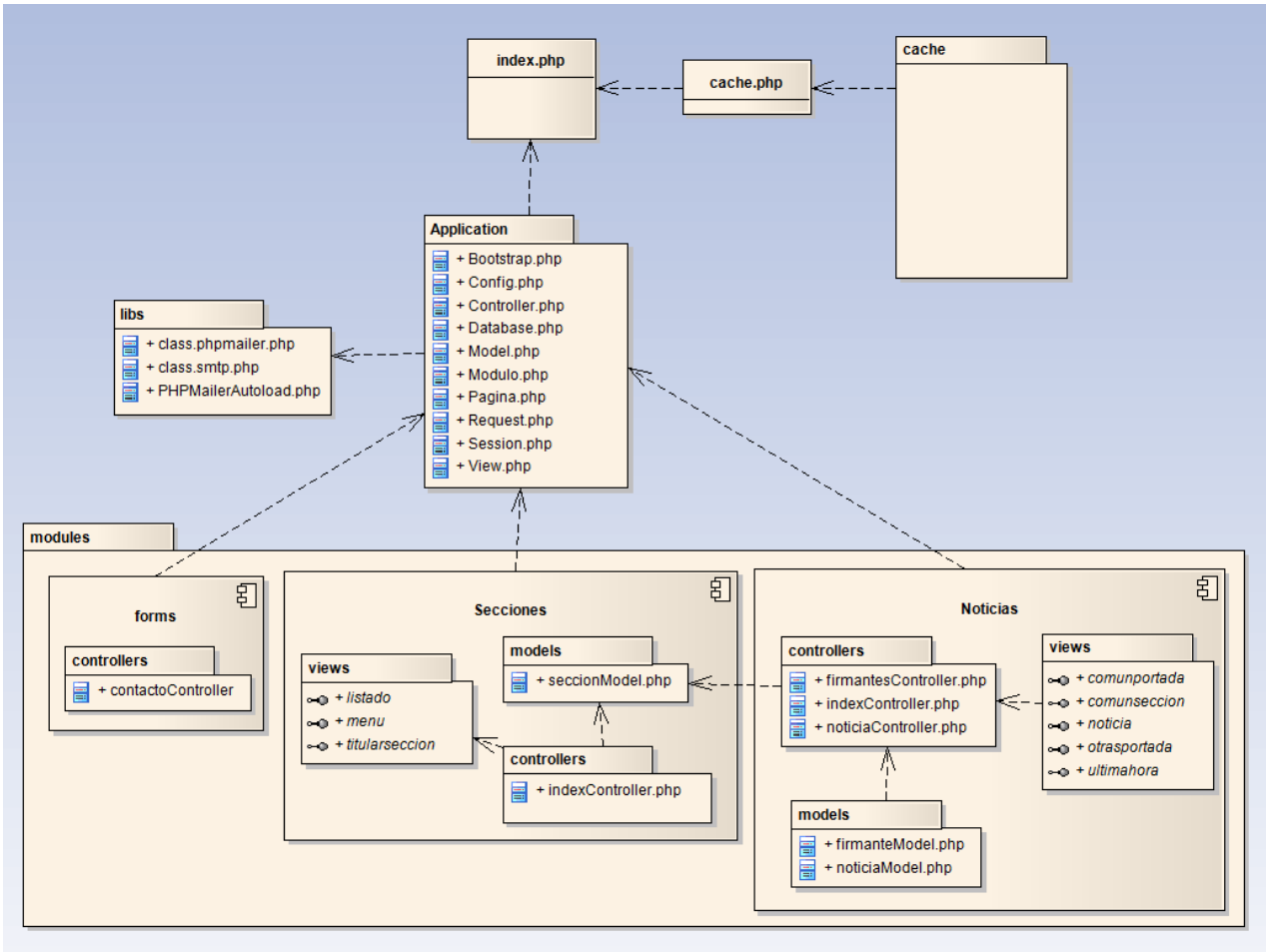
Assuming the software is used in a publishing environment, content would be organised into news articles and sections. There would also be a page for every section and every article entered into the system.

Creating a section should not require configuring a new page and specifying its modules. Instead, the section page should be understood as a container for all sections.

The same applies to news: the article page would encompass all articles.

Other pages will probably also be needed to display static content, modules from other components, or a mixture of content, as on the home page, index, which includes modules from several components.

Basic front-end architecture



The page concept

Most content management systems on the market use a system very similar to the one described, including modules, category management and organisation into components.

One task that remains complicated in every system I know is placing particular content at a particular position, a very common requirement for site administrators.

In Joomla, for example, front-end content is configured through modules: each module's scope is defined by specifying the pages and positions where it should appear, together with the privilege level if an ACL system is in use.

One aim of this final-year project is to outline a system that, unlike the others, configures front-end content through pages rather than through modules.

The point is to configure pages by choosing which modules they display and how, instead of choosing where each module appears. It may seem a minor distinction, but in my view it is a major step forward in managing and organising content.

Pages are the final products or views of our application: they form the front-end output.

To simplify matters, we view the whole front end as a collection of pages. The first argument in the URL therefore identifies the page. Here are some examples for our publishing scenario:

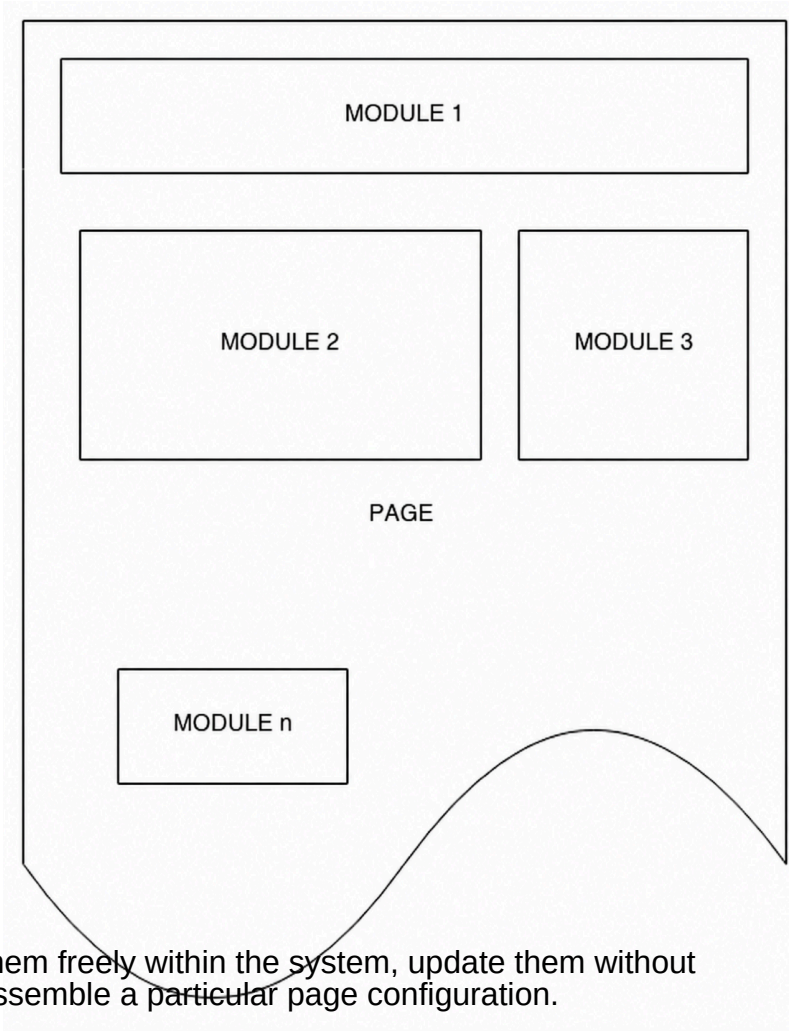
Page	Example URL
Home	https://borjaabad.com
About us	https://borjaabad.com/about-us
Privacy policy	https://borjaabad.com/privacy-policy
Contact	https://borjaabad.com/contact
Section	https://borjaabad.com/section/3/culture
News	https://borjaabad.com/article/123/exhibition-well-received

To summarise:

https://borjaabad.com/pagina/arg/s/

The module concept

Modules are the building blocks of our site. They encapsulate blocks of content that we embed at different positions on a page.



A component can have many modules. The news component, for example, could have a module instance showing the detailed view of a particular article, another showing breaking news, another showing the most-read articles, and so on.

Encapsulation allows us to move them freely within the system, update them without compromising the application, or assemble a particular page configuration.

A page will normally have a main module. The page's description, title and other values should be set by that module, which represents the page's underlying purpose.

Front-end implementation

Unlike the back end, the front end will use modules rather than components.

Each module will be based on the MVC architectural pattern.

The website's index.php receives the URL's GET request and calls the relevant page based on the first argument.

Our index.php will be relatively simple: it includes the requested page. The page contains the logic and calls all its modules, which implement the MVC model. Each module inherits a run method, which the page uses to create it.

The system analyses the URL in the request. If the first argument identifies an existing page, it includes and interprets that page.

PAGE → LOAD MODULES → HTML RESPONSE

```
public function procesarPlantilla(){

   $plantilla = file_get_contents($this->rutaPlantilla);
                preg_match_all('/\[\[.*\]\]/i', $plantilla, $this->_modulos);
                $this->_modulos =
   $this->cargaModulos($this->_modulos[0]);

   if($this->_modulos = $this->procesaModulos($this->_modulos))
                foreach($this->_modulos as $mod)
                  $plantilla = str_replace($mod->string,$mod->buffer,
   $plantilla);

   $this->SetPlantillaProcesada($plantilla);
              }
```

The previous code uses a regular expression to find modules contained in a page, looking for strings such as '[[noticias::index::ultimahora]]'. After processing each module, it replaces these strings with the output buffer obtained from that module's run method.

The page concept and page selector are very practical. They make the code more human-readable by organising it into files containing module calls, just as we see them on the website.

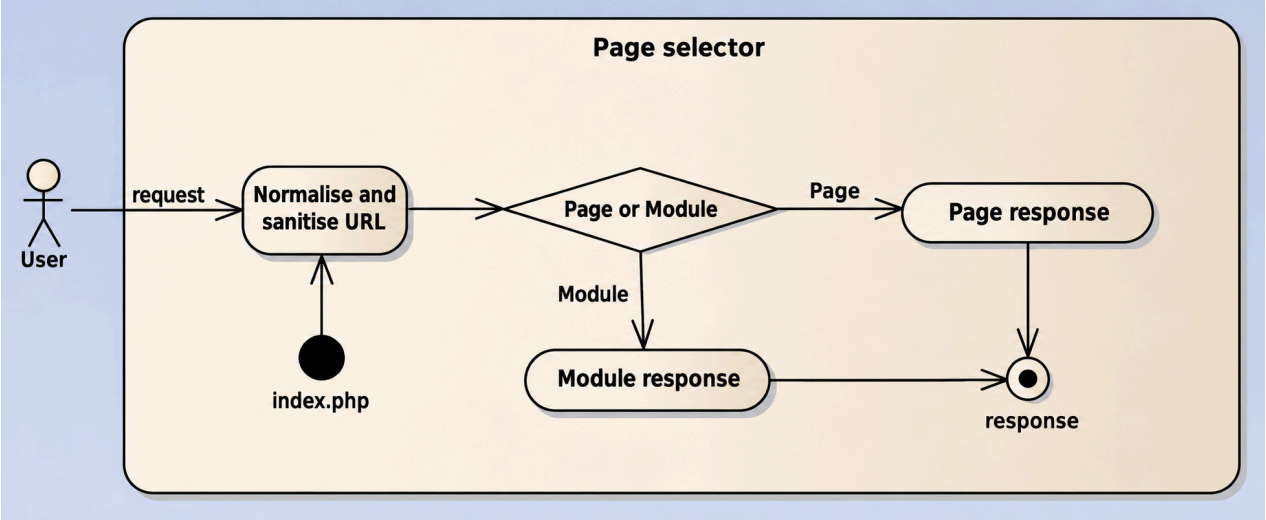
This degree of encapsulation and organisation would also allow caching at different levels, including pages and even modules. Here, caching is implemented at page level in index.php itself, but the system is ready for module-level caching. That could suit systems where particular areas, such as the user area, cannot be cached.

Direct access to a module

Pages work well up to this point, but some services offered by the website may not make sense as pages: a web service, for example, or a controller handling AJAX requests.

Pages must therefore not be the only way to call module controllers: we also need to access modules directly through a URL.

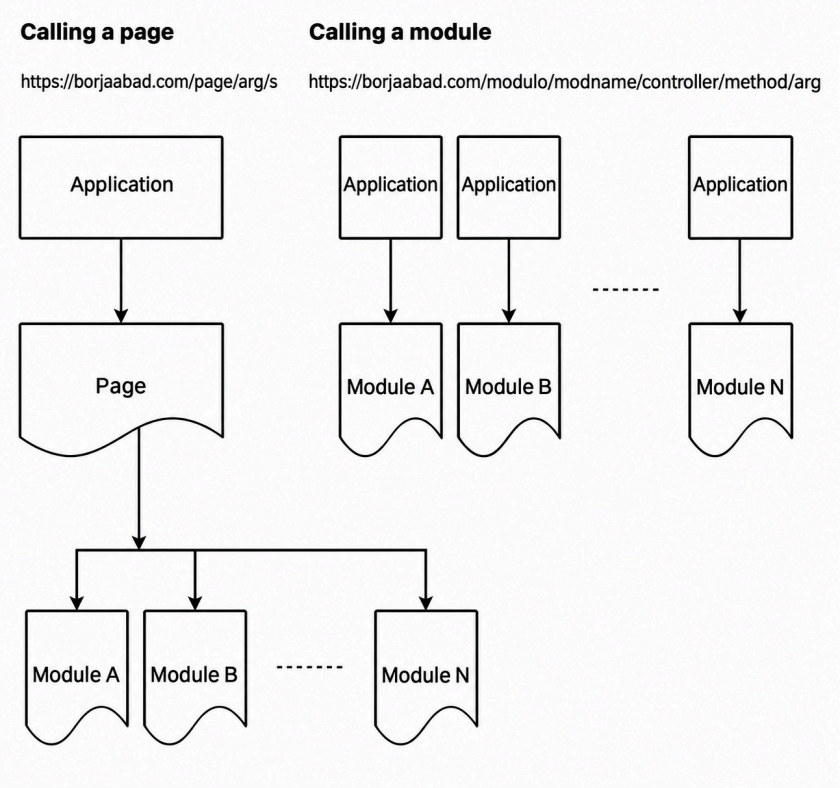
This introduces a new element: the page selector.



For this purpose, the first argument of our request contains the string 'modulo'.

```
http://cms.um/modulo/noticias/votes/vote/positive/123
```

https://borjaabad.com/modulo/modname/controller/method/argument/arg/s



The illustration shows that the system is designed around pages, which are assumed to constitute the website.

When a page loads, all its modules already have access to every Application class. A single instance of the system thus manages the loading of the page's modules.

Here is an excerpt from the Request class implementing the above:

```
//https://borjaabad.com/modulo/modname/controlador/metodo/arg/s
if ($this->first == 'modulo') {//Load module
	$this->_modulo = strtolower(array_shift($url));
	$this->_controlador = strtolower(array_shift($url));
	$this->_metodo = strtolower(array_shift($url));
	$this->_argumentos = $url;
	$modulo = $this->_modulo;
}
```

Module metadata passed to the page

Defining the page concept raises further questions. Common page metadata includes attributes such as title, description and keywords. Other resources, including JS and CSS files, also need to be considered.

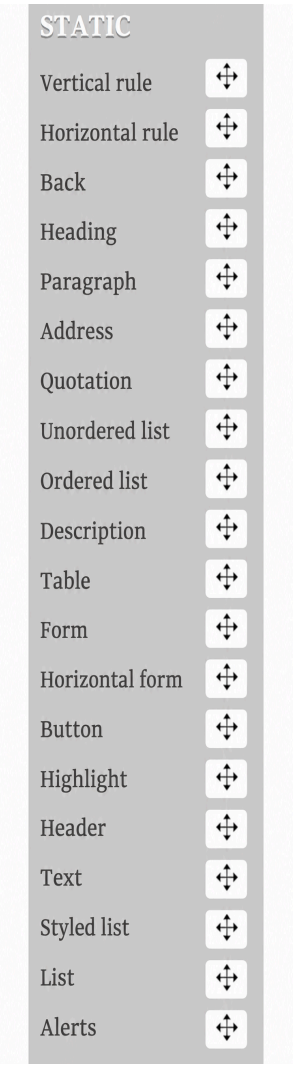
A module may need to define the page's title, description or keywords.

When defining a website layout, all pages normally share some CSS and JS. A particular module may also require an external script.

Modules can call methods on the page instantiating them to set these values. This is partly implemented in the application.

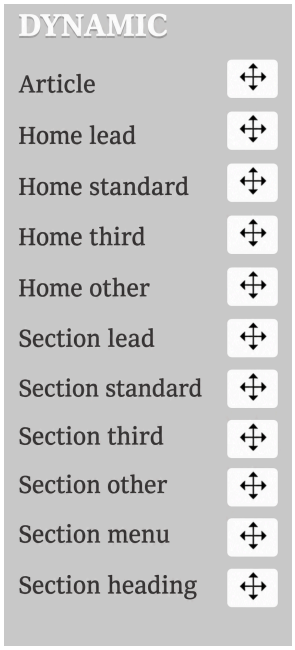
Static modules

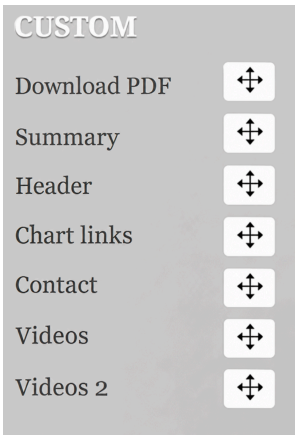
These modules are embedded in the application and are considered generic, commonly used elements. They consist of plain HTML, editable through a floating WYSIWYG editor in the page creation interface.



Dynamic modules

These are custom modules displaying information generated by components. They are configured in modules.xml and implement MVC, so they can receive variables through GET. The article module, for example, will expect a parameter.





The programmer creates them and supplies them to the client or user. The designer can access their style sheets and views. Basic users can only use these modules.

Custom modules

These arise from the need to customise and persist static modules. WYSIWYG facilities let users create content and store it in the system for inclusion in pages. It is stored in the database as HTML markup.

index.php

```
<?php
        define('DS', DIRECTORY_SEPARATOR);
        define('ROOT', realpath(dirname(__FILE__)) .
DS);
        define('APP_PATH', ROOT . 'application' . DS);


   require_once APP_PATH . 'Cache.php';
            $cache = new Cache();


   if($cache->isCached()){
              $cache->response($cache->buffer);
            exit;
          }



   try{
          //Application
          require_once APP_PATH . 'Config.php';
          require_once APP_PATH .
   'Request.php';
          require_once APP_PATH .
   'Bootstrap.php';
          require_once APP_PATH .
   'Controller.php';
          require_once APP_PATH .
   'Database.php';
          require_once APP_PATH . 'Model.php';
          require_once APP_PATH . 'View.php';
          require_once APP_PATH . 'Pagina.php';
          require_once APP_PATH .
   'Modulo.php';


       ob_start();
                $peticion = new
       Request();

       //Page selector

       if($peticion->paginaModulo($peticion)=='pagina')
                Bootstrap::cargaPagina(PAGINA);
              else
                Bootstrap::run($peticion);


       $buffer = ob_get_contents();


   $cache->set($buffer);
          }
          catch (Exception $e){
              $e->getMessage();
          }
        ?>
```

Cache system

When considering a suitable caching system, I wanted it to be simple and fast, without compromising information integrity by displaying outdated website content.

As index.php and the front-end architecture diagram show, Cache.php is loaded before the application. If a cache check succeeds, the system exits without loading the application.

```
if($cache->isCached()){
        $cache->response($cache->buffer);
      exit;
    }
```

If the resource is not cached, the page must be processed to produce the response. The resulting response is then cached to avoid processing it again on the next request.

```
$cache->set($buffer);
```

If the resource is not cached, the page must be processed to produce the response, and the newly requested resource is then saved.

On-demand caching

A cache must always have a limit, both for resource usage and for performance. As already discussed, many systems use time-based on-demand caching. This means the system remains alert and

periodically collects expired cache entries at a predefined interval. This is one of the most widely used approaches.

Instead, I propose limiting the number of stored resources rather than their storage time.

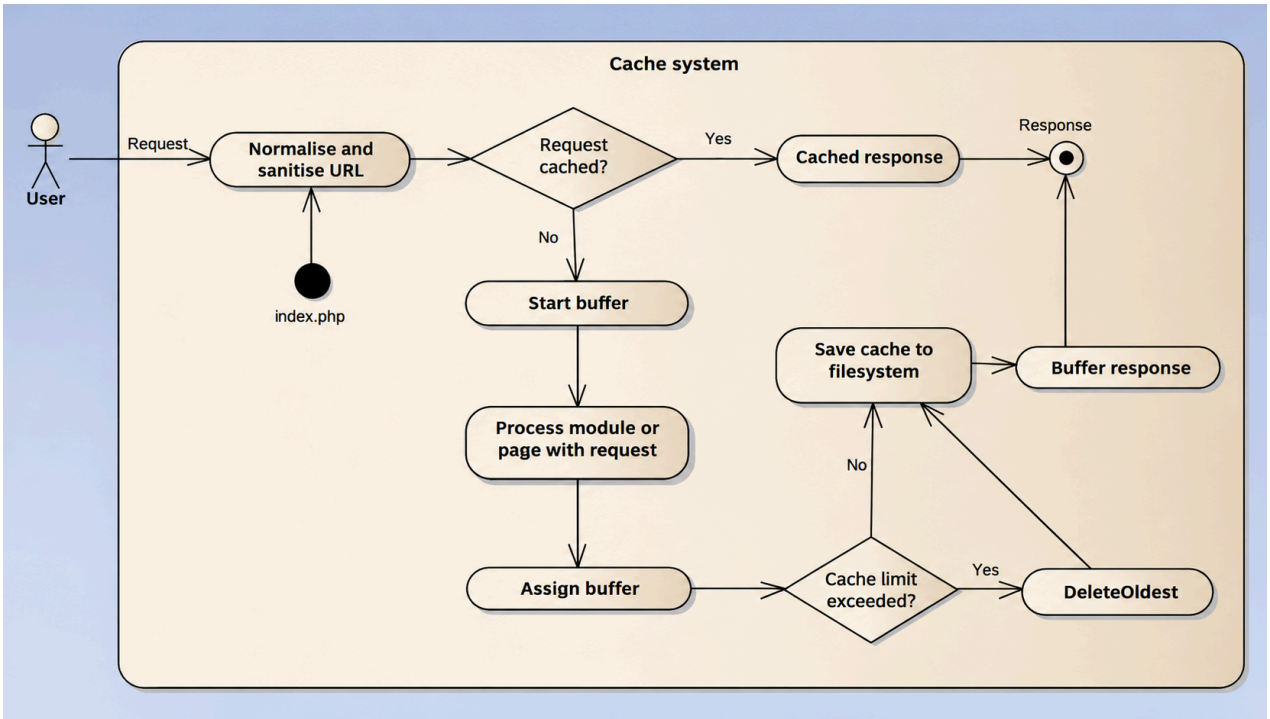
```
$cacheFile = $this->folder.$this->request;
        $dir = opendir($this->folder);


//If the maximum file count is reached, delete the oldest file
        if($this->countFiles() >= $this->limit){
          if($this->deleteOldest())
              if(file_put_contents($this->folder.$this->request,
$buffer));
                  return true;
        }
        else{
            if(file_put_contents($this->folder.$this->request,
$buffer));
                  return true;
        }
```

With a fixed limit, the next request for an uncached resource stores it in the cache and removes the oldest stored resource to stay within the limit.

The system will thus keep the resources most in demand at that moment in its cache.

Unlike time-based systems, which must monitor content expiry and consume resources doing so, this cache renews itself automatically within the established limits.



To prevent caching from compromising information integrity during content updates, generic back-end methods should clear the cache whenever content is modified, deleted or added. This part is not implemented, but it is the intended design.

One possible implementation would use a database stored procedure, triggered by any INSERT, UPDATE or DELETE operation that changes content, to set a field to true. Each time the application runs, it would check this value to decide whether to use the existing cache or discard it as obsolete.

Another possibility would be a manually invoked back-end method that clears the cache after every CRUD operation in the models.

This would ensure that the end user never sees outdated content.

CHAPTER V - LESSONS LEARNED

ANALYSIS AND INTERPRETATION OF RESULTS

In my view, the result is very satisfactory and the stated objectives have been achieved.

Performance

Only after completing the project was I able to test CMS UM's performance against existing products. See the appendix 'Internal study of response times in existing CMS platforms'.

That study shows that Joomla and WordPress, with or without caching, can serve between 7 and 8 requests per second for a typical piece of content.

Under the same conditions, CMS UM serves around 40 requests per second. The surprising result comes when caching is enabled: the web server can serve more than 1,200 requests per second.

It is too early to declare victory, but it is a good start. The idea is to keep running these tests throughout the CMS's development, so that we remain aware of performance losses as complexity increases.

Usability

Apart from opinions from people close to me, I have no data or studies that quantify usability, only my own experience. Once the modules that make up our site and its styles have been defined,

creating pages seems very quick and simple. This aspect is somewhat subjective and depends on user preferences, but I believe the stated aims have been achieved.

The project's application can be seen at https://borjaabad.com

CONCLUSIONS

With work and effort, I believe that the path towards a custom CMS serving as the foundation for almost all of a software producer's web projects can be useful, improving the organisation's productivity and competitiveness.

However different websites may be, they have much in common. Content editors increasingly want familiar resources such as article management, advertising, SEO, multimedia galleries, videos, forums and shops. These can be encapsulated in components and reused for different clients. This makes software production much more efficient, gives clients a more complete product, and allows effort to focus on their business model rather than on administering their website.

REMAINING WORK AND RECOMMENDATIONS

The singleton pattern should be applied when creating module instances, returning the previous state. This would improve performance when many modules use the same component, since only one instance of their model would be needed.
Choosing drag-and-drop libraries or WYSIWYG editors means relying on third-party software and adding hundreds of files to the project. This may become a weak point if these features are not properly implemented or encapsulated. It would therefore

be useful to place greater emphasis on metrics for evaluating the choice of suitable supporting software.

For example, the HTML produced by WYSIWYG editors could be analysed to determine which offers cleaner, more standard markup and best meets the CMS's styling requirements.

As the number of modules grows, the proposed visual interface would probably not scale, requiring another system of submenus or drop-down controls.

For caching, a site with entirely dynamic areas that cannot be cached, such as a user area, would need module-level rather than page-level caching.

BIBLIOGRAPHY

Content Management Systems in Knowledge Management
Authors
Jesús Salinas, Bárbara de Benito, Victoria Marín, Juan Moreno, María Eugenia Morales
Publisher
Universitat Illes Balears

____

Software Engineering
Author
Roger S. Pressman
Publisher
McGraw Hill

____

Patterns For Dummies
Author
Steve Holzner
Publisher
Wiley Publishing, Inc

____

OTHER RESOURCES

Introduction to Content Management Systems: CMS
http://www.slideshare.net/santillan/introduccin-a-los-sistemas-de-gestin-de-contenidos-cms

Content management systems
http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_contenidos

Content management
Title: Articles about content management systems
Author: James Robertson
http://www.steptwo.com.au/category/papers/content-management

Library inventory

Front-end graphical interface
Responsive website: Twitter Bootstrap
CSS
Journalist theme by Bootswatch

Back-end graphical interface
Responsive website based on KamiDesign KATO
Layoutit for drag-and-drop structure generation
CKEditor as the WYSIWYG editor for articles
Aloha Editor as the floating WYSIWYG editor for page layout

Image and a point of interest

Image from a site covering CMS news, Superbwebsitebuilders.com, in an article about the Cmscritic.com awards for different CMS platforms. Awards range from best open-source CMS to best enterprise CMS, site builders, forums and smaller systems.
In 2013, voters chose WordPress as the open-source category winner.
Awards:
http://www.cmscritic.com/critics-choice-cms-awards/
http://superbwebsitebuilders.com/critics-choice-cms-award/
Image:
http://superbwebsitebuilders.com/wp-content/uploads/2013/07/logos-CMS.jpg

The rise of websites based on database-free CMS platforms
http://www.interdixit.com/esp/noticia.php?pag=auge_los_sitios_web_basados_cms_sin_bases_datos

Desktop metaphor
http://es.wikipedia.org/wiki/Met%C3%A1fora_de_escritorio

Providers of CMS usage statistics
Statistics provider:
http://w3techs.com/
CMS data:
http://w3techs.com/technologies/overview/content_management/all
About W3Techs:
http://w3techs.com/about

GLOSSARY

ACL
Access control list: a security technique or system that assigns privileges and/or roles to entities in the system.
Spanish: lista de control de acceso.

Back end
The part of a website visible only to its administrators, normally used to manage data, the business model and the structure of the public site.
Abbreviation: back.

Buffers
Temporary allocations of memory used by processes to store processed information incrementally. They can also absorb incoming information.
Spanish: amortiguador.

Cache
A system that stores processing results in anticipation of future requests, avoiding the need to process them again.

CRUD
Create, read, update or delete: the principal operations performed on information stored in a database.
Spanish: crear, leer, actualizar o eliminar.

CMS
Content management system: software that allows users without advanced technical knowledge to organise information, add, delete or edit it, and present it as they wish. These are usually web applications.
Spanish: gestor de contenidos.

Drag and drop
A technique that lets users visually drag objects into destination containers.
Spanish: arrastrar y soltar.

Extreme Programming
An agile software development technique that addresses problems as they arise rather than anticipating them through exhaustive analysis.
Spanish: programación extrema.

Framework
A collection of tools, files, libraries, functions and other resources, together with a basic architecture, that provides developers with a starting point for their applications.
Spanish: marco de trabajo.

Front end
All the freely accessible public web pages that constitute a website.
Abbreviation: front.

GitHub
A system for maintaining software versions so that they are accessible and recoverable. Based on Git.

INSERT, UPDATE or DELETE
SQL keywords used to insert, update or delete information in a database. Together with SELECT, they provide the CRUD operations.
Spanish: insertar, actualizar o eliminar.
Interface
The definition of rules for communication between computer systems.

Agile methodologies
Methods, techniques and recommendations that facilitate software development tasks. Examples include Scrum and XP.

Output
The pages that have been interpreted and processed: the result of processing.
Spanish: salida.

Open source
Software projects whose source code is publicly available.
Spanish: código abierto.

MVC pattern
Model-view-controller (MVC) is a software architectural pattern separating an application's data and business logic from its user interface. It introduces a controller that receives user requests and passes information from models to views.

Performance
The level of responsiveness or efficiency experienced when using a system or program.
Spanish: rendimiento.

PHP
The most widely used programming language for creating web pages. Web servers interpret it, process the specified logic and return HTML to the user.

RUP
Rational Unified Process is the software development process attributed to Microsoft in the original text.
Spanish: proceso unificado de Rational.

Request
An HTTP protocol request. A request is followed by a response.
Spanish: requerimiento.

Repository
A code store maintaining a copy and version history of developers' projects. Ordinary files may also be stored. Examples include Git, SVN and Mercurial.

Response
An HTTP protocol response to the initial request.
Spanish: respuesta.

Site builders
Applications for creating websites quickly and easily. Their information management capabilities are basic rather than advanced.
Spanish: constructor de sitios.

SEO
Search engine optimisation: techniques and considerations used by editors and developers to help a website meet standards and rank well in search results.
Spanish: optimización de motores de búsqueda.

Tracker
A tool created to track particular information. It may revisit a source periodically to update itself. Used together, trackers gather information from external, public, remote sources.
Spanish: rastreador.

True
A computing concept used to evaluate a condition as true. It is also a reserved word in many programming languages.
Spanish: verdadero.

UML
Unified Modeling Language: a language that primarily uses diagrams to represent software flows, processes, states, descriptions, views and architectures.
Spanish: lenguaje unificado de modelado.

Web service
A technique for communicating data between different applications. For example, a mobile application running on a user's device can consume a web service containing remote data to display information.
Abbreviation: WS.

APPENDICES

External comparative study of Joomla and WordPress

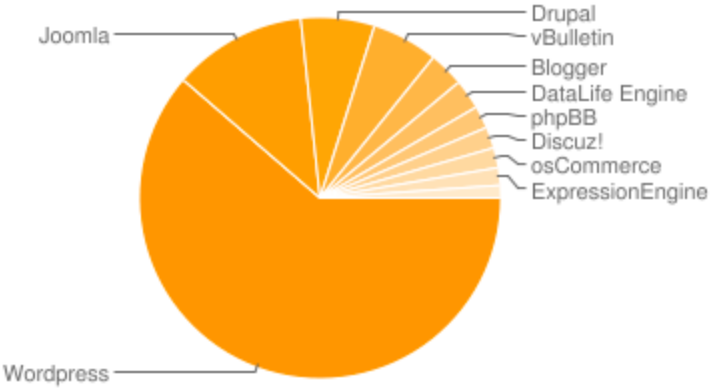
Complete study:
http://redgiantdesign.co.za/rg-design-blog/wordpress-vs-joomla-2013-infographic.html

Comparative infographic:
http://redgiantdesign.co.za/images/easyblog_images/42/WordPress-vs-Joomla.png

Produced by the co-founder of the well-regarded Red Giant Design studio, it offers interesting information on these two major platforms, their statistics and their evolution.

A series of articles unravels their workings, analyses their structure and features, and compares them in various ways.





Putting dynamic content in perspective

Content changes far less frequently than it is accessed. For this analysis I have access to private information from a friendly website, www.vivetorre.com, which provided its records. It is a typical local news portal using a traditional CMS, Editmaker.

The following table lists 20 articles, their last modification and creation dates, visits to date, and links to the articles:

Article link	Last modified	Created	Visits
http://www.vivetorre.com/noticia/3210/x/x	2012-12-24	2012-12-21	243
http://www.vivetorre.com/noticia/3202/x/x	2012-12-24	2012-12-21	58
http://www.vivetorre.com/noticia/3197/x/x	2012-12-24	2012-12-21	78
http://www.vivetorre.com/noticia/3195/x/x	2012-12-21	2012-12-21	72
http://www.vivetorre.com/noticia/3194/x/x	2012-12-21	2012-12-21	83
http://www.vivetorre.com/noticia/3193/x/x	2012-12-21	2012-12-21	98
http://www.vivetorre.com/noticia/3192/x/x	2012-12-21	2012-12-21	76
http://www.vivetorre.com/noticia/3191/x/x	2012-12-21	2012-12-21	73
http://www.vivetorre.com/noticia/3190/x/x	2012-12-21	2012-12-21	129
http://www.vivetorre.com/noticia/3189/x/x	2012-12-21	2012-12-21	79
http://www.vivetorre.com/noticia/3188/x/x	2012-12-21	2012-12-21	135
http://www.vivetorre.com/noticia/3174/x/x	2012-12-21	2012-12-21	121
http://www.vivetorre.com/noticia/3186/x/x	2012-12-21	2012-12-21	74
http://www.vivetorre.com/noticia/3187/x/x	2012-12-21	2012-12-21	70
http://www.vivetorre.com/noticia/3185/x/x	2012-12-21	2012-12-21	57
http://www.vivetorre.com/noticia/3183/x/x	2012-12-21	2012-12-21	78
http://www.vivetorre.com/noticia/3182/x/x	2012-12-21	2012-12-21	75
http://www.vivetorre.com/noticia/3181/x/x	2012-12-21	2012-12-21	84
http://www.vivetorre.com/noticia/3180/x/x	2012-12-21	2012-12-21	72
http://www.vivetorre.com/noticia/3179/x/x	2012-12-21	2012-12-21	71

Modification rate

The available information does not reveal how many times an article was modified, but it does show which articles changed. I estimate that those updated were modified once on average.
Only 3 of the 20 articles were modified after their creation date. This makes sense: here, as in many cases, website updates follow a print edition, with articles uploaded periodically.

Modification rate → 3 modified articles / 20 articles in total = 0.15

Access rate

Analysing how many times each article was accessed and calculating the average gives the access rate.

Access rate → 1,826 total visits / 20 = 91.3

We can conclude that the vast majority of server requests come from visitors browsing the site. In other words, back-end requests that modify content are minimal, no more than 1%:

Number of front-end requests >>> Number of back-end requests

The example shows that content is accessed much more frequently than it changes. We should therefore

put its dynamic nature in perspective: from the system's point of view, the information is less dynamic than one might initially think.

In conclusion:

Front-end access rate >>> Back-end modification rate

Note:

One could argue that analysing a single website does not represent the whole population, but the argument seems common sense to me and potentially applicable to any content management system.

There are more consumers of information than editors producing it. A few journalists can write for hundreds, thousands or millions of users: communication is a mass service. A highly specialised scientific or governmental site might have little traffic and a high update rate, but this is not typical of CMS-based websites.

Internal study of response times in existing CMS platforms

The tests used the same hardware and a basic sample page, with 10 concurrent requests and a total of 1,000 requests to the server.

The same test was performed on the following CMS platforms:

Joomla
C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/um_joomla/
Total time: 138.29 s
Requests per second: 7.23

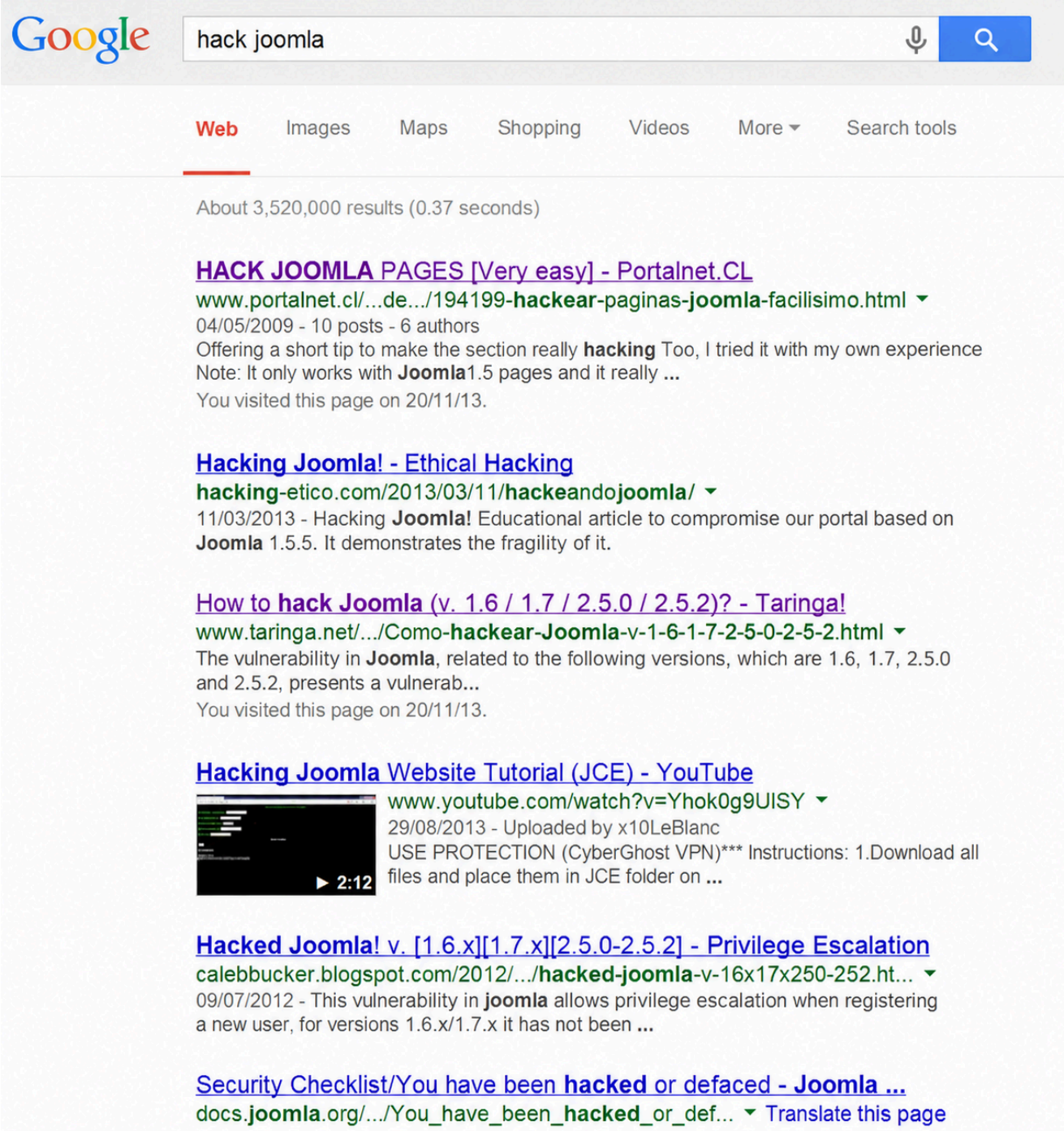
Joomla with built-in caching
C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/um_joomla/
Total time: 122.29 s
Requests per second: 8.19

Wordpress
C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/um_wordpress/
Total time: 124.77 s
Requests per second: 8.01

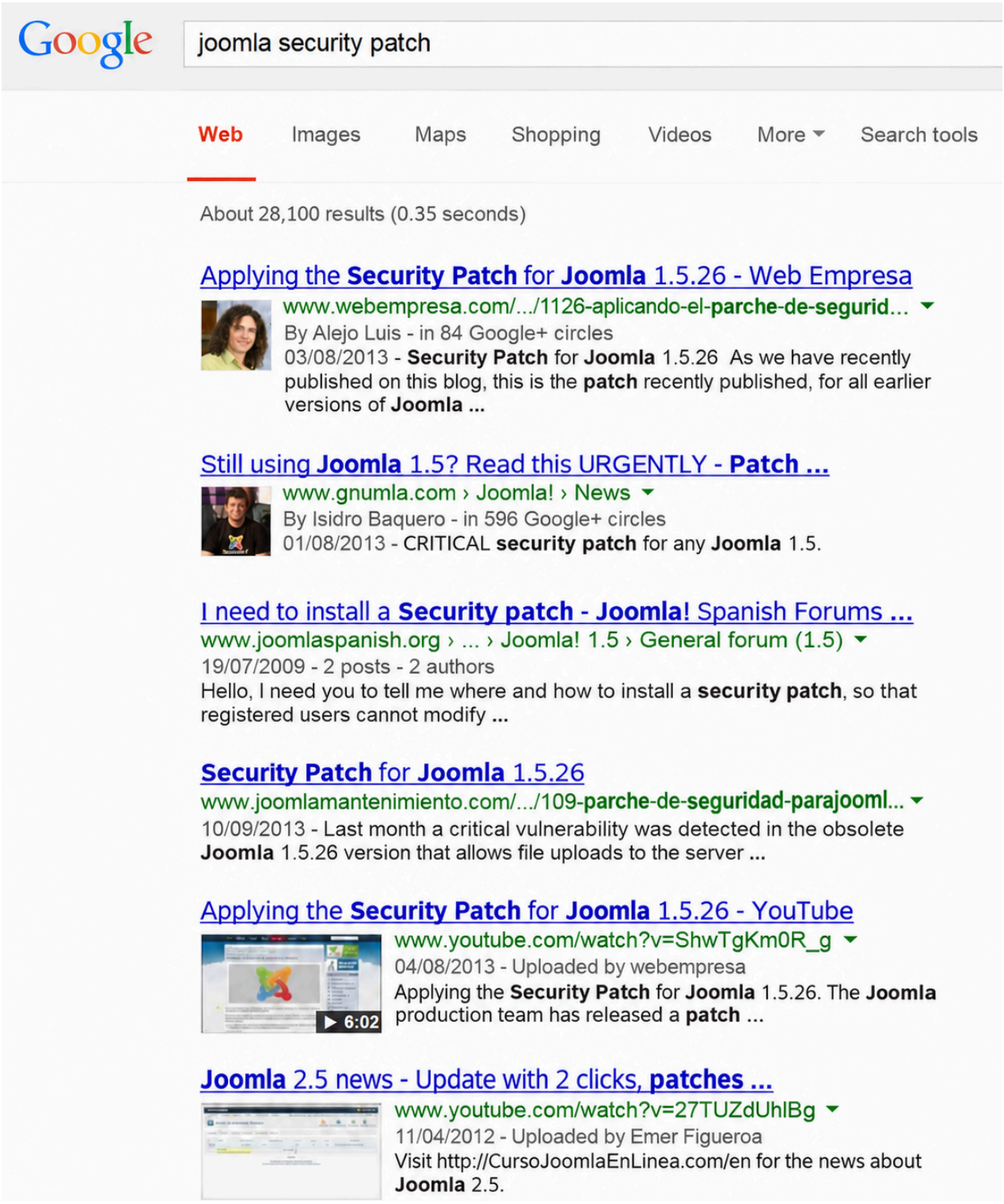
CMSUM
C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/cmsum/
Total time: 26.7 s
Requests per second: 40.13

CMS UM with on-demand caching
C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/cmsum/
Total time: 0.81 s
Requests per second: 1235.27

Internet searches
https://www.google.es/search?q=hack+joomla



https://www.google.es/search?q=parche+seguridad+joomla



Questionnaire 1

Juan Capistran

Freelance developer and entrepreneur

CEO Creative Cocos

http://creativecocos.com/

What software should a web software producer use as the foundation for its projects? There are as many valid options as there are preferences. What kind of brush should a painter use? What set of tools should a mechanic use? To focus on my particular case, I mainly develop Rails, PHP and ASP applications on Windows, so my basic tools are:
- IIS, Apache and Webmin as local servers.
- MySQL, PostgreSQL and Riak as database servers.
- Notepad++ and RubyMine as code editor and IDE.
- Every available browser, with web development plugins for each.
- Various validators, code minifiers, consoles and so on.

Is it appropriate to build a custom web system instead of using a product already on the market?
It depends on the circumstances, the goal, the client and the budget. Is it appropriate to build a custom house instead of buying a prefabricated one? There is no universally correct answer. Reusing, adapting and configuring is generally easier, but it is not always possible or desirable. In general, I prefer a well-built custom application to a well-built generic one. Finally, there are different levels of code reuse. The fact that

an application was custom-built does not mean we had to reinvent the wheel.

Can a single web system form the basis of almost all a software producer's projects? Yes, particularly when considering existing web development frameworks such as Ruby on Rails or Yii.

Questionnaire 2

Carlos Peralta

Degree in Computer Science

Team Leader Doopress

Systems Administrator

http://doopress.com/

What software should a web software producer use as the foundation for its projects?
I understand these are called software stacks. I suggest LAMP because of its widespread use and the number of systems built with it: Linux, Apache, MySQL and PHP. The final P can also stand for Python or Perl, but PHP is the most popular.

Is it appropriate to build a custom web system instead of using a product already on the market?
If the available software is open source and supported by a community, choosing an existing product is undoubtedly preferable. Its popularity should be assessed to estimate how long support will last and how many people will provide it over the medium and long term. For commercial systems, I believe cost is the most relevant factor. A custom system may sometimes be the only way to satisfy the requirements, but its medium- and long-term maintenance by the people or companies that developed it must be considered.

Can a single web system form the basis of almost all a software producer's web projects?
I do not think the distinction between 'web system' and 'web project' is particularly clear or relevant. If this refers to technologies, a stack such as LAMP is not always the foundation of a web project. There might only be Apache and PHP, with another operating system underneath, another database engine, or languages compiled at operating-system level, such as C++.

Questionnaire 3



What software should a web software producer use as the foundation for its projects?
Depending on the scale and requirements of a project, different frameworks can help us develop it.
If our project does not require anything very specific, we could base it on a CMS such as Joomla or Drupal. This can save considerable work: some code, such as article and user/ACL management, is already developed, and third-party modules, components and plugins save further programming. These CMS platforms also have large developer communities and documentation we can consult at any time.
For a larger project, frameworks such as Zend or Symfony can help us structure and organise our code.

Is it appropriate to build a custom system instead of using a product already on the market?
Custom development has advantages and disadvantages.
First, developing an entire custom system can be very expensive, depending on its scale. It also requires an experienced programming team following well-structured working practices,

strict planning and organisation, and a design that allows the system to scale.
A custom system can be highly optimised because it avoids passing through classes and functions included in general-purpose products intended for broader use.

It also avoids exposure to security vulnerabilities in widely used products, which can be attacked quickly once discovered. However, this means we must be even more careful about our own code's security, since we do not benefit from a community identifying and correcting these problems.

Conversely, we lose the benefit of code and documentation developed by an expert community and continuously improved.

Can a single web system form the basis of almost all a software producer's web projects?
This depends greatly on the producer's projects. Generally, yes: producers normally use a base system to save development work and offer clients more competitive prices.
Some specific projects will not use a base system, either because it does not meet their requirements or because clients demand otherwise.
In any case, software producers accumulate code scripts over time that they reuse in future projects.