Instituto Politécnico
de Castelo Branco
Escola Superior
de Tecnologia

Framework de Desenvolvimento de Aplicações Web

Mestrado em Desenvolvimento de Software e Sistemas Interactivos

Tiago Filipe de Andrade Alves

Orientadores
Doutor José Carlos Meireles Monteiro Metrôlho
Doutor Fernando Reinaldo Da Silva Garcia Ribeiro

Maio de 2013

Framework de Desenvolvimento de Aplicações Web

Tiago Filipe de Andrade Alves

Orientadores

Doutor José Carlos Meireles Monteiro Metrôlho

Doutor Fernando Reinaldo Da Silva Garcia Ribeiro

Relatório de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Desenvolvimento de Software e Sistemas Interactivos, realizada sob a orientação científica do Doutor José Carlos Meireles Monteiro Metrôlho e co-orientação do Doutor Fernando Reinaldo Da Silva Garcia Ribeiro, do Instituto Politécnico de Castelo Branco.

Maio de 2013

Composição do júri

Presidente do júri

Doutor Osvaldo Arede dos Santos, Professor Adjunto do Instituto Politécnico de Castelo Branco.

Vogais

Doutor José Manuel Cabral, Professor Auxiliar da Escola de Engenharia da Universidade do Minho;

Doutor Alexandre José Pereira Duro da Fonte, Professor Adjunto do Instituto Politécnico de Castelo Branco;

Doutor José Carlos Meireles Monteiro Metrôlho, Professor Adjunto do Instituto Politécnico de Castelo Branco (Orientador);

Doutor Fernando Reinaldo Silva Garcia Ribeiro, Professor Adjunto do Instituto Politécnico de Castelo Branco (Co-orientador).

Agradecimentos

Aos meus pais, à minha avó e à minha madrinha, pelo incentivo em continuar os estudos, por todo o apoio e amizade, e pela companhia em longas noites de trabalho.

À Cátia por todo o amor, carinho e compreensão, e por não me deixar desistir de seguir os meus ideais, mesmo quando por vezes duvidei dos mesmos.

Ao Cordeiro, Sara, Pedro, Patrícia, Paulo e Fontes pela camaradagem, pela companhia, por toda a farra que me ajudou a desanuviar um pouco de todo o trabalho em mãos. Também a vós devo grande parte do conhecimento e motivação que tenho para seguir esta área.

Ao Catarino, Guilherme e Hiugo, pelo apoio, motivação e contributo que deram ao longo deste trabalho, por me ajudarem e motivarem a crescer profissionalmente.

À malta do Lab que me acompanhou e apoiou durante esta última fase.

Um agradecimento especial aos Professores José Metrôlho e Fernando Ribeiro, por todo apoio e confiança que me deram ao longo do meu percurso nesta instituição.

Resumo

Neste documento apresenta-se uma *framework* de desenvolvimento de aplicações web. Este projeto conta com um sistema base concebido sobre PHP e num paradigma totalmente orientado a objetos. A *framework* disponibiliza diversos métodos através dos quais se torna possível integrar uma aplicação específica sobre uma base comum a diversos tipos de projetos. Este sistema foi criado de forma a disponibilizar ao programador diversas vantagens na construção das suas aplicações, sem que o mesmo necessite de se preocupar com a sua implementação, uma vez que estas estão embutidas no funcionamento desta *framework*. Um dos principais objetivos deste projeto está em focar o esforço de desenvolvimento num problema específico, deixando de parte o problema comum da criação de uma arquitetura base para cada aplicação.

Palavras chave

Framework, Model View Controller, Model View Presenter, Web.

Abstract

This document contains information about a framework for web application development. This project is based on a PHP system in a fully object oriented paradigm. This framework provides various methods that allow the integration of a specific application in a common base for any type of project. This system was designed in order to provide the programmer a number of advantages in the construction of its applications, without the need to worry about the implementation, since these advantages are embedded in the operation of this framework. A major goal of this project is to focus the development efforts on a specific problem, leaving aside the common problem of creating a base architecture for each application.

Keywords

Framework, Model View Controller, Model View Presenter, Web.

Índice geral

Índice de figuras

Lista de tabelas

Lista de abreviaturas, siglas e acrónimos

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

DHTML – Dynamic HTML

DOM – Document Object Model

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment,

MVC – Model View Controller

MVP – Model View Presenter

PDO – PHP Data Objects

PDT – PHP Development Tools

PHP – PHP Hypertext Preprocessor

POO – Programação Orientada a Objetos

RAD – Rapid Application Development

RBAC – Role Based Access Control

SEO – Search Engine Optimization

URI – Uniform Resource Indentifier

URL – Uniform Resource Locator

WSDL – Web Service Definition Language

1. Introdução

1.1. Enquadramento e Motivação

A motivação por detrás da conceção de uma *framework* web está comummente focada na resolução de um ou vários problemas, que poderão abranger diversas áreas ou camadas de uma arquitetura de uma aplicação *web*, quer na construção de código de servidor, bem como na conceção da componente que será apresentada a um utilizador através de um cliente. Estas *frameworks* são normalmente dedicadas à produção de código robusto, estruturado, modular e reutilizável, seguindo uma *design pattern* apropriada a este propósito, tal como a *Model View Controller* (MVC). A *Model View Controller* é uma *design pattern* focada na arquitetura de *software* que separa a informação por três componentes, em que cada qual controla uma parte dessa informação, enquanto que é garantida a interação entre elas [1]. A utilização deste tipo de *patterns* permite a simplificação da construção de código, e a sua correcta estruturação. A MVC é observada em maior detalhe no capítulo 3.1.4.

Uma *framework PHP Hypertext Preprocessor* (PHP) [2] MVC, regularmente inclui as componentes presentes numa estrutura MVC, adicionando ainda o suporte para a criação de uma aplicação *web* completa. Estas possuem capacidades para responder aos vários problemas apresentados pelas camadas distintas de uma aplicação *web*, apresentando uma forte componente *server-sided*, bem como uma forte integração com uma, ou várias, componentes *client-sided*, na qual o foco se estende à integração de ferramentas de *templating*, tais como o *Smarty* [3] ou o *Twig* [4], e/ou ferramentas de *scripting*, como o *Sencha* [5] ou o *jQuery* [6], criando assim uma camada de abstração que permite a utilização de *Dynamic HyperText Markup Language* (DHTML) [7] de uma forma simplificada e fortemente integrada com as componentes de *backend*.

A *framework* apresentada neste documento pretende inserir-se no conceito descrito acima, alterando a sua estrutura base de forma a suportar os princípios de uma MVC tradicional, favorecendo no entanto a utilização de uma abordagem *Model View Presenter* (MVP). Esta arquitetura distingue-se por delegar o controlo dos elementos da *View* ao *Controller*, que neste paradigma assume o nome de *Presenter*, tornando a *View* num elemento com lógica reduzida ou até mesmo inexistente [8].

A motivação da criação da mesma prende-se na necessidade de possuir uma arquitetura devidamente estruturada, utilizando para este fim o conceito base de uma *MVC Framework*, aliado a um sistema que possibilite a criação de uma componente de *frontend* menos focada em automatizações por parte da própria plataforma.

Este trabalho diverge assim das *frameworks* PHP MVC habituais na medida em que utiliza uma arquitetura que proporciona liberdade de escolha nas ferramentas e metodologias a utilizar, provê um conceito que favorece a utilização da MVP em detrimento da MVC tradicional, e que não está inteiramente focada em integrar e basear-se noutras ferramentas já existentes, mas sim na simplificação da construção de código sobre uma estrutura que reforça a utilização de boas práticas no desenvolvimento de aplicações.

1.2. Objetivos

O objetivo deste projeto é a conceção uma *framework* com as seguintes características:

- Assente sobre derivações do modelo MVC;
- Estrutura segmentada em elementos modulares;
- Seguir o paradigma de programação orientada a objetos (POO);
- Instanciar objetos e iniciar ações através da interpretação de pedidos de um cliente;
- Criar um ambiente de desenvolvimento simples, que permita focar o esforço do *developer* na aplicação específica e não na sua estrutura base;
- Permitir a integração de outras tecnologias ou conceitos na plataforma existente, tais como *PHP Data Objects* (PDO) e *Role Based Access Control* (RBAC).

1.3. Organização do Documento

O presente documento encontra-se dividido em 5 capítulos, contendo toda a informação sobre o desenvolvimento deste projeto.

No primeiro capítulo é feita uma introdução ao âmbito no qual este trabalho se insere e são apresentados os objetivos definidos para o mesmo.

O segundo capítulo apresenta o estado da arte, onde é feita uma recolha de informação sobre plataformas semelhantes, comparando-as entre si e com o trabalho a desenvolver.

O terceiro capítulo apresenta toda a fase de desenvolvimento, descrevendo metodologias utilizadas, diagramas de apoio ao desenvolvimento, a arquitetura utilizada e características adicionais da *framework*.

No quarto capítulo são apresentados testes e otimizações efetuadas ao código, culminando com um exemplo de implementação e utilização

No quinto capítulo são apresentadas as conclusões extraídas com o desenvolvimento deste trabalho e o trabalho futuro a realizar.

2. Estado da arte

Dentro do âmbito que se insere este trabalho, foram identificadas as *frameworks* concebidas em PHP mais populares que recorrem a uma estrutura MVC, ou outra derivada desta. As *frameworks* apresentadas neste estudo foram selecionadas face à sua popularidade, extraída pelo volume de pesquisa de termos relacionados com as mesmas, por votação pública, e por resultados obtidos em motores de pesquisa de ofertas de trabalho para programadores *web freelancer* [9, 10, 11, 12]. Através dos estudos referidos anteriormente, foram selecionadas as *frameworks CakePHP*, *CodeIgniter*, *Yii* e *Zend* como as mais relevantes no âmbito deste trabalho.

2.1. CakePHP

CakePHP [13] é uma *framework open source* escrita em PHP e modelada a partir de conceitos extraídos de *Ruby on Rails*. Esta *framework* teve origem a partir de uma versão da *Rapid Application* Framework, escrita por Michal Tatarynowicz em 2005.

Esta *framework* tem como principais características:

- Configuração inicial quase inexistente;
- Possui uma biblioteca de utilitários completa;
- Possui convenções próprias para guiar o programador na criação de cada aplicação;
- Possui ferramentas de validação de dados e formulários, e sistemas de segurança na sua biblioteca base.

2.2. CodeIgniter

O *CodeIgniter* [14] é uma *framework open* source, desenhada para o desenvolvimento rápido de projetos. Esta foi criada em 2006 pela *EllisLab* e possui as seguintes características:

- Baseada numa arquitetura MVC;
- Estrutura leve e com uma biblioteca reduzida
- Possui um desempenho elevado;
- Compatibilidade com maioria dos provedores de alojamento
- A interface de acesso às bibliotecas é simples e bem estruturada
- *Framework* bem documentada, com inúmeros exemplos de utilização;
- Configuração inicial quase inexistente;
- Orientada a objetos.

2.3. Yii

A *Yii* [15] é uma *framework open source* escrita em PHP 5 que promove um *design* aplicacional limpo e reutilizável, e o desenvolvimento rápido de aplicações.

A criação desta *framework* deu-se em Janeiro de 2008, pelo seu fundador Qiang Xue. Em Dezembro de 2008, após um ano de desenvolvimento, a versão 1.0 da *Yii* foi lançada formalmente ao público.

Algumas características relevantes da *Yii* são:

- Baseada numa arquitetura MVC;

- Implementa *Database Access Objects* (DAO), *Query Builder*, *Active Record* e mecanismos de migração de Bases de dados;
- Validação de formulários e de dados;
- *Widgets* com suporte de *Asynchronous JavaScript and XML* (AJAX);
- Suporte de autorização e autenticação;
- Mecanismo de personalização e alteração de temas;
- Geração automática de *Web Service Definition Language* (WSDL) complexos e gestão de pedidos por *Web Service*;
- Internacionalização e localização de aplicações.

Esta *framework* incorpora ainda ideias e trabalho de outras *frameworks* semelhantes, tais como a *Prado*, *Symfony* e *Joomla*.

Um dos pontos chaves promovidos pela *Yii* é o seu alto desempenho face a outras *frameworks* tais como a *CodeIgniter*, a *CakePHP* e a *Zend*, atribuindo esta vantagem à utilização extensiva da técnica *lazy loading*.

2.4. Zend

A *Zend Framework 2* [16] é *open source* e destina-se à criação de aplicações e serviços *web* utilizando a versão 5.3. do PHP e superiores. Esta foi criada em 2006 e é mantida pelos seus criadores, a *Zend Technologies.*

Os componentes desta *framework* são desenhados para possuir o mínimo de dependências de outros componentes. Esta arquitetura permite aos *developers* utilizar apenas os componentes essenciais à sua aplicação. Embora possam ser usados em separado, os componentes da biblioteca padrão formam uma *framework* poderosa e extensível.

Algumas das características desta *framework* são:

- Arquitetura com dependências mínimas;
- Assente sobre uma estrutura MVC sólida e extensível com suporte de *templating*;
- Possibilidade de desenvolver aplicações modulares, onde cada módulo possui as suas próprias *Views, Controllers e Models*;
- Abstração de base de dados.

2.5. Análise Comparativa

Na Tabela 1 apresentam-se as diversas *frameworks* mencionadas no capítulo 2, juntamente com algumas das características mais relevantes para comparação com os objetivos propostos para este projeto.

Tabela 1 - Comparação de características entre *frameworks*

Framework	PHP4	PHP5	MVC	MVP	DB Abstraction	Modules	DB Objects	Templates	AJAX	Auth Module
CakePHP	✔	✔	✔		✔	✔	✔		✔	✔
CodeIgniter	✔	✔	✔		✔	✔	✔	✔		
Yii		✔	✔		✔	✔	✔	✔	✔	✔
Zend		✔	✔		✔	✔	✔	✔	✔	✔
Framework a desenvolver		✔	✔	✔	✔	✔	✔		✔	✔

As características apresentadas são:

- **PHP4:** Suporte para a versão 4 do PHP;
- **PHP5:** Suporte para a versão 5 do PHP;
- **MVC:** Assente sobre uma estrutura MVC ou suporta a mesma;
- **MVP:** Assente sobre uma estrutura MVP ou suporta a mesma;
- **DB Abstraction**: Possui modelo de abstração de dados relacionais;
- **DB Objects:** Utiliza Objetos de Dados;
- **Templates:** Utiliza ou é baseado num sistema de *Templates*;
- **AJAX:** Possui suporte para eventos AJAX;
- **Auth Module:** Possui um módulo interno que permita a gestão de autenticação.

As *frameworks* apresentadas são, em sua maioria muito semelhantes ao trabalho aqui apresentado, sendo a *Yii* e a *CakePHP* aquelas que mais se aproximam dos objetivos propostos e que possuem um comportamento mais próximo do pretendido nesta implementação.

A possibilidade de desenvolvimento em MVP estrita, em detrimento da MVC, deixando possível o suporte para esta última, torna-se uma das principais diferenças deste trabalho.

Por se tratar de uma versão menos recente e limitada, não é dado suporte a PHP 4 neste trabalho. Esta versão possui limitações que impedem a correta construção de uma *framework* inteiramente orientada a objetos.

O sistema de *templating* não constitui um objetivo deste trabalho por se considerar que a não inclusão do mesmo será uma maior vantagem na adoção desta *framework*, uma vez que permite ao programador escolher o sistema que mais se adequa ao trabalho que pretende desenvolver.

Face ao exposto anteriormente, será desenvolvida uma *framework* baseada nas características mais comuns, implementando alguns conceitos únicos à mesma. Esta framework deverá ser capaz de utilizar na integra o conceito MVC e MVP, propiciando a

correcta utilização de cada classe envolvida nestas arquiteturas, permitindo ainda a segmentação de vários grupos MVC em módulos independentes entre si. Deverá possibilitar o tratamento de pedidos por AJAX, reencaminhando os mesmos de forma distinta dos pedidos normais. O suporte básico para a utilização de objectos de dados, permitindo a fácil integração dos mesmos com os *Models* de cada aplicação, deverá estar incluído. A *framework* deverá possuir ainda características que lhe permitam criar uma base para vários tipos de aplicações, como por exemplo, um módulo de autenticação de utilizadores e suporte para múltiplos idiomas.

3. A *Framework*

Neste capítulo estão contidas as informações relevantes à fase de desenvolvimento do projeto. Este refere as metodologias utilizadas, as suas iterações, o ambiente utilizado para o desenvolvimento desta *framework,* a arquitetura deste projeto, o ciclo de vida dos componentes, os vários diagramas que refletem o funcionamento e comportamento de cada aplicação, as otimizações, testes e documentação efetuados ao código, terminando com um exemplo de utilização da *framework*.

3.1. Desenvolvimento

3.1.1. Metodologia de Desenvolvimento

No âmbito deste projeto, foram identificados diversos fatores, tais como prazos para desenvolvimento e necessidade de possuir versões intermédias para a criação de aplicações piloto, que contribuíram para a adoção de uma metodologia capaz de lidar com os prazos de entrega estabelecidos e tempo de desenvolvimento entre versões. Perante estas necessidades, foi seguida uma metodologia capaz de lidar com prototipagem rápida, suscetível a mudanças súbitas tanto a nível de código como no planeamento da aplicação, culminando em implementação e testes em ambientes reais. Neste sentido adotou-se uma metodologia de desenvolvimento baseada na *Rapid Application Development* (RAD) [17].

3.1.2. Iterações no Processo de Desenvolvimento

Durante a fase de desenvolvimento foram produzidas 4 versões da plataforma, sendo que cada uma correspondeu também a uma iteração no ciclo de desenvolvimento. Das 4 versões desenvolvidas, duas foram consideradas como sendo versões finais e representativas dos objetivos do projeto, sendo que as versões intermédias consistiram em adição de funcionalidades e correções de *bugs.* A segunda versão da *framework*, apresentada neste documento, reaproveitou conceitos e componentes provenientes da primeira versão. Os componentes da *package* "core" foram redesenhados na sua totalidade, com o intuito de otimizar o funcionamento da *framework,* bem como de permitir expandir a mesma.

Cada versão foi acompanhada de um projeto num ambiente real de desenvolvimento, direcionado para aplicações *web* públicas e funcionais para um público-alvo. Esta estratégia serviu como base essencial ao aperfeiçoamento do projeto, uma vez que tornou um conceito teórico, sob o qual o mesmo assentou inicialmente, aplicável em situações práticas com um elevado grau de exigência sobre o trabalho desenvolvido.

Estas versões foram aplicadas a projectos específicos com necessidades distintas, tendo sido identificada a cada etapa uma base comum que foi melhorada no decorrer das várias iterações. Através das respostas obtidas por cada cliente, a cada fase do projeto, foi possível gerar incrementos nas versões produzidas, adicionando funcionalidades em cada etapa, o que permitiu também recolher informação útil à melhoria da arquitetura da *framework*.

O ciclo de desenvolvimento terminou ao fim da quarta iteração, uma vez que na mesma se alcançaram os objetivos deste projeto, e não foram identificadas lacunas ou necessidades dentro do âmbito ao qual se pretendia dar suporte.

3.1.3. Ambiente de Desenvolvimento

Desde a fase embrionária deste projeto foram identificadas as tecnologias e linguagens de programação a utilizar, escolhidas de acordo com a natureza do trabalho a desenvolver.

A base funcional deste projeto é integralmente assente sobre PHP, uma linguagem que conta com uma vasta comunidade de desenvolvimento pelo que é comummente adotada por provedores de pacotes de alojamento *web* e *web developers* [18]. O PHP é distribuído sob a *PHP License* v3.01 pelo *PHP Group* [19], potenciando a sua utilização sem custos de licença associados.

Esta linguagem foi, nas suas versões mais recentes (5.x), alvo de uma migração para a *Zend Engine* 2.0 [20], alterando na íntegra o modelo de POO utilizado, tornando-a mais completa e melhorando o seu desempenho [21], enquanto aproxima o seu paradigma de programação a outras linguagens já bem estabelecidas neste âmbito, tais como o C++ ou Java.

A construção desta plataforma contou com a utilização de diversas ferramentas de desenvolvimento, ou de apoio ao desenvolvimento, consideradas como as mais adequadas para atingir os objetivos a que se propunha este projeto.

Integrated Development Environment (IDE)

O IDE *Eclipse* [22], em conjunto com o *plugin PHP Development Tools* (PDT) [23] e o *debugger Xdebug* [24] auxiliam na criação projetos complexos em PHP. Esta combinação apresenta as características típicas de um IDE convencional, tais como *code completion* e *syntax coloring* e *checking*, além de criar um elo virtual entre classes e outras dependências do projeto. A adição da ferramenta *Xdebug* ao *Apache* [25] possibilita a depuração de código em tempo real no ambiente em que se insere o consumidor final da aplicação (e.g. um *web browser*), permitindo a adição de *break-points* e inspeção de elementos, entre outros.

Browser

A utilização de um *browser* no processo de desenvolvimento torna-se imprescindível, dado que o mesmo possibilita fazer pedidos HTTP e, subsequentemente, visualizar os resultados obtidos após interpretação desse mesmo pedido.

Como referido anteriormente, esta ferramenta permite ainda, em conjunto com o *Xdebug*, efetuar verificações e depuração de código.

3.1.4. Arquitetura

A arquitetura base de cada aplicação, apresentada na Figura 1, assenta sobre 3 camadas abstratas distintas:

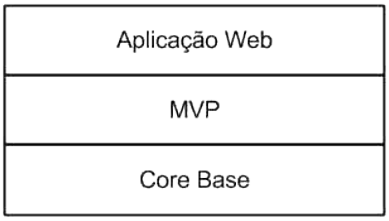

Figura 1 - Arquitetura Aplicacional

O grupo *Core Base* serve de apoio a todo o projeto, contento os elementos de mais baixo nível, que fornecem suporte a todos as outras camadas.

A camada de *Aplicação Web* define todo o conjunto de artefactos produzido para cada aplicação concreta. Este conjunto pode ser composto por *Controllers, Views, Models, scripts,* entre outros recursos necessários à aplicação. Esta é a camada a desenvolver em cada aplicação, que assentará sobre a camada de *Model View Presenter* (MVP) concebida para suportar as necessidades de cada aplicação individual, estabelecendo relação com a camada *Core Base* de forma a automatizar o processo de utilização das componentes da *Aplicação Web*.

MVC

A derivação da metodologia imposta pela *design pattern* MVC conferiu a este projeto uma forma de dividir o código dos diversos componentes por entidades responsáveis por uma parte do sistema. Esta estrutura reforça a criação de código estruturado, bem como a sua simplificação, uma vez que cada classe que a compõe deverá apenas possuir código dentro da função que possui. Desta forma, torna-se possível afirmar que a plataforma concebida assenta sobre vários componentes altamente modulares, seja pela natureza da sua funcionalidade ou responsabilidade, seja pelo princípio de construção imposta a estes componentes. Esta distribuição de responsabilidades permite a reutilização, integração e manutenção de grande parte dos componentes de forma não condicionada, aumentando assim a escalabilidade da plataforma.

Na representação da MVC (Figura 2), a *View* e o *Controller* possuem dependência do *Model*, no entanto o *Model* é independente destas duas classes, tornando a sua conceção e testes independentes da apresentação visual [26].

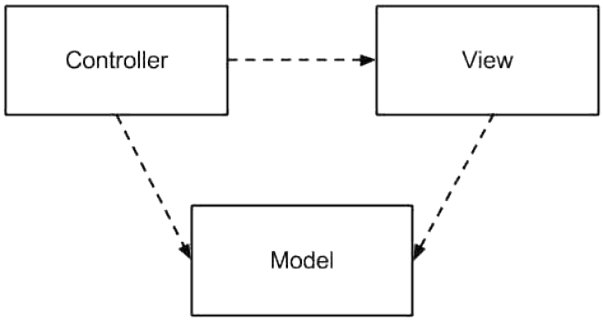

Figura 2 - *Model View Controller*

A classe *Model* é tipicamente correspondente a dados e comportamentos de uma entidade individual, que poderá ter dependências de outras entidades. Esta encapsula atributos e comportamentos que permitem alterar o seu estado ou obter informações sobre a própria entidade. A *Model* normalmente corresponde a uma entidade numa base de dados, ou a outra estrutura de dados persistente.

A View é uma representação visual da informação, apresentando informação proveniente do *Controller* ou *Model*.

O *Controller* interpreta ações de um cliente, informando o modelo ou a *View* de alterações que lhe sejam pertinentes.

Existem, no entanto, variações a este modelo clássico, que definem outro tipo de relações entre as classes, bem como variações à própria MVC, que embora usem o conceito da

estruturação pelas 3 entidades responsáveis, definem novas formas de interação entre eles, alterando também a responsabilidade de cada uma das classes deste modelo.

Neste trabalho foram utilizados dois conceitos derivados da MVC original, adequados à gestão do *Controller* a utilizar e à forma como o mesmo se relaciona com a *View* e *Model*, estes conceitos são o *Front Controller* e o *MVP* na vertente *Passive View,* respetivamente.

Front Controller

De forma a evitar a centralização da lógica aplicacional numa só classe derivada da *Controller*, foi utilizada a *pattern Front Controller* [27].

A *Front Controller* (Figura 3) permite estruturar uma aplicação de forma a permitir a reutilização e flexibilidade de código, e segmentar o mesmo por múltiplas entidades, reduzindo a complexidade da sua análise. Esta abordagem centraliza todos os pedidos numa classe apenas, tendo esta a responsabilidade de criar e executar uma ação específica, num C*ontroller* específico, de acordo com o pedido efetuado.

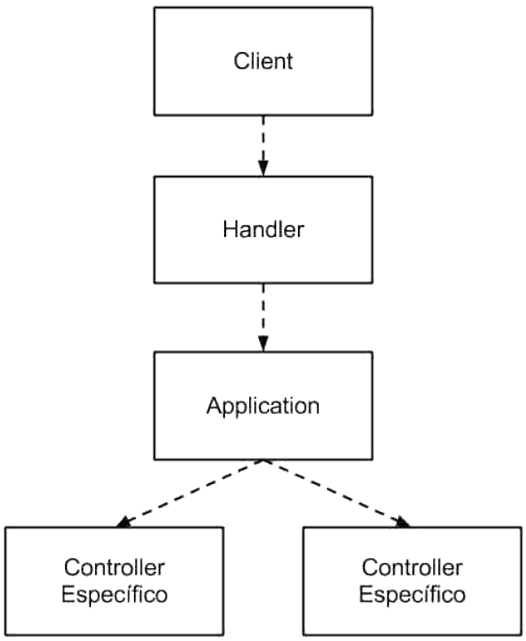

Figura 3 - *Front Controller*

Tendo como base a metodologia utilizada na *Yii Framework* [28], foi concebido um sistema semelhante, no qual todos os pedidos de um cliente são encaminhados para um *handler*, responsável por esta mesma centralização, e que os encaminhará para a aplicação, que por sua vez fará a gestão e adequação de um pedido a um determinado *Controller*. A especificação deste *handler* poderá ser observada com maior detalhe no capítulo "Ciclo de Vida dos Componentes".

MVP *Passive View*

Após a determinação do *Controller* e ação a executar, é dado o início à *MVP Passive View* (Figura 4), a qual irá ser utilizada para fazer a gestão do relacionamento entre classes desta arquitetura.

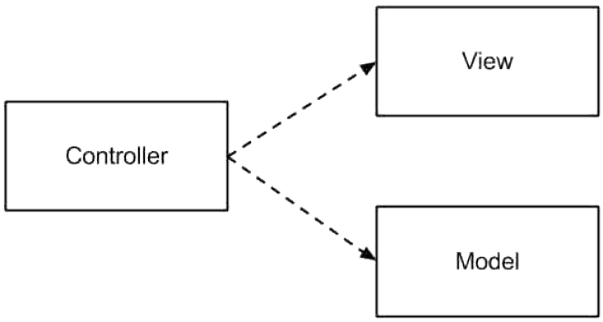

Figura 4 - *Model View Presenter (Passive View)*

Nesta representação a *View* passa a ser considerada um componente passivo, na medida em que deixa de possuir código de ligação ao *Model*, e o *Controller* irá assumir a responsabilidade de atualizar a *View* e *Model* [8].

3.1.5. Ciclo de Vida dos Componentes

Tal como demonstrado na arquitetura desta plataforma, os pedidos feitos por um cliente serão centralizados e geridos da forma mais adequada a esse pedido.

A responsabilidade pela inicialização de toda a plataforma fica então a cargo de dois elementos apresentados na Figura 5, o ficheiro de configuração distribuída, denominado por *.htaccess,* e o ficheiro de raiz do diretório, *root.php*. Estes dois elementos têm um ciclo de vida reduzido e a sua execução é sequencial, ou seja, após a interpretação de cada elemento o resultado obtido é passado ao próximo elemento, e estes deixam de desempenhar qualquer papel até que outro pedido seja feito. Os elementos *App* e *Controller* executam em paralelo, sendo as ações do último despoletadas pelo primeiro, terminando o seu ciclo de vida após o resultado de uma ação ter sido completamente executado.

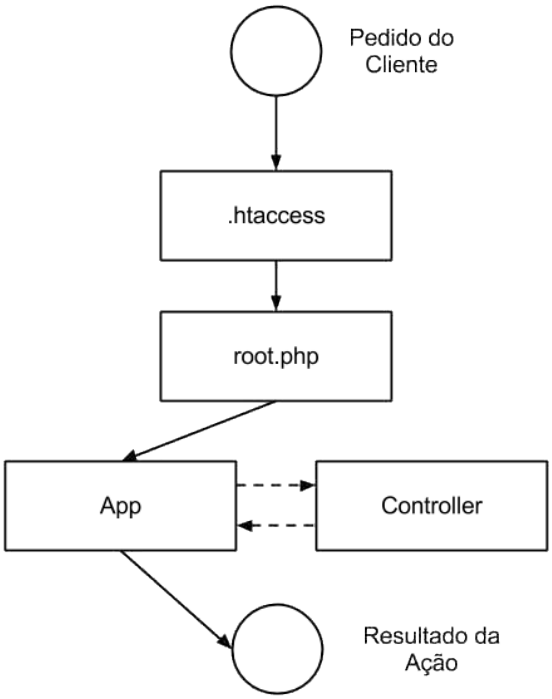

Figura 5 - Ciclo de Vida dos Componentes

.htaccess

O *.htaccess* é um ficheiro de configuração distribuída que provê meios de configuração com base em diretórios. Um ficheiro, contendo uma ou mais diretivas de configuração é colocado num diretório específico, e as diretivas aplicam-se a esse diretório, ou subdiretórios do mesmo [29].

O ficheiro *.htaccess* situado na raiz do projeto contém a seguinte informação:

```
# secure htaccess file
<Files .htaccess>
  order allow,deny
  deny from all
</Files>

# disable directory browsing
Options All -Indexes

DirectoryIndex root.php

Options +FollowSymLinks
RewriteEngine On

#redirect every request to the main file
RewriteCond %{SCRIPT_FILENAME} !-d
RewriteCond %{SCRIPT_FILENAME} !-f
RewriteCond %{REQUEST_URI} (.*)$
```

A ponte entre a interpretação do pedido pelo servidor *Apache* e o ficheiro *root.php*, no qual é definido que o mesmo se tratará do índice do diretório.

As *Rewrite Conditions* [30] definidas informam o servidor que os pedidos feitos neste nível hierárquico de diretorias deverão ser reencaminhados para a raiz do projeto. São exceções a esta condição pedidos a diretórios ou a ficheiros individuais (condições `!-d` e `!-f`, respetivamente). Estas exceções permitem carregar recursos individuais, tais como *scripts*, ficheiros de folha de estilo, imagens ou qualquer outro ficheiro necessário por parte do cliente.

As diretrizes *Files* e *Options* servem como que um mecanismo simples de segurança, impedindo um cliente de navegar no sistema de ficheiros do projeto, ou de visualizar os ficheiros em determinado diretório.

Note-se que o conteúdo deste tipo de ficheiros deverá ser migrado, em fase de produção, para as configurações do servidor principal [29]. Ou seja, o código mencionado anteriormente deverá ser colocado entre as seguintes *tags*:

```
<Directory /deploymentDirectory></Directory>
```

Neste caso, `/deploymentDirectory` deverá corresponder ao diretório onde o projeto estará alojado no sistema de ficheiros do servidor *Apache*.

root.php

O ficheiro *root.php* é responsável por carregar e registar o *AutoLoader,* classe responsável pela ligação dinâmica de classes. Faz parte também das suas responsabilidades registar os diretórios base do projeto, indispensáveis ao bom funcionamento da plataforma, e instanciar

a classe *App*, bem como executar a mesma, o que fará com que se dê o processo de interpretação do pedido.

App e Controller

Estes dois componentes são os responsáveis pela interpretação de cada pedido feito e a sua tradução em artefactos aplicacionais que servirão para estabelecer comunicação entre a aplicação individual e a *framework*. Apresentado na Figura 6 estão os ciclos de vida de ambos componentes.

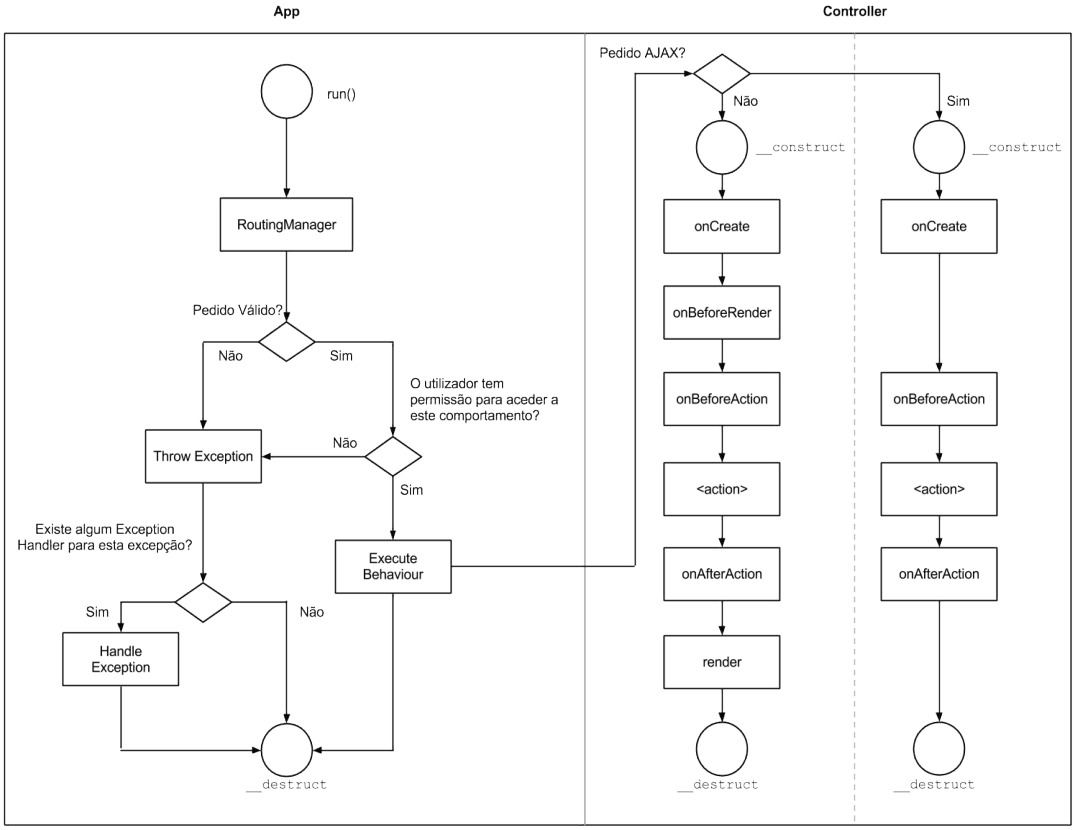

Figura 6 - Ciclo de vida da *App* e *Controller*

A *App,* em conjunto com outras classes, será responsável pela interpretação do pedido proveniente do cliente e por invocar os múltiplos eventos residentes no *Controller*. Esta está segmentada em duas partes fundamentais:

- ***RoutingManager*:** Na inicialização da *App* os dados são enviados para esta classe, que interpreta o pedido, procede à comparação do mesmo relativamente a expressões regulares [31] contidas na classe de configuração da *framework,* e devolve um "comportamento" para a *App*. Este comportamento indica qual o módulo a ser utilizado, qual o *Controller* a ser instanciado, qual a *action* a executar e quais os argumentos a enviar para a mesma. Caso o comportamento devolvido não corresponda a uma classe válida, não possua o método indicado, ou os argumentos do mesmo não condigam com os especificados pelo método, a aplicação lançará uma exceção (*Error404Exception*) a sinalizar um pedido inválido. Caso o pedido seja válido, o mesmo será ainda validado quanto à permissão de acesso por parte do utilizador, lançando uma exceção (*Error403Exception*) a sinalizar a falta de permissão de acesso

nos casos aplicáveis. Em todos os casos em que um comportamento é válido, dar-se-á início ao ciclo de vida do *Controller.*

- ***Exception Handler*:** Um pedido inválido poderá ou não ser tratado por uma classe específica, *Error404Controller* ou *Error403Controller* de acordo com o erro a tratar. Caso não exista nenhuma classe para gerir a exceção, a própria aplicação tratará a mesma, mostrando-a no ecrã, caso a configuração do projeto indique que o mesmo não está em produção, ou caso contrário, terminando a sua execução sem produzir nenhuma mensagem. Em qualquer dos casos será produzido um *status code* [32] de acordo com o tipo de erro produzido, seguindo as boas práticas de *Search Engine Optimization* (SEO) [33].

Após a instanciação do *Controller* é disparado um conjunto de eventos sequenciais que poderão ser geridos através de *overriding* de métodos nas classes que herdem desta. A sequência de métodos invocados alterar-se-á dependentemente do tipo de pedido que for feito, sendo que um pedido com indicação de AJAX [34] não invocará métodos que estejam relacionados com interpretação de *Views* ou *render*, de forma a otimizar o tempo de resposta. Deste componente fazem parte:

- ***onCreate***: Invocado durante a criação do *Controller*, uma vez que este processo é feito automaticamente pela aplicação, este método de conveniência poderá ser utilizado como alternativa ao construtor real da classe;
- ***onBeforeRender***: Este método é invocado apenas em pedidos que não tenham indicação como sendo feitos por AJAX. É executado previamente à ação e a qualquer preparação relacionada com a apresentação de componentes visuais;
- ***onBeforeAction***: Método invocado imediatamente antes da execução de qualquer ação;
- ***action***: O método invocado neste ponto corresponderá à ação indicada no pedido feito por *HTTP*;
- ***onAfterAction***: Método invocado imediatamente após a execução de uma ação;
- ***render:*** Num pedido não identificado como AJAX, o método *render* será invocado com o intuito de obter todos os componentes visuais tratados durante o ciclo de vida e a integrar os mesmos no documento final que constitui a componente visual de uma aplicação. Na cadeia de eventos constituinte do ciclo de vida do *Controller*, este é o único evento que não permite ser reescrito.

3.2. Modelação

No decorrer das diversas iterações que constituem o projeto aqui apresentado, foi produzida toda uma documentação de apoio, bem como os diagramas fulcrais ao acompanhamento, manutenção e utilização desta *framework*. Estes artefactos foram alvo de constante reavaliação e refinação, sendo produzidas novas esquematizações para cada iteração produzida até ao fecho deste projeto.

Os diagramas apresentados pretendem ser representativos do funcionamento da *framework,* tendo sido escolhidos de acordo com o tipo de projeto, focando-se completamente na sua natureza.

3.2.1. Diagrama de Componentes

Os componentes existentes na *framework* estão segmentados em 3 grandes grupos, como é possível observar na Figura 7, os componentes base, contidos no grupo *Core*, os componentes de cada aplicação específica, contidos no grupo *Web App,* e os componentes de configuração dessa aplicação, contidos no grupo *Config*.

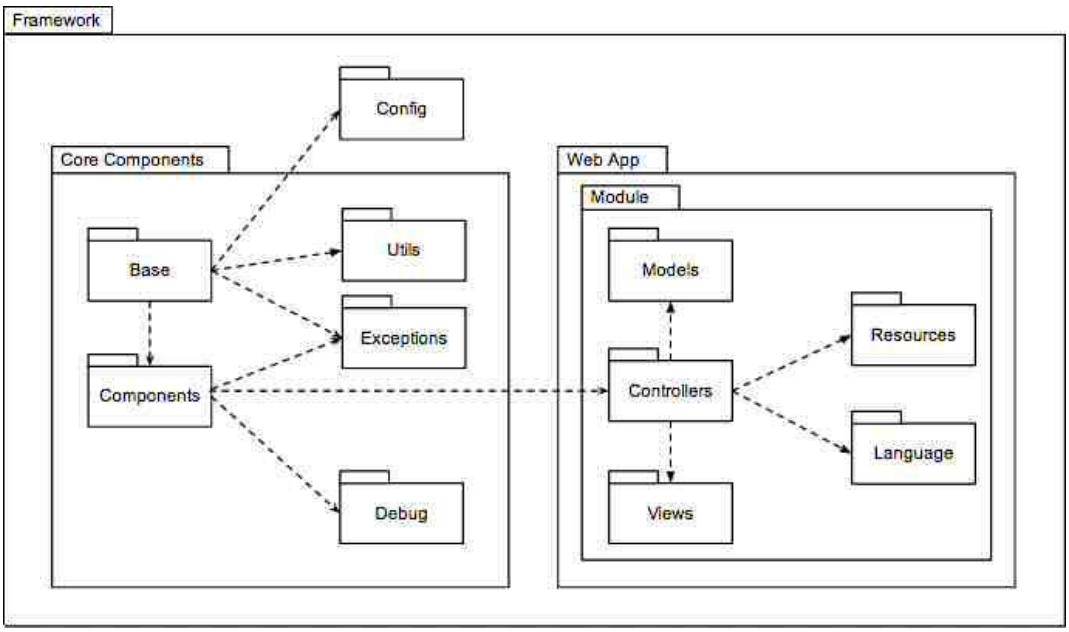

Figura 7 - Diagrama de Componentes

De seguida apresenta-se cada um destes grupos com maior detalhe.

Base Package

Neste grupo estão contidos os elementos base que constituem a *framework*. Esta está segmentada em subgrupos de funcionalidades específicas.

- ***Base***: Deste grupo fazem parte todos os elementos constituintes do ciclo de vida da *App*. Elementos estes que constituem a base da *framework*, ainda que não sendo parte integrante do conceito MVC, são os mesmos os responsáveis pela criação desta estrutura, pela validação de acessos, pela interpretação de pedidos HTTP e pelo tratamento de pedidos inválidos;
- ***Components***: Neste grupo estão contidos todos os elementos constituintes da MVC, bem como outros componentes adicionais, tais como os de suporte a modelos e criação do documento *web* base;
- ***Debug***: Neste grupo estão contidas ferramentas de depuração de código, adequadas para o uso em ambientes no qual não seja possível depender do ***debugger*** do IDE de desenvolvimento;
- ***Exceptions***: Neste grupo estão contidos um conjunto de classes, diretamente descendentes da superclasse *Exception*, que assinalam um conjunto de exceções específicas, úteis no processo de desenvolvimento de cada projeto;

- ***Utils***: Este grupo contém ferramentas utilitárias, que embora não constituam parte integrante do núcleo desta *framework*, poderão ser utilizadas em conjunto com o mesmo, alargando as possibilidades da mesma e expandindo o seu conceito básico a conceitos mais complexos, capazes de fornecer resposta a projetos mais exigentes em termos de funcionalidades.

Web App Package

Este grupo é constituído por um ou mais módulos, correspondentes a partes funcionais da aplicação específica a desenvolver sobre esta *framework*. Cada módulo deverá reproduzir a estrutura indicada no diagrama apresentado na Figura 7, de forma a possibilitar à *framework* o seu uso de uma forma natural e automatizada.

- ***Controllers***: Este grupo deverá conter todos os *Controllers* de uma aplicação;
- ***Models***: Neste grupo deverão estar contidos todos os modelos referenciados pelo módulo;
- ***View***: Neste grupo deverão estar contidas as várias V*iews* do módulo, segmentadas por subdiretórios, em que cada deverá reproduzir o nome do *Controller* à qual está associado. Dentro de cada subdiretório deverão estar contidos os conjuntos de *Views* associados ao módulo. Excecionalmente, V*iews* que sejam comuns a todo o projeto (e.g. *layouts*) poderão situar-se na base do grupo, i.e. uma *View* que pertença um *Controller* deverá estar situada numa diretoria que possua o mesmo nome que esse *Controller*, no entanto, de forma a permitir a reutilização de *Views*, caso esta seja comum a *Controllers* distintos, poderá estar situada na raiz da directoria que contém todos os diretórios de *Views*;
- ***Resources***: Neste conjunto deverão estar contidos recursos adicionais ao projeto, tais como scripts e folhas de estilo. Cada *Controller* possibilita a adição de um ou mais recursos desta categoria, que poderão ser externos ou internos. No último caso apresentado, o *Controller* irá automaticamente procurar os recursos neste grupo, simplificando a tarefa de inclusão dos mesmos uma vez que existe uma abstração do *Uniform Resource Identifier* (URI) necessários para a indicação da localização do ficheiro;
- ***Language***: A componente *Language* é obrigatória apenas em projetos que usem recursos multi-idioma. Caso o projeto se enquadre nesses moldes, deverão existir subcomponentes dentro deste, no qual cada um deverá ser identificado de acordo com os códigos de país especificados como sendo suportados nos ficheiros de configuração da aplicação.

Note-se que esta estrutura, contrariamente à presente no conjunto de componentes base, não é completamente imutável, no sentido em que possibilita agrupar classes comuns a módulos distintos, com uma simples invocação da classe *AutoLoader* no sentido de adicionar um grupo comum ao conjunto de diretórios de referência de classes.

Config

Neste grupo situam-se as diversas classes de configuração da *framework*.

3.2.2. Diagrama de Classes

O diagrama de classes apresentado na Figura 8 representa o relacionamento entre as diversas classes do projeto. Neste diagrama foram omissas classes de utilitários e de exceções, visto que os mesmos, embora fazendo parte deste projeto, não constituem uma parte do funcionamento do seu núcleo.

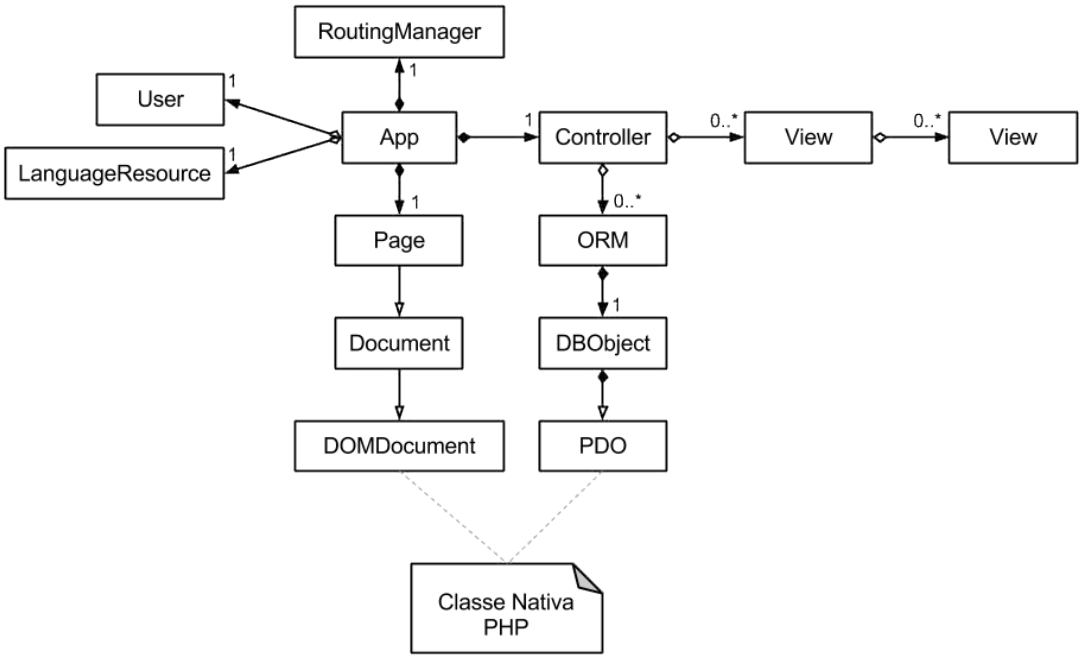

Figura 8 - Diagrama de Relacionamento entre Classes

Descreve-se de seguida, de uma forma geral, as responsabilidades de cada classe:

- ***App*:** A *App* centraliza todos os comportamentos da aplicação. Todos os pedidos de um cliente serão tratados, validados e resolvidos em componentes lógicos nesta classe. Esta faz uso de diversas outras classes para possibilitar o alcance dos seus objetivos:
    - *Routing Manager:* Utilizada para fazer a interpretação de pedidos do cliente;
    - *Page*: Utilizada para fornecer uma representação visual, resultante das ações do cliente;
    - *User*: Utilizada para fazer a verificação de permissões de acesso de determinado utilizador a ações específicas;
    - *Language Resource*: Utilizada na interpretação de idioma proveniente do cliente e gestão de idiomas da aplicação.
- ***Controller*:** Esta classe centraliza toda a lógica de uma aplicação. A forma e nome como é instanciada dependem da interpretação do pedido do cliente e das regras definidas nos ficheiros de configuração. Cada *Controller* deverá agregar conteúdo lógico respetivo a uma parte da aplicação, logo, deverão ser criados novos *Controllers* que herdem o comportamento desta classe e que representem uma secção agregada da aplicação. Este elemento poderá possuir inúmeras *Views*, as

quais serão uma representação visual de uma determinada ação, ou não possuir nenhuma *View* em casos em que o *Controller* não lide com elementos gráficos (e.g. pedidos por *AJAX*);

- ***View*:** A *View* é responsável pela conversão de documentos HTML ou PHP para um formato completamente orientada a objetos. Esta classe fornece ainda meios de simplificação à manipulação de elementos internos a cada *View*. Cada *View* poderá conter outros elementos da sua classe, os quais, após serem anexados, serão fundidos com os elementos já existentes na *View* principal;
- ***ORM*:** O ORM representa um *Model* por omissão. Este fornece meios simplificados para ligações a base de dados, utilizando uma classe de configuração da mesma. Por se considerar que este tipo de modelos seria possivelmente o mais complexo de implementar, foi criada uma integração com a *framework*, como forma de simplificação deste tipo de implementação. Por este motivo, é parte integrante desta classe, uma instância da classe *DBObject*. Cada *Model* específico deverá ser uma *subclasse* da *ORM*, de forma a herdar o seu comportamento. Se o *developer* pretender, poderá utilizar uma implementação proprietária, sem que tal cause qualquer impacte na plataforma;
- ***DBObject*:** A classe *DBObject* estabelece uma relação simples entre o *ORM* e uma base de dados. Esta classe é uma especialização da classe PDO [35], utilizando ainda os ficheiros de configuração de *framework* para simplificar os acessos a bases de dados;
- ***PDO*:** Esta classe, presente nativamente nas bibliotecas de PHP a partir da versão 5.1, fornece meios de comunicação com bases de dados, utilizando objetos de dados para a sua representação. Esta fornece ainda mecanismos de prevenção contra injeção de SQL, *prepared statements*, e diversas estruturas lógicas de dados para a representação dos resultados de cada operação efetuada;
- ***Page*:** Esta classe estabelece um relacionamento entre a *App* e as representações dos elementos visuais destinados ao cliente. O conteúdo visual criado em cada *Controller*, e fornecido pela *App* aquando o término deste, é tratado de forma a construir uma representação gráfica destes elementos. Esta classe é uma especialização da classe *Document;*
- ***Document*:** A classe *Document* estabelece uma relação muito próxima com a classe nativa PHP, *DOMDocument* [36]. A função desta classe é fornecer meios de configuração do documento final, que será apresentado ao cliente. Esta lida com elementos como o título de página, *metatags e encoding* do documento, além de fornecer meios às suas subclasses, para converterem objetos do tipo *View* em *DOMNodes* [37];
- ***DOMDocument*:** Esta classe nativa de PHP fornece meios de construção e manipulação de documentos *web*;
- ***User*:** A *User* representa um utilizador do sistema. Esta está fortemente ligada com a *App* e o *Controller,* uma vez que a primeira faz um pedido à segunda de forma a obter informações sobre a operação pedida por um cliente. A informação da operação devolvida pela App, normalmente constituindo num nome da permissão

necessária, é então passada para a *User,* que deverá fazer a validação da permissão de acesso*.* Uma aplicação que faça uso do conceito de utilizadores e níveis de acesso deverá possuir um modelo de *User* que representem os utilizadores dessa aplicação, herdando os comportamentos da classe *User*. Desta forma, a implementação do mecanismo de verificação de permissões deverá ficar a cargo de cada subclasse, adaptando assim o conceito de utilizador às necessidades de cada aplicação;

- ***LanguageResource*:** A *LanguageResource* é uma classe utilitária, utilizada pela *App* para determinar o idioma do cliente. Esta classe fornece métodos para a obtenção de recursos de idioma estáticos, definidos nos recursos do projeto em diretórios identificados com o código do país que os quais representam.

Cada classe poderá ser observada com maior detalhe no Anexo I, onde é apresentada a documentação produzida relativamente às mesmas.

3.2.3. Diagramas de Sequência

Os diagramas de sequência apresentados neste capítulo são representações de todo o fluxo de funcionamento da *framework*, desde o processamento de um pedido até às fases de materialização desse pedido numa resposta.

Note-se que, embora os pedidos sejam iniciados por um cliente, o mesmo foi omisso dos diagramas por motivos de simplicidade da esquematização de eventos, uma vez que o cliente apenas faz iniciar o processo através de um pedido HTTP, terminando a sua interatividade com o sistema nesse instante.

Processar Pedido

Ao receber um pedido, a *Root* irá encarregar-se de criar uma instância da *App* e executá-la. Esta irá obter uma instância da *RoutingManager* e encaminhar-lhe o pedido para que esta o valide e retorne sobre a forma de um conjunto de comportamentos. Os comportamentos devolvidos à *App* serão então validados. Na Figura 9 é possível observar este comportamento.

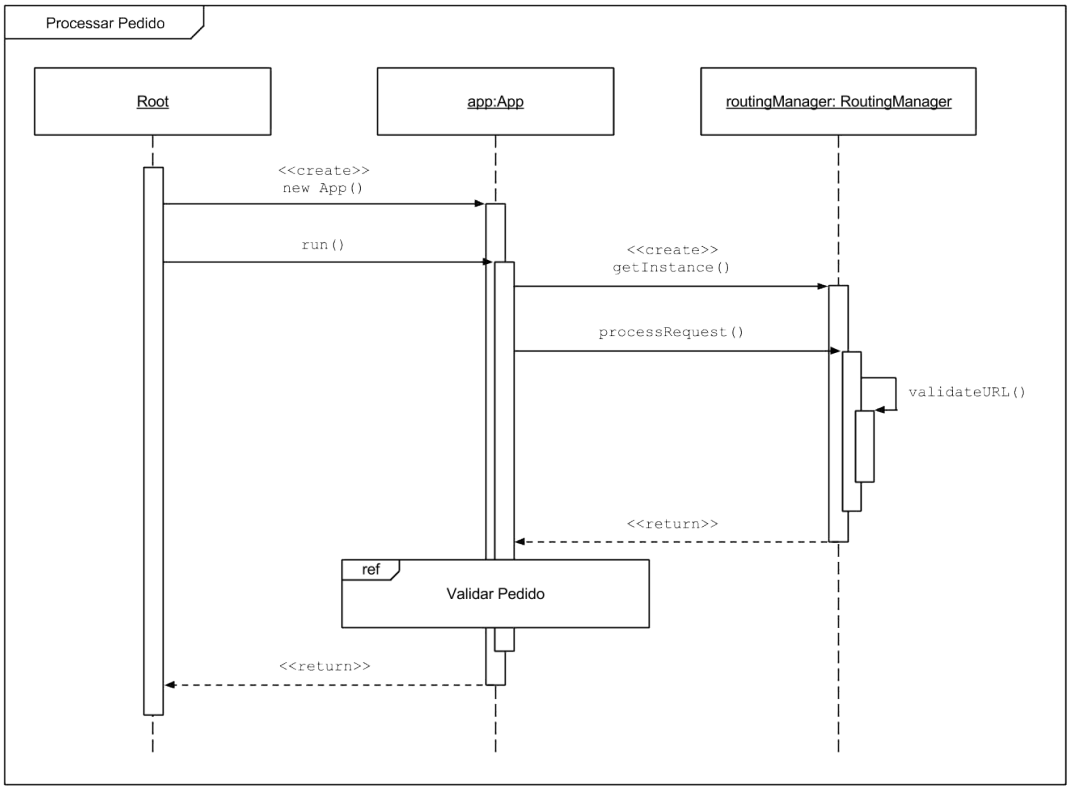

Figura 9 - Diagrama de Sequência – Processar Pedido

Validar Pedido

A validação de um pedido consiste na tentativa de instanciação de um *Controller*, por parte da *App.* Caso o pedido não corresponda a nenhum elemento válido será lançada uma exceção *Error404Exception*, que poderá ser tratada através do mecanismo de gestão de exceções.

Nos casos em que o pedido é válido, é dado lugar à validação de permissões por parte do utilizador autenticado no sistema. No fim do processo de validação, caso exista um *Controller* com um pedido válido, o mesmo será executado. O diagrama apresentado na Figura 10 demonstra a validação deste pedido.

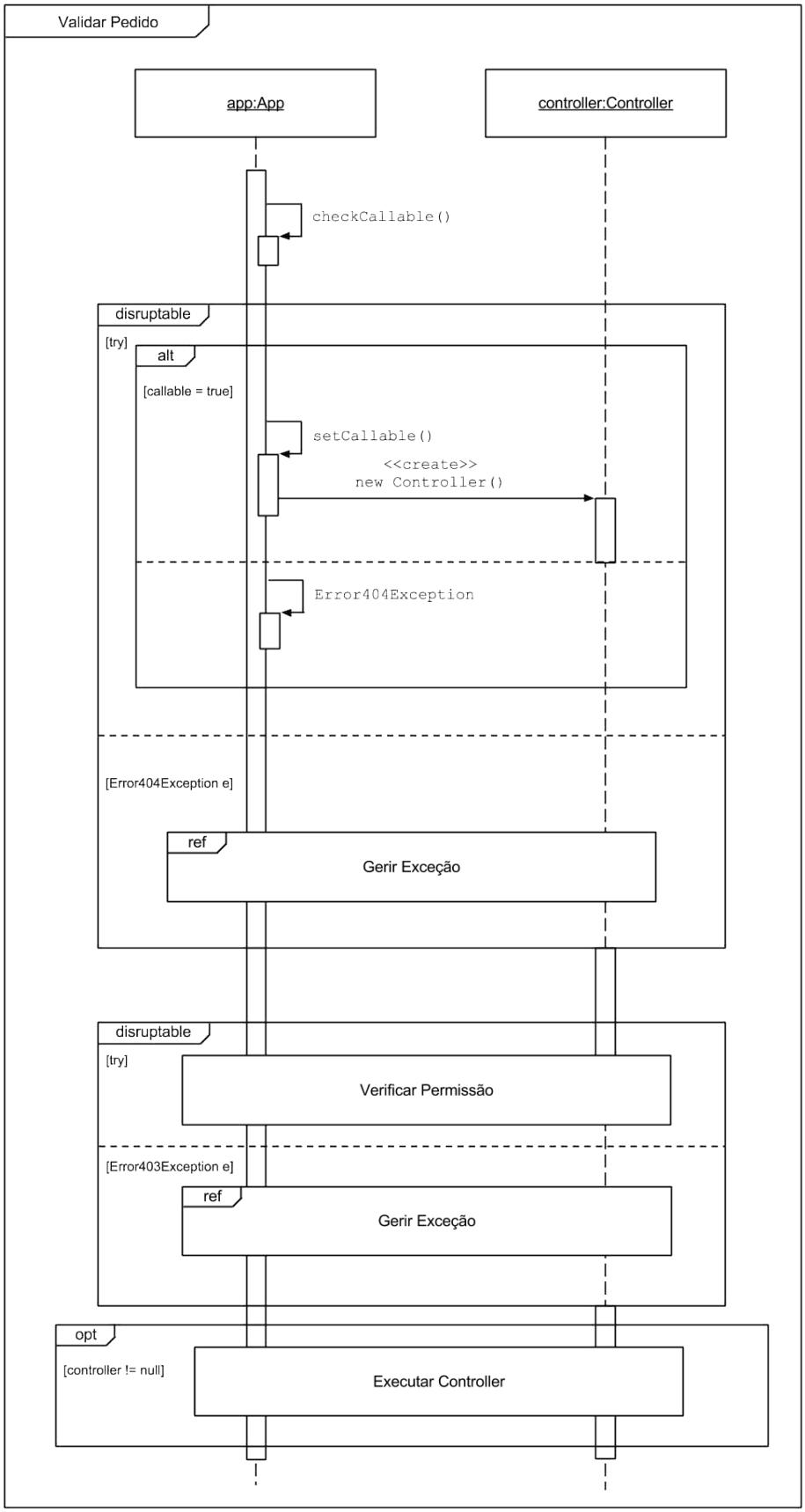

Figura 10 - Diagrama de Sequência – Validar Pedido

Verificar Permissão

Na verificação de permissão de execução de uma ação por parte de um utilizador (Figura 11) é verificada a existência de um modelo de utilizador, caso este não exista, este processo de validação de permissão é ignorado. A autenticação é feita através da obtenção da permissão de acesso necessária para executar determinada ação e a comparação com as permissões do utilizador. Se o utilizador não possuir permissões para executar a ação, é lançada uma exceção *Error403Exception*.

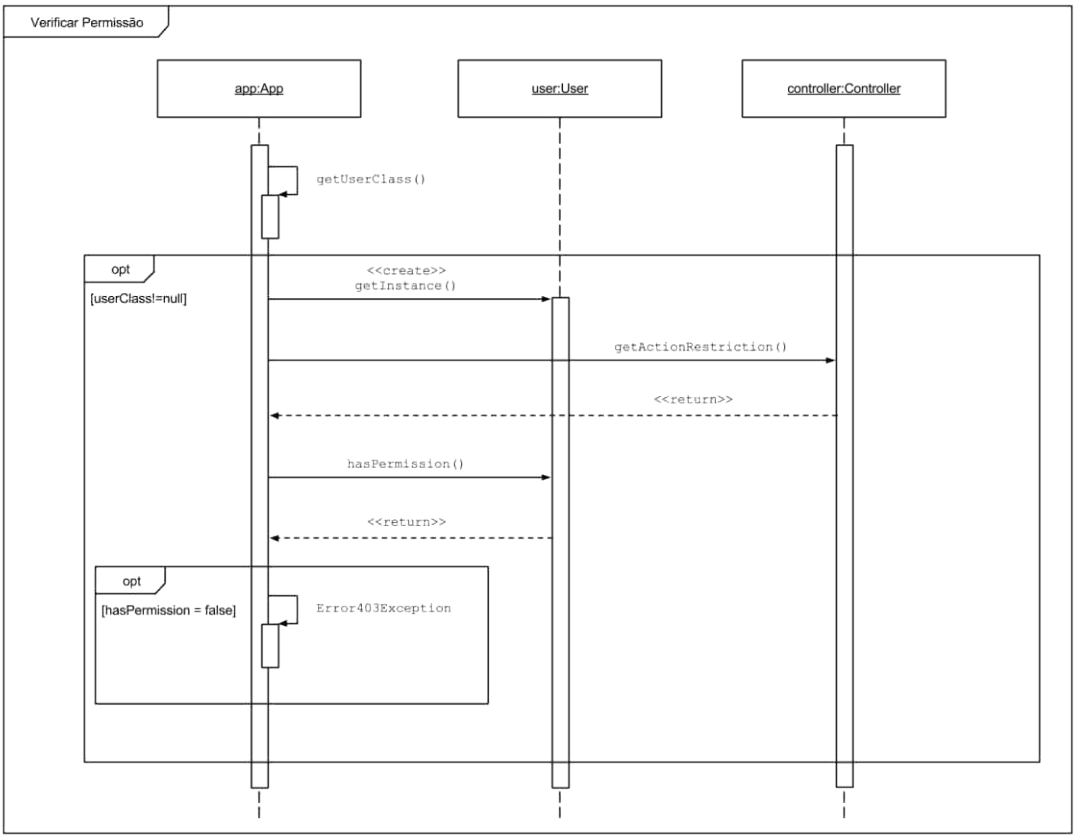

Figura 11 – Diagrama de Sequência - Verificar Permissão

Gerir Exceção

No caso de uma exceção ter sido lançada no decorrer do processo de validação, será feita uma tentativa de conversão do tipo de exceção num *Controller* definido pelo programador, no qual a exceção será tratada. Este processo pode ser observado na Figura 12.

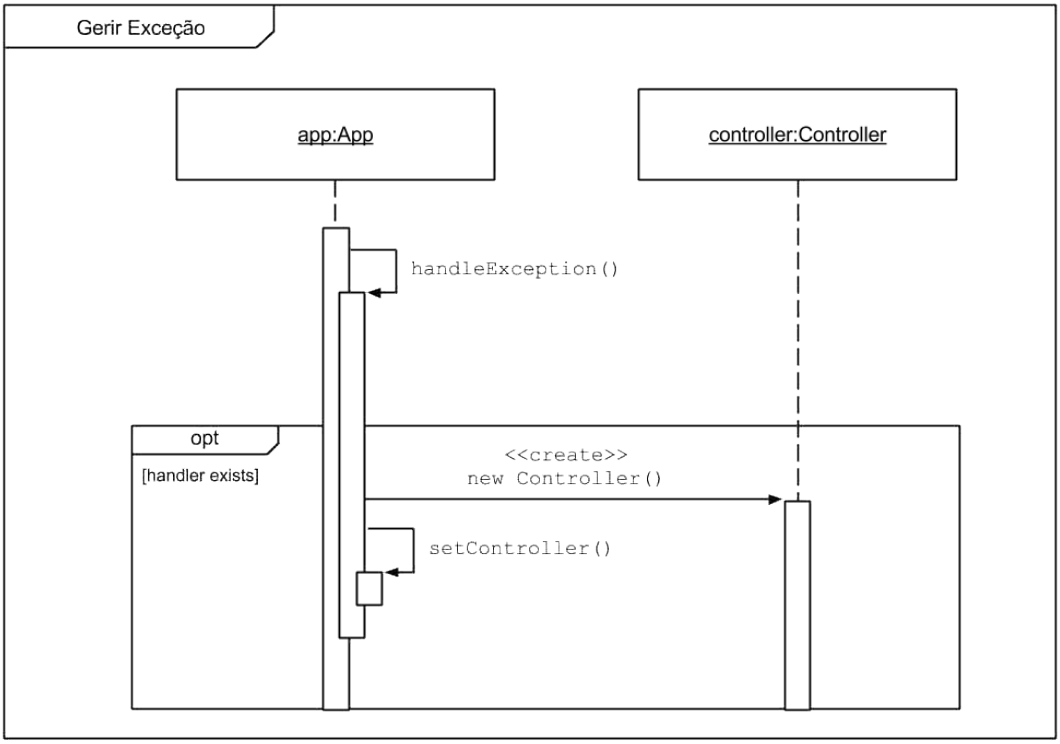

Figura 12 - Diagrama Sequência – Gerir Exceção

Executar *Controller*

A execução do *Controller*, apresentada na Figura 13, conta com diversos pedidos efetuados entre a *App* e o *Controller* específico, esta interação poderá ser vista em maior detalhe na secção 3.1.5 deste documento.

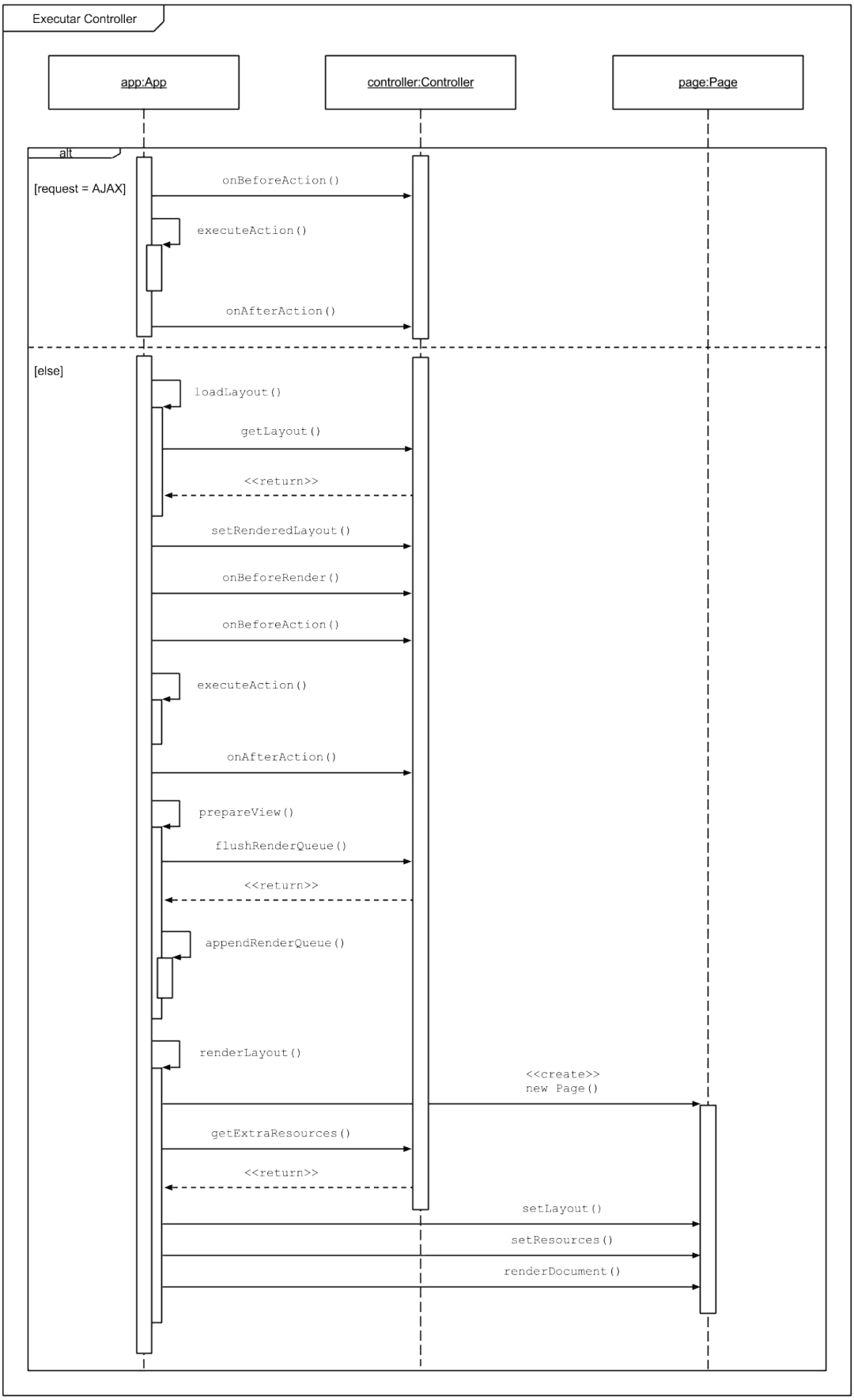

Figura 13- Diagrama de Sequência - Executar *Controller*

Render View

A interação de *Render View* é dada no interior do componente *View*, aquando existe um pedido de avaliação e interpretação de um recurso que contenha elementos visuais (e.g. um ficheiro HTML). Neste caso é feito um pedido à *View* por parte do *Controller*, para que esta processe o recurso especificado. Este processo pode ser analisado com maior detalhe na secção 3.3.6. O diagrama na Figura 14 apresenta este processo em maior detalhe.

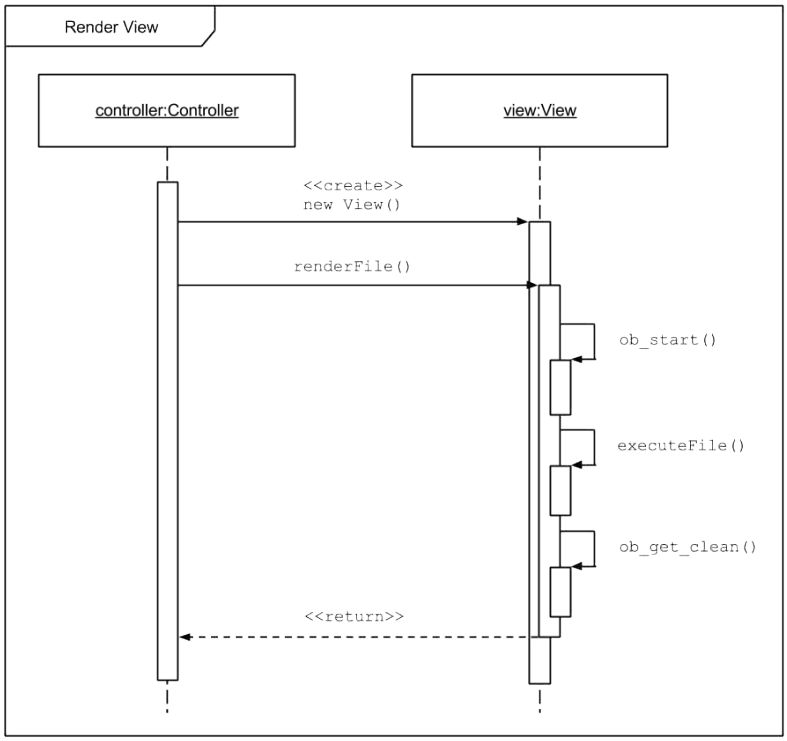

Figura 14 - Diagrama de Sequência - *Render View*

3.3. Outras Características

A *framework* concebida possui outras características que, direta ou indiretamente, ajudam a definir o seu funcionamento e objetivo.

3.3.1. SEO *Uniform Resource Locator* (URL)

A utilização de *SEO URLs* permite a representação da estrutura lógica de uma aplicação *web* de uma forma intuitiva e simples, através da utilização de palavras como forma representativa das secções da aplicação, residindo a sua maior vantagem no fato de tornar uma aplicação relevante em motores de pesquisa.

A utilização do conceito neste projeto demonstra as suas vantagens na transparência com que elementos de código são criados mediante os pedidos feitos por um utilizador, num processo em que o mesmo se encontra abstraído deste fator, além de possibilitar todas as características inerentes ao conceito de SEO.

3.3.2. Auto Loading

Com o recurso a técnicas de *Auto Loading* possibilitou-se a recriação de um ambiente de programação muito semelhante a linguagens com o Java, nas quais existe a possibilidade de fazer referências lógicas entre dependências nas classes constituintes de um programa. Com

recurso a esta funcionalidade, inserida a partir da versão 5 do PHP, foi possível criar um gestor de referências, denominado por *AutoLoader,* que dinamiza o processo de criação de uma aplicação, no qual as dependências entre classes são automaticamente criadas, mediante a necessidade das mesmas, sendo apenas necessário em casos considerados mais extremos a adição de uma nova *package* ao *AutoLoader*. As dependências básicas (i.e. *Controllers*, *Views* e *Models*) são, por omissão, adicionadas de forma automática a cada novo pedido feito ao servidor, sendo as mesmas adequadas ao pedido em si, de forma a otimizar o seu processamento.

3.3.3. Integração com *Document Object Model* (DOM)

Como medida auxiliar de construção de elementos visuais, bem como com o intuito de manter o projeto o mais próximo possível do paradigma de POO, os mesmos são construídos sobre o DOM, tirando partido de todas as suas potencialidades, ao mesmo tempo que possibilita a expansão de cada elemento. A integração do DOM propagou-se por 3 componentes, fulcrais neste âmbito:

View

Este componente manipula documentos *Hypertext Markup Language* (HTML) [38], fornecendo métodos auxiliares sobre os mesmos, permitindo a integração com outras *Views*, além de possibilitar a manipulação direta do documento através da exposição dos *DOMNodes* que o constituem.

Page

Este componente é responsável pela integração das diversas *Views* num único documento, integração de bibliotecas adicionais, folhas de estilo e scripts.

Document

O componente *Document* integra o componente *Page*, criando uma base estruturada de um documento de HTML típico, com possibilidade de definição de características típicas do mesmo, tais como título e codificação de caracteres. Este elemento herda os comportamentos do componente *DOMDocument*, presente nativamente no PHP, tornando-se assim a base de qualquer documento e o último elemento a ser processado na hierarquia criada para este projeto.

3.3.4. Escalabilidade

A aplicação do paradigma de POO tornou-se bastante valiosa neste contexto, uma vez que através da correta utilização das características intrínsecas ao paradigma, e a aplicação de *design patterns*, torna-se possível a extensão e adição de funcionalidades de cada componente. Esta característica torna-se bastante útil aquando a necessidade de se construir novos componentes, ou alterar o seu comportamento, sem a necessidade de alterar os componentes já existentes, constituintes do núcleo do projeto. Por esta possibilidade de adaptação da versão existente da plataforma a novas versões, com novas características, ou a projetos específicos, tornam esta plataforma altamente modular, e por consequência, escalável e flexível.

3.3.5. Facilidade de Utilização

A preocupação constante com a aceitação da plataforma por parte do seu público-alvo, outros *developers*, e com a simplicidade de criação de novas aplicações assentes sobre a mesma, levaram à criação de uma plataforma simples e fácil de utilizar, focando a tarefa do programador à construção dos componentes presentes num modelo MVC típico. As diversas camadas de abstração concebidas permitem afastar o programador de código de mais baixo nível, e focá-lo na produção de código referente ao comportamento, aos componentes lógicos e ao aspeto gráfico da sua aplicação, abstraindo-o das tarefas de transições entre *Controllers*, ações e outro tipo de código considerado como base fundamental para qualquer aplicação.

3.3.6. Render Engine

O sistema de transformação de um ficheiro HTML / PHP numa *View*, embora simples, prova ser uma ferramenta bastante útil e poderosa.

A utilização de uma metodologia de inclusão de ficheiros com recurso a *output buffering* [39] permite, para além da conversão de um ficheiro sem produzir qualquer *output* para o cliente, a inclusão de código PHP dentro do próprio ficheiro a converter. Neste caso torna-se possível a passagem de valores entre *Controller* e *View*, permitindo assim seguir um modelo MVC em detrimento da MVP imposta pela *framework*. Um diagrama ilustrativo deste processo pode ser observado na Figura 14.

3.3.7. Suporte de múltiplos idiomas

Dado que uma das vantagens do desenvolvimento de recursos *web* é a sua possível divulgação a nível mundial, foi incluído, juntamente com os outros componentes de suporte ao *software* desenvolvido, uma classe com capacidades para gestão de ambiente que possuem múltiplos idiomas, sendo que a utilização deste componente é opcional.

Na presença de um ambiente multilingue, a plataforma está apta a comparar um tipo de língua que seja indicado através de um URL com os tipos de língua suportados, presente nas configurações da aplicação. Neste tipo de eventos, a plataforma comutará automaticamente os atributos internos da componente de idioma de forma a tornar a leitura deste tipo de recursos o mais transparente possível para o programador, sendo estes acedidos através de uma interface única, sem que exista a necessidade de preocupação com a determinação do idioma ativo para o fazer corresponder a um recurso específico.

3.3.8. *Role Based Access Control* (RBAC)

A necessidade de um controlo de acesso a módulos, grupos de ações, funcionalidades ou operações, é uma realidade constante no paradigma de desenvolvimento de aplicações multiutilizador [40]. Como forma de resposta a esta necessidade, foi integrada na plataforma a filosofia de controlo de acessos baseada em cargos. Contrariamente ao paradigma de controlo de acesso baseado em utilizadores, este permite a atribuição de operações simples, correspondentes ao nível com maior granularidade, a *roles*, correspondentes ao nível de menor granularidade, e a associação de ambos a um ou mais utilizadores [41]. Com esta abordagem torna-se possível detalhar o papel de cada utilizador numa aplicação, tornando a sua envolvência o mais próximo possível das suas competências dentro de uma unidade organizacional.

A implementação deste conceito conta com duas partes chave, a classe *Controller* e a classe *User*, das quais a definição última ficará completamente dependente do tipo de implementação que for pretendida para cada projeto específico:

- ***Controller*:** A classe *Controller* possui um método, denominado por *getAccessRules*, que deverá retornar um conjunto de ações associadas a um tipo de utilizador ou permissão. Mediante as permissões necessárias para cada ação, será permitida, ou negada, a autorização para a execução das mesmas por parte de um utilizador;
- ***User*:** O método *checkAccess* da classe *User* será invocado a cada ação realizada, ou se for invocada uma verificação manual à mesma. Este método irá determinar se o utilizador tem ou não permissão para executar determinada ação. A implementação deste método poderá ser adequada a cada projeto em curso, ou seja, não está imposta uma metodologia de validação. A adoção desta estratégia permite a livre escolha da tecnologia ou metodologia utilizada, possibilitando a utilização de ligação direta a bases de dados, *web services*, ou métodos de validação estáticos.

Note-se que, tal como referido na apresentação do contexto deste projeto, estes componentes, ainda que integrados neste trabalho, constituem uma parte opcional do mesmo, significando isto que, caso a especificidade de um projeto em causa assim o determine, estes componentes poderão ser ignorados com segurança, fator que não causará qualquer constrangimento no normal funcionamento da *framework*.

4. Otimização, Testes e Documentação

Durante o processo de criação deste trabalho houve especial atenção em produzir uma ferramenta direcionada para a utilização por parte de um público geral. Baseado neste objetivo foram feitas otimizações de código durante o desenvolvimento, testes ao código produzido no fim desta mesma fase, e produzida uma documentação rica e descritiva das classes criadas.

4.1. Otimização de Código

De forma a tornar o código produzido o mais otimizado possível, foram utilizadas duas ferramentas existentes para este propósito, que acompanharam o desenvolver dos diversos componentes criados:

- ***PHP Code Sniffer*:** Utilizado com o fim de adequar o código produzido a normas e convenções existentes, auxiliando também na produção da documentação que acompanha o mesmo;
- ***PHP Mess Detector*:** Utilizado na deteção de código inatingível, *loops* complexos e condicionais não otimizadas.

Através da utilização destas duas ferramentas garantiu-se que o código da *framework, utilizado* nas fases finais de teste não possui nenhum constrangimento relativamente a código mal produzido, permitindo focar o mesmo na resolução e otimização de comportamentos, e não no código base que constitui estes comportamentos.

4.2. Testes

4.2.1. Ambiente de testes

Os testes apresentados foram efetuados num servidor *Apache* com o PHP 5.4, instalado num computador com o sistema operativo *OS X 10.8.3* com um processador *Core i5* de 2.4GHz e 4Gb de RAM.

Estes testes tiveram como objetivo a averiguação de pontos em que existissem perdas de desempenho devido a processamento indevido, causado por possíveis falhas de otimização do código da *framework*. Para este efeito foi utilizado o *profiler* presente no *Xdebug* para a recolha de informação sobre cada pedido executado na plataforma. Os ficheiros *cachegrind* gerados em cada execução foram posteriormente analisados na ferramenta *Kcachegrind*, servindo esta para interpretar a informação recolhida, apresentando-a sob a forma de gráficos e tabelas.

O teste realizado consistiu na reprodução do exemplo criado para efeitos de demonstração, presente na secção 4.4, uma vez que o mesmo cobre o funcionamento do núcleo da *framework*.

4.2.2. Resultados dos testes

Após a avaliação da aplicação foi obtido o grafo presenta na Figura 15.

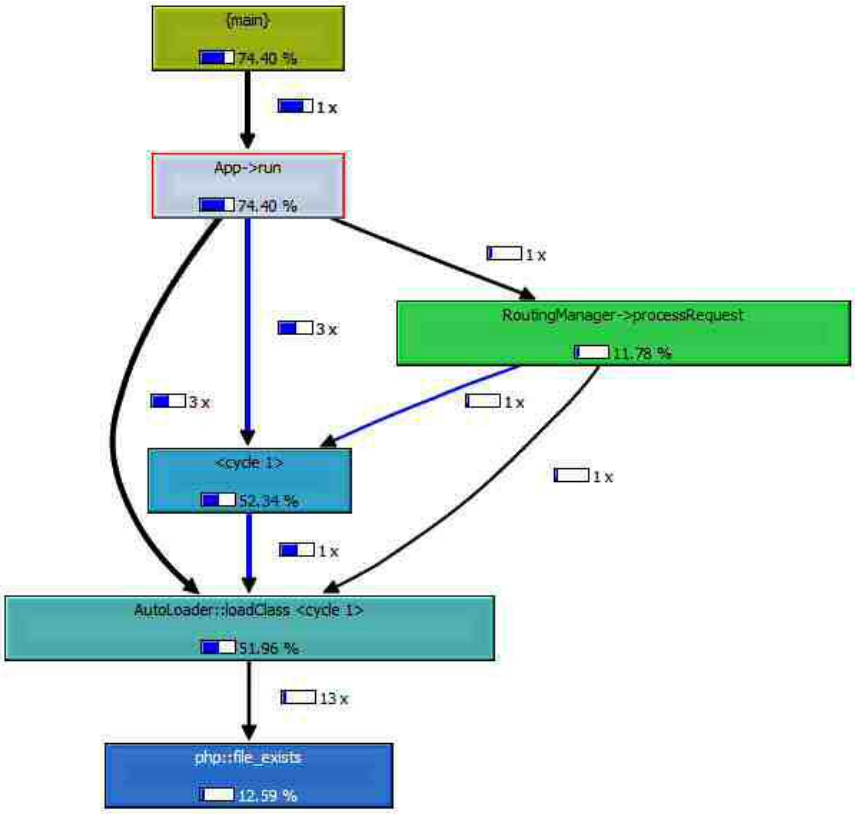

Figura 15 – Primeiro *Profiling* da Aplicação

Conforme é possível observar, grande parte do processamento está centralizado no método *run*, uma vez que é a partir do mesmo que é executado todo o código fulcral ao núcleo de cada aplicação. Neste mesmo método, pode-se observar que a utilização do *AutoLoader* consome cerca de 51.96% do processamento total efetuado. Após inspeção da classe *AutoLoader*, foi possível concluir que o sistema de localização de classes no sistema de ficheiros era o comportamento causador deste consumo. De forma a resolver este problema foi implementado um método de *cache* da localização de classes.

Como é possível observar na Figura 16, com uma nova análise da aplicação, já com o sistema de *caching* implementado, reduziu o tempo despendido no *AutoLoader*, reduzindo-o de 51.96% para 31.56%, aumentando assim o desempenho da *framework*.

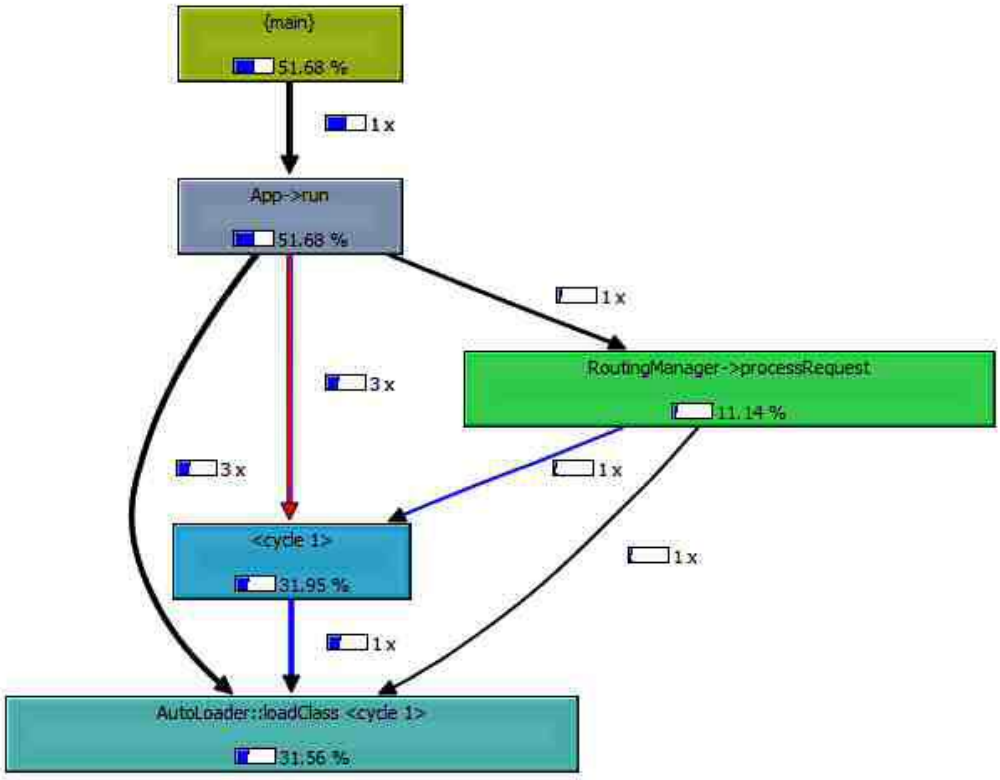

Figura 16 – Segundo *Profiling* da Aplicação

4.2.3. Comparação com outras plataformas

Como forma de comparação, o exemplo utilizado nesta sessão de testes foi recriado nas diversas *frameworks* referenciadas no capítulo 2 e posteriormente executado, de forma a extrair informação sobre o desempenho de cada uma. Para este efeito foram utilizadas as funções *sys_getloadavg()*, *memory_get_usage() e getrusage(),* nativas ao PHP. Os resultados obtidos podem ser visualizados na Tabela 2, onde se apresenta uma média obtida após 10 execuções do exemplo em cada plataforma, nas quais se obteve informação sobre a carga do processador, a memória RAM utilizada e o tempo despendido no processamento da aplicação.

Tabela 2 - Resultados de Teste de Desempenho

Framework	System Load Average	RAM (KB)	Tempo Despendido (ms)
CakePHP	0.836281	502.21	9
CodeIgniter	0.919202	13921.12	12
Yii	0.981933	2277.82	11
Zend	0.990023	31022.02	13
Framework Desenvolvida	0.784179	325.87	10

É possível concluir, a partir da análise dos resultados obtidos nos testes, que o fato de esta *framework* possuir poucas dependências de classes internas e possuir uma biblioteca sem dependências de outros recursos externos ajudam no seu desempenho.

4.3. Documentação

Durante o processo de desenvolvimento foi produzida uma documentação cuidada de cada classe, método e variável, de forma a possibilitar a fácil manutenção e interpretação de código. Para este efeito foi utilizada a convenção de anotações ao código estabelecida pelo *phpDocumentor* [42].

A utilização deste tipo de anotações permite, entre outros, gerar documentação estruturada de forma automática, baseada nas diversas anotações situadas entre o código lógico de cada aplicação, e fornecer ao IDE uma forma de reforço às sugestões providas pelo mesmo, uma vez que o PHP é uma linguagem que não possui um sistema de *strong typing.*

A documentação produzida apresenta o detalhe interno de cada classe, que poderá ser visualizado no Anexo I, no qual está presente a documentação gerada, que, tal como referido, apresenta em maior detalhe os atributos destas classes bem como a visibilidade, o tipo de dados de retorno e argumentos dos seus métodos.

4.4. Exemplo de utilização

Como exemplo de utilização desta *framework* foi utilizado um hipotético *website* / *app web* que consiste num catálogo *online* dos produtos de determinado negócio.

Para efeitos de demonstração apresenta-se um exemplo de desenvolvimento para uma página de apresentação de detalhes de um produto específico.

O desenvolvimento incide sobre 3 partes fulcrais, correspondentes às componentes de uma MVC, as quais se passam a explicar:

Model

Neste exemplo, a *Model* é uma entidade estática, de nome *Produto*, servindo apenas para efeitos demonstrativos. Este representa um produto, com os seus atributos e comportamentos.

```
class Produto
{
	private $id = 3;
	private $nome = 'Tomate';
	private $calibre = 'Cal.67/82/102 kg';
	private $descricao = '<div class="cabecalho">CONSERVAÇÃO</div>(...)';
	private $preco = '0,99';
	private $imgURL = './res/imgcontent/3.jpg';

	public static function load($id)
	{
		return new Produto();
	}

	public function getId()
	{
		return $this->id;
	}
```

```
    public function getNome()
    {
        return $this->nome;
    }

    public function getCalibre(){
        return $this->calibre;
    }

    public function getDescricao()
    {
        return $this->descricao;
    }

    public function getPreco()
    {
        return $this->preco;
    }

    public function getImgURL()
    {
        return $this->imgURL;
    }

}
```

View

A *View* conta com uma estrutura que permite a visualização dos campos do produto, utilizando o atributo *data-id* em elementos que necessitem de ser manipulados. Esta *View* é denominada por *detalhe*, uma vez que apresenta os detalhes do produto.

```
<div class="produto">
    <div class="titulo">
        <h1 data-id="nome"></h1>
        <h2 data-id="calibre"></h2>
        <div data-id="preco"></div>
    </div>
    <div class="conteudo">
        <img data-id="imagem" />
        <div data-id="descricao"></div>
        <a href="../">Voltar</a>
    </div>
</div>
```

Controller

Após criada a entidade responsável pela representação lógica dos produtos e da vista que irá apresentar graficamente essa informação, já se torna possível utilizar estes dois elementos em conjunto no *Controller*.

Neste caso de desenvolvimento específico, o *Controller* apenas conta com a ação correspondente à intenção de um cliente de visualizar os detalhes de um determinado produto.

Esta ação pode ser indicada através da inclusão de */produtos/x/*, onde *x* corresponderia ao identificador do produto, logo após o endereço do *website*.

```
class ProdutosController extends Controller
```

```
{
	//Definir a view "layout" como view por omissão deste controller
	protected $layout = 'layout';

	public function onBeforeRender($layout)
	{
		/*
		Antes de existir um render da página,
		adicionar a folha de estilos "style"
		*/
		$this->addStyleSheet('style');
	}


	public function actionView($id)
	{
		//Obter a view específica para esta ação
		$view = $this->getView('detalhe');

		//Carregar produto com este id
		$produto = Produto::load($id);

		//Inserir texto diretamente nos campos respetivos
		$view->insertText('nome', $produto->getNome());
		$view->insertText('calibre', $produto->getCalibre());
		$view->insertText('preco', $produto->getPreco().'€ / kg');

		//definir o atributo "src" da imagem
		$view->findElementById('imagem')
		->setAttribute('src', $produto->getImgURL());

		//injetar HTML na view
		$view->appendFromString('descricao', $produto->getDescricao());

		//assinalar a view para ser interpretada
		$this->renderView($view);
	}

   }
```

Resultados Finais

O resultado das ações anteriores produz o seguinte código:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
	<title>Horta Biológica - Covilhã</title>
	<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
	<link type="text/css" rel="stylesheet"
	href="/framework/webapp//res/css/style.css" media="all">
</head>
<body>
	<div class="header"></div>
	<div data-id="container">
		<div class="produto">
		<div class="titulo">
			<h1 data-id="nome">Tomate</h1>
			<h2 data-id="calibre">Cal.67/82/102 kg</h2>
			<div data-id="preco">0,99€ / kg</div>
		</div>
```

```
            <div class="conteudo">
                  <img data-id="imagem" src="./res/imgcontent/3.jpg">
                  <div data-id="descricao">
                  <div class="cabecalho">CONSERVAÇÃO</div>(...)</div>
                  <a href="../">Voltar</a>
            </div>
            </div>
      </div>
</body>
</html>
```

Este código constitui uma representação visual do processamento efetuado, tal como se pode verificar no resultado obtido na Figura 17.

Figura 17 - Resultado do Exemplo

5. Conclusão e Trabalho Futuro

O término deste trabalho foi dado face ao cumprimento dos objetivos propostos inicialmente. Com o mesmo podem-se realizar todas as ações propostas nos objetivos iniciais, tendo havido ainda espaço para melhoria e otimização dos mesmos.

A conceção de uma *framework* baseada na MVP *Front Controller* e *Passive View* mostrou-se ser uma mais-valia na criação, manutenção e testes ao código produzido com a mesma.

A utilização de um paradigma completamente orientado a objetos necessitou da criação de componentes de adaptação de elementos fora deste paradigma, o que permitiu também fazer uma otimização sobre o processo usual de desenvolvimento, uma vez que permite encapsular comportamentos que, de outra forma, não estariam presentes neste tipo de elementos.

Tal como proposto inicialmente, a estrutura concebida é altamente modular, o que permite a evolução dos diversos componentes de forma independente e sem prejuízo para o funcionamento da *framework*.

A criação de um "ecossistema" que possibilita a direção de recursos de desenvolvimento para uma aplicação específica, tornando a sua estrutura base abstrata, provou tornar o processo de desenvolvimento mais rápido, estruturado e, por consequência, mais passível a alterações, otimizando assim a tarefa de desenvolvimento.

Os resultados obtidos após os testes mostraram-se bastante satisfatórios, face aos resultados produzidos pelas *frameworks* já existentes, demonstrando um bom desempenho do código produzido, reforçando a sua boa construção e esquematização, fazendo com que o trabalho tenha correspondido com sucesso aos objetivos propostos inicialmente.

Como forma de evolução do trabalho aqui apresentado, é objetivo futuro a criação de um *website* de promoção da *framework*, na qual se fará a distribuição livre da mesma. Este objetivo tem como intuito a criação de uma comunidade de apoio e suporte, bem como a inserção e utilização desta *framework* noutros âmbitos e contextos, como forma de recolha de *feedback*, para tornar possível melhorias futuras ou acréscimos de funcionalidades à mesma.

Um dos grandes objetivos futuros consiste em efetuar revisões e análises ao *workflow* atual, com vista à melhoria do código existente, baseado na informação recolhida da utilização da *framework* em diversos ambientes de desenvolvimento e em projetos com necessidades distintas.

Referências

[1] T. Reenskaug, "MVC XEROX PARC 1978-79," 12 06 2010. [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. [Acedido em 11 4 2013].

[2] The PHP Group, "General Information," 11 4 2013. [Online]. Available: http://www.php.net/manual/en/faq.general.php#faq.general.what. [Acedido em 11 4 2013].

[3] New Digital Group, Inc., "Smarty FAQs," 28 12 2012. [Online]. Available: http://smarty.incutio.com/?page=SmartyFrequentlyAskedQuestions. [Acedido em 11 4 2013].

[4] Sensio Labs, "About," 2012. [Online]. Available: http://twig.sensiolabs.org/. [Acedido em 11 4 2013].

[5] Sencha Inc., "Products," 2013. [Online]. Available: http://www.sencha.com/products/. [Acedido em 11 4 2013].

[6] The jQuery Foundation, 2013. [Online]. Available: http://jquery.com/. [Acedido em 11 4 2013].

[7] W3C, "Dynamic HTML," 21 11 2012. [Online]. Available: http://www.w3.org/Style/#dynamic. [Acedido em 11 4 2013].

[8] Microsoft, "Model-View-Presenter Pattern," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff647543.aspx. [Acedido em 13 3 2013].

[9] P. Korop, "Top 5 most popular PHP frameworks of 2012," webcoderpro.com, 6 12 2012. [Online]. Available: http://webcoderpro.com/blog/top-5-most-popular-php-frameworks-of-2012/. [Acedido em 2 5 2013].

[10] PHPFrameworks.com, "Top 10 PHP Frameworks," 2013. [Online]. Available: http://www.phpframeworks.com/top-10-php-frameworks/. [Acedido em 2 5 2013].

[11] E. Bizina, "Top 5 PHP Frameworks Infographic," Zfort Group, 5 4 2013. [Online]. Available: http://www.zfort.com/blog/top-5-php-frameworks-infographic/. [Acedido em 2 5 2013].

[12] Web Revisions, "PHP framework – The Best PHP Framework for 2013," 5 1 2013. [Online]. Available: http://webrevisions.com/tutorials/php-framework-the-best-php-framework-for-2013/. [Acedido em 2 5 2013].

[13] Cake Software Foundation, Inc., "CakePHP," 2013. [Online]. Available: http://cakephp.org/. [Acedido em 12 05 2013].

[14] Ellis Lab, "CodeIgniter," 2013. [Online]. Available: http://ellislab.com/codeigniter. [Acedido em

12 5 2013].

[15] Yii Software LLC, 2013. [Online]. Available: http://www.yiiframework.com/. [Acedido em 12 5 2013].

[16] Zend Technologies Ltd, 2013. [Online]. Available: http://framework.zend.com/. [Acedido em 12 5 2013].

[17] E. Gottesdiener, *RAD Realities: Beyond the Hype to How RAD Really Works,* 1995.

[18] BuiltWith, "Framework Usage Statistics," 12 4 2013. [Online]. Available: http://trends.builtwith.com/framework. [Acedido em 18 4 2013].

[19] The PHP Group, "The PHP License, version 3.01," 2012. [Online]. Available: http://www.php.net/license/3_01.txt. [Acedido em 11 4 2013].

[20] The PHP Group, "Migrating from PHP 4 to PHP 5," 5 4 2013. [Online]. Available: http://php.net/manual/en/faq.migration5.php. [Acedido em 11 4 2013].

[21] The PHP Group, "New Object Model," 5 4 2013. [Online]. Available: http://www.php.net/manual/en/migration5.oop.php. [Acedido em 11 4 2013].

[22] The Eclipse Foundation, 2013. [Online]. Available: http://eclipse.org/. [Acedido em 11 4 2013].

[23] The Eclipse Foundation, "PHP Development Tools," 2013. [Online]. Available: http://projects.eclipse.org/projects/tools.pdt. [Acedido em 11 4 2013].

[24] D. Rethans, 2013. [Online]. Available: http://xdebug.org/. [Acedido em 11 04 2013].

[25] The Apache Software Foundation, 2012. [Online]. Available: http://apache.org/. [Acedido em 11 4 2013].

[26] Microsoft, "Model-View-Controller," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff649643.aspx. [Acedido em 13 3 2013].

[27] Microsoft, "Front Controller," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff648617.aspx. [Acedido em 14 3 2013].

[28] Yii Software LLC, "Model-View-Controller (MVC)," 2013. [Online]. Available: http://www.yiiframework.com/doc/guide/1.1/en/basics.mvc. [Acedido em 14 4 2013].

[29] The Apache Software Foundation, "Apache HTTP Server Tutorial: .htaccess files," 2013. [Online].

Available: http://httpd.apache.org/docs/2.2/howto/htaccess.html. [Acedido em 12 3 2013].

[30] The Apache Software Foundation, "Apache Module mod_rewrite," 2011. [Online]. Available: http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html. [Acedido em 14 4 2013].

[31] The Open Group, "Regular Expressions," 1997. [Online]. Available: http://pubs.opengroup.org/onlinepubs/007908799/xbd/re.html. [Acedido em 14 4 2013].

[32] e. a. RFC 2616 Fielding, "Status Code Definitions," 6 1999. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. [Acedido em 14 4 2013].

[33] SEOmoz, "HTTP Status Codes," 2012. [Online]. Available: http://www.seomoz.org/learn-seo/http-status-codes. [Acedido em 14 4 2013].

[34] J. J. Garret, "Ajax: A New Approach to Web Applications," 18 2 2005. [Online]. Available: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. [Acedido em 14 4 2013].

[35] The PHP Group, "The PDO Class," 12 4 2013. [Online]. Available: http://pt2.php.net/manual/en/class.pdo.php. [Acedido em 17 4 2013].

[36] The PHP Group, "The DOMDocument class," 2013. [Online]. Available: http://php.net/manual/en/class.domdocument.php. [Acedido em 14 4 2013].

[37] The PHP Group, "The DOMNode class," 2013. [Online]. Available: http://php.net/manual/en/class.domnode.php. [Acedido em 14 4 2013].

[38] W3C, "A vocabulary and associated APIs for HTML and XHTML," 17 12 2012. [Online]. Available: http://www.w3.org/TR/html5/. [Acedido em 14 4 2013].

[39] The PHP Group, "Output Buffering Control," 19 4 2013. [Online]. Available: http://www.php.net/manual/en/book.outcontrol.php. [Acedido em 22 4 2013].

[40] SANS Institute, "Role-Based Access Control: The NIST Solution," *SANS Institute Reading Room,* p. 2, 2003.

[41] incits, "Role-Based Access Control INCITS CS1 St d d S i INCITS CS1 Standards Series," 13 1 2012. [Online]. Available: http://securityfeeds.com/drupal/sites/default/files/CS1%20RBAC%20Standards%20(DU_13Jan2012_Weil)_v1.1.pdf. [Acedido em 14 4 2013].

[42] phpDocumentor, "PHPDoc Reference," [Online]. Available: http://www.phpdoc.org/docs/latest/for-users/phpdoc-reference.html. [Acedido em 20 4 2013].

Glossário

Application Programming Interface

Conjunto de instruções e padrões que formam um protocolo de comunicação entre componentes de *software*.

Asynchronous Javascript And XML

Grupo de tecnologias utilizado em conjunto, no lado do cliente, para a obtenção de dados de servidor de forma assíncrona.

Auto Loading

Mecanismo para a obtenção e ligação dinâmica de classes de um projeto PHP.

Client-Sided

Software presente no lado de um cliente.

Controller

Parte de uma MVC que estabelece a ligação com os restantes componentes.

Debugger

Ferramenta utilizada para depuração e análise de código.

Design Pattern

Solução reutilizável para um problema comum em determinado contexto.

Document Object Model

Convenção utilizada para representação de documentos em forma de objetos.

Dynamic Hypertext Markup Language

Tipo de tecnologia *client-sided* utilizada para criar conteúdo *web* interativo.

Framework

Abstração de *software* que fornece funcionalidades básicas para construir ferramentas assentes na mesma.

Hypertext Transfer Protocol

Protocolo de aplicação para sistemas de informação hipermédia.

Integrated Development Environment

Aplicação que fornece meios para o desenvolvimento de *software*. Normalmente constituído por um editor de código, ferramentas de automatização, *debugger*, compilador e interpretador.

Lazy Loading

Técnica utilizada para deferir a inicialização de um objeto até ao momento em que o mesmo seja necessário.

Model

Parte de uma MVC que representa uma entidade que possui atributos e comportamentos.

Model View Controller

Design Pattern que segmenta o código produzido por unidades com responsabilidades específicas, o *Model*, a *View* e o *Controller*.

Model View Presenter

Design Pattern semelhante à *Model View Controller*, que difere na forma como os componentes comunicam entre si, alterando ligeiramente as responsabilidades de cada.

Open-Source

Filosofia que promove o acesso universal ao código fonte de um projeto.

Plugin

Software que fornece uma característica ou funcionalidade adicional a uma aplicação já existente.

Profiler

Ferramenta que utiliza uma variedade de técnicas para colecionar dados de vários tipos de um determinado artefacto de *software*.

Rapid Application Development

Metodologia de desenvolvimento que promove iterações rápidas suscetíveis a alterações. Esta metodologia defende um planeamento reduzido em favor de uma prototipagem rápida.

Regex

Regex ou *Regular Expression* é uma expressão que define um padrão que possui meios concisos e flexíveis para encontrar fragmentos de texto.

Render

Processo para gerar uma representação gráfica, ou imagem, a partir de determinada informação.

Role Based Access Control

Acesso a privilégios baseados no papel que um utilizador desempenha num sistema.

Search Engine Optimization

É o processo de otimizar um elemento *web* com vista a aumentar a sua visibilidade em motores de pesquisa.

Server-Sided

Software presente no lado de um servidor.

Status Code

Valores numéricos que indicam o estado da resposta de um pedido HTTP.

Templating

Técnica que possibilita a definição de componentes reutilizáveis.

Uniform Resource Identifier

Identificador de um recurso que é composto pelo *Uniform Resource Location* associado ao *Uniform Resource Name*.

Uniform Resource Location

Parte de um *Uniform Resource Identifier* que identifica o caminho até determinado recurso, não incluindo o nome desse recurso.

View

Componente que faz parte da MVC e que é responsável pela representação visual de um modelo.

Web Service

Meio de comunicação entre duas aplicações de *software* através da *World Wide Web*.

Anexos

Anexo I - Documentação das Classes

Nota: A documentação de classes aqui apresentada foi gerada parcialmente com recurso à ferramenta *ApiGen*.

Package Overview

Packages summary

config	BaseConfig, Config, DBConfig
core	
core\base	App, AutoLoader, RoutingManager
core\components	Controller, DBObject, Document, ORM, Page, User, View
core\debug	Logger
core\exceptions	Error403Exception, Error404Exception, IdNotFoundException, LanguageResourceNotFoundException, LayoutNotFoundException, RenderTargetNotFoundException, ResourceNotFoundException, UnspecifiedLayoutException, ViewNotFoundException
core\utils	LanguageResource

Package: config

Class BaseConfig

Base configuration class. This class provides a basic set of information that is necessary, and helpful, to other dependencies in this project.
This class **must not** be extended directly. In order to provide specific information to the project configuration `Config` should be used instead.

Direct known subclasses
Config

Abstract
Package: config

Author: Tiago Alves
Version: 1.0
Located at config/BaseConfig.php

Methods summary

final public	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getUserClassgetUserClass( )
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getDeploymentDirgetDeploymentDir( ) Gets the deployment directory
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getDeploymentDirNamegetDeploymentDirName( ) Gets the deployment directory
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getDefaultLayoutgetDefaultLayout( ) Gets the default layout of this web app
final public array	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getModulesgetModules( ) Gets the modules of this web app
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getDefaultModulegetDefaultModule( ) Gets the default module of this web app
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getDefaultControllergetDefaultController(string ***$moduleName***) Gets the default controller for a given module
final public array	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getURLRoutingRulesgetURLRoutingRules( ) Get the URL routing rules
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getAppNamegetAppName( ) Gets the app's name
final public	C:\timwe\workspace\web\documentation\class-BaseConfig.html -

string	_getDefaultLanguagegetDefaultLanguage( ) Gets the app's default language
final public array	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getAvailableLanguagesgetAvailableLanguages( ) Gets the country code of the available languages
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getAppCharsetgetAppCharset( ) Gets the app's default charset
final public mixed	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getAppBaseURLgetAppBaseURL( ) Gets the application base URL
final public string	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _getClientPathgetClientPath( ) Gets the client base path to incluse scripts and stylesheets
final public boolean	C:\timwe\workspace\web\documentation\class-BaseConfig.html - _isDevModeisDevMode( ) Check if the app is in development state

Properties summary

protected static Config	*$instance*	null	Singleton Instance
protected string	*$deploymentDir*	'./'	The base directory where the web project will be deployed
protected string	*$deploymentDirName*	'webapp'	The base directory name where the web project will be deployed
protected string	*$appName*	null	The name of the app
protected array	*$availableLanguages*	array('en')	The app's available languages **IMPORTANT:**The default language is the first one declared in

			the array
protected string	*$appCharset*	'UTF-8'	The app's charset
protected string	*$defaultModule*	**null**	The app's default module name
protected array	*$modules*	**array**()	The app's available modules
protected array	*$defaultControllers*	**array**()	The app modules' default controllers **Note:** The array pair / value must identify the module and it's default controller, respectively
protected array	*$urlRoutingRules*	**array**()	The app's routing rules **Note:** The array pair / value must identify the url regex match and it's default behaviour, respectively
protected string	*$defaultLayout*	**null**	The app's default layout name
protected boolean	*$devMode*	**true**	Defines whether or not the app is in development mode
protected string	*$userClass*	**null**	Defines the user class to be used for RBAC

Class Config

Configuration class. This class provides a basic set of information pertaining to the current application.
Elements from the `BaseConfig` should be overriden in order to accommodate project specific settings.

BaseConfig
└ **Config**

Package: config
Author: Tiago Alves

Version: 1.0
Located at config/Config.php

Methods summary

public static Config	C:\timwe\workspace\web\documentation\class-Config.html - _getInstancegetInstance( )

Methods inherited from BaseConfig

getAppBaseURL(), getAppCharset(), getAppName(), getAvailableLanguages(), getClientPath(), getDefaultController(), getDefaultLanguage(), getDefaultLayout(), getDefaultModule(), getDeploymentDir(), getDeploymentDirName(), getModules(), getURLRoutingRules(), getUserClass(), isDevMode()

Properties inherited from BaseConfig

$appCharset, $appName, $availableLanguages, $defaultControllers, $defaultLayout, $defaultModule, $deploymentDir, $deploymentDirName, $devMode, $instance, $modules, $urlRoutingRules, $userClass

Class DBConfig

Database basic configuration. This class allows ORM based models to access a common database configuration.

Package: config
Author: Tiago Alves
Version: 1.0
Located at config/DBConfig.php

Methods summary

public static DBConfig	C:\timwe\workspace\web\documentation\class-DBConfig.html - _getInstancegetInstance( )
public string	C:\timwe\workspace\web\documentation\class-DBConfig.html - _getDBServergetDBServer( ) Gets the database's server
public string	C:\timwe\workspace\web\documentation\class-DBConfig.html - _getDBUsernamegetDBUsername( ) Gets the database's username
public string	C:\timwe\workspace\web\documentation\class-DBConfig.html - _getDBPasswordgetDBPassword( ) Gets the database's password

public string	C:\timwe\workspace\web\documentation\class-DBConfig.html - _getDBNamegetDBName( ) Gets the database name
public string	C:\timwe\workspace\web\documentation\class-DBConfig.html - _getDBCharsetgetDBCharset( ) Gets the database's default charset

Properties summary

protected static Config	$instance	null	C:\timwe\workspace\web\documentation\class-DBConfig.html - $instance Singleton Instance

Package: core\base

Class App

Main class that handles all the initial and final processing behind the framework. It is responsible for request interpretation and translation into usable controllers and actions. and for constructing the final document which will contain the results of the executed action.

Package: core\base
Author: Tiago Alves
Version: 1.0
Located at core/base/App.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-App.html - _runrun( ) Executes the application

Class AutoLoader

Manager for autoloaded classes. This manager loads classes and keeps a registry of locations where the dependencies may be loaded from.

Package: core\base
Author: Tiago Alves
Version: 1.0
Located at core/base/AutoLoader.php

Methods summary	
public static	C:\timwe\workspace\web\documentation\class-AutoLoader.html - _registerregister( )
public static	C:\timwe\workspace\web\documentation\class-AutoLoader.html - _loadClassloadClass(string **$className**) Loads a class
public static	C:\timwe\workspace\web\documentation\class-AutoLoader.html - _addDirectoriesaddDirectories(array **$dirArray**) Adds an array of directories to be used by the autoloader
public static	C:\timwe\workspace\web\documentation\class-AutoLoader.html - _addDirectoryaddDirectory(string **$dir**) Adds a single directory to be used by the autoloader

Class RoutingManager

Translate requests into behaviours. This class is responsible for request evaluation, comparing and resolving into accepted behaviours, as specified in BaseConfig.

Package: core\base
Author: Tiago Alves
Version: 1.0
Located at core/base/RoutingManager.php

Methods summary	
public static	C:\timwe\workspace\web\documentation\class-RoutingManager.html - _getInstancegetInstance( )
public string	C:\timwe\workspace\web\documentation\class-RoutingManager.html - _parseControllerNameparseControllerName(string **$controllerName**) Parses the requested controller into a valid controller name
public string	C:\timwe\workspace\web\documentation\class-RoutingManager.html - _parseActionNameparseActionName(string **$actionName**) Parses the requested action into a valid action name
public array	C:\timwe\workspace\web\documentation\class-RoutingManager.html - _splitUrlsplitUrl(string **$url**) Splits an URL into tokens

public array	C:\timwe\workspace\web\documentation\class-RoutingManager.html - _processRequestprocessRequest(string **$url**, BaseConfig **$config**) Processes the given request in search of a valid behaviour

Package: core\components

Class Controller

This class represents a base to a controller in the MVC pattern. Any controller in this framework should extend this class in order to provide the basic functioning to each controller.

Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/Controller.php

Methods summary

final public	C:\timwe\workspace\web\documentation\class-Controller.html - _ControllerController(string **$moduleName**, mixed **$controllerName**, mixed **$invokedAction**) Creates a controller
public	C:\timwe\workspace\web\documentation\class-Controller.html - _onCreateonCreate( ) Convenience method, called after the controller constructor
public	C:\timwe\workspace\web\documentation\class-Controller.html - _onBeforeActiononBeforeAction( ) Convenience method, called before invoking an action method
public	C:\timwe\workspace\web\documentation\class-Controller.html - _onAfterActiononAfterAction( ) Convenience method, called after invoking an action method
public	C:\timwe\workspace\web\documentation\class-Controller.html - _onBeforeRenderonBeforeRender(View **$layout**) Convenience method, called before the view is echoed on the display
protected array	C:\timwe\workspace\web\documentation\class-Controller.html - _getRestrictionsgetRestrictions( ) This method should be overriden in order to restric user

	access to actions
final public string|null	C:\timwe\workspace\web\documentation\class-Controller.html - _getActionRestrictiongetActionRestriction(string $actionName) Method invoked by the application in order to determinate if an action has any restrictions associated with it
final protected View	C:\timwe\workspace\web\documentation\class-Controller.html - _getViewgetView(string $name, boolean $controllerSpecific = true) Gets a view by a given name
final protected	C:\timwe\workspace\web\documentation\class-Controller.html - _renderViewrenderView(View $view, boolean $appendToTop = false) Indicated the view to me rendered
final public	C:\timwe\workspace\web\documentation\class-Controller.html - _setRenderedLayoutsetRenderedLayout(View $layout) Sets the rendered layout
final protectedView	C:\timwe\workspace\web\documentation\class-Controller.html - _getLayoutViewgetLayoutView( ) Gets the rendered layout view
final public array	C:\timwe\workspace\web\documentation\class-Controller.html - _flushRenderQueueflushRenderQueue( ) Flushes the current render View queue to be added to the page
final public string	C:\timwe\workspace\web\documentation\class-Controller.html - _getLayoutNamegetLayoutName( ) Returns the controller's layout name
final public string	C:\timwe\workspace\web\documentation\class-Controller.html - _getNamegetName( ) Gets this controller's name
final public string	C:\timwe\workspace\web\documentation\class-Controller.html - _getModuleNamegetModuleName( ) Gets the module name associated with this controller
final public string	C:\timwe\workspace\web\documentation\class-Controller.html - _getInvokedActiongetInvokedAction( ) Gets the name of the action that was invoked (accordingly to the url naming)
final protected	C:\timwe\workspace\web\documentation\class-Controller.html - _addScriptaddScript(string $name, boolean $externalScript = false) Adds a script to the controller

final protected	C:\timwe\workspace\web\documentation\class-Controller.html - _addStyleSheetaddStyleSheet(string $name, boolean $externalStyleSheet = false) Adds a styleSheet to the controller
final public array	C:\timwe\workspace\web\documentation\class-Controller.html - _getScriptsgetScripts( ) Gets the script list
final public array	C:\timwe\workspace\web\documentation\class-Controller.html - _getStyleSheetsgetStyleSheets( ) Gets the styleSheet list
final public array	C:\timwe\workspace\web\documentation\class-Controller.html - _getExternalStyleSheetsgetExternalStyleSheets( ) Gets the external styleSheet list
final public array	C:\timwe\workspace\web\documentation\class-Controller.html - _getExternalScriptsgetExternalScripts( ) Gets the external script list
final protected	C:\timwe\workspace\web\documentation\class-Controller.html - _redirectredirect(string $url, string $responseCode = 302) Redirects a request
final protected string	C:\timwe\workspace\web\documentation\class-Controller.html - _getClientPathgetClientPath( ) Get the client path
final protected string	C:\timwe\workspace\web\documentation\class-Controller.html - _getBaseURLgetBaseURL( ) Gets the base URL for this controller

Properties summary

protected string	$layout	null	C:\timwe\workspace\web\documentation\class-Controller.html - $layout Defines which specific layout should be used in this controller Leave it null to use the default layout, specified in the Config class
protected string	$name	null	C:\timwe\workspace\web\documentation\class-Controller.html - $name The name of this controller as it was called by the URL request

Class DBObject

This class provides basic database access to any class that might need it. It uses the configuration provided in DBConfig in order to ensure an efortless usage.

PDO

└ **DBObject**

Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/DBObject.php

Methods summary

public static DBObject	C:\timwe\workspace\web\documentation\class-DBObject.html - _getInstancegetInstance( ) Get a single instance of this class

Methods inherited from PDO

__construct(), __sleep(), __wakeup(), beginTransaction(), commit(), errorCode(), errorInfo(), exec(), getAttribute(), getAvailableDrivers(), inTransaction(), lastInsertId(), prepare(), query(), quote(), rollBack(), setAttribute()

Constants inherited from PDO

ATTR_AUTOCOMMIT, **ATTR_CASE**, **ATTR_CLIENT_VERSION**, **ATTR_CONNECTION_STATUS**, **ATTR_CURSOR**, **ATTR_CURSOR_NAME**, **ATTR_DEFAULT_FETCH_MODE**, **ATTR_DRIVER_NAME**, **ATTR_EMULATE_PREPARES**, **ATTR_ERRMODE**, **ATTR_FETCH_CATALOG_NAMES**, **ATTR_FETCH_TABLE_NAMES**, **ATTR_MAX_COLUMN_LEN**, **ATTR_ORACLE_NULLS**, **ATTR_PERSISTENT**, **ATTR_PREFETCH**, **ATTR_SERVER_INFO**, **ATTR_SERVER_VERSION**, **ATTR_STATEMENT_CLASS**, **ATTR_STRINGIFY_FETCHES**, **ATTR_TIMEOUT**, **CASE_LOWER**, **CASE_NATURAL**, **CASE_UPPER**, **CURSOR_FWDONLY**, **CURSOR_SCROLL**, **ERRMODE_EXCEPTION**, **ERRMODE_SILENT**, **ERRMODE_WARNING**, **ERR_NONE**, **FETCH_ASSOC**, **FETCH_BOTH**, **FETCH_BOUND**, **FETCH_CLASS**, **FETCH_CLASSTYPE**, **FETCH_COLUMN**, **FETCH_FUNC**, **FETCH_GROUP**, **FETCH_INTO**, **FETCH_KEY_PAIR**, **FETCH_LAZY**, **FETCH_NAMED**, **FETCH_NUM**, **FETCH_OBJ**, **FETCH_ORI_ABS**, **FETCH_ORI_FIRST**, **FETCH_ORI_LAST**, **FETCH_ORI_NEXT**, **FETCH_ORI_PRIOR**, **FETCH_ORI_REL**, **FETCH_PROPS_LATE**, **FETCH_SERIALIZE**, **FETCH_UNIQUE**, **MYSQL_ATTR_COMPRESS**, **MYSQL_ATTR_DIRECT_QUERY**, **MYSQL_ATTR_FOUND_ROWS**, **MYSQL_ATTR_IGNORE_SPACE**, **MYSQL_ATTR_INIT_COMMAND**, **MYSQL_ATTR_LOCAL_INFILE**, **MYSQL_ATTR_SSL_CA**, **MYSQL_ATTR_SSL_CAPATH**, **MYSQL_ATTR_SSL_CERT**, **MYSQL_ATTR_SSL_CIPHER**, **MYSQL_ATTR_SSL_KEY**,

MYSQL ATTR USE BUFFERED QUERY, **NULL EMPTY STRING**, **NULL NATURAL**, **NULL TO STRING**, **PARAM BOOL**, **PARAM EVT ALLOC**, **PARAM EVT EXEC POST**, **PARAM EVT EXEC PRE**, **PARAM EVT FETCH POST**, **PARAM EVT FETCH PRE**, **PARAM EVT FREE**, **PARAM EVT NORMALIZE**, **PARAM INPUT OUTPUT**, **PARAM INT**, **PARAM LOB**, **PARAM NULL**, **PARAM STMT**, **PARAM STR**

Class Document

This class provides a DOMDocument base to the web project.

DOMImplementation

└ **Document**

Direct known subclasses
Page

Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/Document.php

Methods summary

protected	C:\timwe\workspace\web\documentation\class-Document.html - _DocumentDocument(string *$title*, string *$lang*, string *$encoding*) Creates an HTML document
public DOMElement	C:\timwe\workspace\web\documentation\class-Document.html - _addStyleSheetaddStyleSheet(string *$url*, string *$media* = 'all') Links a stylesheet to the document
public DOMElement	C:\timwe\workspace\web\documentation\class-Document.html - _addScriptaddScript(string *$url*) Links an external javascript script to this document
protected DOMElement	C:\timwe\workspace\web\documentation\class-Document.html - _createElementcreateElement(string *$tagname*, string *$value* = **null**) Creates a DOMElement from a tag name
protected DOMNode	C:\timwe\workspace\web\documentation\class-Document.html - _importElementimportElement(

	DOMNode *$element*) Imports a node into this document
protected DOMNode	C:\timwe\workspace\web\documentation\class-Document.html - _getElementById getElementById(string *$id*) Gets an element by a given ID
protected	C:\timwe\workspace\web\documentation\class-Document.html - _addMetaTag addMetaTag(string *$name*, string *$content*) Adds a meta tag to the document
protected	C:\timwe\workspace\web\documentation\class-Document.html - _setDescription setDescription(string *$dec*) Adds a description metatag
protected	C:\timwe\workspace\web\documentation\class-Document.html - _setKeywords setKeywords(string *$keywords*) Adds a keyword metatag
protected	C:\timwe\workspace\web\documentation\class-Document.html - _appendToBody appendToBody(DOMElement *$element*) Appends an element to this document's body
protected	C:\timwe\workspace\web\documentation\class-Document.html - _getLanguage getLanguage( ) Gets the page language
public string	C:\timwe\workspace\web\documentation\class-Document.html - _renderDocument renderDocument( ) Retrieve the document in HTML format

Methods inherited from DOMImplementation

createDocument(), createDocumentType(), getFeature(), hasFeature()

Class ORM

This class provides a base to create models, accordingly to the MVC pattern, that need a connection to a database.

Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/ORM.php

Methods summary

final public	C:\timwe\workspace\web\documentation\class-ORM.html - _ORMORM(mixed *$data* = **null**)
public	C:\timwe\workspace\web\documentation\class-ORM.html - __set__set(mixed *$name*, mixed *$value*)
public	C:\timwe\workspace\web\documentation\class-ORM.html - __get__get(mixed *$name*)
public	C:\timwe\workspace\web\documentation\class-ORM.html - __isset__isset(mixed *$name*) As of PHP 5.1.0
public	C:\timwe\workspace\web\documentation\class-ORM.html - __unset__unset(mixed *$name*) As of PHP 5.1.0

Properties summary

protected DBObject	*$dbo*	**null**	C:\timwe\workspace\web\documentation\class-ORM.html - $dbo The active PDO connection
protected boolean	*$newRecord*	**true**	C:\timwe\workspace\web\documentation\class-ORM.html - $newRecord Control whether this is a new record or a reload of an existing one
protected array	*$data*	**array**()	C:\timwe\workspace\web\documentation\class-ORM.html - $data Location for overloaded data.

Class Page

This class provides a bridge between the logical elements contained within View elements and a Document instance.

DOMImplementation

└ Document

└ **Page**

Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/Page.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-Page.html - _PagePage(mixed *$title*, mixed *$language*, mixed *$encoding*)
public	C:\timwe\workspace\web\documentation\class-Page.html - _addViewaddView(View *$view*) Adds a View instance to the body of the page
public	C:\timwe\workspace\web\documentation\class-Page.html - _appendNodeToElementappendNodeToElement(DOMNode *$node*, mixed *$id*)
public	C:\timwe\workspace\web\documentation\class-Page.html - _appendViewToIdappendViewToId(View *$view*, string *$id*) Appends a View instance to an element with the specified Id
protected	C:\timwe\workspace\web\documentation\class-Page.html - _appendViewToBodyappendViewToBody(View *$view*) Appends a View instance to the body of the document
public	C:\timwe\workspace\web\documentation\class-Page.html - _appendViewToTagappendViewToTag(View *$view*, string *$tagname*) Appends a View instance to an element of the specified tag type

Methods inherited from Document

Document(), addMetaTag(), addScript(), addStyleSheet(), appendToBody(), createElement(), getElementById(), getLanguage(), importElement(), renderDocument(), setDescription(), setKeywords()

Methods inherited from DOMImplementation

createDocument(), createDocumentType(), getFeature(), hasFeature()

Class User

This class provides a base to a User's access control.

Abstract
Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/User.php

Methods summary

public static User	C:\timwe\workspace\web\documentation\class-User.html - _getInstancegetInstance( ) Get a single instance of this class
abstract public	C:\timwe\workspace\web\documentation\class-User.html - _onCreateonCreate( ) Event invoked whenever a User instance is created by the App
abstract public boolean	C:\timwe\workspace\web\documentation\class-User.html - _hasPermissionhasPermission(string *$itemName*) Checks whether this user has permission to access the specified item

Class View

This class is part of the MVC pattern and it provides methods to manipulate html artifacts.

Package: core\components
Author: Tiago Alves
Version: 1.0
Located at core/components/View.php

Methods summary

final public	C:\timwe\workspace\web\documentation\class-View.html - _ViewView(string $viewFilename) Creates a View instance from a valid HTML / XML file
public DOMElement	C:\timwe\workspace\web\documentation\class-View.html - _findElementByIdfindElementById(mixed $id) Finds an element by a given id
public DOMElement	C:\timwe\workspace\web\documentation\class-View.html - _findElementByTagNamefindElementByTagName(mixed $tagName) Finds an element by a given tag
public	C:\timwe\workspace\web\documentation\class-View.html - _appendViewappendView(string $id, View $view) Appends another view to a given id
public	C:\timwe\workspace\web\documentation\class-View.html - _appendViewOnTopappendViewOnTop(the $id, View $view, integer $position = 0) Appends a view on top of a specified target, or on another specified position, counting from the top
public	C:\timwe\workspace\web\documentation\class-View.html - _addViewaddView(View $view) Adds a view to the bottom of this one
public DOMElement	C:\timwe\workspace\web\documentation\class-View.html - _createElementcreateElement(string $name, string $value = null) Creates a new Element that can be appended to this view
public DOMElement	C:\timwe\workspace\web\documentation\class-View.html - _insertTextinsertText(string $id, string $text) Convenience method to insert text in a given element
public DOMNode	C:\timwe\workspace\web\documentation\class-View.html - _appendElementappendElement(string $id, DOMNode $element) Appends an element to another

public DOMElement	C:\timwe\workspace\web\documentation\class-View.html - _insertElementinsertElement(string $id, string $elementType, string $value = null) Convenience method to create and insert an element in this view
public	C:\timwe\workspace\web\documentation\class-View.html - _removeElementremoveElement(DOMNode $element) Removes an element from the view
public	C:\timwe\workspace\web\documentation\class-View.html - _removeElementByIdremoveElementById(string $id) Remove an element from the view by it's given data-id
public DOMNodeList	C:\timwe\workspace\web\documentation\class-View.html - _getViewContentgetViewContent( ) Gets the view content
public DOMDocumentFragment	C:\timwe\workspace\web\documentation\class-View.html - _appendFromStringappendFromString(string $id, mixed $string) Converts a string into a document fragment

Package: core\debug

Class Logger

Utility class that can be used to create log files

Package: core\debug
Author: Tiago Alves
Version: 1.0
Located at core/debug/Logger.php

Methods summary

public static	C:\timwe\workspace\web\documentation\class-Logger.html - _loglog(string $string, string $logname) Log a given string into a log file

public static	C:\timwe\workspace\web\documentation\class-Logger.html - clearLogclearLog(string $logname) Clears the contents of a log file

Package: core\exceptions

Class Error403Exception

Exception that represents a 403 error equivalent

Exception

└ **Error403Exception**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/Error403Exception.php

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class Error404Exception

Exception that represents a 404 error equivalent

Exception

└ **Error404Exception**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/Error404Exception.php

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class IdNotFoundException

Exception thrown whenever a specified Id isn't found

Exception

└ **IdNotFoundException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/IdNotFoundException.php

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class LanguageResourceNotFoundException

Exception thrown whenever a language resource isn't found in the filesystem

Exception

└ **LanguageResourceNotFoundException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located atcore/exceptions/LanguageResourceNotFoundException.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-LanguageResourceNotFoundException.html - _LanguageResourceNotFoundExceptionLanguageResourceNotFoundException (mixed *$resourceName*, mixed *$stringResourceId*, mixed *$countryCode*)

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class LayoutNotFoundException

Exception thrown whenever a specified layout isn't found in the filesystem

Exception

└ **LayoutNotFoundException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/LayoutNotFoundException.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-LayoutNotFoundException.html - _LayoutNotFoundExceptionLayoutNotFoundException(mixed *$filename*, mixed *$searchedDir*)

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class RenderTargetNotFoundException

Exception thrown whenever a suitable container target isn't found to render a view

Exception

└ **RenderTargetNotFoundException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/RenderTargetNotFoundException.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-RenderTargetNotFoundException.html - _RenderTargetNotFoundExceptionRenderTargetNotFoundException()

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class ResourceNotFoundException

Exception thrown when a specified isn't found on the filesystem

Exception

└ **ResourceNotFoundException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/ResourceNotFoundException.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-ResourceNotFoundException.html - _ResourceNotFoundExceptionResourceNotFoundException(mixed *$name*)

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class UnspecifiedLayoutException

Exception thrown when there are views to render, but a layout wasn't specified to render them

Exception

└ **UnspecifiedLayoutException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/UnspecifiedLayoutException.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-UnspecifiedLayoutException.html - _UnspecifiedLayoutExceptionUnspecifiedLayoutException( )

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Class ViewNotFoundException

Exception thrown when a specified view isn't found in the filesystem

Exception

└ **ViewNotFoundException**

Package: core\exceptions
Author: Tiago Alves
Version: 1.0
Located at core/exceptions/ViewNotFoundException.php

Methods summary

public	C:\timwe\workspace\web\documentation\class-ViewNotFoundException.html - _ViewNotFoundExceptionViewNotFoundException(mixed *$view*, mixed *$dir*)

Methods inherited from Exception

__construct(), __toString(), getCode(), getFile(), getLine(), getMessage(), getPrevious(), getTrace(), getTraceAsString()

Properties inherited from Exception

$code, *$file*, *$line*, *$message*

Package: core\utils

Class LanguageResource

Utility class that can be used in multi-language applications. The language is automatically set by Application whenever a URL contains information about a language, which can be specified in`Config`

Package: core\utils
Author: Tiago Alves
Version: 1.0
Located at core/utils/LanguageResource.php

Methods summary

public static Language	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _getInstance`getInstance( )` Get a single instance from this class
public	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _setActiveModule`setActiveModule( string` ***$moduleName*** `)` Sets the active module
public	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _setActiveLanguage`setActiveLanguage( string` ***$countryCode*** `)` Sets the country code of the active language
public string	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _getActiveLanguage`getActiveLanguage( )` Gets the active language's country code
public string	C:\timwe\workspace\web\documentation\class-LanguageResource.html -

	_getDefaultLanguagegetDefaultLanguage( ) Gets the app's default language
public array	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _getAvailableLanguagesgetAvailableLanguages( ) Gets the country code of the available languages
public	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _getLanguageAvailabilitygetLanguageAvailability(string *$countryCode*) Checks if a language is available
public string	C:\timwe\workspace\web\documentation\class-LanguageResource.html - _getMessagegetMessage(string *$resourceName*, string *$stringResourceId*) Gets a message in the active language