Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

iFrame - Framework para o Desenvolvimento de Aplicações Web

Luiz Antônio da Silva

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Eduardo Adilio Pelinson Alchieri

Brasília
2014

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Wilson Henrique Veneziano

Banca examinadora composta por:

Prof. Dr. Eduardo Adilio Pelinson Alchieri (Orientador) — CIC/UnB
Prof. Dr. Edison Ishikawa — CIC/UnB
Prof. Marcos Fagundes Caetano — CIC/UnB

CIP — Catalogação Internacional na Publicação

da Silva, Luiz Antônio.

iFrame - Framework para o Desenvolvimento de Aplicações Web / Luiz Antônio da Silva. Brasília : UnB, 2014.
81 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2014.

1. iFrame, 2. framework, 3. MVC, 4. PHP, 5. HTML, 6. CSS,
7. MySQL, 8. SCRUM, 9. Sistemas Web

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

iFrame - Framework para o Desenvolvimento de Aplicações Web

Luiz Antônio da Silva

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Eduardo Adilio Pelinson Alchieri (Orientador)
CIC/UnB

Prof. Dr. Edison Ishikawa
CIC/UnB

Prof. Marcos Fagundes Caetano
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 01 de julho de 2014

Dedicatória

Dedico este trabalho a meu filho Miguel, que acompanhou minha jornada na UnB desde seu nascimento, cuja compreensão me permitiu seguir em frente e concluir o ensino superior.

Agradecimentos

Agradeço a meu filho, pelo simples fato de existir, o que me dá forças para sempre seguir em frente a fim de alcançar um futuro melhor, a minha esposa Priscila, que muito me auxiliou a ter paz, tranquilidade e a manter a calma nos momentos mais difíceis para a construção deste trabalho, a meu orientador, o professor Eduardo Alchieri que me permitiu e incentivou no desenvolvimento deste projeto, e a diversos outros professores do CIC e de outros departamentos que me apoiaram durante minha passagem pela Universidade de Brasília.

Resumo

Este trabalho tem por finalidade apresentar o desenvolvimento de um *framework* web, afim de demonstrar suas principais funcionalidades, instalação e funcionamento além de uma breve explanação sobre as linguagens utilizadas em sua construção, seu conceito e a metodologia aplicada em seu desenvolvimento.

Por fim, este *framework* será utilizado em um estudo de caso para o desenvolvimento de uma aplicação web, denominada "intranet", o que permitirá a coleta e analise de dados sobre seu funcionamento e eficácia.

Palavras-chave: iFrame, framework, MVC, PHP, HTML, CSS, MySQL, SCRUM, Sistemas Web

Abstract

This work aims to present the development of a web framework, in order to show its core functionality, installation and operation plus a brief explanation about the languages used in its construction, its concept and the methodology used in its development.

Finally, this framework will be used in a case study for the development of a web application, called "intranet", which will collect and analyze data about their operation and efficiency.

Keywords: iFrame, framework, MVC, PHP, HTML, CSS, MySQL, SCRUM, Web System

Sumário

Lista de Figuras

Capítulo 1

Introdução

Com a crescente evolução dos meios de comunicação e o crescimento do fluxo de informações ostensivas ou restritas, de caráter auxiliador à tomada de decisão, ou simplesmente informativo, torna-se cada vez mais necessário o desenvolvimento de métodos de transmissão de informações mais eficazes, eficientes e seguros.

Para tal, este trabalho propõe a construção de um framework em PHP, usando como padrão de projeto (Desing Patterns) o modelo MVC (Model-View-Controller), que será a base de um estudo de caso na construção de uma aplicação web denominada "intranet".

O termo intranet, foi utilizado pela primeira vez em 24 de Abril de 1995, num artigo de autoria técnica de Stephen Lawton [9], intitulado *Intranets fuel growth of Internet access tools*, cuja tradução seria aproximadamente, **Intranets o combustível de crescimento das ferramentas de acesso a Internet**, publicado na revista Digital News & Reviews.

A intranet é uma rede privada que assenta sobre a suíte de protocolos da Internet, baseada em protocolos TCP / IP e seus conceitos, como o paradigma cliente-servidor. Em outras palavras, trata-se de uma rede privada dentro de uma organização que está de acordo com os mesmos padrões da Internet, como o apresentado na figura 1.1, porém acessível apenas por funcionários ou outros membros que possuam autorização de acesso.

Através da intranet as organizações militares podem tramitar mensagens e documentos privados de forma mais segura, além do que, seus utilizadores podem ter acesso à Internet através de servidores de firewall que têm a função de selecionar as mensagens em ambas as direções garantindo que a segurança seja preservada. Outro benefício da intranet é permitir um maior controle e gerencia de seu pessoal, reduzindo a burocracia, aumentando a produtividade, criando maior flexibilidade e versatilidade em todo o processo.

Ao criar um site para a intranet, deve ser dada uma atenção especial aos detalhes que permitirão a todos encontrar de maneira fácil e ágil as informações que procuram. Para tal ela deve ser bem organizada e com design gráfico agradável, geralmente sendo concebida e organizada para se concentrar num processo de um Seção ao invés do Departamento como um todo. Quando parte de uma intranet é acessível a clientes, parceiros, fornecedores ou simplesmente pessoas de fora do âmbito da organização, ela passa a ser conhecida com extranet.

Os gestores e desenvolvedores de aplicações web possuem muitas ferramentas à sua disposição, permitindo assim, facilitar o processo de criação de um Portal para a intranet, de modo que ela possa estar sempre acompanhando as novas tendências e utilizando os mais atuais mecanismos de segurança. Dentre elas podemos citar os editores de html, xml,

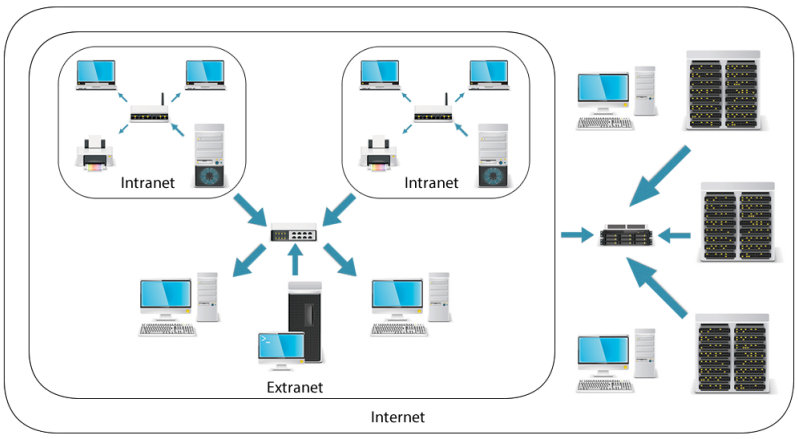


Figura 1.1: Internet, Extranet e Intranet

php, java, ajax e outras linguagens utilizadas no desenvolvimento, bases de dados como por exemplo MySQL, PostgreSQL, Oracle dentre outros com a finalidade de armazenamento de informações e documentos e, formas de interfaces com applets java e java script.

Além da interface, a intranet conta com diversas aplicações que facilitam o trabalho dos membros de uma organização. Estas aplicações podem fazer parte de um determinado pacote de aplicativos de uma interface padrão de intranet ou simplesmente serem inseridos individualmente na rede, tendo uma gestão centralizada ou não.

Cabe a interface de intranet garantir a interligação dessas aplicações, permitindo um acesso rápido e seguro a cada uma delas, podendo ainda sugerir as mais recomendas ou usadas em cada setor da organização.

Em resumo, a intranet tornou-se uma ferramenta auxiliadora na integração dos membros de uma organização, permitindo uma comunicação mais ágil e segura, facilitando o trabalho em equipe, reduzindo a produção dispendiosa de papel e contribuindo para a evolução dos sistemas web.

1.1 Problema

Após uma análise no processo de desenvolvimento e manutenção do Portal da *intranet* de uma determinada Instituição, pode-se perceber alguns problemas.

O Portal web de interface e gerenciamento de conteúdo, ou como ficou conhecido por apenas *"intranet"*, abrange vários sistemas, dentre eles uma aplicação de divulgação de informações, sistema de Avisos e Agenda, uma aplicação de controle e aplicação de materiais em estoque, sistema de Material, uma aplicação de solicitação de serviços, sistema de Suporte, uma aplicação de gerencia de recursos humanos, sistema de Pessoal etc. Foi constatado que toda a intranet e seus sistemas foram desenvolvidos utilizando a lingua-

gem PHP 4.0, de forma estruturada, com conexão direta a um banco de dados do tipo MySQL.

Devido ao modelo adotado na época de sua implementação a *intranet* apresenta alguns problemas que dificultam a migração de seus sistemas para adequar-se a métodos e técnicas de desenvolvimento mais modernas, assim como permitir o uso de novas tecnologias de desenvolvimento.

1.2 Justificativa

A fim de possibilitar o uso de novas tecnologias de desenvolvimento web, como conexões ajax, reuso de classes e métodos através de Orientação a Objetos, dentre outras, na "intranet", este trabalho apresentará um estudo de caso realizado utilizando um framework web para a implementação de uma "nova intranet" e seus sistemas agregados. O framework a ser analisado é o iFrame, o qual vem sendo desenvolvido seguindo a metodologia de produção SCRUM e utilizando o padrão de projeto (Desing Patterns) MVC, ambos apresentados posteriormente.

1.3 Objetivo Geral

Demonstrar que o iFrame é um framework robusto, seguro, fácil de usar e capaz de atender às necessidades dos desenvolvedores da "nova Intranet".

1.4 Objetivos Específicos

O estudo de caso com o iFrame visa atingir os seguintes objetivos:

- garantir a reutlização das classes implementadas nos diversos sistemas da "nova intranet", evitando assim a duplicação desnecessária de código;
- apresentar documentação clara e concisa sobre seu funcionamento;
- permitir futuro compartilhamento de classes de ajuda (Helpers) entre desenvolvedores;
- proporcionar maior agilidade no desenvolvimento de novos sistemas.

1.5 Resultados Esperados

Desenvolver com sucesso um modulo da "nova Intranet" usando o iFrame como base do projeto, de forma fácil e ágil, além de analisar e testar o iFrame em um ambiente de produção e demonstrar sua eficiência e robustez.

Capítulo 2

Desenvolvimento

Este capítulo apresentará um pouco sobre as duas linguagens utilizadas para a construção do iFrame, o HTML e o PHP.

Além disso irá abordar sobre a utilização de CSS com o auxílio do framework Bootstrap utilizado e distribuído pela Globo.com.

Será apresentado o Padrão de Projeto ou, Desing Patterns, MVC junto ao conceito de framework web.

Por fim um pouco sobre o MySQL, utilizado para o armazenamento de dados e a metodologia de desenvolvimento SCRUM, que possibilitou a gestão do projeto.

Ao longo das seções deste capítulo diversas referencias serão feitas, através de imagens do código do próprio iFrame, de forma a exemplificar a utilização de cada linguagem, modelo ou banco de dados.

2.1 HTML

HTML (*HyperText Markup Language, ou Linguagem de Marcação de Hipertexto,*) é uma linguagem de marcação utilizada para produzir páginas na Web sendo interpretadas diretamente por navegadores. A tecnologia é fruto da junção entre os padrões HyTime (padrão para a representação estruturada de hipermídia e conteúdo baseado em tempo) e SGML (padrão de formatação de textos).

O HTML original (e outros protocolos associados como o HTTP) foram criados por Tim Berners-Lee, um físico britânico, usando uma estação NeXTcube com o ambiente de desenvolvimento NeXTSTEP. Na época a linguagem não era uma especificação, mas uma coleção de ferramentas para resolver um problema de Tim: a comunicação e disseminação das pesquisas entre ele e seu grupo de colegas. Sua solução, combinada com a então emergente internet pública (que tornaria-se a Internet) ganhou atenção mundial.

A estrutura básica de um documento, assim como o demonstrado na figura 2.1 apresenta as seguintes marcações muito utilizadas no código do iFrame e de qualquer aplicação web:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="description" content="a descrição do seu site em no máximo 90 caracteres">
        <meta name="keywords" content="escreva palavras-chaves curtas, máximo 150 caracteres">
        <title>Título do Documento</title>
    </head>
    <body>

<!– Aqui fica a página que será visível para todos, onde pode-se inserir textos, imagens, links para outras páginas, etc, geralmente usa-se: –>

        <div>
            Tag para criar uma "caixa" ou um bloco, mais utilizada com Cascading Style Sheets (Folhas de Estilo em Cascata)
        </div>
        <span>
            Tag para modificação de uma parte do texto da página
        </span>
        <img src="endereço_de_uma_imagem.jpg"/>
        <a href="http://www.icriacoes.com.br»
            Criações para a Internet
        </a>
    </body>
</html>
```

HTML 5

HTML5 (*Hypertext Markup Language*, versão 5) é uma linguagem para estruturação e apresentação de conteúdo para a World Wide Web e é uma tecnologia chave da Internet originalmente proposto por Opera Software. Esta nova versão traz consigo importantes mudanças quanto ao papel do HTML no mundo da Web, através de novas funcionalidades como semântica e acessibilidade. O HTML5 será o novo padrão para HTML, XHTML, e HTML DOM.

Em particular, HTML5 adiciona funções sintáticas. Elas incluem as tags de <video>,<audio>, e elementos <canvas>, assim como a integração de conteúdos Salable Vector Graphics, ou gráficos vetoriais escaláveis, que substituem o uso de tags <object> genéricas. Estas funções são projetadas para tornar mais fácil a inclusão e a manipulação

Figura 2.1: iFrame - HTML5

de conteúdo gráfico e multimidia na web sem ter de recorrer a plugins proprietários e APIs.

Outros novos elementos, como <section>,<article>, e <nav>, são projetados para enriquecer o conteúdo semântico dos documentos. Novos atributos têm sido introduzidos com o mesmo propósito, enquanto alguns elementos e atributos têm sido removidos. Alguns elementos, como <a>, e <menu> têm sido mudados, redefinidos ou padronizados.

Exemplo de código HTML5 para reproduzir áudio sem a necessidade de *plug-ins*:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>Áudio em HTML5</title>
    </head>
    <body>
        <audio controls autoplay>
            <source src=“audio.og” />

<!– Mensagem explicando que o navegador não suporta áudio ou o formato usado.–>

            <p>Seu navegador não suporta áudio HTML5 ou o formato Opus.</p>
        </audio>
    </body>
</html>
```

2.2 CSS

Cascading Style Sheets (ou simplesmente CSS) é uma linguagem de folhas de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Em vez de colocar a formatação dentro do documento, o desenvolvedor cria um link (ligação) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal basta, portanto modificar apenas um arquivo.

Sintaxe

O CSS tem uma sintaxe simples e utiliza uma série de palavras em inglês para especificar os nomes de diferentes estilos de propriedade de uma página. Uma folha de estilo consiste de uma lista de regras. Cada regra ou conjunto de regras consiste de um ou mais seletores e um bloco de declaração. Uma declaração de bloco é composta por uma lista de declarações entre chaves. Cada declaração em si é uma propriedade, dois pontos (:), um valor, então um ponto e vírgula (;).

Em CSS, seletores são usados para declarar quais elementos de marcação um estilo se aplica, uma espécie de expressão correspondente. Os seletores podem ser aplicados a todos os elementos de um tipo específico, ou apenas aqueles elementos que correspondam a um determinado atributo; elementos podem ser combinados, dependendo de como eles são colocados em relação uns aos outros no código de marcação, ou como eles estão aninhados dentro do objeto de documento modelo.

A Pseudo-classe é outra forma de especificação usada em CSS para identificar os elementos de marcação, e, em alguns casos, ações específicas de usuário para o qual um bloco de declaração especial se aplica. Um exemplo freqüentemente utilizado é o hover, pseudo-classe que se aplica um estilo apenas quando o usuário "aponta para" o elemento visível, normalmente, mantendo o cursor do mouse sobre ele. Isto é anexado a um seletor como em *a:hove* ou *#elementid:hover*. Uma pseudo-classe seleciona elementos inteiros, tais como *:link* ou *:visited*, considerando que um pseudo-elemento faz uma seleção que pode ser constituída por elementos parciais, tais como *:first-line* ou *:first-letter*.

Aqui está um exemplo que resume as regras acima:

```
selector [, selector2, ...][:pseudo-class] {
        property: value;
        property2: value2;
}
/* comment */
```

2.2.1 Bootstrap

Baseado no Twitter Bootstrap, o Globo Bootstrap [11] é um *framework* de *front-end* poderoso, totalmente intuitivo, a fim de facilitar o desenvolvimento dos elementos de interface nos sites da Globo.com.

Podendo ser também usado em outros projetos como um *guideline* (diretrizes) a fim produzir de forma consistente os padrões consolidados do *Twitter* e da *Globo.com*. Um projeto que pretende contribuir ativamente com o Twitter Bootstrap, ele é uma ferramenta poderosa que também pode ser usada para facilitar na padronização e nas melhores

```
155  input[type="search"] {
156    -webkit-box-sizing: content-box;
157       -moz-box-sizing: content-box;
158            box-sizing: content-box;
159    -webkit-appearance: textfield;
160  }
161
162  input[type="search"]::-webkit-search-decoration,
163  input[type="search"]::-webkit-search-cancel-button {
164    -webkit-appearance: none;
165  }
```

Figura 2.2: iFrame: CSS3

```
26          <div class="container">
27
29              <div class="row-fluid">
30
31                  <div class="alert alert-info span4">
32
33                      <h4 align="center">Padrão de Projeto<br/>MVC</h4>
34
35                  </div>
36
                    <div class="alert alert-info span4">
```

Figura 2.3: iFrame: Bootstrap

práticas de desenvolvimento HTML/CSS e Javascript, tanto para iniciantes no desenvolvimento web, quanto para os avançados que desejam dar um passo além em interações mais complexas.

Com design bastante intuitivo e elegante, com dimensões e proporções calculadas de forma a manter a proporção das larguras, permitiu que praticamente tudo no Bootstrap fosse feito através de um grid que vai de 1 a 12, podendo ser *responsive* ou não, dependendo apenas da inclusão de alguns arquivos css que controlam o *responsive*.

Exemplificado na figura 2.3, o Bootstrap da Globo.com foi largamente utilizado no iFrame, a fim de facilitar o processo de desenvolvimento, devido a sua facilidade de implementação e removendo a necessidade da criação de uma identidade única para o framework.

2.3 PHP

Acrônimo recursivo para *"PHP: Hypertext Preprocessor"*, originalmente Personal Home Page é uma linguagem interpretada livre, usada originalmente apenas para o desenvolvimento de aplicações presentes e atuantes no lado do servidor, capazes de gerar conteúdo dinâmico na World Wide Web.

```
24      public function excluirArquivo(){
25
26          $parameters = new GetParametersHelper();
27          $dados['parametros'] = $parameters->getParameters();
28
29          $arquivo = "./files/" . $dados['parametros']['arquivo'];
30
31          $upload = new UploadHelper();
32          $upload->deleteFile($arquivo);
33
34          $this->view("ExemploUpload", $dados);
35
36      }
37
38  }
39
```

Figura 2.4: iFrame: PHP

Figura entre as primeiras linguagens passíveis de inserção em documentos HTML, dispensando em muitos casos o uso de arquivos externos para eventuais processamentos de dados. O código é interpretado no lado do servidor pelo módulo PHP, que também gera a página web a ser visualizada no lado do cliente.

A linguagem surgiu em meados de 1994, como um pacote de programas CGI criados por *Rasmus Lerdorf* [10], com o nome Personal Home Page Tools, para substituir um conjunto de scripts Perl que ele usava no desenvolvimento de sua página pessoal. Em 1997 foi lançado o novo pacote da linguagem com o nome de PHP/FI, trazendo a ferramenta Forms Interpreter, um interpretador de comandos SQL. Mais tarde, *Zeev Suraski* [17] desenvolveu o analisador do PHP 3 que contava com o primeiro recurso de *orientação a objetos* [16], que dava poder de alcançar alguns pacotes, tinha herança e dava aos desenvolvedores somente a possibilidade de implementar propriedades e métodos.

Pouco depois, *Zeev* e *Andi Gutmans* [6], escreveram o PHP 4, abandonando por completo o PHP 3, dando mais poder à máquina da linguagem e maior número de recursos de orientação a objetos. O problema sério que apresentou o PHP 4 foi a criação de cópias de objetos, pois a linguagem ainda não trabalhava com apontadores ou *handlers* (manipuladores), como são as linguagens Java, Ruby e outras. O problema fora resolvido na versão atual do PHP, a versão 5, que já trabalha com *handlers*. Caso se copie um objeto, na verdade copiaremos um apontador, pois, caso haja alguma mudança na versão original do objeto, todas as outras também sofrem a alteração, o que não acontecia na PHP 4.

O PHP trata-se de uma linguagem extremamente modularizada, o que a torna ideal para instalação e uso em servidores web. Diversos módulos são criados no repositório de extensões *PECL* (PHP Extension Community Library) e alguns destes módulos são introduzidos como padrão em novas versões da linguagem. É muito parecida, em tipos de dados, sintaxe e mesmo funções, com a linguagem C e C++. Pode ser, dependendo da configuração do servidor, embarcada no código HTML. Existem versões do PHP disponíveis para os seguintes sistemas operacionais: *Windows, Linux, FreeBSD, Mac OS, OS/2, AS/400, Novell Netware, RISC OS, AIX, IRIX e Solaris.*

Linguagem altamente utilizada, constitui cerca de 90% do código do iFrame, desde

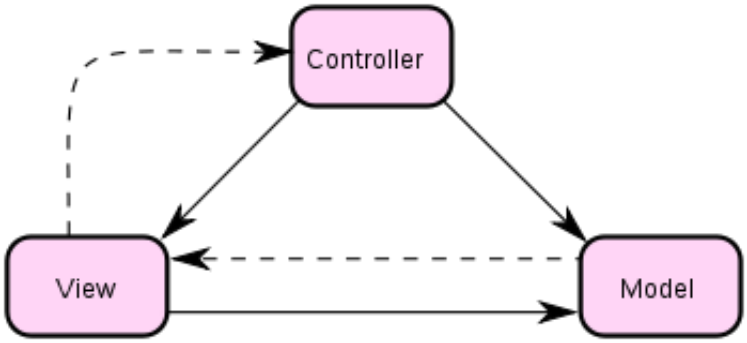


Figura 2.5: Modelo MVC

sua estrutura base, até os métodos mais especializados. Segue um pequeno exemplo de sua utilização na figura 2.4, cujo código apresenta um método de exclusão de arquivos de uma pasta e sua referência no banco de dados.

2.4 MVC

Model-view-controller (MVC), em português modelo-visão-controlador, é um modelo de arquitetura de software que separa a representação da informação, e a interação do usuário com ele, simplificado na figura 2.5. O *model* (modelo) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma *view* (visão) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. a *controller* (controladora) faz a mediação da entrada, convertendo-a em comandos para o *model* ou *view*. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos.

O padrão MVC foi descrito pela primeira vez em 1979 por Trygve Reenskaug [14], que trabalhava no Smalltalk, na Xerox PARC. A implementação original é descrita em profundidade no artigo de Steve Burbeck, PhD com o título: "*Applications Programming in Smalltalk-80: How to use Model–View–Controller*".

Além de dividir a aplicação em três tipos de componentes, o desenho MVC define as interações entre eles:

- Uma *controller* pode enviar comandos para sua *view* associada para alterar a apresentação da *view* e do *model* (por exemplo, percorrendo um documento). Ele também pode enviar comandos para o *model* para atualizar seu estado (por exemplo, editando um documento).
- Um *model* notifica suas *views* e *controllers* associadas quando há uma mudança em seu estado. Esta notificação permite que as *views* produzam saídas atualizadas e que as *controllers* alterem o conjunto de comandos disponíveis. Uma implementação passiva do MVC monta estas notificações, devido a aplicação não necessitar delas ou a plataforma de software não suportá-las.

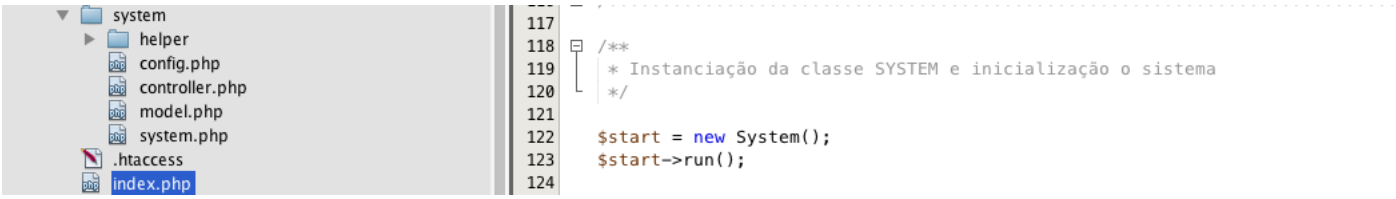


Figura 2.6: iFrame: Principais Arquivos do Modelo MVC

- A *view* solicita do *model* a informação que ela necessita para gerar uma representação de saída.

Na figura 2.6, podemos ver os dois arquivos principais do iFrame, baseados no modelo MVC, o *controller.php* e a *model.php* assim como o método *run()* que inicializa todo o framework, melhor explicado no capítulo 3.

Um caso prático é uma aplicação web em que a *view* é um documento HTML (ou derivado) gerado pela aplicação. A *controller* recebe uma entrada GET ou POST após um estímulo do utilizador e decide como processá-la, invocando objetos do *model* para tratar a lógica de negócio, e por fim trazendo uma *view* para apresentar a saída.

Com o aumento da complexidade das aplicações desenvolvidas, sempre visando a programação *orientada a objeto*, torna-se relevante a separação entre os dados e a apresentação das aplicações. Desta forma, alterações feitas no layout não afetam a manipulação de dados, e estes poderão ser reorganizados sem alterar o layout.

Esse padrão resolve este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois, que seria a controladora.

2.5 Framework

Um *framework* (ou arcabouço), em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação.

Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle. *"Framework é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação."* — Fayad e Schmidt [4].

- *framework conceitual* é um conjunto de conceitos usado para resolver um problema de um domínio específico, não se tratando de um software executável, mas sim de um modelo de dados para um domínio;
- *framework de software* compreende de um conjunto de classes implementadas em uma linguagem de programação específica, usadas para auxiliar no desenvolvimento de um software.

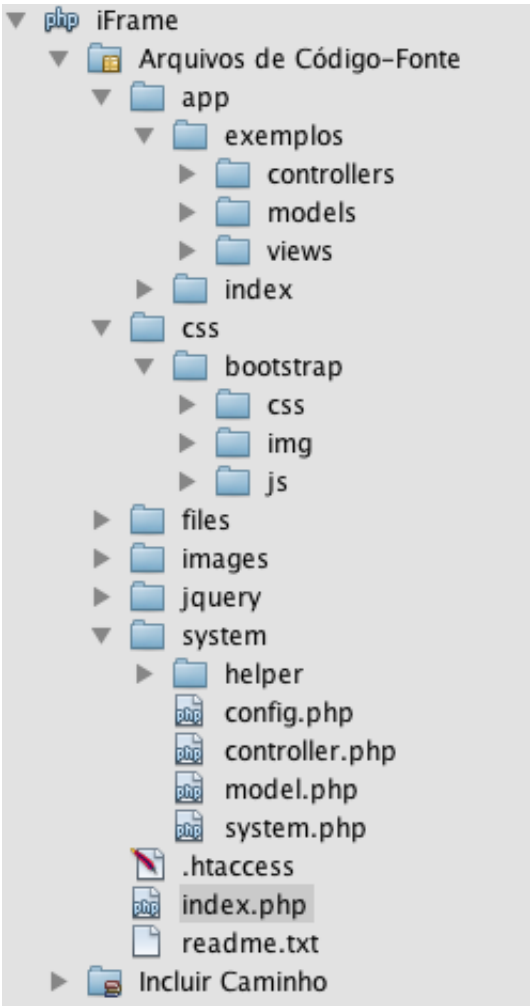


Figura 2.7: iFrame: Estrutura do Framework

O framework atua onde há funcionalidades em comum a várias aplicações, porém para isso as aplicações devem ter algo razoavelmente grande em comum para que o mesmo possa ser utilizado. Padrões de projeto de software não se confundem com frameworks, pois padrões possuem um nível maior de abstração. Um framework inclui código, diferentemente de um padrão de projeto. Um framework pode ser modelado com vários padrões de projeto, e sempre possuem um domínio de uma aplicação particular, algo que não ocorre nos padrões e projeto de software.

Frameworks possuem vantagens, tais como:

- maior facilidade para a detecção de erros, por serem peças mais concisas de software;
- concentração na abstração de soluções do problema que se está tratando;
- eficiência na resolução dos problemas e otimização de recursos.

Os frameworks podem ser divididos em:

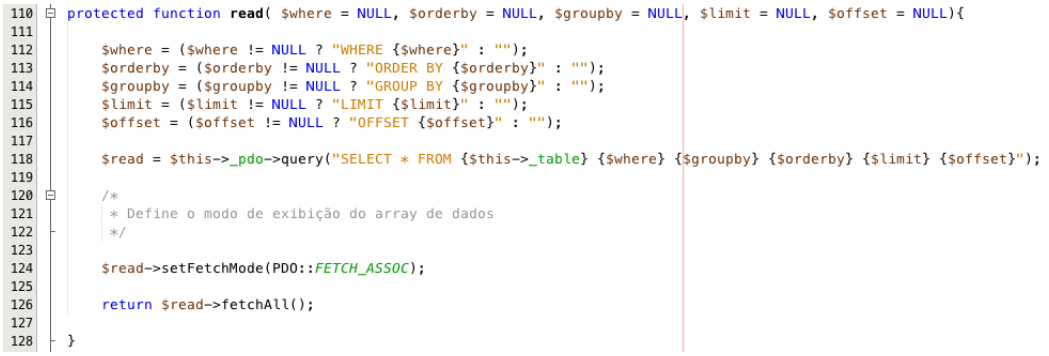
- Frameworks verticais: confeccionados através da experiência obtida em um determinado contexto específico. Esses são mais comumente chamados de frameworks especialistas. Tentam resolver problemas de um domínio e são usados em vários softwares do mesmo domínio. Exemplos: framework financeiro e de recursos humanos. Após alguns projetos em um domínio específico, serão percebidos pontos semelhantes entre estes projetos, e é com base nestes pontos, que será construído o framework vertical (especialista).
- Frameworks horizontais: não dependem do domínio da aplicação e podem ser usados em diferentes domínios.Exemplos: Interfaces gráficas, persistência, transação.

Especificamente em orientação a objetos, framework é um conjunto de classes com objetivo de reutilização de arquitetura de software, provendo um guia para uma solução em um domínio específico de software, como o ilustrado na figura **??**. O framework se diferencia de uma simples biblioteca, pois esta se concentra apenas em oferecer implementação de funcionalidades, sem definir a reutilização de uma solução de arquitetura.

Um framework possui ainda os Frozenspots e os Hotspots

- Frozenspots são as partes fixas de um framework, também conhecidos como hook points. São serviços já implementados pelo framework. Normalmente realizam chamadas indiretas aos hotspots.
- Hotspots são as partes flexíveis de um framework. São pontos extensíveis, necessitam de complementação por funcionalidades/serviços que devem ser implementados, eles são partes nos quais os programadores que usam o framework adicionam o seu código para especificar uma funcionalidade de sua aplicação.

Na figura 2.7, é possível ver a estrutura do *iFrame*, assim como os *frozenspots*, exemplificados pelos arquivos da pasta *system* e os *hotspots* sendo os arquivos da pasta *helpers*.



```
110  protected function read( $where = NULL, $orderby = NULL, $groupby = NULL, $limit = NULL, $offset = NULL){
111
112      $where = ($where != NULL ? "WHERE {$where}" : "");
113      $orderby = ($orderby != NULL ? "ORDER BY {$orderby}" : "");
114      $groupby = ($groupby != NULL ? "GROUP BY {$groupby}" : "");
115      $limit = ($limit != NULL ? "LIMIT {$limit}" : "");
116      $offset = ($offset != NULL ? "OFFSET {$offset}" : "");
117
118      $read = $this->_pdo->query("SELECT * FROM {$this->_table} {$where} {$groupby} {$orderby} {$limit} {$offset}");
119
120      /*
121       * Define o modo de exibição do array de dados
122       */
123
124      $read->setFetchMode(PDO::FETCH_ASSOC);
125
126      return $read->fetchAll();
127
128  }
```

Figura 2.8: iFrame: MySQL

2.6 MySQL

O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo.

Foi criado na Suécia por suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius, que têm trabalhado juntos desde a década de 1980. O MySQL hoje suporta Unicode, Full Text Indexes, replicação, Hot Backup, GIS, OLAP e muitos outros recursos de banco de dados.

Os *Sistemas Gerenciadores de Bancos de Dados*(SGBD) são usados em muitas aplicações, enquanto atravessando virtualmente a gama inteira de softwares de computador. O SGBD é o método preferido de armazenamento/recuperação de dados/informações para aplicações multi-usuários grandes onde a coordenação entre muitos usuários é necessária. Até mesmo usuários individuais os acham conveniente, entretanto, muitos programas de correio eletrônico e organizadores pessoais estão baseados em tecnologia de banco de dados standard.

2.7 Metodologia SCRUM

O *SCRUM* é um processo de desenvolvimento interativo e incremental para gerenciamento de projetos e desenvolvimento ágil de software, tendo seu processo básico apresentado na figura 2.9. Ele possui seu foco no gerenciamento de projeto da organização onde é difícil planejar à frente. Mecanismos do Controle de Processo Empírico, onde ciclos de feedback constituem o núcleo da técnica de gerenciamento que são usadas em oposição ao tradicional gerenciamento de comando e controle. É uma forma de planejar e gerenciar projetos trazendo a autoridade da tomada de decisão a níveis de propriedade de operação e certeza.

Jeff Sutherland, John Scumniotales e Jeff McKenna conceberam, documentaram e implementaram o *SCRUM*, conforme descrito abaixo, na empresa Easel Corporation em1993, incorporando os estilos de gerenciamento observados por Takeuchi e Nonaka. Em 1995, Ken Schwaber formalizou a definição de *SCRUM* e ajudou a implantá-lo no desenvolvi-

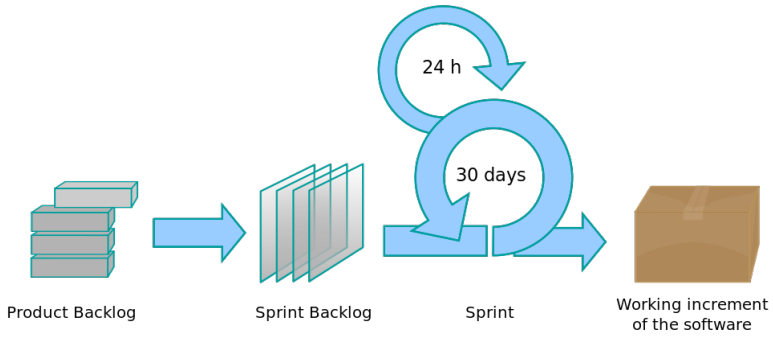


Figura 2.9: Processo SCRUM

mento de softwares em todo o mundo.

Principais Características

- clientes se tornam parte da equipe de desenvolvimento (os clientes devem estar genuinamente interessados na saída);
- entregas frequentes e intermediárias de funcionalidades 100% desenvolvidas;
- planos frequentes de mitigação de riscos desenvolvidos pela equipe;
- discussões diárias de status com a equipe;
- há discussão diária na qual cada membro da equipe responde às seguintes perguntas:
  - O que fiz desde ontem?
  - O que estou planejando fazer até amanhã?
  - Existe algo me impedindo de atingir minha meta?
- transparência no planejamento e desenvolvimento;
- reuniões frequentes com os stakeholders (todos os envolvidos no processo) para monitorar o progresso;
- problemas não são ignorados e ninguém é penalizado por reconhecer ou descrever qualquer problema não visto;
- locais e horas de trabalho devem ser energizadas, no sentido de que "*trabalhar horas extras*" não necessariamente significa "*produzir mais*".

Como outras metodologias de desenvolvimento ágil, o *SCRUM* pode ser implementado através de uma ampla gama de ferramentas. Muitas empresas utilizam ferramentas de software universais, como planilhas para construir e manter artefatos como o *backlog* do *sprint*. Há também pacotes de software open-source e proprietários dedicados à gestão de produtos no âmbito do processo *SCRUM*. Outras organizações implementam o *SCRUM*

Figura 2.10: Quadro de Atividades (Scrum Board)

sem o uso de quaisquer ferramentas de software, e mantêm seus artefatos na forma de cópias impressas, como papel, quadros e notas.

Muitas organizações têm sido resistentes às metodologias introduzidas em baixos níveis da organização. Porém, a adaptabilidade do *SCRUM* permite que ele seja introduzido de forma *stealth*, ou seja invisível, usando os três passos:

- Agende uma demonstração do software com seu cliente em um mês a partir de agora;
- Como equipe, tome um mês para deixar o software pronto para uma demo, com funcionalidades prontas, não simplesmente telas;
- Na demonstração, obtenha feedback e use-o para guiar o seu próximo mês de trabalho de desenvolvimento.

SCRUM Solo

O *SCRUM* é baseado em pequenas equipes. Ele permite a comunicação entre os membros da equipe. Entretanto, há uma grande quantidade de softwares desenvolvidos por programadores solos. Um software sendo desenvolvido por um só programador pode ainda se beneficiar de alguns princípios do *SCRUM*, como: um *product backlog*, *backlog de sprint*, um *sprint* e uma *sprint review*.

Capítulo 3

iFrame

O *iFrame*, ou *internet framework*, conforme apresentado nos capítulos anteriores, encontra-se em sua versão 1.0, desenvolvida utilizando as linguagens *HTML 5* e *PHP 5.4*, com o auxilio do Globo Bootstrap, um poderoso framework css, banco de dados *MySQL 5.5* e usando como base o padrão de projeto (Desing Patterns) *MVC*, *model-view-controller*.

Mesmo em sua versão de desenvolvimento 1.0, a primeira versão a ser implementada no desenvolvimento da "nova intranet", o iFrame já apresenta diversos recursos para o desenvolvimento de sistemas voltados a uma intranet, como métodos facilitadores de inserção de telas de *login/logout*, criação dinâmica de relatórios de informações diversas fornecidas diretamente do banco de dados, construção de galerias de imagens, dentre outros apresentados em sua interface inicial de *Boas Vindas*.

Nesta tela, conforme a Figura 3.1, são apresentados alguns exemplos de utilização do *iFrame*, com demonstrações de seu funcionamento e o código necessário para realizá-lo, permitindo ao desenvolvedor escolher de forma fácil e com clareza quais métodos pretende utilizar para atingir seus objetivos, já ciente dos resultados esperados através dos exemplos fornecidos.

Como será demonstrado logo a seguir, o *iFrame*, possui rápida instalação e fácil configuração, permitindo que em pouco tempo sua metodologia de desenvolvimento seja assimilada pelos desenvolvedores.

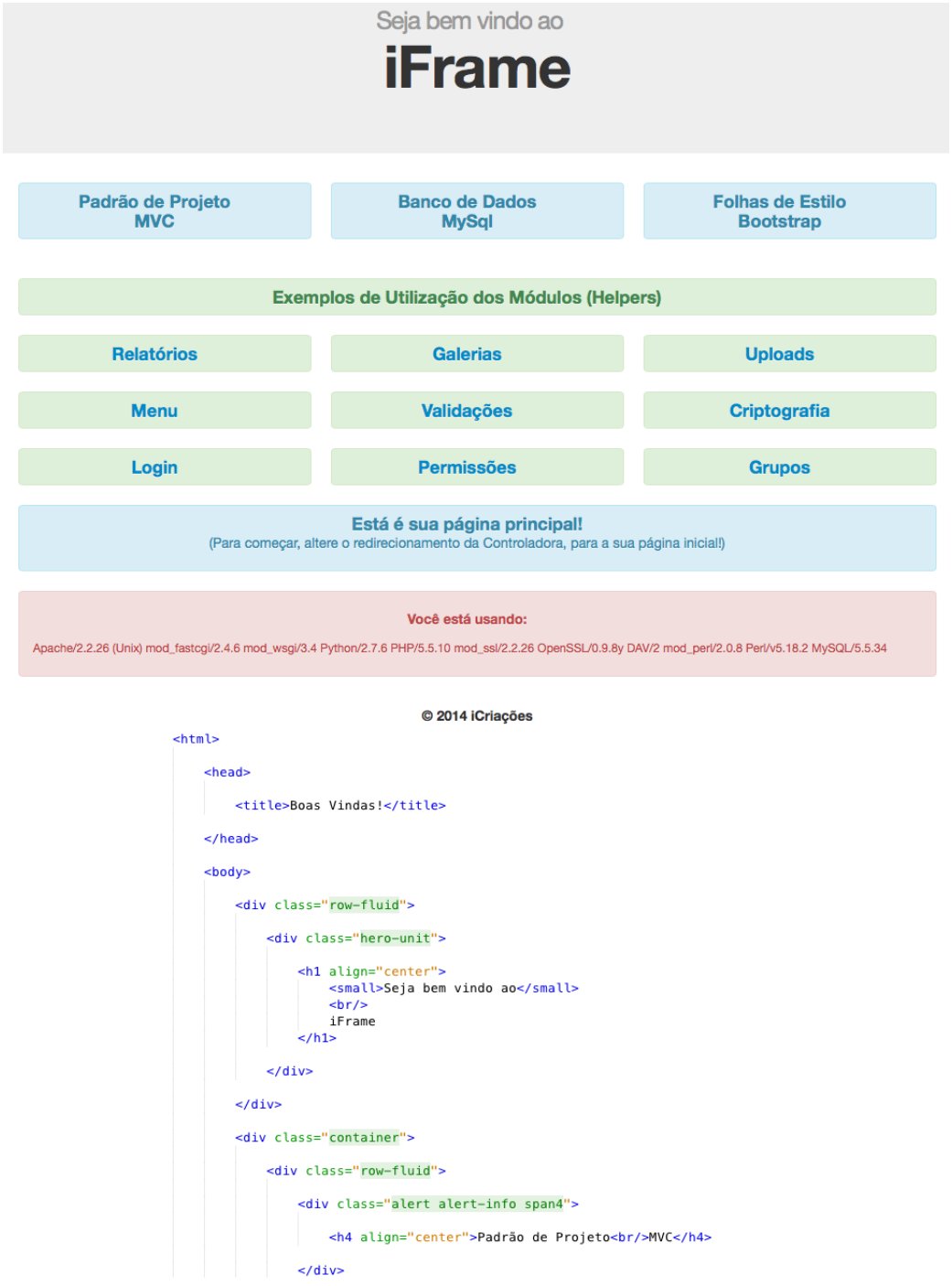


Figura 3.1: iFrame: Tela de Boas Vindas

3.1 Instalação e Estrutura

Para iniciar os trabalhos de desenvolvimento, utilizando o *iFrame*, basta que o desenvolvedor realize o download do arquivo *iFrame.zip* no endereço de compartilhamento do 4Shared, *http://www.4shared.com/zip/P1wWLlaWba/iFrame.html*, e o descompacte no diretório raiz do servidor. Após isso, o desenvolvedor deverá criar um banco de dados no SGBD do MySQL, com o nome de seu projeto e importar as tabelas contidas no arquivo *iframe.sql*, que encontra-se na pasta *install*. As tabelas com prefixo "*ifr_*" são de uso do *iFrame* e devem ser mantidas sem alteração.

A próxima etapa é a configuração do iFrame. Para isso o arquivo *config.php*, contido na pasta *install*, conforme figura 3.2, deverá ser aberto e editado nos trechos em destaque, referentes ao caminho da pasta raiz, o nome do *host*, o usuário, senha e o nome do banco de dados. Depois desta breve configuração o *iFrame* estará pronto para ser usado, acessando o diretório raiz do servidor web onde foi instalado.

Seguindo o padrão de projeto *MVC*, o *iFrame* faz uso de *Models*, modelos de acesso a banco de dados e execução de regras de negócio, *Views*, interfaces de apresentação de informações e requisição de dados, e *Controllers*, controladoras de solicitação de informações e acesso, tanto em seus módulos de desenvolvimento (tratados na "nova intranet" como *sistemas*), quanto em seu núcleo principal.

O *iFrame* é dividido em duas pastas principais, a pasta *system*, que possui os arquivos base de funcionamento do *framework* no padrão *MVC*, e a pasta *app* que possui as aplicações, ou sistemas, desenvolvidos, apresentado na figura 3.2. Cada pasta, dentro da pasta *app*, possui três outras pastas, que são: pasta *controllers*, responsável por armazenar os arquivos do tipo *nomeController.php*, pasta *models*, contendo os arquivos *nomeModel.php*, e a pasta *views*, com os arquivos do tipo *nomeView.php*.

A pasta *system*, armazena os arquivos *controller.php*, responsável por direcionar as ações do iFrame como um todo, resolvendo as chamadas de função e exibindo as *views* solicitadas, *model.php*, responsável pala conexão com o banco de dados e a disponibilização dos principais métodos de acesso a banco para todos os sistemas, e *system.php*, que é o arquivo responsável pelo tratamento das *urls amigáveis* e execução do *framework* como um todo. Além destes arquivos a pasta *system* também contém a pasta *helpers*, que por sua vez armazena todos o arquivos do tipo *nomeHelper.php*, que são os ajudantes do sistema, realizando atividades como validação de datas, criptografia de dados, dentre outras tarefas necessárias para o funcionamento dos demais sistemas embutidos no *framework*.

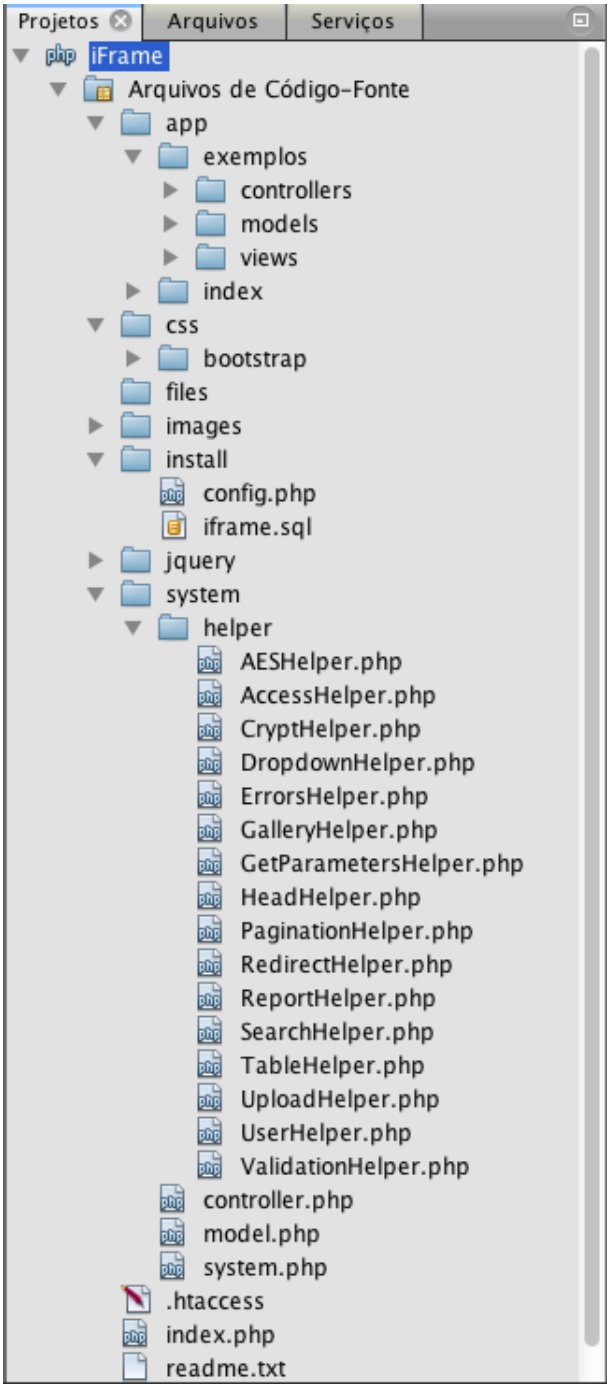


Figura 3.2: iFrame: Estrutura de Arquivos

3.2 Funcionamento

A base do funcionamento de um *framework* em *MVC*, como é o caso do *iFrame*, é a utilização de url's amigáveis, permitindo que as classes sejam acionadas via *$_GET* de forma mais elegante e protegida, impedindo que os arquivos *.php* sejam acessados diretamente. Diferente de outros *frameworks* que utilizam apenas os parâmetros *Controller* e *Action*, o *iFrame* trabalha com três parâmetros básicos passados pela url, o *System*, a *Controller* e a *Action*.

Exemplo de url no iFrame

- http://iFrame/exemplos/galeria/exemploGaleria

Neste exemplo temos:

- *iFrame* é o nome do projeto a ser desenvolvido, podendo ser substituído no momento de criação do projeto;
- */exemplos* é o nome do sistema(app) que foi chamado, identificando o nome da pasta dentro da estrutura do *iFrame*;
- */galeria* é o nome da classe(class) que está sendo chamada, identificando o nome do arquivo da controladora, nesse caso *GaleriaController.php*;
- */exemploGaleria* é o nome do método que está sendo executado com essa chamada.

Através do *mod_rewrite*, ou módulo de reescrita, do *Apache*, a url informada no navegador é enviada para o *iFrame*, que a recebe e trata na *class System*, onde ela é fragmentada, repassando as informações (nome do sistema, da controladora e do método) separadamente para cada método de tratamento, que por sua vez armazenará essas informações em variáveis disponibilidades para o sistema. Outros parâmetros também podem ser enviados para o sistema usando a url, porém eles devem ser repassados sempre aos pares na forma de *key/value*, ou seja, *chave/valor*, como no exemplo a baixo:

- http://iFrame/exemplos/relatorio/exemploRelatorio/tipo/editar/codigo/000001

Onde:

- *tipo e codigo* são as *key*(chaves); e
- *editar e 000001* são os *value*(valores).

Dessa forma o *iFrame* poderá interpretar ações diferentes em uma mesma página.

Finalmente, ao tratar e armazenar cada informação repassada pela url, o método *run*, da classe *System*, irá verificar se a pasta contendo o sistema desejado existe, em caso positivo irá verificar se o arquivo da controladora escolhida encontra-se na pasta e realizará um *require_once*, para incluir o arquivo e garantir que isso ocorrerá apenas uma vez. Depois a classe será instanciada, para que seus métodos fiquem acessíveis e por fim o método selecionado será executado.

A classe *Model* possui a função de conexão com o banco de dados através da classe PDO, nativa do PHP, além de realizar as funções básicas de acesso a banco conhecidas como **CRUD** [13], que são:

- *Create*, inseri um registro no banco de dados através da função SQL *insert*;
- *Read*, lê um único registro ou a tabela toda usando a função SQL *select*;
- *Update*, atualiza um registro específico ou toda a tabela através da função SQL de mesmo nome; *Delete*, apaga um registro específico ou todos os registros da tabela também utilizando uma função SQL de mesmo nome.

Por ultimo, mas não menos importante a classe *Controller*, é responsável por realizar o tratamento dos parâmetros repassados pela classe *System* e incluir o arquivo da *view* solicitada através do método *require_ once*.

Como apresentado acima, cada *app*, ou seja, sistema embutido no *iFrame*, possui suas próprias *models, views e controllers*, porém elas herdam os métodos de suas respectivas "*mães*", através do comando *extends*, permitindo assim uma maior segurança durante o uso do *framework*, pois as classes "*mães*" mantém seu acesso restrito e limitado.

A função das *models filhas*, ou seja, as *models* de cada *app*, tem a função principal de fornecer métodos de acesso a determinadas tabelas ou até mesmo a informações específicas que são constantemente solicitadas pelos usuários do sistema, além de realizar certas lógicas de regras de negocio e algumas validações. As *controllers filhas*, são responsáveis pela execução de determinadas lógicas, a fim de direcionar as requisições oriundas das *views e models filhas*, assim como realizar a exibição das *views filhas* através do método da *controller* principal.

Como apresentado na figura 3.3, ao inserir uma *url* no navegador, a classe *System* recebe esse endereço e o separa em três importantes variáveis: *$_ system*, *$_ controller* e *$_ action*. De posse dessas variáveis a classe *System* localiza a pasta selecionada e o arquivo da controladora escolhida usando o método *file_ exists()*, inclui o arquivo usando o *require_ onde""* e depois instancia a classe através do *new*. Depois de tudo isso feito, realiza a chamada do método, que no caso apresentado na figura, finaliza com a exibição de alguma tela, através da chamada do método *view*, pertencente a classe *Controller* principal, que tem a função de incluir e apresentar o arquivo de exibição solicitado.

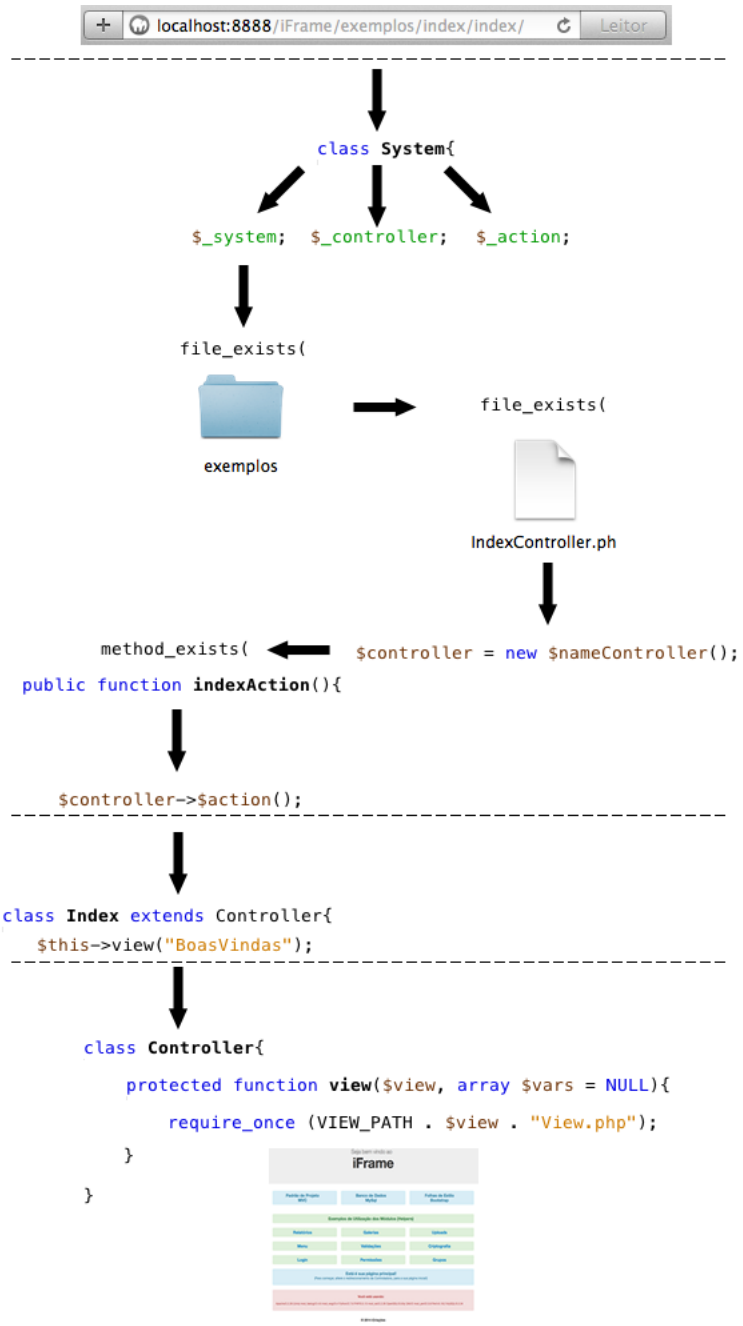


Figura 3.3: iFrame: Fluxo de Funcionamento

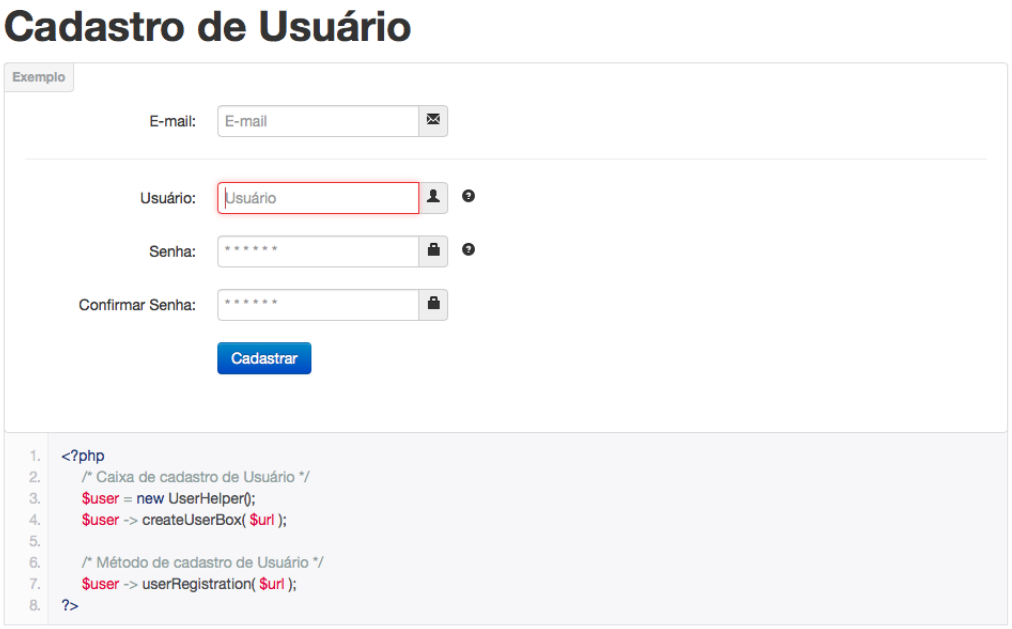


Figura 3.4: iFrame: Tela de Cadastro de Login

3.3 Segurança

Tendo como foco o desenvolvimento de uma intranet para uma Organização Militar, o iFrame necessita prover alto grau de segurança ao acesso dos dados cadastrados e documentos enviados, registrar as ações criticas realizadas pelos usuários, como por exemplo o apagar de arquivos e/ou registros, a alteração de dados pessoais e até mesmo a aplicação de materiais oriundos de depósitos.

Para sanar tal necessidade o *iFrame* possui uma autenticação baseada em usuários e grupos de acesso, onde os usuários cadastrados são alocados em um grupo, herdando suas permissões de acesso. Os grupos de acesso foram construídos usando uma estatura de árvore.

O iFrame possibilita de forma fácil a criação de uma tela de *Cadastro de Login*, apenas com a chamada de uma função já predefinida, conforme figura 3.4, contendo tratamento contra possíveis tentativas de invasão através de *SQL Inject* [12]. O usuário cadastra seu *login*, *senha* e *e-mail*, como um cadastro básico e posteriormente, o desenvolvedor pode acrescentar outros formulários de cadastro para dados mais específicos.

A senha inserida, passa por um processo de *verificação de força* através da comparação de caracteres especiais e pontuação dos caracteres escolhidos, por exemplo, o usuário que inserir uma senha com mais de 8 caracteres terá mais pontos que um usuário que inserir apenas 6 caracteres.

Após a analise de força da senha e a confirmação através do campo *confirmar senha*, o iFrame irá executar a criptografia da senha, através da função *crypt* do PHP, utilizando o método de criptografia *Blowfish* com um *salt*, salto de caracteres, de 22 caracteres gerados randomicamente, e convertidos na *base 64*, e um custo de 28 (256) ciclos de encriptação,

exemplificado pelo fragmento de código apresentado na figura 3.5. Esta senha após ser criptografada será armazenada no banco de dados para ser utilizada posteriormente no processo de entrada no sistema.

Após o cadastro o usuário poderá efetuar login no sistema através de uma tela já definida pelo *iFrame* com a chamada de um método para a criação do botão de *login*, que automaticamente verifica se o usuário está logado e o transforma em um botão de *logoff*, e um método para a verificação do *login* e *senha* inserido, já realizando o tratamento contra *SQL Inject* e validando as informações com o banco de dados, conforme a figura 3.6.

Por fim, o *iFrame* disponibiliza um método para troca de senha, conforme figura 3.7, no qual são realizados todos os procedimentos para a validação da *Senha Atual* com o banco de dados, e posterior criptografia da *Nova Senha*, após sua confirmação.

O acesso aos diversos *app's* embutidos, ou desenvolvidos posteriormente para o *iFrame*, podem fazer uso da *app* padrão de acesso do próprio *iFrame*. Essa *app* funciona através do conceito de que um *usuário* pertence a um *grupo*, que possui certas *permissões* de acesso a cada *sistema* específico. Desta forma, ao criarmos um novo usuário ele será automaticamente alocado no grupo "*Padrão*", tendo permissões básicas de acesso, e podendo ser direcionado para outro grupo, com mais permissões pelo administrador do sistema.

Para fazer uso desse *app*, basta que o usuário, através da interface do aplicativo, crie seus *sistemas* e *permissões* específicos, associando os mesmos aos grupos desejados. De posse do *sistema* e da permissão desejada o desenvolvedor pode realizar uma chamada de função da classe *AccessHelper*, exemplificada na figura 3.8 para verificar se o usuário possui aquela permissão específica solicitada, mesmo que ela seja própria de seu grupo ou herdada de um grupo *pai*, sendo que por definição todos os grupos são *filhos* do grupo "*Padrão*". Essa verificação só é possível através de métodos de verificação recursiva da classe *AccessHelpers* disponibilidade pelo *iFrame*.

Outra classe de segurança implementa pelo *iFrame* é a *AES* [7], Advanced Encryption Standard, que permite criptografar a variável de sessão que contem o *login* do usuário logado. Dessa forma, o *iFrame* pode verificar as permissões do usuário sem que haja o risco de algum usuário mal intencionado insira manualmente um login inválido, ou de outra pessoa, simulando seu login.

```php
class CryptHelper {

    /*
     * Método de encriptação selecionado (bcrypt/blowfish)
     */

    protected $_saltPrefix = "2x";

    /*
     * Custo de ecriptação (quantidade de ciclos, sendo o valor uma
     * potência de 2)
     */

    protected $_defaultCost = 8;

    /*
     * Caracter inicial do salt
     */

    protected $_startLength = 7;

    /*
     * Quantidade de caracteres do salt
     */

    protected $_saltLength = 22;

    /*
     * Função gerar a senha a ser inserida no banco
     */

    public function hash($string, $cost = null) {

        if (empty($cost)) {

            $cost = $this->_defaultCost;

        }

        /*
         * Gera o salt aleatório
         */

        $salt = $this->generateRandomSalt();

        /*
         * Gera o Hash, usando o salt
         */

        $hash = $this->generateHashString((int)$cost, $salt);

        /*
         * Encripta a string com o hash, usando o bcrypt/blowfish
         */

        $crypt = crypt($string, $hash);

        return $crypt;

    }
```

Figura 3.5: iFrame: *Helper* de Criptografia

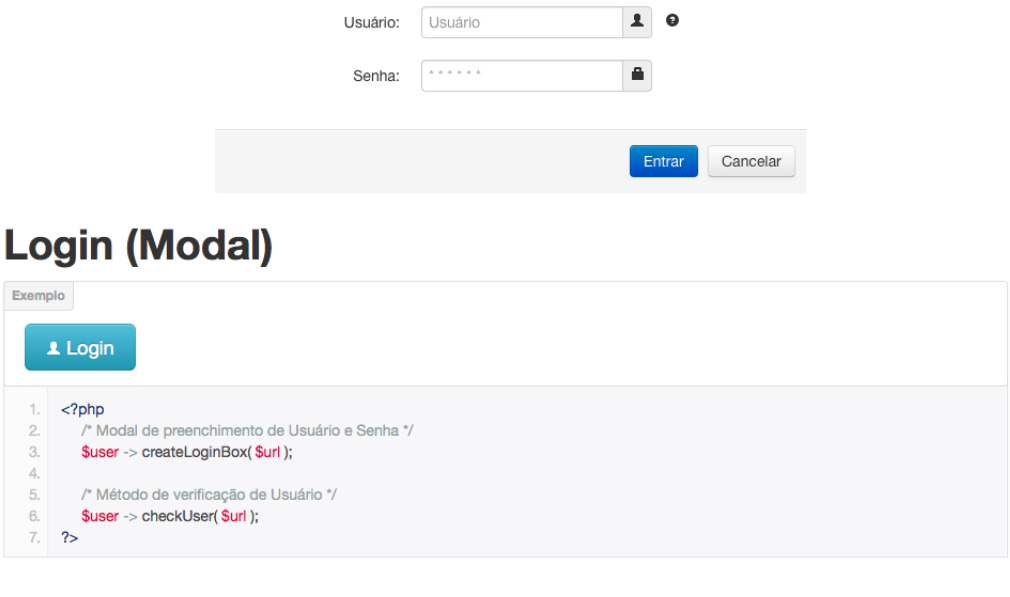


Figura 3.6: iFrame: Tela de Login

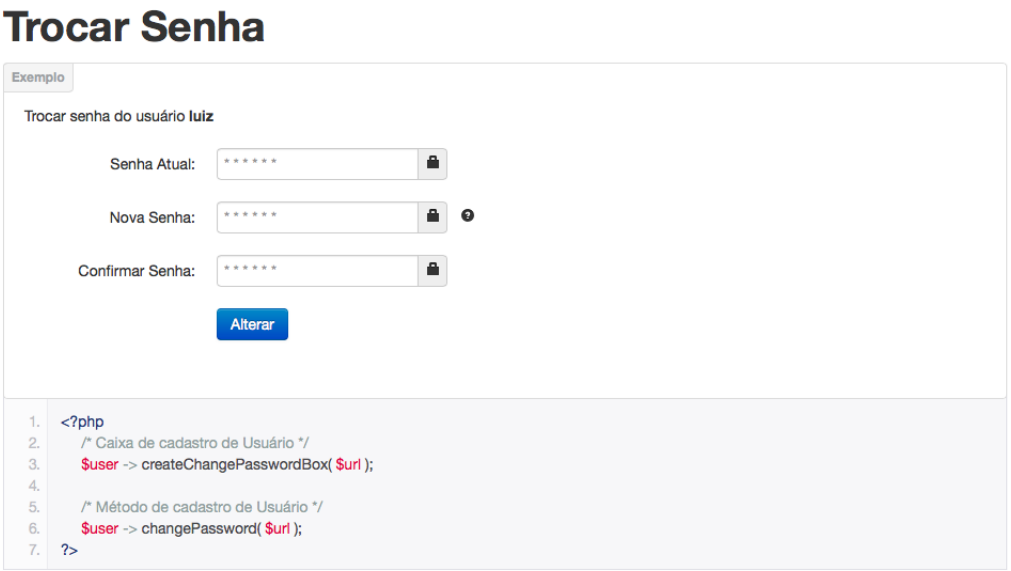


Figura 3.7: iFrame: Trocar Senha

```php
class AccessHelper extends Model{

    /*
     * Variáveis utlizadas pelo sistema para passagem dinâmica de parametros
     */

    private $_result;
    private $_sistema;
    private $_permissao;

    public function checkAcesso($sistema, $permissao){

        /*
         * Método inicial para verificação de Acesso
         * $sistema = Sistema a ser acessado
         * $permissao = Nível de permissão exigida
         */

        $user = isset($_SESSION['user']) && !empty($_SESSION['user']) ? $_SE

        if($user == ""){

            /*
             * Caso o usuário não esteja logado, retorna automáticamente
             * uma resposta Falsa e finaliza a verificção
             */

            return FALSE;

        }
        elseif($user == "admin"){

            return TRUE;

        }
        else{

            /*
             * Caso o Usuário esteja logado e os dados constem nas
             * devidas variáveis de sessão o sistema atribue seus valores
             * as variaveis da classe e chama o próximo método de
             * validação
             */

            $this->_tabela = "login";

            $dados['login'] = $this->read("login='" . $_SESSION['user'] . "'

            foreach ($dados['login'] as $login) {

                $codigoPessoa = $login['codigoPessoa'];

            }

            $this->_sistema = $sistema;
            $this->_permissao = $permissao;

            $this->checkGrupo($codigoPessoa);

            if($this->_result == TRUE){

                /*
                 * Caso o ultimo método de validação retorne Verdadeiro
                 * a autorização de acesso é concedida, finalizando
                 * o processo
                 */

                return TRUE;

            }

        }

    }
```

Figura 3.8: iFrame: Arquivo AccessHelper.php

```php
function __autoload($file)
{

    if (file_exists(HELPER_PATH . $file . ".php")) {

        include_once HELPER_PATH . $file . ".php";

    } elseif (file_exists(MODEL_PATH . $file . ".php")) {

        include_once MODEL_PATH . $file . ".php";

    } else {

        $error = new ErrorsHelper();
        $error->error(
            "A Model ou Helper ' " .
            $file . " ' não foi encontrada! <br/>
            Favor entrar em contato com o Administrador do Sistema!", "danger"
        );


    }

}
```

Figura 3.9: iFrame: Método de _autoload

3.4 Reusabilidade

Uma das principais vantagens do uso de um *framework* no desenvolvimento de uma aplicação, são as soluções prontas que ele traz, e a possibilidade de reutilização de outras soluções, sejam aquelas feitas por terceiros ou reaproveitadas do próprio *framework*. O *iFrame* não poderia ser diferente, por isso ele trabalha com *Helpers*, ou classes ajudantes. As *Helpers* são classes desenvolvidas com o intuito de facilitar o trabalho do desenvolvedor, trazendo soluções para problemas recorrentes, como validação de datas, criptografia de informações, upload de arquivos, dentre outros.

Devido a uma metodologia de construção mais abrangente e genérica, as *Helpers* não dependem de nada do sistema, podendo ser migradas para outra instalação do *iFrame*, garantindo assim um potencial de compartilhamento de soluções para diversos requisitos que possam surgir durante o desenvolvimento. Dessa forma um programador pode criar um *Helper* e disponibilizá-la no *iFrame* para que os outros possam usá-la de forma automática, já que ao ser instanciada, uma classe é automaticamente incluída nos arquivos do *iFrame* através de um método de _ *autoload*, conforme figura 3.9.

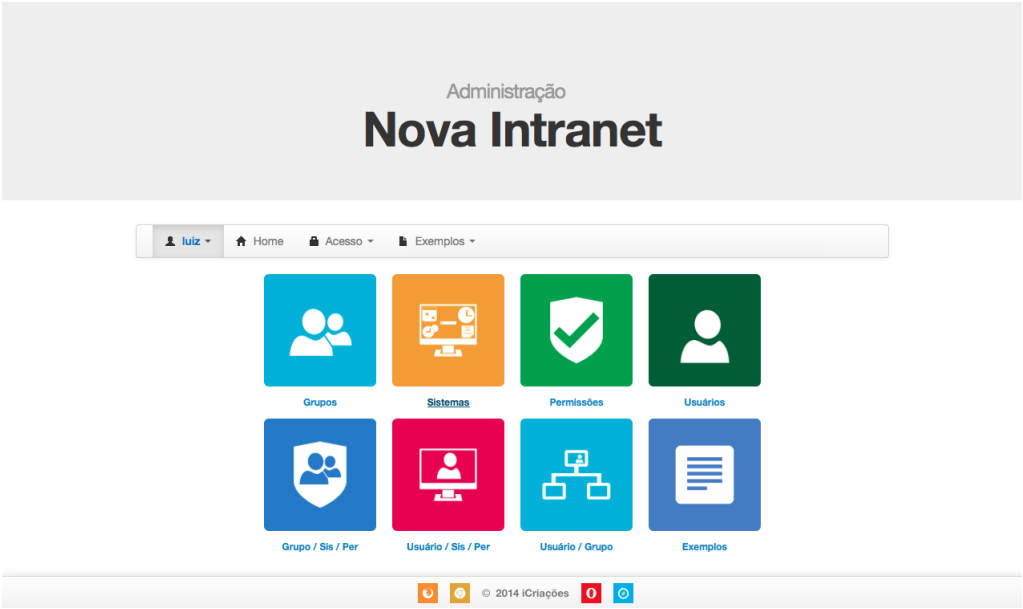


Figura 3.10: Nova Intranet - Tela de Administração

3.5 Estudo de Caso - Nova Intranet

Como proposto nesse trabalho, a "nova intranet", cuja interface de administração consta na figura 3.10, está sendo desenvolvida usando o framework *iFrame v1.0* como base do projeto. As classes de criptografia, cadastro de usuário e permissões de acesso, foram todas implementadas na *app* de login e no controle de acesso, exemplificada na figura 3.11.

Os diversos relatórios constantes na "nova intranet" fazem uso dos métodos de construção dinâmica de relatórios do *iFrame*, como visto na figura 3.12, assim como a *app* criada para controle e disponibilização de conteúdo, que além desses métodos também utilizam os métodos da *helper* de upload, conforme figura **??**.

A equipe de desenvolvimento envolvida no processo de construção desta "nova intranet", é composta por um Analista, responsável pelo planejamento e documentação do projeto e dois programadores, que acumulam a função de gerencia do banco de dados.

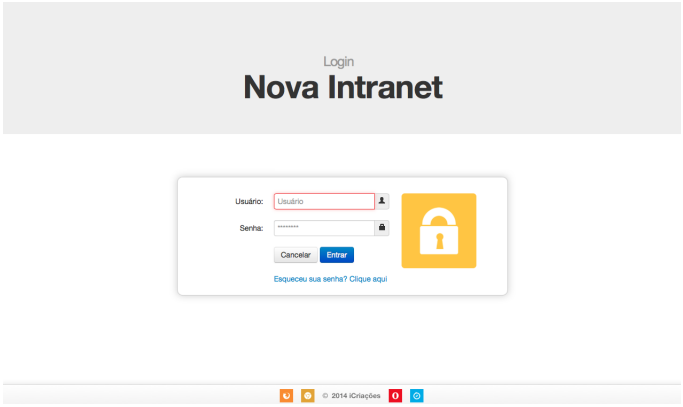


Figura 3.11: Nova Intranet - Tela de Login

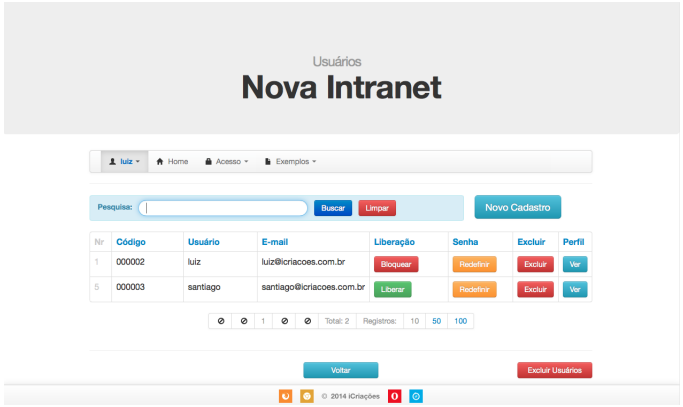


Figura 3.12: Nova Intranet - Relatório de Usuários

Capítulo 4

Conclusão

A antiga intranet da Instituição acima citada como alvo do estudo de caso proposto por este trabalho, foi toda desenvolvida usando PHP 4.0 estruturado, sem nenhum reaproveitamento de código, ou padrão de projeto. O tempo para preparação e estudo de novas metodologias de desenvolvimento era curto, o número de programadores reduzido, e a experiência compartilhada por outros programadores sobre a utilização de *frameworks* robustos, como o *CodeIgniter* [5] ou o Zend framework [18] mostrava que estes *frameworks* possuem um complicado aprendizado e estrutura muito rígida para o acréscimo de novos módulos.

A opção sugerida foi a construção de um *framework* próprio desde sua estrutura base, de forma que fosse possível o desenvolvimento de módulos independentes entre si, porém que pudessem se relacionar caso necessário. Outra vantagem dessa opção era a possibilidade de gerar uma documentação completa de suas funcionalidades desde o inicio do projeto. Por fim, através do uso de "ajudantes", cada programador poderia criar seus próprios métodos que auxiliariam no desenvolvimento do sistema como um todo.

Portanto o *iFrame* provou ser um *framework* robusto, permitindo o desenvolvimento de diversos tipos de *app*, seguro, com a utilização de *helpers* de criptografia para a codificação dos dados sensíveis do sistema, fácil de ser instalado, utilizado e até mesmo expandido, devido a sua estrutura simples e organizada e por fim capaz de atender as necessidades de desenvolvedores de "*intranets*" e demais aplicações web.

Os arquivos base do *iFrame* encontram-se disponibilizados para download no seguinte endereço eletrônico http://www.4shared.com/zip/P1wWLlaWba/iFrame.html, do site de compartilhamento de arquivos do 4Shared, para que programadores interessados possam fazer uso deste framework e auxiliar em seu crescimento.

Referências

[1] Tim Berners-Lee. Físico, cientista da computação e professor do mit. *World Wide Web Consortium*, March 1989. http://pt.wikipedia.org/wiki/Tim_Berners-Lee.

[2] Edgar Frank Codd. Relational model of data for large shared data banks. *Association for Computing Machinery*, pages 377–387, jun 1970. http://pt.wikipedia.org/wiki/Edgar_Frank_Codd.

[3] Pablo Dall'oglio. *PHP-GTK Criando Aplicações Gráficas com PHP*. São Paulo: Novatec, 2007.

[4] M. E. Fayad e D. C. Schmidt. Object-oriented application framworks. *Comunications of the ACM*, 40:10, 1997. 11

[5] Ellislab. Codeigniter, php framework. *Ellislab.com Inc*, 2006. http://ellislab.com/codeigniter. 32

[6] Andi Gutmans. Co-fundador zend technologies. *Apache Software Foundation*, 1999. http://pt.wikipedia.org/wiki/Andi_Gutmans. 9

[7] Steve Borg e Vincent Rijmen Joan Daemen. The design of rijndael: Aes - the advanced encryption standard. *Springer-Verlag*, 2002. 25

[8] Steve Jobs. Thoughts on flsh. *apple.com*, April 2010. http://www.apple.com/hotnews/thoughts-on-flash/.

[9] Stephen Lawton. Instranets fuel growth of internet access tools. *Digital News & Review*, April 1995. http://www.afab.com/DNR_intranets.htm. 1

[10] Rasmus Lerdorf. Systems design engineering. *University of Waterloo*, November 1968. http://http://toys.lerdorf.com/. 9

[11] Alexandre Magno. Globo bootstrap. *Globo.com*, 2012. http://blog.alexandremagno.net/2012/08/globo-boostrap/. 7

[12] Fulvio F. Neto. Desenvolvimento seguro com banco de dados relacional e stored procedure no ambiente web utilizando mysql e php. *Caderno de Estudos Tecnologicos, FATEC, SP*, 2013. 24

[13] Rubens Prates. *MYSQL Guia de Consulta Rápida*. Editora Novatec, 2009. 21

[14] Trygve Mikkjel Heyerdahl Reenskaug. Cientista da computação norueguês. *Universidade de Oslo*, 1979. http://pt.wikipedia.org/wiki/Trygve_Reenskaug. 10

[15] Ken Schwaber. Agile project management with scrum. *Microsoft Press*, feb 2004. http://en.wikipedia.org/wiki/Ken_Schwaber.

[16] Carlos Sica. *PHP Orientado a Objetos: Fale a Linguagem da Internet*. Rio de Janeiro - RJ: Ciência Moderna, 2006. 9

[17] Zeev Suraski. Co-fundador zend technologies. *Apache Software Foundation*, 1999. http://pt.wikipedia.org/wiki/Zeev_Suraski. 9

[18] Zend Technologies. Zend framework 2. *Zend Technologies Ltd*, 2006. http://framework.zend.com/about/. 32