ADIANTI FRAMEWORK PARA PHP

9ª edição

PABLO DALL'OGLIO

CAPÍTULO 1

Introdução

Este capítulo apresenta os conceitos fundamentais presentes no Adianti Framework. Aqui você conhecerá as premissas de desenvolvimento do framework, suas principais características, sua arquitetura, estrutura de diretórios e como se dá o fluxo de execução de seu funcionamento.

1.1 Características

Aqui veremos as características do Adianti Framework.

Foco em aplicações de negócio

Durante a construção do Adianti Framework, não houve a intenção de criar um Framework generalista, possibilitando o desenvolvimento de qualquer aplicação com ele, como um site ou um blog. Apesar de ser possível realizar tais atividades com o Adianti Framework, ele nasceu dentro de um contexto de desenvolvimento de sistemas de gestão, sistemas de informação, mais popularmente conhecidos como ERP's. Ele não prima pela flexibilidade de configuração visual de seus componentes. Por outro lado, ele apresenta componentes bem elaborados, de alto nível, robustos, prontos e fáceis de serem utilizados para a criação de aplicações de negócio.

Desejamos que o desenvolvedor que utiliza o Adianti Framework consiga criar as interfaces do sistema de maneira ágil, escrevendo um código-fonte limpo, orientado a objetos, elegante e de fácil manutenção. Queremos que o desenvolvedor tenha tempo para focar nas regras de negócio, não em detalhes de implementação, para tal, reunimos as principais bibliotecas existentes para criar bons componentes.

Baseado em padrões

Todo o Adianti Framework é baseado em padrões de projeto. Ao todo, ele utiliza em sua implementação mais de 20 padrões de projeto. Dentre eles, podem ser citados: Factory Method, Singleton, Registry, Strategy, Decorator, Active Record, Identity Field, Foreign Key Mapping, Association Table Mapping, Class Table Inheritance, Composite, Query Object, Layer Supertype, Repository, Model View Controller, Front Controller, Template View, Remote Facade, Lazy Initialization, dentre outros.

Qualidade de código

Todo o framework foi desenvolvido seguindo rígidas regras de escrita de código-fonte. Para provar tal característica, as aplicações de exemplo desenvolvidas no framework podem rodar com o nível máximo de erros habilitado no php.ini e nenhuma mensagem de Warning ou Notice será exibida em tela. Além disso, você não verá a presença de nenhuma "@" (usada para suprimir erros) no código-fonte do framework, muito menos a utilização de recursos como variáveis globais.

Orientado a componentes

Diferentemente de outros Frameworks que levam o desenvolvedor a mesclar HTML e PHP em templates, no Adianti Framework esta será uma situação de exceção. A ideia do Framework é construir sempre que possível interfaces por meio de componentes. Caso você precise de uma funcionalidade visual muito específica, então use o mecanismo de templates do Framework, que permite utilizar HTML para compor uma interface. Não gostamos de escrever código HTML, CSS, jQuery, Bootstrap, dentre outras tecnologias dentro do código-fonte da aplicação. Estes códigos devem ficar escondidos da aplicação, encapsulados dentro de componentes. Assim, o desenvolvedor não precisa ser grande conhecedor destas tecnologias para construir interfaces avançadas, basta conhecer os componentes. Como exemplo, para criar componentes utilizados em um formulário, declaramos da seguinte maneira:

```
<?php
$nome        = new TEntry('nome');
$senha       = new TPassword('senha');
$nascimento  = new TDate('nascimento');
$genero      = new TCombo('genero');
$curriculo   = new TText('curriculo');
$estilo      = new TRadioGroup('estilo');
$habilidades = new TCheckGroup('habilidades');
```

Pequeno e fácil de instalar, e open source

Todo o Framework foi concebido para ser pequeno e de fácil instalação. Desta forma, rodar um aplicativo que utiliza o Adianti Framework será tão simples quanto descompactá-lo, uma vez que você já tenha o ambiente com os pré-requisitos configurados. Não existem pré-requisitos escondidos ou proprietários para rodar uma aplicação feita com o Adianti Framework.

Abstração de base de dados

O Adianti Framework privilegia a independência da aplicação em relação ao banco de dados, mediando esta comunicação por meio de uma camada de abstração que permite desenvolver aplicações complexas sem escrever nenhuma linha de SQL, deixando esta tarefa para o framework. Para tal, são criadas classes para representar as tabelas de um banco de dados, seguindo o padrão de projeto Active Record. Com isso, a mesma aplicação poderá rodar sobre diferentes bancos de dados sem necessidade de reescrita. Além disso, padrões de segurança já implementados no framework evitam ataques como SQL Injection.

```
<?php
$customers = Customer::where('gender', '=', 'M')
                     ->where('name', 'like', 'A%')
                     ->load();

foreach ($customers as $customer)
{
    print $customer->phone;
}
```


Templates para a construção de sistemas

O framework fornece uma arquitetura para criação de novos sistemas. Com ele diversas atividades como a manipulação de base de dados e a construção de interfaces serão muito mais ágeis. Porém ao criarmos um novo sistema também precisamos de funcionalidades pré-definidas tais como: controle de login, controle de permissões, armazenamento de logs, dentre outros. Você poderia, com base no framework, implementar o seu próprio controle de login e de permissões. Porém, como essas funcionalidades são relativamente comuns, deixamos tais recursos prontos para você ganhar ainda mais velocidade no início de seu projeto. Tais funcionalidades não estão implementadas diretamente no Framework, mas estão presentes nos templates. Um template é uma estrutura pré-definida sobre o framework que contém funcionalidades para acelerar ainda mais a criação da aplicação. Podemos destacar o controle de login, controle de permissão de acesso, montagem de menus, registro de logs de acesso, registro de logs de SQL e registro de logs de inserção, alteração e exclusão de registros, bem como a comunicação entre usuários, e o compartilhamento de documentos.

Um template contém o próprio Adianti Framework e além disso, fornece vários ingredientes adicionais pré-definidos. O template da aplicação em si contém programas prontos, classes de modelo, base de dados (para as permissões e logs), elementos em HTML e CSS. No site do Framework são disponibilizados alguns templates básicos, mas você possui toda a liberdade de criar um template adaptado às suas necessidades, ou mesmo alterar o visual dos templates disponibilizados, ajustando cores, fontes, e posições de elementos.

Toda a aplicação que rodará dentro do layout é escrita somente com os conhecimentos do Framework, utilizando os seus componentes. Na figura a seguir temos um exemplo de um dos temas do Template disponibilizado no site.

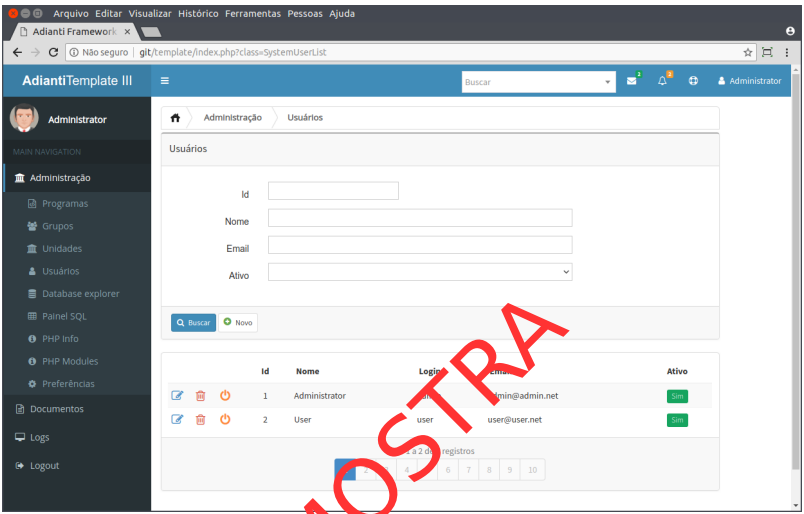


Figura 1 Aplicação rodando com um Template

Obs: Existem templates baseados em Bootstrap e Material design, dentre outros. Você pode utilizar um destes e adaptar às suas necessidades, ou criar um totalmente novo.

1.2 Arquitetura

Uma aplicação criada com o Adianti Framework normalmente será dividida em classes com diversos papéis, dentre os quais podemos citar:

- **Model**: Uma entidade do modelo. Estas entidades manipulam dados e desempenham algumas regras de negócio. São representadas por classes armazenadas em `app/model`. Ex: `Cliente`, `Venda`, `Pedido`;
- **View**: Interface visual na fronteira entre o sistema e o usuário. Pode ser representada por um Template HTML (`app/resources`), ou um grupo de objetos compondo um objeto maior (`app/view`). Estes objetos geralmente lidam com aspectos de apresentação ou coleta de informações ao usuário final;
- **Controller**: Responsável por receber ações vindas de uma classe View e tomar ações. Coordena a sequência de atividades em resposta a uma ação. Para isto, geralmente interage com vários objetos Model para oferecer uma resposta ao objeto View;
- **Service**: Responsável por prestar um serviço para a aplicação (serviço interno), como, por exemplo, processar uma regra de negócio complexa, ou prestar um serviço para outra aplicação (ex: REST Service). Representada por uma classe armazenada na pasta `app/service`.

1.3 Estrutura de diretórios

A tabela a seguir apresenta a estrutura de diretórios utilizada pelo Framework.

Tabela 1. Estrutura de diretórios

Pasta	Conteúdo
app	Contém a aplicação desenvolvida
config	Arquivos de configuração da aplicação e do BD
control	Classes controladoras da aplicação
database	Arquivos de banco de dados (Ex: Sqlite)
forms	Formulários em XML, criados no Form Designer
images	Imagens da aplicação
lib	Bibliotecas e componentes específicos da aplicação
model	Classes de modelo da aplicação (entities)
output	Arquivos temporários gerados (relatórios)
reports	Arquivos criados pelo Designer de documentos
resources	Fragmentos HTML para usar em templates
service	Serviços disponibilizados pela aplicação
templates	Templates da aplicação (Layout)
view	Classes de apresentação reutilizáveis
lib	Bibliotecas
adianti	Adianti Framework
base	Classes controladoras padrão
control	Classes que gerenciam o fluxo de controle
core	Núcleo do Framework (execução, carga, etc)
database	Camada de acesso a banco de dados
images	Imagens utilizadas pelo Framework
include	Arquivos incluídos pelo Framework (js, css)
log	Classes de log de operações
registry	Classes para controle de registro (sessão, cache)
service	Serviços Back-end para componentes
validator	Validadores de formulários
widget	Biblioteca de componentes de apresentação
wrapper	Wrappers para componentes complexos
bootstrap	Biblioteca Bootstrap utilizada pelo framework
jquery	Biblioteca jQuery utilizada pelo framework
vendor	Bibliotecas instaladas pelo Composer
cmd.php	Utilitário para execuções em linha de comando
download.php	Script auxiliar para download de arquivos
engine.php	Motor de execução do framework
index.php	Ponto de entrada da aplicação Web
init.php	Inicialização do framework
menu.xml	Estrutura do menu da aplicação

CAPÍTULO 2

Instalação e configuração

Neste capítulo vamos aprender a dar os primeiros passos com o framework. Vamos abordar a instalação dos pré-requisitos de ambiente em Linux, Windows e MacOS, bem como os passos para instalação do Framework.

2.1 Instalação do ambiente

2.1.1 Ambiente Linux

Nesta seção, será abordada a instalação dos pré-requisitos para rodar o Adianti Framework em Linux Ubuntu 18.04. Dessa forma, instalamos o Apache, o PHP, bem como o suporte aos bancos de dados SQLite, PostgreSQL e Mysql. O suporte ao SQLite é fundamental, visto que a maioria das aplicações de exemplo que acompanham o Framework utilizam este banco de dados.

A instalação do ambiente Web é simples e rápida, vejamos:

```
# sudo su
# apt-get update
# apt-get install apache2 libapache2-mod-php php php-sqlite3 php-pgsql php-mysql php-xml
# service apache2 restart
```

Certifique-se de que o Apache tenha habilitada a interpretação dos arquivos `.htaccess`, uma vez que o Framework utiliza este arquivo para proteger o acesso a determinadas pastas com arquivos de configuração, como `app/config`. A seguir, temos um trecho de código que deve estar presente no `apache2.conf` ou `httpd.conf` dentro da cláusula `Directory`, para habilitar esta configuração.

```
AllowOverride All
```

A configuração do PHP é definida pelo arquivo `php.ini`, que no caso do Ubuntu 18.04, fica localizado em `/etc/php/7.2/apache2/php.ini`. Para ambientes de desenvolvimento, recomendamos as cláusulas a seguir, que por sua vez, habilitam todas os ti-

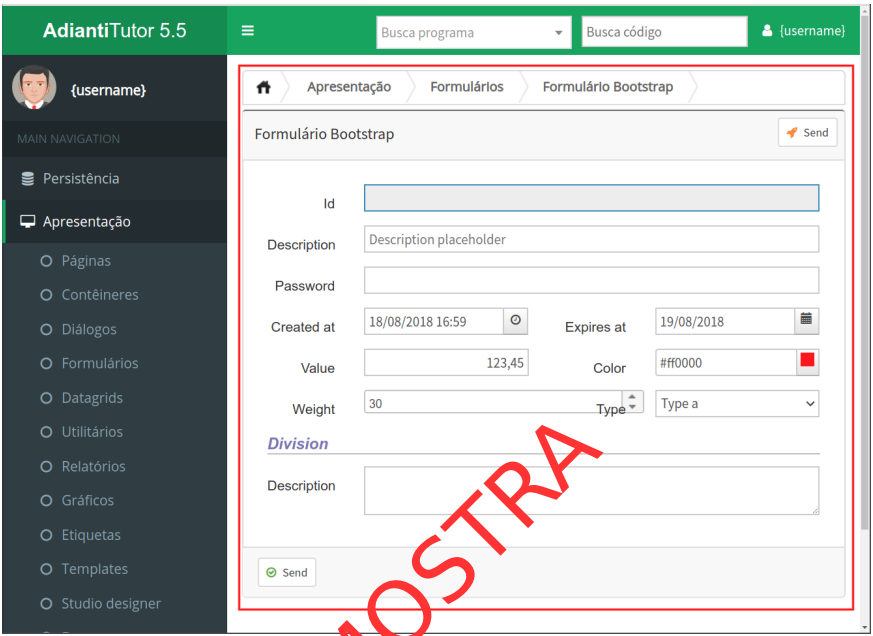


Figura 3 Template com o conteúdo em destaque ao centro

2.4.1 O index

Quando executarmos uma aplicação Adianti Framework, o arquivo `index.php` é interpretado. Isto ocorre na primeira vez em que o usuário acessa o sistema ou sempre que o mesmo solicitar a recarga da página. Isto ocorre por que toda a ação realizada é carregada por meio de requisições Javascript assíncronas, por meio do back-end `engine.php`, não necessitando recarga da página.

A seguir apresentamos o `index.php`, página de entrada (front-end) Web. Ele inicia com a carga do `init.php`, que por sua vez realiza a inicialização do framework, com o carregamento das classes, e definição de configurações. Em seguida é iniciada uma nova seção. `TSession` é a classe responsável por manipular a sessão. Logo após, realizamos a leitura do arquivo `menu.xml`, que contém o menu da aplicação por meio da classe `TMenuBar`. O resultado do menu é armazenado na variável `$menu_string` e usado em seguida dentro do layout.

Em seguida é realizada a leitura do layout da aplicação: `layout.html`. O layout contém a estrutura visual da aplicação e será comentado em seguida. Algumas substituições de variáveis são realizadas, e a variável `$menu_string` (resultado do `menu.xml`) entra na posição demarcada por `{MENU}` dentro do layout. Após, o conteúdo do template é exibido em tela por meio do comando `echo`. Ao final, é verificado se o parâmetro "`class`" está presente na URL. Caso afirmativo, a classe correspondente é carregada por meio do método `AdiantiCoreApplication::loadPage()`.

```
<body>
    <div class="adianti_container">
        <div class="body">
            <div id="menuDiv">
                <div class="tutor-navbar">
                  <div class="tutor-navbar-inner">
                    {MENU}
                  </div>
                </div>
            </div>
            <div id="adianti_div_content" class="content"></div>
            <div id="adianti_online_content"></div>
        </div>
    </div>
</body>
</html>
```

2.4.4 O menu

Um dos recursos utilizados pelo `index.php` é o menu da aplicação, armazenado no arquivo `menu.xml`. Este arquivo é processado pela classe `TMenuBar` e o resultado disto é um menu hierárquico como pode ser visto na próxima figura. É importante notar que ao usar um dos Templates disponibilizados, o menu é renderizado lateralmente, mas a estrutura do XML é a mesma que apresentada a seguir.

No capítulo em que abordaremos o Template do Framework, que já acompanha cadastros de usuários, grupos, e permissões de acesso, veremos que o menu da aplicação é filtrado conforme o perfil do usuário logado. Assim, o usuário somente poderá visualizar e acessar as opções que tiver acesso.

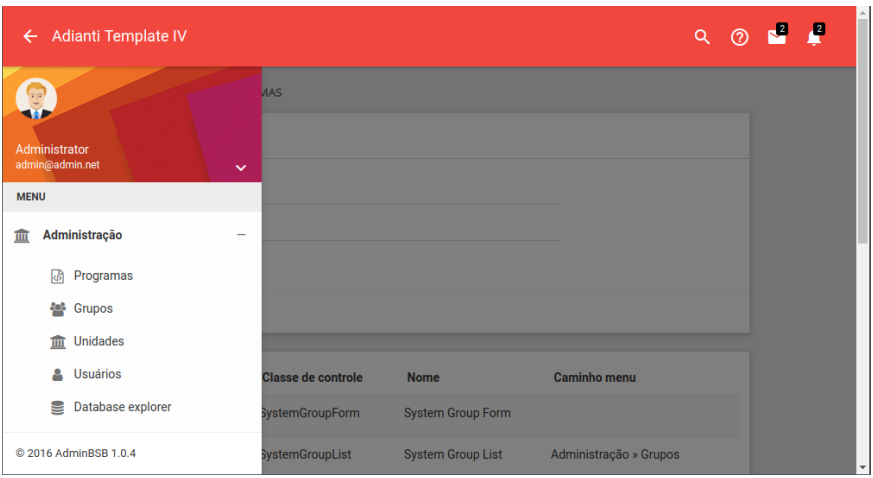


Figura 4 Exemplo de menu

O menu da aplicação é escrito no formato XML como demonstrado a seguir. O menu é renderizado por meio da classe `TMenuBar`, que faz a sua interpretação.

CAPÍTULO 3

Modelos e persistência

Todo o acesso a bancos de dados dentro do Adianti Framework passa por uma camada orientada a objetos que torna a maioria das operações transparentes ao desenvolvedor. Esta camada de manipulação visa dar maior agilidade na criação das aplicações, bem como fazer com que todo o acesso aos dados ocorra de uma maneira orientada a objetos. Assim, na grande maioria das vezes, você não precisará mais escrever os tradicionais INSERT, UPDATE e DELETE, mas irá simplesmente executar métodos sobre objetos, e internamente o Framework tratará de executar os comandos SQL corretos.

3.1 Modelo utilizado

3.1.1 Modelo de classes

Ao longo deste capítulo, trabalharemos com vários exemplos envolvendo acesso a banco de dados. Mas antes de pensar na estrutura do banco de dados, modelamos a estrutura da aplicação com seus objetos, atributos, métodos e relacionamentos. Para os exemplos deste capítulo, elaboramos um modelo simplificado com as classes: `Customer` (cliente), `Category` (categoria), `City` (cidade), `State` (estado), `Contact` (contato), `Skill` (habilidade), `Sale` (venda), `SaleItem` (item da venda), e `Product` (produto). As relações entre os objetos são as seguintes:

- ❑ Um objeto `Customer` (cliente) está associado a um objeto `City` (cidade);
- ❑ Um objeto `Customer` (cliente) está associado a um objeto `Category` (categoria). A categoria pode ser: frequente, casual, varejista, etc;
- ❑ Um objeto `Customer` (cliente) possui uma composição com nenhum ou vários objetos do tipo `Contact` (contato). Isto significa que um objeto `Contact` (objeto parte) é parte exclusiva somente um objeto `Customer` (objeto todo);

- ❑ Um objeto `Customer` (cliente) possui uma agregação com nenhum ou vários objetos `Skill` (habilidade). Isto significa que um objeto `Skill` (objeto parte) pode ser parte compartilhada de diferentes objetos `Customer` (objeto todo);
- ❑ Um objeto `City` (cidade) está associado a um objeto `State` (estado);
- ❑ Um objeto `Sale` (venda) está associado a um objeto `Customer` (cliente);
- ❑ Um objeto `Sale` (venda) possui uma composição com nenhum ou vários objetos `SaleItem` (item da venda). Cada objetos `SaleItem` (parte) será parte exclusiva do objeto `Sale` (todo);
- ❑ Um objeto `SaleItem` (item da venda) está associado a um objeto `Product` (produto).

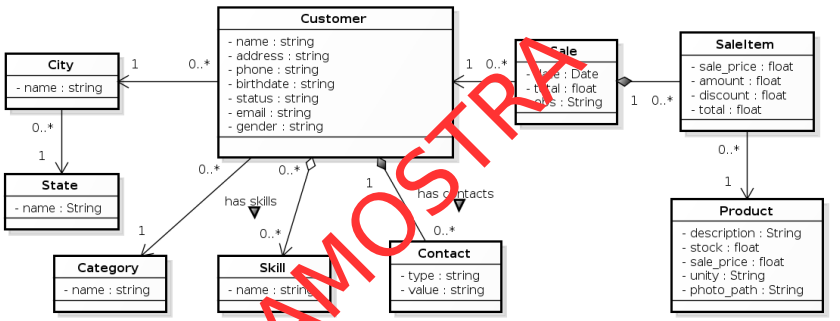


Figura 6 Modelo de classes

Obs: Você encontra maiores informações sobre orientação a objetos em PHP com o livro "*PHP Programando com orientação a objetos*",do mesmo autor.

3.1.2 Modelo relacional

Um modelo de classes pode ser facilmente convertido em um modelo relacional por meio de técnicas de mapeamento objeto-relacional (M.O.R.). A partir do modelo de classes exposto anteriormente, derivamos um modelo relacional contendo as tabelas para armazenar os objetos da aplicação. As seguintes conversões foram realizadas:

- ❑ O objeto `Customer` (cliente) está relacionado com `Category` (categoria) e com `City` (cidade) por meio de chaves estrangeiras (*foreign key mapping*);
- ❑ O objeto `City` (cidade) está relacionado com `State` (estado) por meio de chaves estrangeiras (*foreign key mapping*);
- ❑ A relação de composição entre `Customer` (cliente) e `Contact` (contato) foi mapeada por meio de uma chave estrangeira (*foreign key mapping*), uma vez que em uma relação de composição, a parte é exclusiva do objeto todo;

- A relação de agregação entre `Customer` (cliente) e `Skill` (habilidade) foi mapeada pela tabela associativa (*association table mapping*) `customer_skill` já que na agregação, a parte não é exclusiva do todo;
- O objeto `Sale` (venda) está relacionado com `Customer` (cliente) por meio de chaves estrangeiras (*foreign key mapping*);
- A relação de composição entre `Sale` (venda) e `SaleItem` (item da venda) foi mapeada por meio de uma chave estrangeira (*foreign key mapping*), uma vez que em uma relação de composição, a parte é exclusiva do objeto todo;
- O objeto `SaleItem` (item da venda) está relacionado com `Product` (produto) por meio de chaves estrangeiras (*foreign key mapping*);

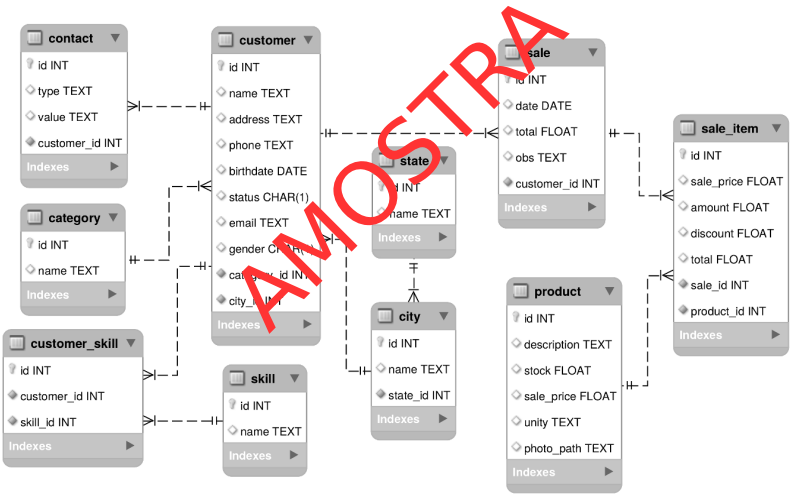


Figura 7 Modelo relacional

3.2 Configuração e acesso ao banco de dados

3.2.1 Criação do banco de dados

O modelo relacional exposto anteriormente é distribuído junto com o código-fonte do tutor no arquivo `app/database/samples.sql`. Além deste, uma base de dados SQLite pronta com dados está no arquivo `app/database/samples.db`. Você pode praticar os exemplos que envolvem base de dados, recriando esta base em outros sistemas gerenciadores como PostgreSQL, ou MySQL se assim preferir, uma vez que o Framework é independente de base de dados como será visto mais adiante.

Obs: A base de dados de exemplos foi suficientemente testada em MySQL, SQLite e PostgreSQL. Para outros bancos de dados, alguns ajustes podem ser necessários.

3.3 Manipulação de objetos

3.3.1 O padrão Active Record

Todos os objetos da camada de modelo de uma aplicação Adianti Framework são objetos Active Record. Um Active Record é um dos padrões de projeto (*design patterns*) mais conhecidos e implementado na maioria dos frameworks. Conforme Martin Fowler, um objeto Active Record é:

> *"An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data."*

Um Active Record é um objeto que representa uma linha de uma tabela (ou view), e encapsula ao mesmo tempo o acesso aos dados e a lógica de negócios daquela entidade. Para exemplificar, podemos pensar em um objeto Livro, que pode ter métodos como `registrarEmprestimo()` e `registrarDevolucao()`, que são métodos de negócio e também pode ter métodos como `store()`, `delete()` e `load()` que são métodos de acesso aos dados (persistência).

Geralmente os objetos Active Record não implementam estes métodos de acesso aos dados `store()`, `delete()` e `load()`, uma vez que nossos objetos teriam misturado em um mesmo local, métodos de negócio e de persistência. Para resolver isso, os objetos Active Record generalizam estas operações, que são implementadas por superclasses. No caso do Adianti Framework, os métodos de persistência são implementados pela classe `TRecord`. Assim, todos os objetos de domínio da aplicação devem ser subclasses de `TRecord`, herdando um conjunto de métodos de persistência.

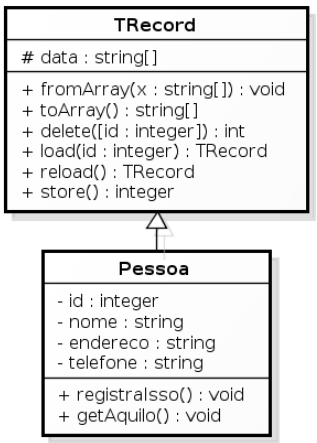


Figura 8 Um objeto Active Record

app/control/dbsamples/ObjectRender.php

```
<?php
class ObjectRender extends TPage
{
    public function __construct()
    {
        parent::__construct();
        try
        {
            TTransaction::open('samples');

            $product = Product::find(4);

            // renderiza atributos entre chaves
            echo $product->render('The product <b>{id}</b> is <u>{description}</u>');

            echo '<br>';

            // avalia a fórmula, substitui atributos entre chaves
            echo $product->evaluate('= {sale_price} * {stock}');

            TTransaction::close();
        }
        catch (Exception $e)
        {
            new TMessage('error', $e->getMessage());
        }
    }
}
```


3.4 Manipulação de coleções

3.4.1 O padrão Repository

O padrão Active Record nos permite trabalhar com um objeto que representa uma linha do banco de dados, ou seja, ele individualiza os registros fazendo com que cada um seja representado por um objeto em memória. E quando desejamos trabalhar com uma coleção (um conjunto) de objetos? Podemos utilizar o padrão Repository. O padrão Repository é implementado por uma classe que disponibiliza uma interface para manipular coleções de objetos como representado pela figura a seguir.

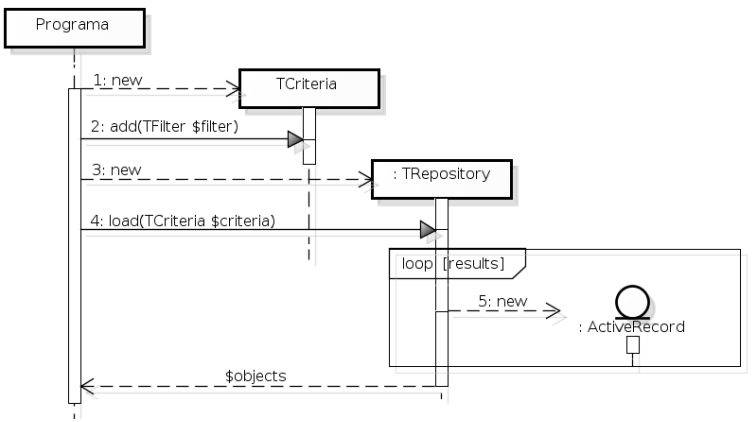


Figura 9 Carregamento de objetos com repositório

3.5.3 Agregação

A agregação também é um relacionamento todo/parte, tal qual a composição. Assim, também temos um objeto "todo" composto de uma ou mais "partes". A diferença primordial entre os dois tipos de relacionamento é que na agregação, um objeto "parte" não é exclusivo do "todo", ou seja, pode pertencer à diferentes objetos "todo". A figura a seguir mostra o exemplo de agregação utilizado neste livro. Neste exemplo, temos objeto `Customer` (cliente) com uma agregação com o objeto `Skill` (habilidades).

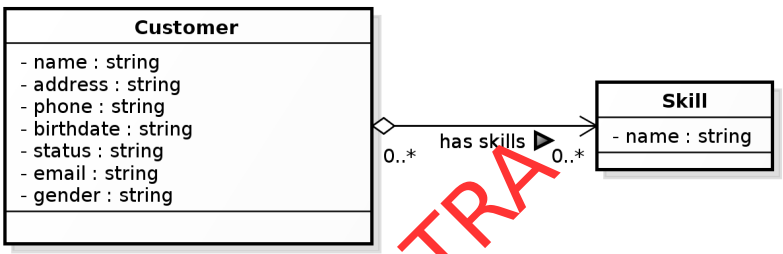


Figura 14 Relacionamento de agregação

Como vimos, na agregação um objeto "parte" pode estar vinculado à diferentes objetos "todo". Neste caso em específico, um cliente pode ter diferentes habilidades (esportes, literatura, música) e uma habilidade pode pertencer à diferentes clientes. Dessa forma, não é possível representar essa estrutura em banco de dados apenas com duas tabelas. Assim, surge a necessidade de uma terceira tabela, chamada de tabela associativa (*Association Table Mapping*). O objetivo desta tabela é armazenar pares de chaves estrangeiras. Nesta tabela, cada linha terá uma chave estrangeira para a tabela `customer` e outra para a tabela `skill`. Assim, poderemos ter vários registros dos relacionamentos entre objetos `Customer` e objetos `Skill`.

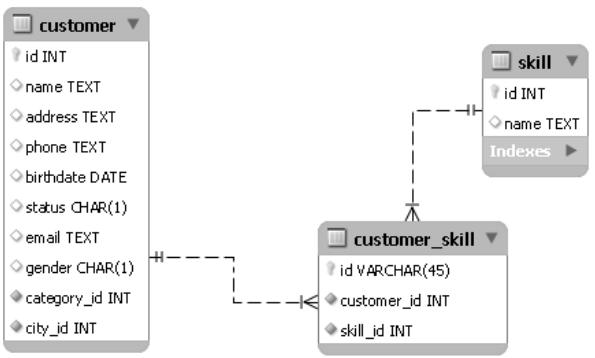


Figura 15 Uma agregação mapeada para o modelo relacional

CAPÍTULO 4

Componentes de apresentação

O foco deste capítulo é explorar os componentes de apresentação do Framework. Portanto, vamos explorar tabelas, painéis e notebooks; componentes para criação de formulários e datagrids; diálogo de mensagens, dentre outros.

4.1 Conceitos básicos

4.1.1 Controlador de páginas

Qualquer elemento a ser apresentado pelo Framework deve estar contido em uma página. Para criarmos uma nova página, devemos criar uma classe. Assim, cada página a ser exibida no framework será uma classe. Existem basicamente dois controladores de páginas: `TPage` e `TWindow`. Qualquer página deve ser subclasse de um desses dois controladores padrão. Enquanto subclasses de `TPage` são exibidas dentro do layout do sistema, subclasses de `TWindow` são exibidas em uma janela (camada).

A seguir temos o exemplo de uma página simples contendo apenas uma mensagem de Hello World. No método construtor da página acrescentamos seu conteúdo, por meio do método `add()`. Aqui estamos utilizando o componente `TLabel`, que representa um rótulo de texto. Neste capítulo, abordaremos outros componentes visuais que podem ser utilizados.

app/control/Presentation/HelloWorld.class.php

```
<?php
class HelloWorld extends TPage
{
    public function __construct()
    {
        parent::__construct(); // é fundamental executar o construtor da classe pai
        parent::add(new TLabel('Hello World'));
    }
}
```

4.2.4 Página na forma de Janela

Qualquer página criada com o Adianti Framework, seja para exibir um conteúdo HTML, um formulário, listagem, dentre outros, também pode ser exibida na forma de janela. Para tal, no lugar de estendermos a classe `TPage`, vamos estender a classe `TWindow`. Nesse caso, podemos usar métodos como `setTitle()`, para definir o título da janela, e `setSize()`, para definir seu tamanho. Todo o conteúdo adicionado à página pelo método `parent::add()` automaticamente é exibido dentro da janela.

app/control/Presentation/Pages/ExternalSystemWindowView.class.php

```
class ExternalSystemWindowView extends TWindow
{
    public function __construct()
    {
        parent::__construct();
        parent::setTitle(_t('PDF inside Window'));
        parent::setSize(0.8, 650);

        $iframe = new TElement('iframe');
        $iframe->id = "iframe_external";
        $iframe->src = "http://www.adianti.com.br/framework_files/template-material/";
        $iframe->frameborder = "0";
        $iframe->scrolling = "yes";
        $iframe->width = "100%";
        $iframe->height = "600px";

        parent::add($iframe);
    }
}
```


A figura a seguir demonstra a janela criada com um sistema externo embarcado.

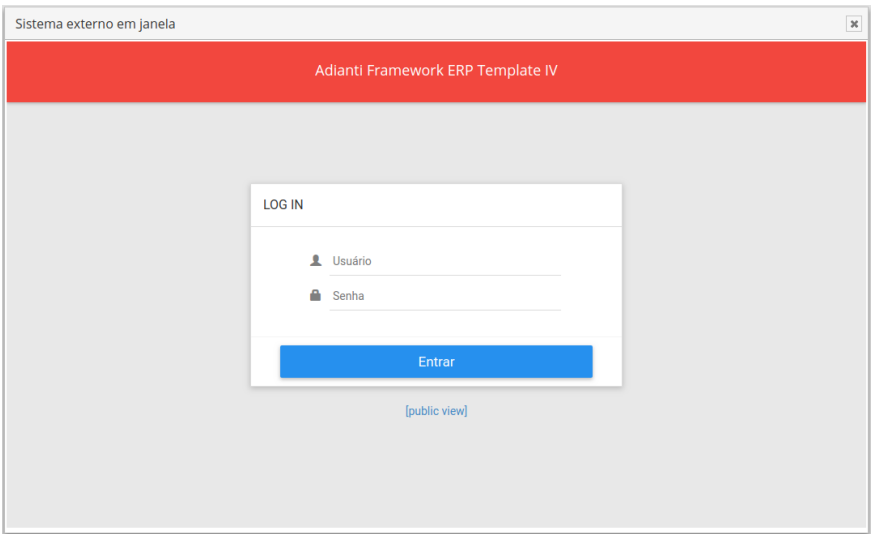


Figura 21 Janela com sistema externo

```
        $row->layout = ['col-sm-2 control-label', 'col-sm-4', 'col-sm-4', 'col-sm-2' ];

        $row = $this->form->addFields( [ new TLabel('Custom 2') ],
                                       [ new TEntry('field_19') ],
                                       [ new TEntry('field_20') ],
                                       [ new TEntry('field_21') ] );
        $row->layout = ['col-sm-2 control-label', 'col-sm-2', 'col-sm-6', 'col-sm-2' ];

        // ...
        $this->form->addAction('Send', new TAction(array($this, 'onSend')),
                                        'fa:check-circle-o green');

        // empacota o conteúdo em uma caixa vertical
        $vbox = new TVBox;
        $vbox->style = 'width: 100%';
        $vbox->add(new TXMLBreadCrumb('menu.xml', __CLASS__));
        $vbox->add($this->form);

        parent::add($vbox);
    }
}
```

Na próxima figura, podemos ver a distribuição automática de campos na tela.

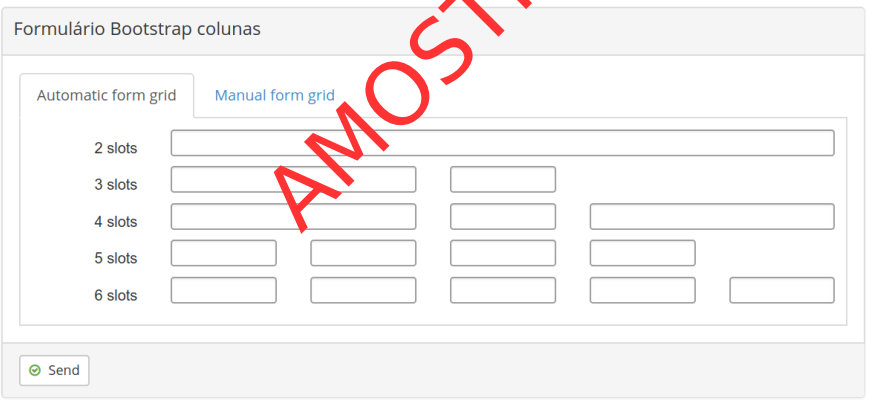


Figura 39 Formulário Bootstrap em colunas – distribuição automática

Na próxima figura, podemos ver a distribuição manual de campos na tela.

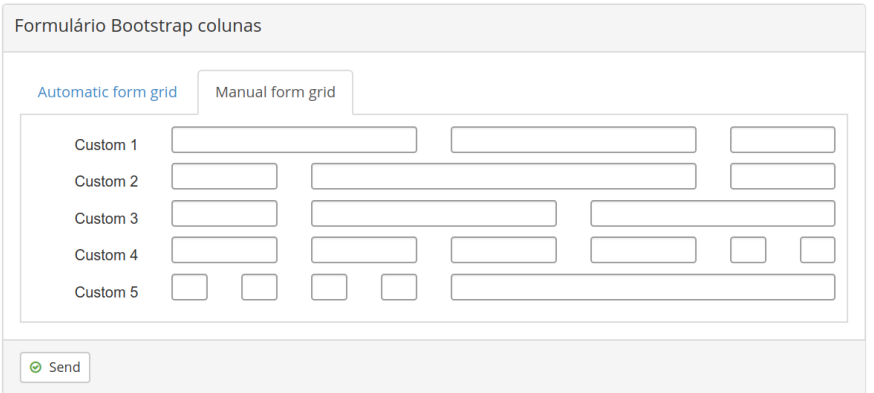


Figura 40 Formulário Bootstrap em colunas – distribuição manual

```
        $this->form->addAction('Send', new TAction(array($this, 'onSend')),
                               'fa:check-circle-o green');

        // empacota tudo usando uma caixa vertical
        $vbox = new TVBox;
        $vbox->style = 'width: 100%';
        $vbox->add(new TXMLBreadCrumb('menu.xml', __CLASS__));
        $vbox->add($this->form);

        parent::add($vbox);
    }
```

O formulário possui uma ação, criada pelo método `addAction()`, que por sua vez está conectada ao método `onSend()`. Este método somente apresenta os dados do formulário em tela para conferência.

```
    public function onSend($param)
    {
        $data = $this->form->getData();
        $this->form->setData($data);

        echo '<pre>';
        print_r($data);
        echo '</pre>';
    }
}
```


A figura a seguir demonstra o formulário criado em execução.

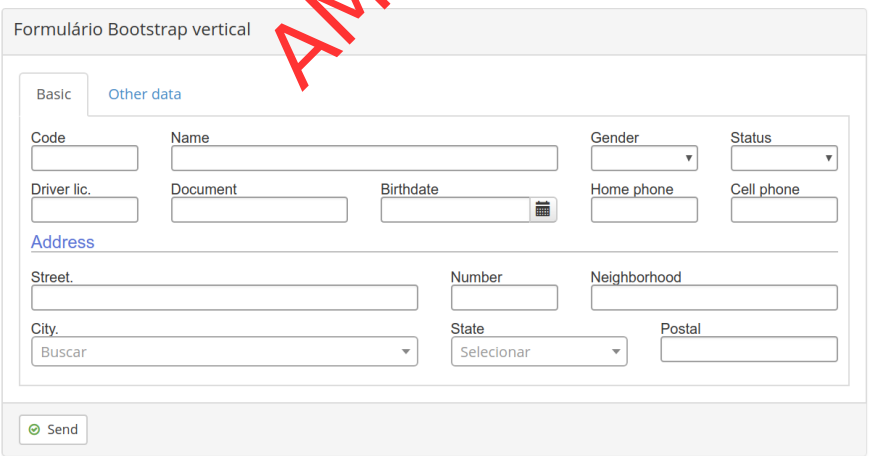


Figura 41 Formulário Bootstrap com rótulos acima

4.5.6 Postagem estática de formulários

Quando enviamos os dados de um formulário por meio de um botão (Salvar, Enviar), somente o núcleo da página é recarregado, e não toda a página. Com isso, a navegação fica bastante fluida. Como o núcleo da página é recarregado, isso nos permite adicionar novos componentes à tela após a postagem ou realizar possíveis atualizações de valores em campos após a postagem da ação. Como por exemplo, podemos citar o campo ID, que geralmente é preenchido após a gravação de um registro novo.

```
        $this->fieldlist->addDetail( new stdClass );
        $this->fieldlist->addDetail( new stdClass );
        $this->fieldlist->addDetail( new stdClass );
        $this->fieldlist->addDetail( new stdClass );
        $this->fieldlist->addDetail( new stdClass );
        $this->fieldlist->addCloneAction();
        $this->form->add($this->fieldlist);

        // cria botão de ação
        $save= TButton::create('save',  array($this, 'onSave'), 'Save', 'fa:save blue');
        $this->form->addField($save);

        $panel = new TPanelGroup(_t('Form field list'));
        $panel->add($this->form);
        $panel->addFooter($save);

        // empacota elementos utilizando uma caixa vertical
        $vbox = new TVBox;
        $vbox->style = 'width: 100%';
        $vbox->add(new TXMLBreadCrumb('menu.xml', __CLASS__));
        $vbox->add($panel);
        parent::add($vbox);
    }
```

O método `onSave()` será utilizado somente para exibir em uma nova janela, todos os dados recebidos. É importante notar que este método é estático, o que faz com que a tela não seja recarregada e os campos mantêm o seu valor mesmo após a postagem. A função `print_r()` é usada para converter o vetor de dados (`$param`), em string, desde que informado o booleano `TRUE` como segundo parâmetro da função.


```
    public static function onSave($param)
    {
        // show form values inside a window
        $win = TWindow::create('test', 0.6, 0.8);
        $win->add( '<pre>'.str_replace("\n", '<br>', print_r($param,
true) ).'</pre>'  );
        $win->show();
    }
}
```

Na próxima figura, podemos ver o resultado da execução deste programa.

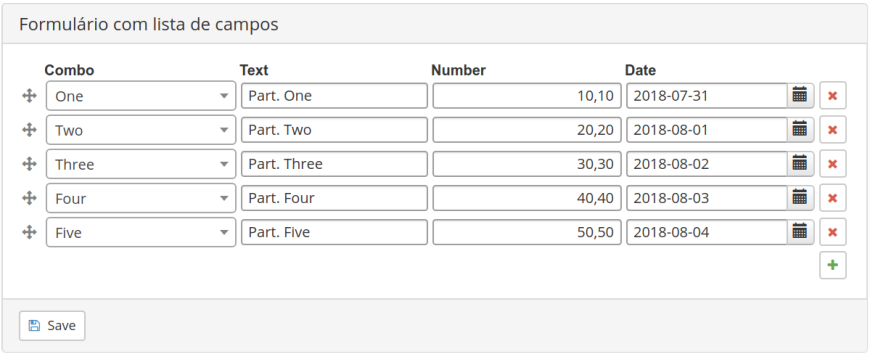


Figura 43 Formulário com múltiplos valores

```
        // cria uma string com os valores dos campos
        $message = 'Radio : ' . $data->radio . '<br>';
        $message.= 'Check : ' . print_r($data->check, TRUE) . '<br>';
        $message.= 'Radio (button) : ' . $data->radio2 . '<br>';
        $message.= 'Check (button) : ' . print_r($data->check2, TRUE) . '<br>';
        $message.= 'Combo : ' . $data->combo . '<br>';
        $message.= 'Combo2 : ' . $data->combo2 . '<br>';
        $message.= 'Select : ' . print_r($data->select, TRUE) . '<br>';
        $message.= 'Search : ' . print_r($data->search, TRUE) . '<br>';
        $message.= 'Unique: '. print_r($data->unique, TRUE) . '<br>';
        $message.= 'Autocomplete: ' . $data->autocomplete;

        // exibe a mensagem
        new TMessage('info', $message);
    }
}
```

Na próxima figura, podemos ver o resultado da execução deste programa.

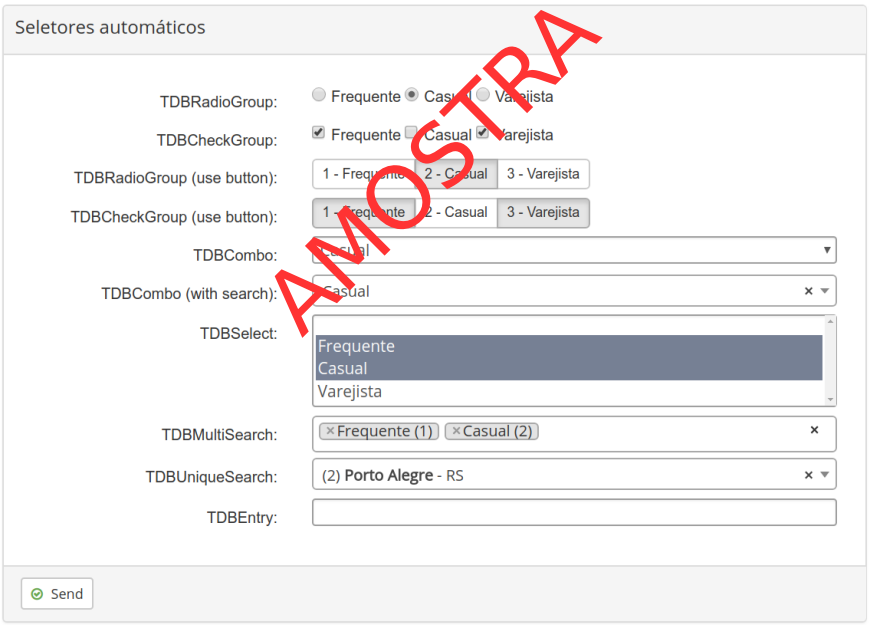


Figura 48 Formulário com seletores automáticos do banco

4.5.15 Interações dinâmicas

Até o momento, vimos vários exemplos de formulários e de componentes. Em todos os casos, somente tivemos acesso aos dados digitados pelo usuário após a postagem do formulário. Mas em muitas situações, precisamos ter acesso aos dados digitados em interações dinâmicas, como em um evento de saída de um campo ou na troca de um elemento de uma combo.

```
        $action3 = new TDataGridAction(array($this, 'onView'));
        $action3->setLabel('View address');
        $action3->setImage('bs:hand-right green');
        $action3->setField('address');

        // cria o agrupamento de ações
        $action_group = new TDataGridActionGroup('Actions', 'bs:th');

        // adiciona as ações ao agrupamento
        $action_group->addHeader('Available Options');
        $action_group->addAction($action1);
        $action_group->addAction($action2);
        $action_group->addSeparator();
        $action_group->addHeader('Another Options');
        $action_group->addAction($action3);

        $this->datagrid->addActionGroup($action_group); // adiciona o agrupamento

        $this->datagrid->createModel(); // cria o modelo

        $panel = TPanelGroup::pack(_t('Datagrid'), $this->datagrid, 'footer');
        parent::add($panel);
    }
    // demais métodos . . .
}
```

Na próxima figura, podemos ver o resultado da execução deste programa.

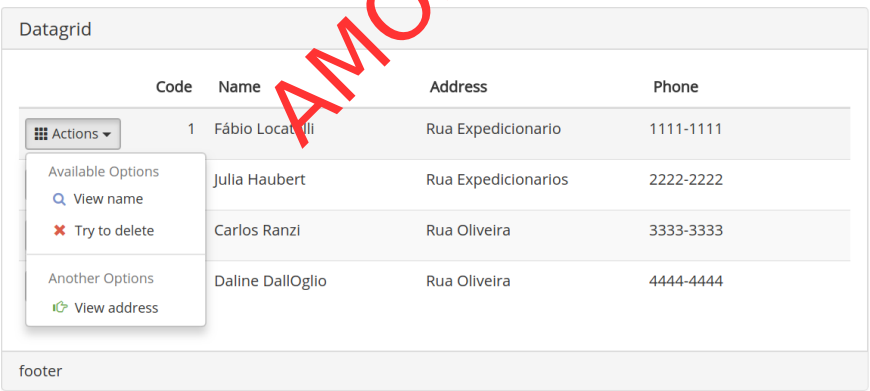


Figura 59 Datagrids com grupos de ações

4.6.5 Datagrids com ações condicionais

Quando criamos uma datagrid, geralmente desejamos exibir todas as suas ações ao usuário. Mas, em alguns casos algumas ações devem ser suprimidas. Em algumas situações, por exemplo, não desejamos exibir o botão de excluir apenas para alguns registros em que essa ação não deve ser executada.

Para permitir esconder a ação do usuário em alguns casos, foi criado o método `setDisplayCondition()` da classe `TDataGridAction`. Este método permite definir um método do usuário como parâmetro. O método definido pelo usuário, que neste caso é `displayColumn()`, receberá o objeto que representa a linha corrente (Active Record) e

```
            // empacota tudo em uma caixa vertical
            $vbox = new TVBox;
            $vbox->style = 'width: 100%';
            $vbox->add( $breadcrumb );
            $vbox->add($this->html);
            parent::add($vbox);
        }
        catch (Exception $e)
        {
            new TMessage('error', $e->getMessage());
        }
    }

    function loadPage()
    {}
}
```

A figura a seguir demonstra a primeira tela do assistente.

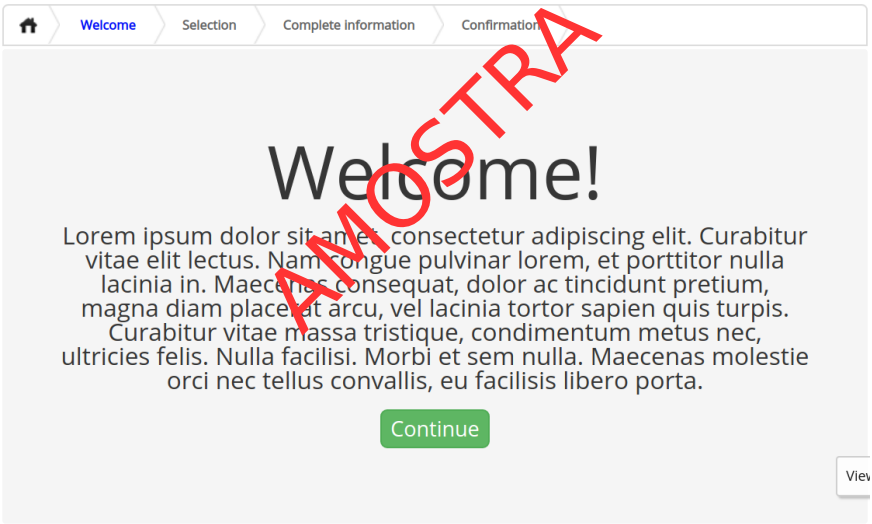


Figura 73 Tela de boas vindas

A segunda página do assistente terá uma datagrid na qual o usuário terá acesso a uma lista de cursos, e terá somente de clicar em um deles para progredir para a próxima etapa. A datagrid possui uma só ação (`onSelect`), que passa como parâmetro o código (`code`) e a descrição (`description`) do curso selecionado. Esta tela também terá uma trilha (`TBreadCrumb`), mas desta vez, o item selecionado será o segundo (Selection). Esta tela também terá um botão de retorno à página anterior (`TActionLink`), que executa um método por meio de uma requisição GET, levando o usuário à página anterior por meio do método `MultiStepRegistration1View::loadPage()`. O botão de retorno é posicionado logo abaixo da datagrid, no rodapé do painel que envolve ela, por meio do método `TPanelGroup::pack()`.

Na próxima figura, podemos ver o resultado da execução deste programa.

Figura 78 Calendário

4.9 Templates e novos componentes

Nessa seção veremos como criar formas de apresentação mais livres para as páginas, por meio de templates HTML, e também por meio de novos componentes.

4.9.1 Template View básico

Por mais que possamos criar componentes de software, existem limitações que devem ser reconhecidas. Não seria prudente nem inteligente construir um framework com milhares de componentes, imaginando todas as necessidades que diferentes projetos possuem. Um Framework é um trabalho inacabado, um arcabouço de software que atenderá talvez 80% das necessidades de um projeto. Sempre que uma necessidade específica não coberta pelo Framework surgir, devemos ter alternativas definidas para sua implementação. No caso do Adianti Framework podemos criar um componente filho de `TElement` em `app/lib`, ou utilizar um Template View.

Partimos para a criação de componentes de software quando a necessidade irá se repetir, tem características claras e uma interface bem definida. Mas às vezes, uma determinada página tem uma necessidade específica de apresentação que não justifica a criação de um componente. Neste caso, podemos utilizar um Template View.

A seguir, você confere parte do relatório gerado no formato PDF.

Customers

Code	Name	Category	Email	Birthdate
1	Andrei Zmievski	Casual	contact@gmail.com	1980-01-01
2	Rubens Prates	Frequente	contact@gmail.com	1990-01-01
3	Augusto Campos	Frequente	contact@gmail.com	1990-01-01
4	Marcelio Leal	Frequente	contact@gmail.com	1990-01-01
5	Manuel Lemos	Frequente	contact@gmail.com	1990-01-01
6	Fábio Locattelli	Frequente	contact@gmail.com	1990-01-01
7	Leonardo Soldatelli	Frequente	contact@gmail.com	1990-01-01
8	Alberto Bengoa	Frequente	contact@gmail.com	1990-01-01
9	Fábio Milani	Frequente	contact@gmail.com	1990-01-01
10	Huberto Meyer	Frequente	contact@gmail.com	1990-01-01
11	Rafael Pavin	Frequente	contact@gmail.com	1990-01-01
12	Eduardo Bacchi	Frequente	contact@gmail.com	1990-01-01

Figura 86 Relatório gerado no formato PDF

4.10.2 Conversão de Templates para PDF

No exemplo anterior, vimos como gerar relatórios em formato de tabela em HTML, PDF, RTF, e XLS. O formato tabular é bom para dados lineares e em grande quantidade. Porém, nem sempre o formato tabular é o mais adequado para relatórios e documentos quando precisamos de uma liberdade maior de criação. Às vezes precisamos de um design mais livre do conteúdo em relação ao documento.

Para apresentar uma forma mais livre de criação de relatórios e documentos, neste exemplo vamos utilizar um Template HTML para montar um relatório, e em seguida converter o mesmo para o formato PDF, e apresentá-lo em uma nova janela para que o usuário possa visualizar, salvar ou imprimir este arquivo.

O exemplo inicia com a utilização da classe `THtmlRenderer` para realizar o carregamento de um Template HTML (`customer_accounts.html`). Em seguida, utilizamos o vetor `$replace` para indicar como o Template será preenchido. Como este exemplo já foi explicado anteriormente no exemplo sobre "Template View com matrizes", vamos focar na conversão do documento em si, não em sua geração.

Após definir pelo vetor `$replace`, como o Template terá suas seções e variáveis preenchidas, utilizamos o método `enableSection()` para habilitar o Template, e preenchê-lo por meio da variável `$replace`. O Template é processado por meio do método `getContents()`, que retornará o resultado do processamento já com o conteúdo.

Para gerar o documento no formato PDF, utilizamos a classe `Dompdf`. Esta classe possui o método `loadHtml()` para carregar o conteúdo HTML, `setPaper()` para definir as propriedades da página, `render()` para renderizar o PDF, e `output()` para gerar a saída do documento, que é escrita em disco pelo método `file_put_contents()`.

Por fim, criamos uma janela (`TWindow`), com o documento em seu interior.

app/control/Presentation/Chart/DashboardView.class.php

```
<?php
class DashboardView extends TPage
{
    function __construct()
    {
        parent::__construct();

        $vbox = new TVBox;
        $vbox->style = 'width: 100%';

        $div = new TElement('div');
        $div->add( $a=new BarChartView(false) );
        $div->add( $b=new LineChartView(false) );
        $div->add( $c=new PieChartView(false) );
        $div->add( $d=new PieChartView(false) );

        $a->style = 'width:50%;float:left;padding:10px';
        $b->style = 'width:50%;float:left;padding:10px';
        $c->style = 'width:50%;float:left;padding:10px';
        $d->style = 'width:50%;float:left;padding:10px';

        $vbox->add(new TXMLBreadCrumb('menu.xml', __CLASS__));
        $vbox->add($div);

        parent::add($vbox);
    }
}
```


Na figura a seguir, podemos visualizar o resultado deste programa.

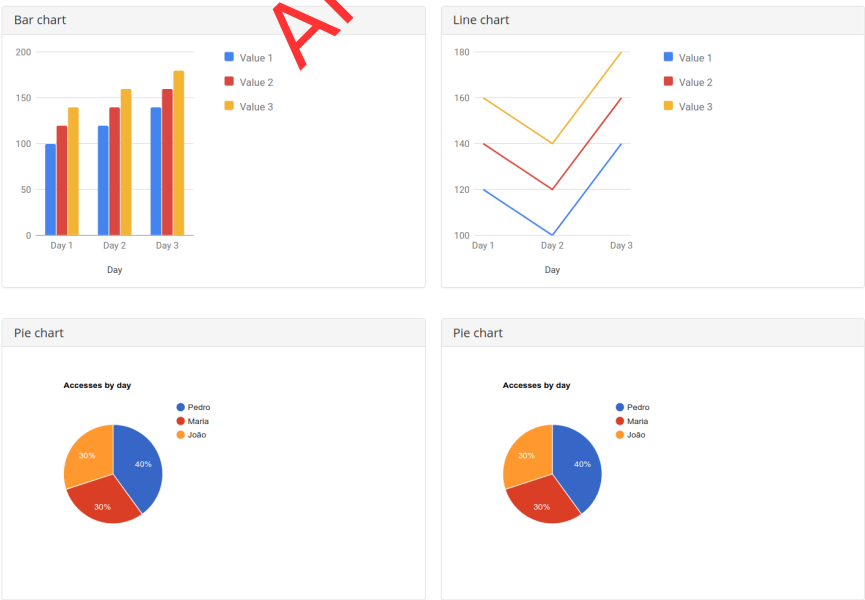


Figura 91 Dashboard

A figura a seguir demonstra o resultado da execução deste programa.

Figura 93 Etiquetas de Barcode

4.12.2 Etiquetas de QRCode

Para demonstrar a criação de etiquetas de QRCode, vamos utilizar o cadastro já existente de produtos, e apresentar um QRCode baseado em seu ID e descrição. Assim como já demonstrado nas etiquetas de códigos de barras, a classe `AdiantiBarcodeDocumentGenerator`, que é responsável pela geração das etiquetas de QRCode, permite que configuremos um modelo (template) para geração das etiquetas. Cada etiqueta pode conter outras informações além do QRCode, tais como códigos, descrições, etc.

Para criar as etiquetas de códigos de QRCode, vamos antes criar um formulário, onde o usuário poderá configurar o modelo (template), onde indicará quais informações deseja exibir nas etiquetas. A figura a seguir demonstra este formulário. No campo do template, o usuário poderá inserir campos como `{$id}`, mas também a posição onde entrará o código de QRCode com a marcação `#qrcode#`.

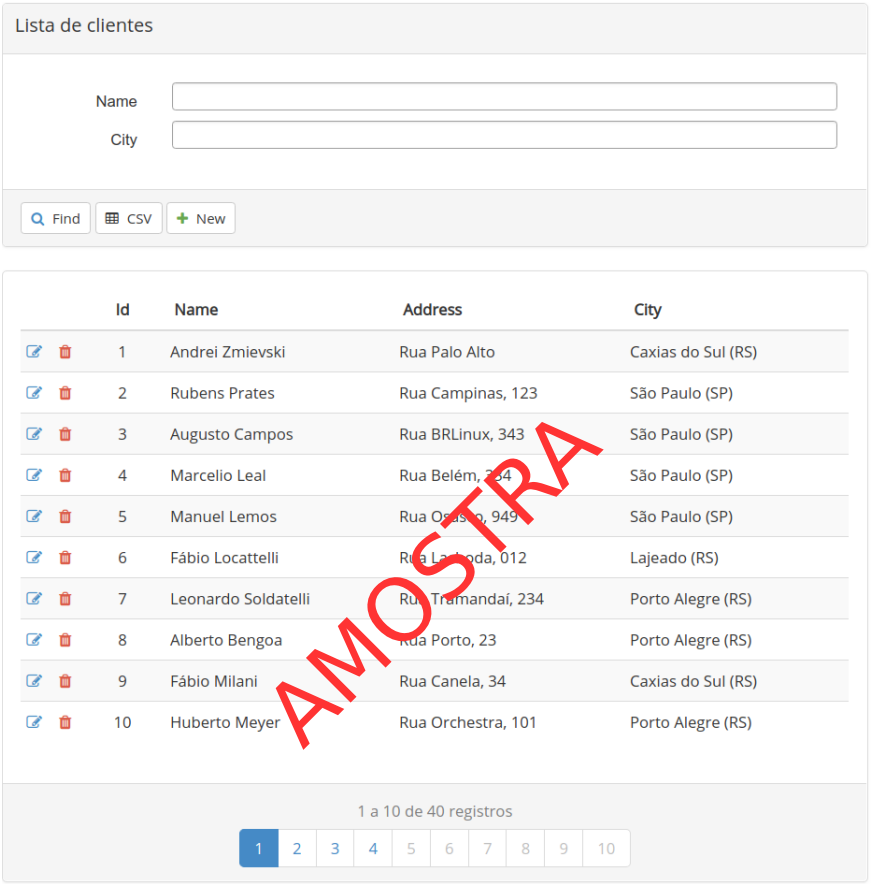


Figura 107 Datagrid de clientes

Conforme pode ser visto no código-fonte a seguir, iniciamos a criação da página pelo formulário (`BootstrapFormBuilder`). Este formulário terá dois campos de buscas (`name` e `city_name`) e três ações, que são: "Find", que utilizará os campos de nome do cliente ou nome da cidade para filtrar a datagrid, por meio do método `onSearch()`; "CSV", que permite exportar os dados da datagrid neste formato, por meio da execução do método `onExportCSV()`, e "Novo", que direcionará o usuário para o formulário de cadastro, representado pela classe `CustomerFormView`, ainda a ser criada.

Após a criação do formulário, é criada a datagrid (`TDataGrid`). A datagrid terá quatro colunas adicionadas por meio do método `addColumn()`, que são: `id`, `name`, `address`, e "`{city->name} ({city->state->name})`", sendo que esta última é representada por uma máscara que é processada para retornar o nome da cidade e o nome do estado, por meio de relações de associação (método `get_city()` na classe `Customer` e método `get_state()` na classe `City`). A datagrid terá duas ações adicionadas pelo método `addAction()`, que são: "Editar", que carrega o objeto selecionado na datagrid para edi-

5.3.5 Formulário mestre-detalhe de vendas

Nos exemplos anteriores, criamos variados tipos de cadastros, mas sempre envolvendo uma única tabela. Neste exemplo, criaremos um formulário para cadastro de vendas, onde teremos um formulário mestre-detalhe, onde o registro mestre terá os dados da venda, e os registros dos detalhes terão os itens da venda.

A figura a seguir demonstra o formulário que criaremos. Na parte superior temos atributos relativos à venda, que é o registro principal. Nesta parte temos atributos como o id, data, cliente, e observação da venda. Já na parte inferior, temos os detalhes da venda. Os detalhes da venda são formados por um formulário de cadastro e edição de itens, bem como de uma datagrid que permite editar ou excluir o registro do item. Os itens possuem informações como o produto, preço, quantidade, e desconto. Ao clicar no botão de editar da datagrid, os dados da linha clicada são transportados para o formulário logo acima para permitir a edição do item. O botão registrar transfere os dados editados para a datagrid. O Salvar, grava a venda, bem como os itens.

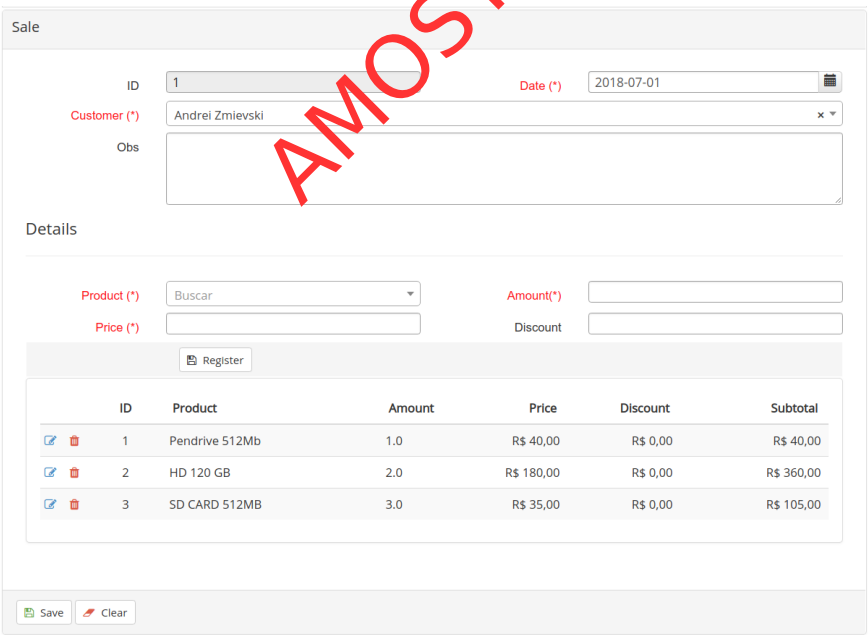


Figura 110 Formulário mestre-detalhe de vendas

Durante a edição de uma venda, os dados dos registros dos detalhes serão armazenados em uma variável de sessão. Isto significa que se o usuário excluir itens, adicionar, ou alterar quantidades, e descontos, estas alterações somente estarão presentes nesta variável de sessão. Os dados dos itens, bem como do registro mestre (data, cliente, observação) somente serão gravados no banco de dados quando o usuário clicar no botão principal para salvar o registro (Save).

5.5.2 Edição de registros em lote

No capítulo 4, criamos uma datagrid contendo campos de entrada de dados. Naquele momento, ainda não havíamos integrado o programa com a base de dados, pois os exemplos tinham como objetivo demonstrar somente características visuais, não de persistência. Neste exemplo, vamos criar uma datagrid em que uma das colunas será exibida na forma de um campo de entrada de dados (`TEntry`), permitindo a alteração de valores diretamente na datagrid. Note que é uma abordagem diferente da edição inline. Enquanto que na edição inline, o campo de edição somente é exibido quando do clique sobre a coluna, na edição em lote, o campo sempre é exibido, permitindo ações como o TAB entre campos. Além disso, é possível utilizarmos outros componentes para edição, tais como a combo, text, senha, dentre vários outros.

Na figura a seguir, podemos conferir o resultado do programa que criaremos a seguir. É possível perceber o formulário de buscas no topo, e a datagrid abaixo, com o campo de preço de venda editável. O formulário ao redor da datagrid não é visível, mas ele está presente, agrupando logicamente os campos.

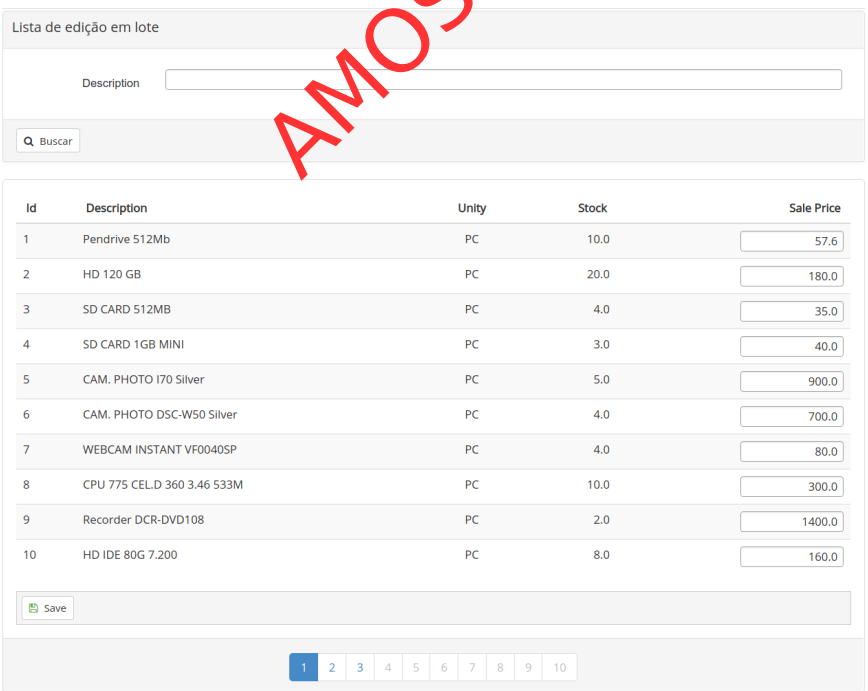


Figura 113 Edição de registros em lote

Para implementar a edição em lote alguns pontos merecem destaque. Em primeiro lugar, é preciso criar um formulário ao redor da datagrid para que este "contenha" os campos criados dentro de suas colunas, e seja responsável pela submissão destas in-

CAPÍTULO 6

Template para criação de sistemas

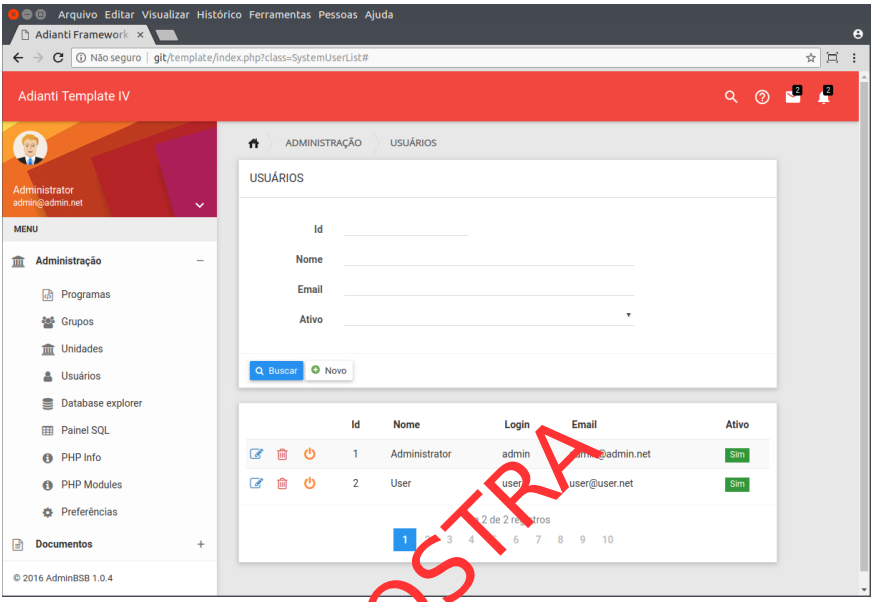
No início, o Adianti Framework era nada mais do que um Framework puro, sendo que cada usuário implementava de sua maneira o controle de login, de permissões de acesso, dentre outros. Até que resolvemos criar o que chamamos de "Template" para fornecer um gabarito padrão para nossos usuários criarem aplicações. O Template permite ao desenvolvedor definir as opções do menu, cadastrar os programas, usuários, grupos, permissões de acesso, sendo que o Template cuida da correta montagem de menus e controle de acesso. Além disso, o Template registra logs de acesso, de alteração de registros, de SQL, e permite aos usuários compartilharem documentos, trocarem mensagens, e receberem notificações do sistema, de uma maneira padronizada.

6.1 Visão geral

Nas primeiras versões do Framework, não havia uma aplicação que demonstrasse, de maneira completa, como implementar um controle de permissões, o que levou diferentes desenvolvedores a criarem mecanismos próprios para esta funcionalidade.

Após um tempo, percebemos que uma necessidade comum devia ser implementada de maneira padronizada, para aumentar a qualidade das aplicações. Dessa maneira criamos o Template, um gabarito para criação de novas aplicações com o Adianti Framework. A cada nova versão, esse gabarito ganha novas funcionalidades. Inicialmente ele oferecia controle de permissões de acesso, por meio do cadastro de usuários, grupos, e programas. Posteriormente foram agregadas funcionalidades de registro de logs de acesso, de alteração de registros e log de SQL, e em seguida de comunicação entre usuários por meio de troca de mensagens e compartilhamento de documentos.

Obs: O Template pode ser baixado em http://www.adianti.com.br/framework-template.



Tema 4 do Template

6.2 Módulo Administração

Nesta seção abordaremos o módulo de Administração, com sua modelagem de classes, tabelas, e funcionalidades.

6.2.1 Diagrama de classes

Na próxima figura, temos o diagrama de classes do sistema, com as classes da camada de modelo relativas ao controle de permissões de acesso (`app/model/admin`). A seguir, temos uma lista com as classes Active Record:

- `SystemUser`: representa um usuário do sistema. Um usuário poderá ter agregação com grupo (`SystemGroup`), e com programa (`SystemProgram`). Isto significa que um usuário poderá possuir vários grupos e vários programas. Além disso, usuário pode ter uma página inicial (*front page*), e estar associado com uma unidade organizacional (`SystemUnit`) principal, dentre várias outras;
- `SystemGroup`: representa um grupo de usuários. Um grupo tem agregação com programa (`SystemProgram`). Isto significa que um grupo poderá possuir vários programas;
- `SystemProgram`: representa um programa. Um programa possui um nome (`name`) e uma classe de controle (`controller`). O atributo `controller` deve ser preenchido com o nome da classe de controle, que desejamos dar acesso.
- `SystemUnit`: representa uma unidade organizacional. Pode ser uma filial ou um departamento de uma organização.

6.2.9 Cadastro de usuários

O cadastro de usuários permite relacionar um usuário a um ou vários grupos, a um ou vários programas, e a uma ou várias unidades. Podemos marcar vários grupos nas checkboxes, o que é representado pela agregação entre `SystemUser` e `SystemGroup`. Também podemos marcar várias unidades individuais ao usuário na área de unidades, o que é representada pela agregação entre `SystemUser` e `SystemUnit`. Também podemos adicionar vários programas individuais ao usuário na área de programas, o que é representada pela agregação entre `SystemUser` e `SystemProgram`.

É importante lembrar que, ao vincularmos o usuário a um grupo, este terá acesso a todos os programas que fazem parte daquele grupo.

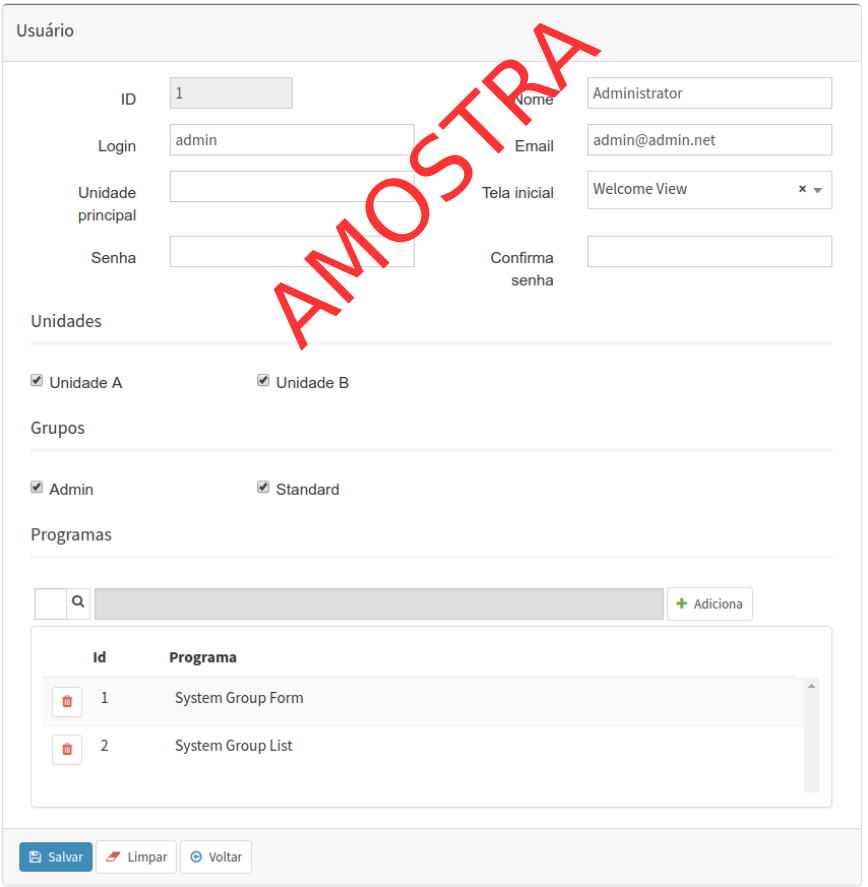


Figura 129 Cadastro de usuários

A figura a seguir apresenta um exemplo de datagrid com rolagem horizontal.

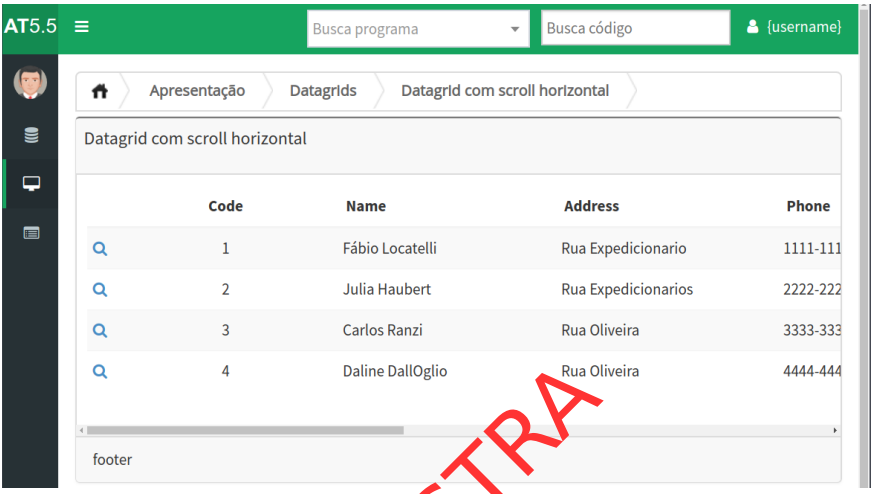


Figura 147 Datagrid com rolagem horizontal

Outra prática que permite exibir mais informações em datagrids sem precisar utilizar muitas colunas, é o popover. O popover, como já visto anteriormente, é um balão exibido quando o usuário passa o mouse sobre o registro. Podemos utilizar este balão para exibir diversas informações relacionadas ao registro.

Exemplo de datagrid com popover

```
$this->datagrid = new BootstrapDatagridWrapper(new TQuickGrid);
$this->datagrid->enablePopover('Item details', '<table> ...{$var} </table>');
```

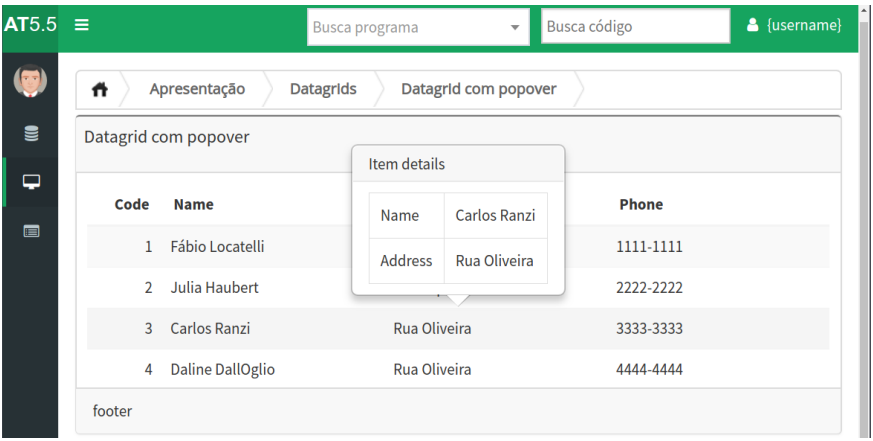


Figura 148 Datagrid com popover

CAPÍTULO 7

Estudos de caso

Após vários capítulos estudando o framework, já nos sentimos preparados para criar as nossas próprias aplicações, uma vez que já vimos como manipular entidades da base de dados, criar uma interface orientada a objetos, e também vimos que o Template nos oferece recursos essenciais na criação de novas aplicações, como: login, controle de permissões, logs, e comunicação, dentre outros. O objetivo deste capítulo é conhecer dois exemplos de aplicações construídas com o Adianti Framework.

7.1 Aplicação Library

A primeira aplicação que vamos estudar, chama-se Library (biblioteca). O objetivo desta aplicação é fornecer os controles mínimos que uma biblioteca precisa para funcionar, sempre visando a demonstração do framework. Esta aplicação não tem como objetivo ser um sistema completo para gerenciamento de bibliotecas, uma vez que já existem inúmeros softwares com esta finalidade.

A aplicação Library fornece algumas funcionalidades como a catalogação de livros, a classificação de livros, cadastro de leitores, autores, assuntos e editoras, empréstimo e devolução de livros, relatórios de livros, leitores e empréstimos, dentre outros.

A aplicação Library é baseada no Template e já herda deste o controle de permissão, compartilhamento de documentos, e também a comunicação entre usuários por trocas de mensagens. Esta aplicação possui diferentes perfis de usuários, sendo que cada perfil possui acesso a um grupo de funcionalidades do sistema. Poderemos ver melhor as funcionalidades acessadas por cada um dos atores, no diagrama de casos de uso. Ao longo deste capítulo, abordaremos como a aplicação está estruturada.

Obs: Baixe o Library no endereço http://www.adianti.com.br/framework-library.

CAPÍTULO 8

Integrações

O objetivo deste capítulo é abordar alguns tópicos adicionais relacionados a integração de outras bibliotecas dentro de nossa aplicação, bem como da integração de nossa aplicação com outras por meio de Web Services.

8.1 Rotas amigáveis

Ao criarmos uma aplicação com o Framework, percebemos que a URL nativa contém a informação da classe e do método que será executado, no seguinte formato:

```
http://www.aplicacao.local/index.php?class=ContactForm&method=onEdit&key=1
```

Como o Framework é utilizado na grande maioria das vezes para criar sistemas corporativos internos, a URL não é tão relevante quanto seria em um web site público ou e-commerce. Mas mesmo em contextos privados e corporativos, é interessante usarmos URL's em um formato amigável para facilitar o compartilhamento de links entre colaboradores. Além disso, o Framework é cada vez mais usado para produzir diferentes tipos de aplicações, como e-commerce, portais corporativos, etc. Dessa forma, vamos verificar como transformar uma URL completa em uma URL resumida:

```
http://www.aplicacao.local/contact-edit?key=1
```

Aplicação de exemplo

Uma aplicação com rotas pré-configuradas é disponibilizada para download junto aos arquivos do Framework. Ela se chama Contacts API, e além de rotas amigáveis possui outros recursos como serviços REST e RESTFul. A aplicação Contacts API já possui rotas pré-definidas para todas as ações previstas no Template, tais como: gerenciamento de programas, grupos de usuários, unidades, preferências, documentos, dentre outros. Caso tenha dúvidas, faça download da Contacts API para verificar uma aplicação já em funcionamento com o recurso de rotas amigáveis.

Adianti Studio

O Adianti Studio é uma IDE Desktop para desenvolvimento PHP que possui uma versão Professional. O Adianti Studio Pro automatiza a criação de código-fonte para o Adianti Framework, tornando muito mais ágil o desenvolvimento de novos sistemas. O Adianti Studio Pro oferece diversos assistentes para geração de código, que são telas passo a passo que atuam sobre uma tabela e geram código, como:

- ❑ Assistente para criação de formulários de edição e consulta;
- ❑ Assistente para criação de listagens com filtros;
- ❑ Assistente para criação de formulários mestre/detalhe;
- ❑ Assistente para criação de relatórios tabulares.

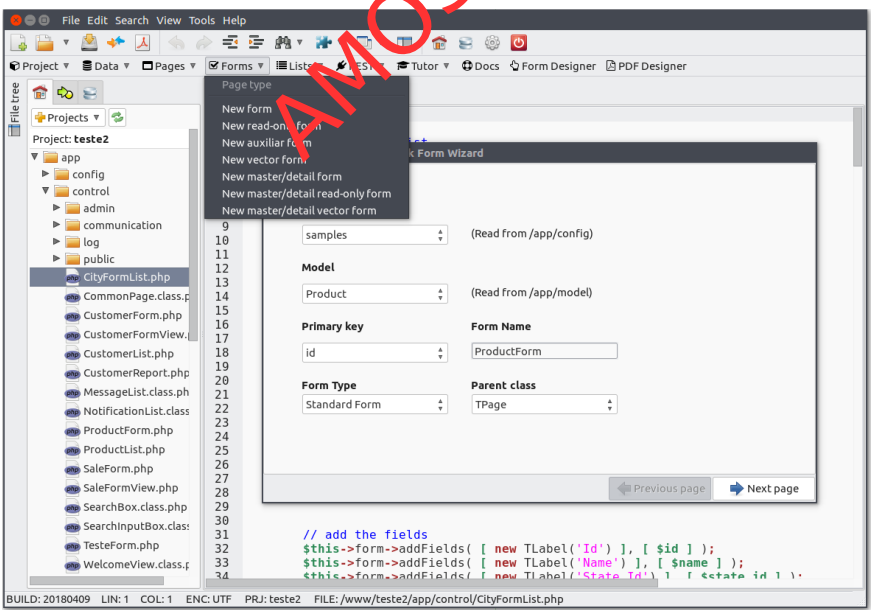


Figura 160 Adianti Studio

Adianti Builder

O Adianti Builder é um construtor online de aplicações, que vai desde a modelagem do banco, até o projeto de telas de formulários, datagrids, relatórios, documentos, gráficos, e outros. Permite realizar a maioria das definições de maneira visual, e também com pontos de edição de código-fonte.

Após modelagem dos dados, é possível criar facilmente formulários, listagens, documentos, relatórios, gráficos, calendários, utilizando definições visuais, e funções de clique e arraste. É possível definir o comportamento de ações como botões de formulário, ações de datagrids, criar códigos personalizados, e muito mais.

O Adianti Builder, além de um construtor visual, também é um editor de código online. Ao final, o usuário poderá realizar download da aplicação, ou enviar diretamente para um de seus servidores por meio da função de deploy.

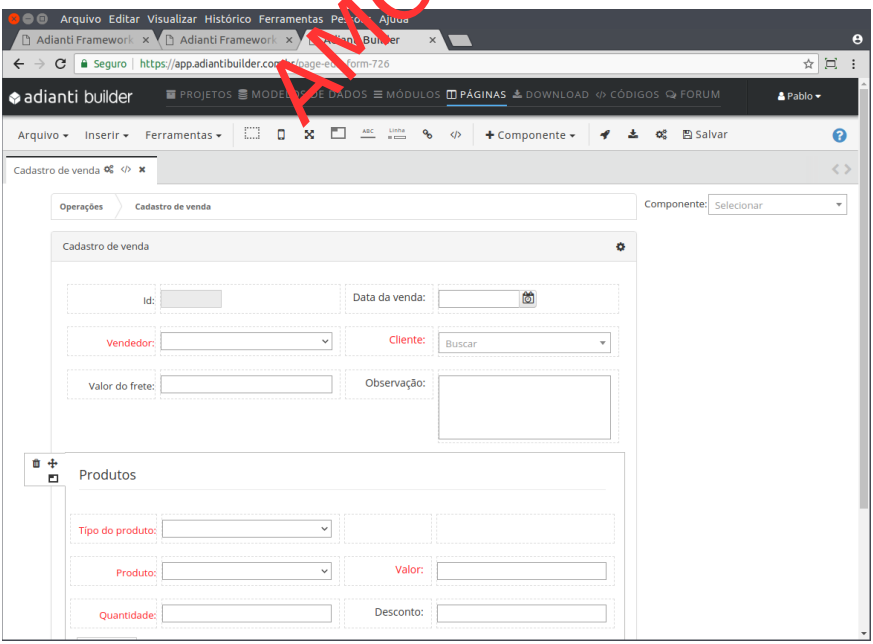


Figura 161 Adianti Builder

Canais Adianti

Não deixe de acompanhar as novidades por nossos canais:

https://www.facebook.com/adiantisolutions

https://twitter.com/adiantisolution

https://www.youtube.com/AdiantiSolutions

Do mesmo autor