

CodeIgniter

Produtividade na criação de
aplicações web em PHP



Casa do
Código

JONATHAN LAMIM ANTUNES

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida

Vivian Matsui

Revisão

Bianca Hubert

Vivian Matsui

Revisão técnica

Carlos Panato

[2016]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

PREFÁCIO

Fiquei muito feliz com o convite de Jonathan Lamim para desenvolver este prefácio. Fica aqui registrado o meu obrigado.

Gostaria de deixar um recado para os leitores, passando por alguns eixos principais: coragem, esforço, dom, compartilhamento e código aberto. Pode parecer que não, mas eles estão conectados, como quase tudo na vida.

Escrever um livro não é tarefa fácil. É preciso coragem, ainda mais atualmente, em que muito do que se busca já se encontra aqui em pequenas porções de informação, ora certas, ora equivocadas. Mesmo assim, o nobre autor se aventurou e se firmou na ideia de desenvolver este projeto. Diante desse posicionamento, bastante comum na comunidade de software livre e código aberto, nos resta aplaudir, ainda mais por se tratar de um tema específico e não feito para uma massa qualquer.

Aliás, penso que desenvolver aplicações e programar profissionalmente não são tarefas para qualquer um. E sabe por quê? Por ser uma arte. E assim como em toda arte, existem artistas. Qualquer um pode rabiscar, mas dizer que qualquer um pode ser Leonardo Da Vinci é, no mínimo, engraçado, para não dizer improvável. Então, me desculpem aqueles que acham que qualquer um pode programar bem.

Pensar logicamente é tarefa de todos, e nisso concordo plenamente. Esta talvez seja a principal razão por que nossos estudantes odeiem tanto matemática e demais matérias que envolvam cálculo: é preciso pensar logicamente. Pode parecer que não, mas a lógica é a base, é o miolo de qualquer desenvolvimento, de qualquer aplicação.

Agora imagine o começo da carreira de alguém que tem o dom de programar, desses que não se encontram em qualquer lugar. Encontrar boa bibliografia de estudo é um martírio para esse tipo de indivíduo. Aliás, encontrar bons livros sobre desenvolvimento é uma tarefa duplamente complexa, principalmente porque dentro de tantas linguagens, softwares, metodologias e definições, aquele que está começando tem um mundo em suas mãos, mas não sabe manuseá-lo. Imagine só como fica a cabeça de alguém que está começando neste admirável mundo novo?

Para quem gosta, seria o mesmo que soltar uma criança em um parque munido de um cartão com créditos infinitos, porém sem ter alguém para dizer como usar tantos brinquedos legais. Muitos vão na tentativa e erro, mas sabemos que nem sempre é assim que funciona, e que tal metodologia não funciona para todos. Esse indivíduo, então, tentará se conectar com outros pares, outros que pensam como ele. Onde esse "garoto de programa" vai parar? Lá mesmo, na internet, onde todo mundo está.

E como não seria diferente, se a linguagem escolhida for de código aberto, será muito mais gostoso :-). As comunidades de desenvolvedores em torno de linguagens de código aberto, não só no Brasil, mas principalmente fora, são bem ativas. Elas vão acolher esse novo membro. Ao desenvolver com código aberto, um mundo de possibilidades se abre, basta visitar repositórios como o GitHub, que abriga inúmeros projetos.

O GitHub, assim como vários outros repositórios de códigos, fazem jus à cultura do compartilhamento, que já tomou conta da internet e dos negócios, e tem como um de seus pilares o ato de compartilhar.

Veja bem: aprendemos desde criança que devemos compartilhar nosso lanche, o material de escola, o alimento. Quando crescemos, dizem que não é assim. Dizem que agora é concorrência e que é

errado compartilhar. Onde você acha que está o erro? Então, apegue-se a esse ideal, desenvolva com qualidade e bola para frente! Quem sabe essa bola não vira um *game* que vai parar na sua loja de aplicativos preferida?

Um abraço e *happy coding*!

— João Fernando Costa Júnior (Iniciativa Espírito Livre)

SOBRE O AUTOR



Tudo começou em 2004, quando entrei para a escola técnica. Lá, estudei informática, e aprendi sobre manutenção, redes de computadores e sistemas operacionais, mas o que me atraiu mesmo foram as matérias relacionadas à programação. Aprendi a programar usando Delphi, depois comecei a estudar JavaScript e HTML, e foi aí que me apaixonei por desenvolvimento para web.

Em 2005, concluí o curso técnico e me mudei do interior de Minas Gerais para o Espírito Santo, onde comecei a ter oportunidades de colocar em prática tudo o que já havia aprendido. Comecei então a escrever artigos sobre desenvolvimento web, dar aulas de programação e informática básica, e auxiliar alunos de uma escola particular durante as aulas no laboratório de informática.

Com o passar do tempo, fui me aprofundando nos estudos sobre desenvolvimento web, passei a colaborar com projetos open source e a visão foi se abrindo ainda mais. Quanto mais eu aprendia, mais eu queria ensinar, compartilhar. Já são mais de 300 artigos escritos, muitas horas de aulas ministradas, várias palestras e hangouts, e ainda sinto que posso compartilhar muito mais conteúdo.

Para conhecer um pouco mais sobre o meu trabalho e meus artigos, acesse os links:

- **Site pessoal:** <http://www.jlamim.com.br>
- **Facebook:** <https://www.facebook.com/jonathanLamim>

AGRADECIMENTOS

Antes de tudo, meus agradecimentos são a Deus, por ter me dado saúde e força para trabalhar neste livro. Sem Ele, nada disso teria sido possível.

À editora Casa do Código pela oportunidade, não só de compartilhar conhecimento, mas também de aprender ainda mais durante o processo de produção deste livro. Vivian, obrigado pela paciência e pelas repetidas revisões de um mesmo capítulo, por conta da minha insatisfação com o conteúdo que eu mesmo escrevi.

Um agradecimento que não tem medida à minha esposa, Juliana, que me incentivou durante todo o período em que estive envolvido nesse trabalho e que compreendeu as vezes em que eu precisei abrir mão de uma sessão de cinema, um filme na TV, uma viagem ou simplesmente horas de bate-papo com ela para me dedicar ao projeto deste livro.

À minha família, que me proporcionou a oportunidade de estudar e poder encontrar na área de TI a minha profissão e a realização de vários sonhos. Pai, mãe e irmãos, obrigado por todo o apoio que sempre me deram e ainda dão.

Obrigado a você que está lendo esta obra, que acreditou que o conteúdo dela pode ser útil para o seu crescimento profissional.

SOBRE O LIVRO

O livro traz, por meio de dois projetos completos, a aplicação de recursos, bibliotecas, boas práticas, dicas e um pouco de teoria sobre o desenvolvimento de sites e sistemas, utilizando o framework CodeIgniter. Com os conhecimentos adquiridos neste livro, você será capaz de desenvolver sites e sistemas com qualidade e rapidez, usando a linguagem PHP e esse framework.

Durante a leitura, você vai encontrar diversos cenários dentro das duas aplicações onde utilizaremos os recursos do CodeIgniter para poder desenvolver a solução ideal, de forma rápida, prática e funcional. A teoria será mesclada com a prática. Assim, enquanto você escreve um código, você aprende o que ele faz, em vez de aprender o que ele faz, para que serve e, somente depois, escrever o código e ver o resultado.

O foco deste livro não é ensinar PHP, HTML, CSS e/ou JavaScript, mas sim ensinar a utilizar o framework CodeIgniter de forma produtiva e eficiente. Para um bom aproveitamento do conteúdo, você deverá ter noções sobre:

- Conhecimento sobre gerenciamento de arquivos (criação de pastas, renomear pasta, compactar e descompactar arquivos);
- Montagem de ambiente de desenvolvimento com PHP, MySQL e Apache em ambiente Linux e Windows;
- PHP;
- MySQL básico;
- HTML básico;
- CSS básico;
- Bootstrap básico.

Para agilizar o processo de estruturação das telas durante os exemplos, vamos usar também o framework Bootstrap (<http://getbootstrap.com>).

Os exemplos completos estarão disponíveis no GitHub (<https://github.com/jlamim/livro-codeigniter>) e nos tutoriais de apoio no portal Universidade CodeIgniter (<http://www.universidadecodeigniter.com.br>).

Caso tenha dúvidas durante a leitura e execução dos exemplos, você pode publicá-las no fórum da Casa do Código, em <http://forum.casadocodigo.com.br/>.

Boa leitura!

UNIVERSIDADE CODEIGNITER

O **Universidade CodeIgniter** é um portal com conteúdo exclusivamente em português sobre o framework CodeIgniter. Nele, você encontrará dicas e tutoriais sobre o framework para os mais diferentes níveis de conhecimento, indo desde o básico ao avançado, sem fazer distinção.

Acesse o site: <http://www.universidadecodeigniter.com.br>.

Qual o objetivo?

O projeto tem como objetivo disponibilizar conteúdo exclusivamente em português sobre o framework CodeIgniter, desde o básico até o avançado.

Como surgiu?

Atualmente, temos muito pouco material sobre o CodeIgniter escrito em português, sejam tutoriais, livros ou até mesmo a documentação oficial. Essa falta de material e o volume constante de dúvidas enviadas aos grupos de discussão nas redes sociais foram o que motivou a criação desse canal de compartilhamento de conhecimento.

Quem cuida do projeto?

O projeto tem como curador Jonathan Lamim, autor deste livro, programador PHP com mais de 10 anos de experiência em desenvolvimento web. Além de gerar conteúdo para o site, ele faz a revisão dos conteúdos enviados por outros colaboradores.

Como ser um autor?

Qualquer desenvolvedor com conhecimento em CodeIgniter pode publicar conteúdo no portal. Para isso, basta preparar um post e enviar para a curadoria pelo e-mail contato@universidadecodeigniter.com.br.

Ao enviar o post, envie as imagens usadas e o código-fonte de exemplo do post, tudo em um único arquivo compactado.

Sumário

1 Introdução ao CodeIgniter	1
1.1 Requisitos mínimos	2
1.2 Instalando o CodeIgniter	3
1.3 Estrutura de arquivos e diretórios do CodeIgniter	6
1.4 Alterando a localização do diretório system	10
1.5 Alterando o idioma das mensagens padrões	11
1.6 Conclusão	12
2 Anatomia de um model	13
2.1 Como carregar um model	14
2.2 Carregando um model no autoload	15
2.3 Conclusão	15
3 Anatomia de um controller	16
3.1 Enviando parâmetros por meio da URL	17
3.2 Nomes reservados	18
3.3 Conclusão	20
4 Anatomia de uma view	22
4.1 Carregando uma view	22
4.2 Enviando dados para a view	24
4.3 Retornando uma view como string	25

4.4 Usando Template Parser na view	26
4.5 Conclusão	26
5 Criando um site institucional — Parte I	28
5.1 Cenário e estrutura inicial	28
5.2 Montando a home	31
5.3 Montando as páginas sobre a empresa e serviços	38
5.4 Criando e configurando as rotas	39
5.5 Passando dados do controller para a view	51
5.6 Comprimindo o HTML de saída com um hook do CI	52
5.7 Conclusão	54
6 Criando um site institucional — Parte II	56
6.1 Configurando o cache para as páginas do site	56
6.2 Criando as páginas de Fale Conosco e Trabalhe Conosco	59
6.3 Criando a página do Fale Conosco	61
6.4 Enviando os dados do formulário de contato por e-mail	69
6.5 Criando a página do Trabalhe Consoco	74
6.6 Conclusão	84
7 Validando formulários	86
7.1 Carregando a library	86
7.2 Aplicando as regras de validação	87
7.3 Regras de validação da library Form Validation	91
7.4 Criando suas próprias regras de validação	95
7.5 Criando mensagens de erro	97
7.6 Executando as regras de validação	99
7.7 Recuperando os dados dos campos do formulário	100
7.8 Exibindo as mensagens de erro	101
7.9 Conclusão	102

8 Enviando e-mails com a library Email	104
8.1 Enviando um e-mail simples	104
8.2 Enviando e-mail usando uma view como template da mensagem	105
8.3 Enviando e-mail com anexo	106
8.4 Envio de e-mail com SMTP	107
8.5 Parâmetros de configuração	108
8.6 Outros métodos da library Email	111
8.7 Conclusão	112
9 Gerenciando sessões com a library Session	113
9.1 Configurando a sessão	113
9.2 Carregando a library e inicializando a sessão	115
9.3 Trabalhando com sessão temporária	115
9.4 Trabalhando com sessão permanente	118
9.5 Armazenando sessões no banco de dados	120
9.6 Armazenando sessões em arquivos físicos	120
9.7 Armazenando sessões com Redis	121
9.8 Armazenando sessões com Memcached	121
9.9 Conclusão	122
10 Upload, download e compressão de arquivos	124
10.1 Upload	124
10.2 Download	130
10.3 Compressão de arquivos	131
10.4 Conclusão	134
11 Implementando CAPTCHA nativo	136
11.1 Carregando o helper	137
11.2 Gerando o CAPTCHA	138
11.3 Adicionando o valor do CAPTCHA à sessão	141

11.4 Exibindo a imagem no formulário	141
11.5 Validando o CAPTCHA	143
11.6 Conclusão	145
12 Criando um encurtador de URLs — Parte I	147
12.1 Sobre o encurtador	147
12.2 Criando a estrutura do projeto e o banco de dados	148
12.3 Preparando as rotas	154
12.4 Criando o model Urls_model	156
12.5 Criando o model User_model	161
12.6 Criando o controller Urls	164
12.7 Conclusão	171
13 Criando um encurtador de URLs — Parte II	173
13.1 Criando o controller User	173
13.2 Criando as views	179
13.3 Conclusão	190
14 Trabalhando com banco de dados	192
14.1 Configurando uma conexão com o banco de dados	192
14.2 Inicializando a library Database	195
14.3 Executando consultas com <code>\$this->db->query()</code>	196
14.4 Query Helper	198
14.5 Query Builder	199
14.6 CRUD	205
14.7 Conclusão	206
15 Paginação de resultados	208
15.1 Introdução à library Pagination	208
15.2 Implementando a paginação no encurtador de URL	212
15.3 Conclusão	217

16 Usando template parser	218
16.1 Introdução	218
16.2 Inicializando a library	220
16.3 Aplicando o template parser na view	220
16.4 Chamando o template parser no controller	221
16.5 Usando o template parse para uma string	223
16.6 Conclusão	226
17 Manipulando imagens	227
17.1 Bibliotecas nativas do PHP suportadas	228
17.2 A library Image Manipulation	228
17.3 Configurando o upload de imagem	229
17.4 Processando o upload	231
17.5 Criando um thumbnail da imagem original	232
17.6 Redimensionando uma imagem	238
17.7 O método resize()	242
17.8 Rotacionando uma imagem	244
17.9 O método rotate()	247
17.10 Recortando uma imagem	248
17.11 O método crop()	251
17.12 Inserindo marca d'água na imagem	252
17.13 O método watermark()	255
17.14 Conclusão	256
18 Trabalhando com Composer	257
18.1 Adicionando, atualizando e removendo dependências	258
18.2 Testando as dependências instaladas	260
18.3 Conclusão	261
19 Poupando tempo de desenvolvimento com funcionalidades nativas do CodeIgniter	262

19.1 Trabalhando com URLs	262
19.2 Trabalhando com textos	266
19.3 Trabalhando com strings	269
19.4 Mapeando diretórios	270
19.5 Conclusão	271
20 Migrando um projeto da versão 2.x para a 3.x	273
20.1 Atualize o diretório system	273
20.2 Atualize o nome das classes	274
20.3 Atualize o arquivo config/mimes.php	275
20.4 Atualize o arquivo config/autoload.php	275
20.5 Mover as alterações da classe Log ou extensões	275
20.6 Atualização para as novas features da library Session	276
20.7 Atualize o arquivo config/database.php	276
20.8 Substitua os templates de erro	276
20.9 Atualize o arquivo config/routes.php	277
20.10 Funções e métodos com mudança do valor retornado	278
20.11 Uso do filtro de XSS	279
20.12 Uso de get_post()	280
20.13 Atualização de mensagens do form_validation	281
20.14 Mudanças menores	281
20.15 Atenção com as funcionalidades descontinuadas	283
20.16 Conclusão	289
21 Mantendo a estrutura de banco de dados atualizada com Migrations	291
21.1 Ajustando as configurações	291
21.2 A lógica de execução das migrations	292
21.3 Projeto prático	293
21.4 Conclusão	300

22 Apêndice A	302
22.1 Como ativar o mod_rewrite no Apache em um servidor Linux	302
22.2 Links úteis	303
23 Apêndice B	304
23.1 Instalando o Redis	304
23.2 Saiba mais sobre o Redis	305
24 Apêndice C	307
24.1 Biblioteca GD	307
24.2 Links úteis	309
25 Conclusão	310
25.1 Links úteis	311

Versão: 19.5.29

INTRODUÇÃO AO CODEIGNITER

"Não é o mais forte que sobrevive, nem o mais inteligente. Quem sobrevive é o mais disposto à mudança." — Charles Darwin

O CodeIgniter (ou **CI**, como muitos chamam, e nós também) é um framework MVC open source, escrito em PHP e mantido atualmente pelo *British Columbia Institute of Technology* e por uma grande comunidade de desenvolvedores ao redor do mundo. Sua simplicidade faz dele um dos mais utilizados e com uma curva de aprendizado bem pequena.

Sua documentação é bem completa e detalhada, facilitando o processo de pesquisa sobre determinada biblioteca ou funcionalidade. Com isso, o tempo gasto com a leitura da documentação diminui, e você pode passar mais tempo trabalhando no código do seu projeto.

Com o CI, é possível desenvolver sites, APIs e sistemas das mais diversas complexidades, tudo de forma otimizada, organizada e rápida. Suas bibliotecas nativas facilitam ainda mais o processo de desenvolvimento, e ainda permitem ser estendidas para que o funcionamento se adapte à necessidade de cada projeto. Diversas bibliotecas de terceiros (*third-party*) estão disponíveis no GitHub, Composer e em outros repositórios de arquivos, e podem ser muito úteis.

Atualmente, ele está na versão 3.0.6, sob a licença MIT, e com muitas melhorias em relação à versão 2.x (que ainda continua disponível), principalmente nas questões de segurança, banco de dados, otimização para o PHP 7 e a possibilidade de utilizar o Composer para o gerenciamento de bibliotecas de terceiros de forma nativa.

Mais detalhes sobre o CI podem ser vistos no site do framework (<https://codeigniter.com>) e em sua documentação (https://codeigniter.com/user_guide).

1.1 REQUISITOS MÍNIMOS

Para um melhor aproveitamento, é recomendado o uso de PHP 5.4 ou mais recente. Ele até funciona com PHP 5.2, mas recomendo fortemente que você não utilize versões antigas do PHP, por questões de segurança e desempenho potenciais, bem como recursos ausentes.

Algumas aplicações fazem uso de banco de dados, e o CodeIgniter suporta atualmente:

- MySQL(5.1+), mysqli e PDO drivers
- Oracle (drivers oci8 e PDO)
- PostgreSQL (postgre a PDO)
- MSSQL (mssql, sqlsrv e PDO)
- Interbase/Firebird (ibase e PDO)
- ODBC (odbc e PDO)

O CI pode ser instalado em sistemas operacionais UNIX (Linux e Mac OS X) e Windows, desde que o ambiente de desenvolvimento com Apache ou Nginx (UNIX) e IIS (Windows) estejam devidamente montados e funcionando.

1.2 INSTALANDO O CODEIGNITER

O processo de instalação do CI é muito simples e fácil de ser executado. O primeiro passo é fazer o download do framework. Para isso, você tem duas possibilidades:

Direto do site

Clique em *Download* na home, conforme mostrado na figura:



Figura 1.1: Download direto pelo site

Pelo repositório no GitHub

Acesse o repositório do projeto no GitHub (<https://github.com/bcit-ci/CodeIgniter>) e faça o download clicando em *Download ZIP*, conforme a figura a seguir:

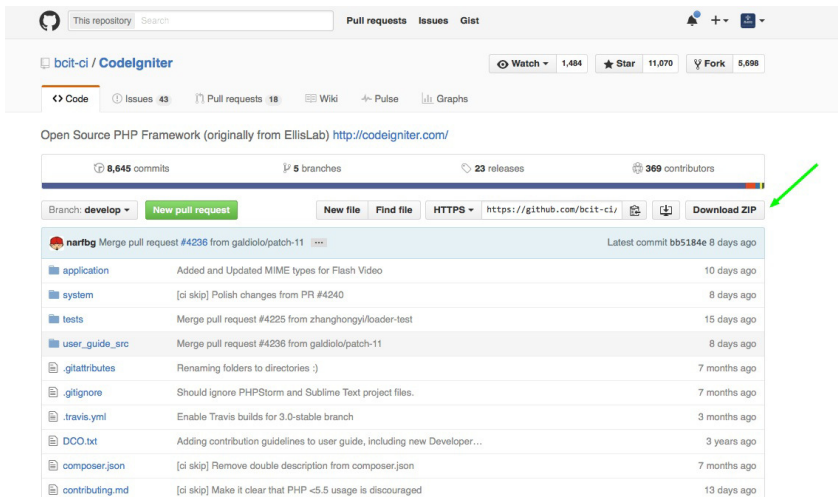


Figura 1.2: Repositório do CodeIgniter no GitHub

Após fazer o download, descompacte o arquivo no diretório onde codificará o projeto, chamando esse diretório de `instalacao-ci`. Mantenha o arquivo ZIP que você baixou em algum local na sua máquina, pois ele pode ser útil no decorrer do livro, e assim não há a necessidade de um novo download.

DIRETÓRIO EXCLUSIVO PARA OS EXEMPLOS DO LIVRO

Crie um diretório exclusivo para os exemplos do livro, pode chamá-lo de `exemplos-livro-ci`. Assim, fica tudo organizado e fácil de você localizar os códigos conforme forem sendo referenciados nos capítulos.

Feito isso, sua estrutura de arquivo no diretório `instalacao-ci` deve ser como a mostrada a seguir:

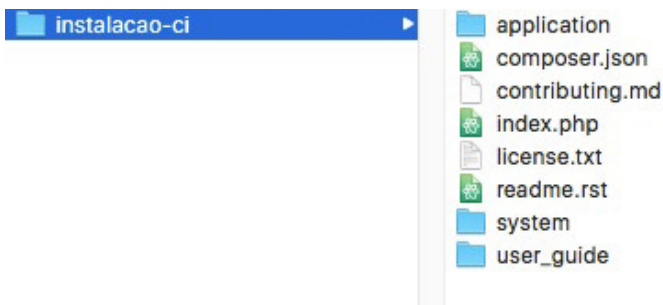


Figura 1.3: Estrutura de arquivos após a instalação

Dos arquivos e diretórios da estrutura, você pode remover o diretório `user_guide`, pois ele contém apenas a documentação do CI, e os arquivos `contributing.md` e `readme.rst`, que trazem informações sobre como contribuir com o desenvolvimento do CI e informações sobre eles, respectivamente.

Feito isso, você pode testar a instalação acessando a URL correspondente ao diretório no seu servidor local. Essa URL pode variar, mas levando em consideração que você esteja utilizando `localhost` e nomeou os diretórios conforme orientado anteriormente, ela seria a seguinte:

<http://localhost/exemplos-livro-ci/instalacao-ci>

E o resultado deverá ser:

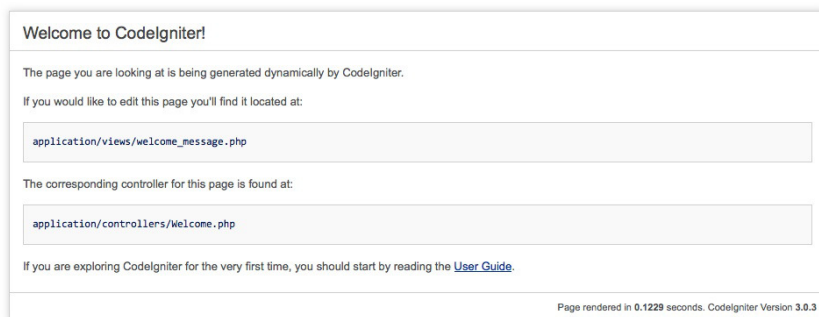


Figura 1.4: Resultado da instalação

Pronto, você já instalou o CI e agora é hora de conhecer um pouco mais sobre a estrutura de arquivos e diretórios dele.

1.3 ESTRUTURA DE ARQUIVOS E DIRETÓRIOS DO CODEIGNITER

Agora que o CodeIgniter já está instalado e funcionando, mostrarei com mais detalhes a estrutura de arquivos e diretórios. É importante conhecê-la para que possa organizar o seu código da melhor maneira possível, e fazer o uso correto dos recursos que o CI disponibiliza.

Diretório *application*

Esse é o diretório da aplicação, onde ficarão todos os arquivos relacionados ao projeto. Ele é composto de 12 diretórios, que farão com que os arquivos do projeto fiquem bem divididos e separados dos arquivos do "*core*" do CI. Assim, você poderá atualizar a versão do CI sem ter de mudar os arquivos do projeto de diretório ou estrutura.

- *cache* — Diretório que armazena os arquivos que são colocados em cache.
- *config* — Diretório que armazena os arquivos de configuração do CI, como por exemplo, *database.php* , *constants.php* , *routes.php* , entre outros que veremos no decorrer dos capítulos.
- *controllers* — Diretório que armazena os arquivos com os controllers do projeto. Ao instalar o CI, ele já vem com um controller criado, que é o *welcome.php* .
- *core* — Diretório usado para estender classes e funcionalidades do *core* do CI, adaptando-se às

necessidades do projeto.

- `helpers` — Diretório que armazena os arquivos com funções que funcionarão como assistentes durante o desenvolvimento. Por exemplo, você pode criar um helper (assistente) para determinado grupo de tarefas realizadas com frequência dentro do projeto, ou então estender as funcionalidades dos *helpers* nativos do CI.
- `hooks` — Diretório que armazena os arquivos que também estendem funcionalidades padrão do CI. Você pode criar um hook para alterar uma funcionalidade padrão, como por exemplo, minificar o código HTML de saída quando o método `load->view()` for executado.
- `language` — Diretório que armazena os arquivos com os dicionários de idiomas, assim você pode desenvolver projetos multi-idíomas de forma fácil, e também utilizar os outros idiomas dos pacotes de tradução oficiais do CI, que podem ser encontrados em <https://github.com/bcit-ci/codeigniter3-translations>.
- `libraries` — Diretório que armazena as *libraries* (bibliotecas) criadas para o projeto, ou *libraries* estendidas do *core* do CI.
- `logs` — Diretório que armazena os arquivos de log, que são configurados em `application/config/config.php`.
- `models` — Diretório que armazena os arquivos onde ficarão as regras de negócio do projeto.
- `third_party` — Diretório que armazena código de terceiros, como classes que podem auxiliar em alguma

rotina do projeto e que foram desenvolvidas por outros programadores e/ou não possuem os padrões do CI.

- **views** — Diretório que armazena arquivos que serão carregados no browser. Você pode inserir outros diretórios dentro dele para organizar os arquivos da melhor maneira possível.

Nem sempre você fará uso de todos os diretórios dentro de `application`, os mais utilizados são: `controllers`, `libraries`, `logs`, `models` e `views`. Mas o uso fica a critério do desenvolvedor e da necessidade do projeto.

Diretório system

Esse diretório armazena o *core* do CI, e o conteúdo dos arquivos contidos nesse diretório não devem ser alterados. Isso porque, para manter o CodeIgniter atualizado, na maioria das versões basta substituir o conteúdo desse diretório pelo conteúdo do mesmo diretório na versão mais recente.

Ele possui apenas seis outros diretórios, muito bem divididos, e com os arquivos nomeados de forma bem intuitiva. Assim, caso queira estudar mais a fundo o funcionamento do CI, você pode fazê-lo analisando o código-fonte dos arquivos desse diretório.

DIFERENÇA DA VERSÃO 2.x PARA A 3.x

A atualização da versão 2.x para a 3.x vai muito além da substituição dos arquivos no diretório `system`. Entre uma versão e outra, foi implementado outro padrão para nomenclatura das classes. Então, além de copiar os arquivos, é necessário alterar o nome das classes nos arquivos do diretório `application`.

A documentação do CI tem um material excelente sobre o processo de migração entre as mais diferentes versões dele. Veja essa documentação em:

http://www.codeigniter.com/user_guide/installation/upgrading.html No capítulo 20. *Migrando um projeto da versão 2.x para a 3.x*, veremos como migrar um projeto da versão 2.x para a versão 3.x.

Arquivos `index.php` e `composer.json`

O arquivo `index.php` é o arquivo base de um projeto feito com CI. Ele carrega o arquivo de *core* necessário para a carga das *libraries*, *helpers*, *classes*, entre outros arquivos do framework e execução do código escrito por você.

Nesse arquivo, você pode alterar a localização dos diretórios `system` e `application`, configurar as exibições de erro para os ambientes do projeto (desenvolvimento, teste e produção, por exemplo), customizar valores de configurações, entre outras possibilidades. Quase não se faz alteração nesse arquivo, mas durante os capítulos serão feitas algumas, para que possam atender aos requisitos dos exemplos.

O arquivo `composer.json` se tornou parte integrante do CI a partir da versão 3, quando foi adicionado o suporte nativo ao Composer (<https://getcomposer.org/>). É nesse arquivo que são adicionadas as configurações de dependências do projeto, para que as bibliotecas sejam instaladas automaticamente a partir da execução de uma linha de comando. Veremos exemplos de uso do Composer no capítulo 18. *Trabalhando com Composer*.

1.4 ALTERANDO A LOCALIZAÇÃO DO DIRETÓRIO SYSTEM

No decorrer deste livro, vamos trabalhar com alguns projetos de forma independente, mas todos usarão a mesma versão do CodeIgniter. Como vamos manter todos no mesmo servidor e sob o mesmo domínio (nesse caso, o `localhost`), podemos utilizar somente um diretório `system` para todos os projetos.

Para fazer isso, é necessário realizar duas operações. A primeira é mover o diretório `system`, que está localizado dentro do diretório `instalacao-ci`, para a raiz. Feito isso, você terá na raiz do diretório com os exemplos, o diretório `instalacao-ci` e `system`.

Agora que você moveu o diretório, é hora de alterar o *path* (caminho) dele no arquivo `index.php`. A alteração será feita conforme o exemplo:

- **Antes:**

```
$system_path = 'system';
```

- **Depois:**

```
$system_path = '../system';
```

Agora, todo projeto que for colocado dentro de `exemplos-`

livro-ci não necessitará mais de ter o diretório `system` , basta alterar a sua localização no arquivo `index.php` .

1.5 ALTERANDO O IDIOMA DAS MENSAGENS PADRÕES

Por padrão, o CI vem configurado para o idioma *inglês*, mas alterar para o português é simples e não exige conhecimentos avançados.

Primeiramente, você vai fazer o download do pacote de tradução acessando o link <https://github.com/bcit-ci/CodeIgniter/wiki/Portugu%C3%AAs-do-Brasil>. Em seguida, descompactar e copiar o diretório do idioma que deseja, no caso do livro, o *portuguese*, que está no diretório `language` , para o diretório `application/language` do projeto `instalacao-ci` .

REPOSITÓRIO DE TRADUÇÕES

Caso você encontre algo traduzido errado, ou conheça um termo mais apropriado, faça um *fork* do repositório, atualize e dê um *pull request* para o repositório principal.

Após ter copiado o diretório com a tradução, vá até o diretório `application/config` e abra o arquivo `config.php` . Nele você vai substituir *english* por *portuguese*, que é o nome do diretório com os arquivos de tradução.

- **Antes:**

```
$config['language'] = 'english';
```

- **Depois:**

```
$config['language'] = 'portuguese';
```

1.6 CONCLUSÃO

Neste capítulo, você aprendeu os seguintes pontos:

- O que é o CodeIgniter (CI);
- Como fazer a instalação e montagem dos ambientes;
- A estrutura de arquivos e diretórios;
- Como alterar a localização do diretório `system` ;
- Como alterar o idioma padrão para o português.

Agora você está pronto para começar a praticar com as funcionalidades das bibliotecas que o CI possui. Nos próximos três capítulos, você aprenderá sobre a anatomia de `models` , `controllers` e `views` no CI e, na sequência, criará o primeiro projeto de exemplo do livro: um site institucional.

Ansioso para começar? Não perca tempo, comece logo a leitura e prática do próximo capítulo.

Bons estudos!

Código-fonte

Sempre que formos criar um novo exemplo, usaremos as configurações aplicadas neste capítulo, e os arquivos base podem ser baixados diretamente no GitHub, em:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-01-base-para-projetos>

Lembre-se de que, além desses arquivos, você vai precisar do diretório `system` , que contém o *core* do CodeIgniter, cujo download você já fez no início do capítulo.

ANATOMIA DE UM MODEL

"Você nunca sabe que resultados virão da sua ação. Mas se você não fizer nada, não existirão resultados." — Mahatma Gandhi

Um *model* é uma classe para trabalhar com as informações do banco de dados. Nela você executa todas as ações necessárias de pesquisa (`SELECT`), adição (`INSERT`), atualização (`UPDATE`) e exclusão (`DELETE`) de informações.

As operações de pesquisa, adição, atualização e exclusão de registros também são conhecidas como CRUD (acrônimo para *Create, Read, Update e Delete**).

Você pode ter um *model* único que lida com essas operações em qualquer tabela, recebendo parâmetros que definem o que fazer e onde fazer, ou então ter um *model* para cada controlador que precise interagir com o banco de dados. Não há uma obrigatoriedade quanto a isso dentro do CodeIgniter, você pode trabalhar com o que for mais confortável e produtivo para você.

Veja a seguir um código de exemplo de um *model*:

```
<?php
if(!defined('BASEPATH')) exit('No direct script access allowed');

class Example_model extends CI_Model{
```

```

function __construct(){
    parent::__construct();
}

function Save($data){
    $this->db->insert('table',$data);

    if($this->db->insert_id()){
        return true;
    }else{
        return false;
    }
}
}

```

Toda classe no CI precisa ser estendida da classe pai, `CI_Model`, e ter o método `__construct()` instanciado para poder herdar as funcionalidades da classe pai. Também precisa ter a primeira letra do nome em maiúsculo, e o nome do arquivo ser igual ao nome da classe. Por exemplo, `application/models/Example_model.php` é o nome que deve ser dado para o arquivo da classe mostrada anteriormente.

Não é obrigatório, mas por uma questão de padrão do CI, é utilizado o sufixo `_model` logo após o nome do *model*.

2.1 COMO CARREGAR UM MODEL

Todos os *models* dentro do CI devem ser armazenados no diretório `application/models`, podendo ser separados em subdiretórios. Para chamar um *model*, você utiliza `$this->load->model('nome_do_model')`. Se o *model* desejado estiver dentro de um subdiretório, então basta informar o nome do diretório antes do nome do *model*: `$this->load->model('sub-diretorio/nome_do_model')`.

Se você quiser modificar o nome do *model* na hora de chamá-lo, para tornar a nomenclatura mais compreensível e/ou adequada ao seu projeto, basta informar o nome que deseja como segundo

parâmetro para o método: `$this->load->model('nome_do_model', 'novo_nome_do_model') .`

2.2 CARREGANDO UM MODEL NO AUTOLOAD

Pode ser que, em algum momento dentro da sua aplicação, você tenha um *model* que seja usado em vários controladores. E ficar carregando esse *model* em cada um deles pode fazer com que uma hora você esqueça de carregar. Para evitar esse problema, você pode adicionar o *model* no autoload do CodeIgniter, diretamente no arquivo `application/config/autoload.php` .

```
$autoload['model'] = array('nome_do_model' => 'novo_nome_do_model');
```

2.3 CONCLUSÃO

Neste capítulo, você aprendeu como deve ser montado um *model* e a sua anatomia. Esse conhecimento será importante para quando iniciarmos o desenvolvimento dos projetos práticos e de exemplos. Nos próximos dois capítulos, você verá sobre a anatomia de Controller e View no CI, complementando assim o estudo da arquitetura MVC dentro do CodeIgniter.

Links úteis

- **Documentação oficial do CodeIgniter sobre Models:**
https://codeigniter.com/user_guide/general/models.html

ANATOMIA DE UM CONTROLLER

"Obstáculos são aqueles perigos que você vê quando tira os olhos de seu objetivo." — Henry Ford

O *controller* é o que dá vida a uma aplicação. Ele determina o que será executado quando uma requisição HTTP é feita. Ele nada mais é do que uma classe com um ou vários métodos que podem ser acessados a partir da URL, ou que estão associados a URLs através das rotas.

Veja a seguir um exemplo de *controller*:

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Example extends CI_Controller {

    function __construct()
    {
        parent::__construct();
        $this->load->model('Example_model');
        $this->load->library('form_validation');
    }

    function Index(){
        $this->load->view('home');
    }

    function Login(){
        $this->load->view('login');
    }
}
```

Vou exemplificar para facilitar a compreensão. Veja a seguinte URL: `www.doma.in/login` .

Essa URL aciona o método `Login()` do controller `Example` , e é associada a ele pela rota `$route['login'] = "Example/Login";` .

Veremos mais detalhes sobre rotas nos capítulos 5. *Criando um site institucional — Parte I*, 6. *Criando um site institucional — Parte II* e 12. *Criando um encurtador de URLs — Parte I*, quando criaremos os projetos do site institucional e encurtador de URLs, respectivamente.

Também poderíamos acessar o método `Login()` sem a necessidade de uma rota associada, simplesmente definindo na própria URL, `www.doma.in/example/login` . Dessa forma, o primeiro segmento da URL (`example`) identifica o *controller*, e o segundo (`login`) identifica o método.

Todo *controller* no CI deve ter o mesmo nome, tanto no arquivo quanto no nome da classe, e começar com a primeira letra maiúscula.

3.1 ENVIANDO PARÂMETROS POR MEIO DA URL

É possível passar parâmetros para métodos de um *controller* que não estejam associados a rotas. Para isso, usamos o método `$this->uri->segment(posicao_do_segmento)` para recuperar o

parâmetro a partir da posição dele na URL. Por exemplo, `www.doma.in/user/edit/1`, em que `1` é o ID do usuário que está editando os dados.

Para recuperar esse ID dentro de um método no controlador, é usado o método `$this->uri->segment(3)`. O parâmetro `3` passado para o método determina que deve ser recuperado o valor do terceiro segmento da URL.

Os segmentos são determinados pelas '/' (barras) após a URL base.

3.2 NOMES RESERVADOS

O CodeIgniter possui uma lista de nomes reservados que não podem ser utilizados em *controllers* e métodos. Veja essa lista a seguir:

Controllers

- `CI_Controller`
- `Default`
- `index`

Funções

- `is_php()`
- `is_really_writable()`
- `load_class()`
- `is_loaded()`
- `get_config()`
- `config_item()`

- `show_error()`
- `show_404()`
- `log_message()`
- `set_status_header()`
- `get_mimes()`
- `html_escape()`
- `remove_invisible_characters()`
- `is_https()`
- `function_usable()`
- `get_instance()`
- `_error_handler()`
- `_exception_handler()`
- `_stringify_attributes()`

Variáveis

- `$config`
- `$db`
- `$lang`

Constantes

- `ENVIRONMENT`
- `FCPATH`
- `SELF`
- `BASEPATH`
- `APPPATH`
- `VIEWPATH`
- `CI_VERSION`
- `MB_ENABLED`
- `ICONV_ENABLED`
- `UTF8_ENABLED`
- `FILE_READ_MODE`
- `FILE_WRITE_MODE`

- DIR_READ_MODE
- DIR_WRITE_MODE
- FOPEN_READ
- FOPEN_READ_WRITE
- FOPEN_WRITE_CREATE_DESTRUCTIVE
- FOPEN_READ_WRITE_CREATE_DESTRUCTIVE
- FOPEN_WRITE_CREATE
- FOPEN_READ_WRITE_CREATE
- FOPEN_WRITE_CREATE_STRICT
- FOPEN_READ_WRITE_CREATE_STRICT
- SHOW_DEBUG_BACKTRACE
- EXIT_SUCCESS
- EXIT_ERROR
- EXIT_CONFIG
- EXIT_UNKNOWN_FILE
- EXIT_UNKNOWN_CLASS
- EXIT_UNKNOWN_METHOD
- EXIT_USER_INPUT
- EXIT_DATABASE
- EXIT__AUTO_MIN
- EXIT__AUTO_MAX

3.3 CONCLUSÃO

Neste capítulo, você aprendeu como deve ser montado um *controller* e a sua anatomia. Esse conhecimento será importante para quando iniciarmos o desenvolvimento dos projetos práticos e exemplos mais adiante. No próximo capítulo, você verá sobre a anatomia de uma View no CI, completando assim o estudo da arquitetura MVC dentro do CI.

Links úteis

- **Documentação oficial do CI sobre Controllers:**
https://codeigniter.com/user_guide/general/controllers.html
- **Documentação oficial do CI sobre Rotas:**
https://codeigniter.com/user_guide/general/routing.html

ANATOMIA DE UMA VIEW

"A perfeição não é alcançada quando já não há mais nada para adicionar, mas quando já não há mais nada que se possa retirar." —
Antoine de Saint-Exupéry

No CodeIgniter, uma *view* nada mais é do que um arquivo HTML, que corresponde a uma tela da aplicação ou fragmento de conteúdo da tela, e que é chamada diretamente pelo *controller* através do método `$this->load->view()` .

Elas ficam armazenadas no diretório `application/views` e podem ser organizadas em subdiretórios. Quanto ao nome de uma *view*, não há uma obrigatoriedade ou um padrão determinado pelo CI, como tem para *controllers* e *models*, o que permite flexibilidade para organizá-las a seu modo.

Mesmo sendo a *view* um arquivo com código HTML, ela deve ser salva com a extensão `.php` . Mas na hora de informar qual *view* será carregada, a extensão não deve ser passada junto do nome da *view*, pois o próprio CI faz esse processo ao validar se a *view* existe e fazer a sua renderização.

4.1 CARREGANDO UMA VIEW

Carregar uma *view* é bastante simples, basta utilizar o método `$this->load->view()` , passando o nome do arquivo como parâmetro.

```
$this->load->view('home');
```

Se a *view* estiver em um subdiretório, então ele deve ser especificado no parâmetro passado:

```
$this->load->view('commons/header');
```

Carregando múltiplas views

É possível carregar múltiplas *views* no mesmo método do *controller*. Isso é feito quando a página é composta por vários fragmentos de código que foram separados em arquivos diferentes no diretório `application/views`.

```
$this->load->view('commons/header');  
$this->load->view('home');  
$this->load->view('commons/footer');
```

Nesse exemplo, usamos uma estrutura básica para montar uma página: cabeçalho, conteúdo e rodapé. O método `$this->load->view()` foi chamado três vezes, de forma a carregar, ordenadamente, as três *views*.

Uma outra possibilidade no carregamento de *views* é fazê-lo dentro de uma *view*, assim você não precisa chamar várias vezes o método no *controller*. Se sabemos que toda página de conteúdo precisa ter o cabeçalho e o rodapé, então colocamos as chamadas do método no início e fim da *view* com o conteúdo da página.

- **View commons/header.php :**

```
<html>  
<head>  
    <title>Título da página</title>  
</head>  
<body>
```

- **View home.php :**

```
<?php $this->load->view('header');  
<p><?=$content?></p>
```

```
<?php $this->load->view('footer');
```

- **View commons/footer.php :**

```
</body>  
</html>
```

4.2 ENVIANDO DADOS PARA A VIEW

Como se sabe, a *view* é a responsável por exibir os dados enviados pelo *controller*. Para que ela receba esses dados, basta adicionar um segundo parâmetro ao método `$this->load->view()`.

Veja o código dentro do controller:

```
$data['title'] = "Título da página";  
$data['content'] = "Conteúdo da página";  
$this->load->view('home', $data);
```

Esse segundo parâmetro é um `array` com as informações que serão exibidas na *view*. Ao recuperar os dados na *view*, eles deixam de ser um `array` para se tornarem variáveis simples. Então, em vez de chamar pelo índice, você chama como variável.

Veja o código da view:

```
<html>  
<head>  
    <title><?=$title?></title>  
</head>  
<body>  
    <p><?=$content?></p>  
</body>  
</html>
```

Se em algum momento você precisar enviar uma lista de dados para a *view*, você poderá tranquilamente executar um *looping* dentro dela para poder listar esses dados.

- **Código dentro do controller:**

```

$data['title'] = "Título da página";
$data['content'] = "Links Importantes";
$data['domains'] = array('www.casadocodigo.com.br', '
www.livrocodeigniter.com.br');
$this->load->view('home', $data);

```

- **Código da view:**

```

<html>
<head>
    <title><?=$title?></title>
</head>
<body>
    <h1><?=$content?></h1>
    <ul>
        <?php foreach($domains as $domain):?>
            <li><?=$domain?></li>
        <?php endforeach; ?>
    </ul>
</body>
</html>

```

4.3 RETORNANDO UMA VIEW COMO STRING

Existe um terceiro parâmetro que pode ser aplicado ao método `$this->load->view()`, que é do tipo `booleano` e determina se a *view* será retornada como `string`, ou se será renderizada no browser.

Esse parâmetro é muito útil, por exemplo, quando temos um sistema com vários templates diferentes para envio de e-mails, e esses templates devem ser carregados para o corpo da mensagem, e não renderizados na tela.

Como o terceiro parâmetro é opcional, ele tem o valor padrão `FALSE`, renderizando a *view* no browser sempre que o método `$this->load->view()` é chamado.

```

$data['destinatario'] = "Jonathan Lamim Antunes";
$data['assunto'] = "Lançamento do livro 'CodeIgniter Teoria na Prática'";
$this->load->view('templates/email', $data, TRUE);

```

Se você não precisar enviar dados para a *view*, basta passar `NULL` como valor do segundo parâmetro.

4.4 USANDO TEMPLATE PARSER NA VIEW

O *Template Parser* é uma biblioteca nativa do CI que permite usar pseudovariáveis no lugar de código PHP. Ao fazer o uso desse recurso, o método usado para carregar a *view* deixa de ser o `$this->load->view()` e passa a ser o `$this->parser->parse()`.

```
<html>
<head>
    <title>{ $title }</title>
</head>
<body>
    <h1>{ $content }</h1>
    <ul>
        {domains}
        <li>{domain}</li>
    {/domains}
</body>
</html>
```

O uso do *Template Parser* torna o código da *view* mais limpo, facilitando o trabalho do desenvolvedor front-end que possui pouco conhecimento de PHP e do CodeIgniter.

Veremos mais detalhes sobre o uso do *Template Parser* no capítulo 16. *Usando template parser*.

4.5 CONCLUSÃO

Neste capítulo, você aprendeu como deve ser montada uma *view* e a sua anatomia. Esse conhecimento será importante para quando iniciarmos o desenvolvimento dos projetos práticos e exemplos

mais adiante. No próximo capítulo, você colocará esse conhecimento em prática criando um site institucional usando o CI e algumas de suas bibliotecas nativas.

Links úteis

- **Documentação oficial do CI sobre Views:**
https://codeigniter.com/user_guide/general/views.html
- **Documentação oficial do CI sobre Template Parser Library:**
https://codeigniter.com/user_guide/libraries/parser.html

CRIANDO UM SITE INSTITUCIONAL — PARTE I

"É fazendo que se aprende a fazer aquilo que se deve aprender a fazer." — Aristóteles

Essa frase parece um pouco complicada de se entender, mas se resume em: **você só aprende fazendo**. E é assim que vai ser a partir de agora. Tudo o que for aprender neste e nos próximos capítulos será na prática, com cenários o mais próximo possível dos reais que você encontrará pelo mercado de trabalho.

Neste e no próximo capítulo, você criará um site institucional utilizando o CodeIgniter, e aprenderá a utilizar recursos como:

- Envio de e-mail
- Configuração de rotas
- Validação de formulários
- Cache
- Log
- Hooks
- Upload de arquivos
- Sessões

Mãos à obra, ou melhor, ao código!

5.1 CENÁRIO E ESTRUTURA INICIAL

Antes de começar a desenvolver, é preciso saber o que será desenvolvido. Não se cria um site sem antes levantar informações com o cliente, por mais simples que este seja. Veja a seguir a composição do site de forma resumida, mas contendo as informações essenciais para implementar usando o CI.

COMPOSIÇÃO DO PROJETO

O site será composto por 5 páginas:

- Home
- Sobre a empresa
- Serviços
- Trabalhe Conosco
- Fale Conosco

As páginas deverão ser compostas por menu no topo e texto. Para as páginas de "Trabalhe conosco" e "Fale conosco", o formulário deverá estar do lado esquerdo, e do lado direito informações de contato como telefones, e-mail, endereço e um mapa.

Com essas informações, já é possível definir o que será necessário para o desenvolvimento do site, e quais os recursos do CodeIgniter serão utilizados. Chegou a hora de montar a estrutura para mais um exemplo.

Para adiantar o processo, faça uma cópia do diretório `instalacao-ci`, que possui as configurações necessárias para esse novo exemplo. Renomeie o diretório copiado para `site-institucional` e teste o acesso para ver se abrirá corretamente a tela padrão do CI.

A URL será <http://localhost/exemplos-livro-ci/site-institucional>.

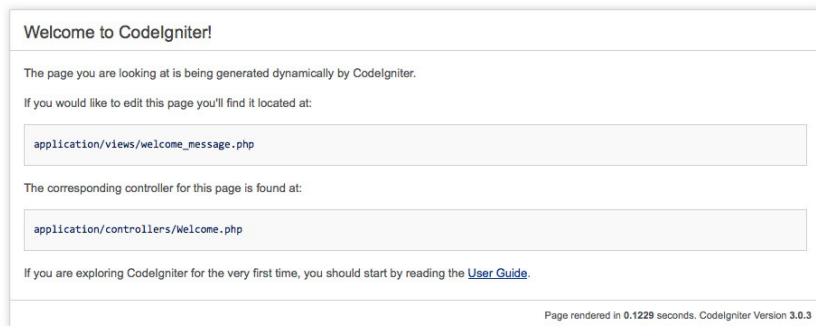


Figura 5.1: Resultado da instalação no diretório site institucional

Para a construção das páginas do site, vamos usar o Bootstrap. Caso ainda não o conheça, veja mais informações em seu site, <http://www.getbootstrap.com>.

É possível utilizar o Bootstrap de duas formas: com os arquivos dele junto dos arquivos da aplicação, ou através de CDNs. Para usá-lo na mesma estrutura de arquivos da aplicação, será necessário fazer o download dele e colocar os arquivos dentro dos respectivos diretórios em `assets`, separando por tipo (`css`, `js`, imagens e outros que vierem junto).

Para os exemplos deste livro, não usaremos os arquivos do Bootstrap junto dos arquivos da aplicação, faremos uso das CDNs, informando apenas o link dos arquivos no código das páginas, conforme listado a seguir:

- CSS:
<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css>
- JS:
[https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bo](https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js)

[otstrap.min.js](#)

- Tema (opcional):
<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css>

Para completar a estrutura de diretórios e arquivos do site, crie o diretório `assets` na raiz do projeto, e dentro dele os diretórios `css`, `img` e `js`. Esses diretórios servirão para armazenar os arquivos adicionais que serão criados ao longo do capítulo.

SOBRE A ESTRUTURA DAS PÁGINAS

A estrutura utilizada para as páginas do site será a de páginas de exemplo do próprio Bootstrap, com modificações para se adequarem ao estudo proposto. Todo o conteúdo usado nos exemplos é fictício e com caráter educacional, apenas.

Em caso de dúvidas sobre a implementação da estrutura das páginas, entre no fórum de discussão sobre o livro, em <http://forum.casadocodigo.com.br>.

5.2 MONTANDO A HOME

Para começar a estruturar uma página usando o CI, é necessário saber que você precisará no mínimo de um *controller* e uma *view*, além de configurar uma ou mais rotas que apontem para essa página. No caso da home, você vai precisar criar um *controller*, uma *view* e atualizar uma informação de rota já existente no arquivo `application/config/routes.php`, para que, ao acessar o site, a home seja exibida.

SOBRE O PADRÃO MVC (MODEL-VIEW-CONTROLLER)

- *Controller* — Responsável por interpretar os dados enviados pelo usuário e efetuar o tratamento necessário, repassando esses dados para a *view* ou para o *model*.
- *Model* — Responsável pelo gerenciamento de dados, podendo repassar esses dados para a *view*.
- *View* — Responsável por exibir os dados obtidos e tratados para o usuário.

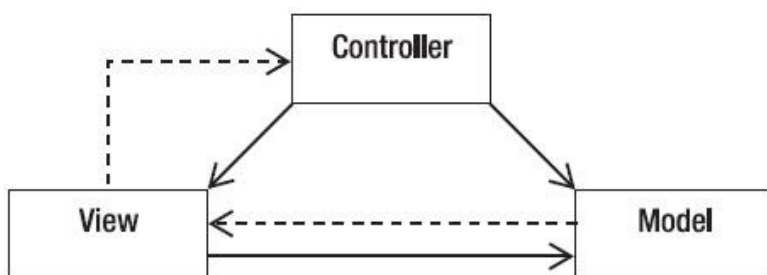


Figura 5.2: Fluxo do MVC

Criando o controller

Vá até o diretório `application/controllers` e crie um arquivo chamado `Institucional.php`. Esse arquivo será o *controller* responsável pela home e pelas páginas com informações sobre a empresa e serviços. Dentro desse arquivo, coloque o código a seguir:

```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');
```

```
class Institucional extends CI_Controller {  
    public function index()  
    {  
        $this->load->view('home');  
    }  
}  
  
?>
```

Foi criada uma classe chamada `Institucional` , estendida da classe de *controllers* padrão do CI, a `CI_Controller` . Dentro dessa classe, foi criado um método chamado `index()` , que vai ser o responsável por carregar a *view* da página principal do site.

O MÉTODO `$this->load->view()` DO CODEIGNITER

Esse método é o responsável por fazer o carregamento das *views* da aplicação, e ele conta com 3 parâmetros, sendo somente o primeiro obrigatório.

```
$this->load->view('view_file',          'data',  
'return_as_data');
```

- `view_file` : é a localização do arquivo da *view* dentro do diretório `application/views` . Pode estar dividido em subdiretórios, e não é necessário informar a extensão do arquivo.
- `data` : é a variável (`array` ou `object`) contendo os dados dinâmicos que serão exibidos na *view*.
- `return_as_data` : é um booleano (`TRUE` ou `FALSE`) que informa se a saída do método vai ser impressa na tela (`FALSE`), ou se vai ser um retorno com o conteúdo da *view* (`TRUE`). Se não informar esse parâmetro, o conteúdo da *view* será impresso na tela.

O próximo passo é criar um arquivo chamado `home.php` em `application/views` , com o seguinte conteúdo:

```
<!DOCTYPE html>  
<html lang="pt_BR">  
  <head>  
    <meta charset="utf-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-sca  
le=1">
```

```

    <meta name="description" content="Exercício de exemplo do capítulo 5 do livro CodeIgniter">
    <meta name="author" content="Jonathan Lamim Antunes">
    <title>Site Institucional</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
    <link href="http://getbootstrap.com/assets/css/ie10-viewport-bug-workaround.css" rel="stylesheet">
    <link href="<?=base_url('assets/css/home.css')?>" rel="stylesheet">
    <!--[if lt IE 9]><script src="http://getbootstrap.com/assets/js/ie8-responsive-file-warning.js"></script><![endif]-->
    <script src="http://getbootstrap.com/assets/js/ie-emulation-modes-warning.js"></script>
    <!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
    <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
</head>
<body>
    <div class="site-wrapper">
        <div class="site-wrapper-inner">
            <div class="cover-container">
                <div class="masthead clearfix">
                    <div class="inner">
                        <h1 class="masthead-brand">LCI</h1>
                        <nav>
                            <ul class="nav masthead-nav">
                                <li class="active"><a href="#">Home</a></li>
                                <li><a href="#">A Empresa</a></li>
                                <li><a href="#">Serviços</a></li>
                                <li><a href="#">Trabalhe Conosco</a></li>
                                <li><a href="#">Fale Conosco</a></li>
                            </ul>
                        </nav>
                    </div>
                </div>
                <div class="inner cover">
                    <h1 class="cover-heading">Ensinando através da prática
</h1>
                    <p class="lead">Até aqui você aprendeu como criar um <i>controller</i>, uma <i>view</i> e a usar a função <i>base_url</i> do helper <i>url</i> utilizando o livro "CodeIgniter: Produtividade na criação de aplicações web em PHP".</p>
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
    </div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11
.3/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/j
s/bootstrap.min.js"></script>
    <script src="http://getbootstrap.com/assets/assets/js/ie10-vie
wport-bug-workaround.js"></script>
</body>
</html>

```

A folha de estilo CSS, assim como as imagens utilizadas para esse exemplo, podem ser obtidas em <https://github.com/jlamim/livro-codeigniter/tree/master/CAP-5e6-css-e-imagens>. Deverão ser gravadas nos respectivos diretórios, dentro de *assets*, para que o resultado visual da codificação fique de acordo com as figuras apresentadas.

O código anterior consiste na estrutura básica de uma página usando HTML e Bootstrap. Contém o `head` que carrega os arquivos CSS e metatags, o `body` onde está o conteúdo da página, e o fechamento da tag `<html>`.

Repare que os arquivos JS foram carregados no final do arquivo, antes do fechamento da tag `<body>`. Isso faz com que o carregamento da página não fique lento caso um script demore para ser executado.

Foi utilizado um fragmento de código PHP junto com um método nativo do CI para inserir a URL absoluta para a folha de estilo da home.

```

<link href="<?=base_url('assets/css/home.css')?>" rel="stylesheet"
>

```

Nesse fragmento de código, foi chamado o método

`base_url()` , que recebeu como parâmetro o path para a folha de estilo.

O MÉTODO `BASE_URL()`

É um dos métodos do *helper* **URL** do CodeIgniter. Ele retorna o caminho passado como parâmetro concatenado com a URL base da aplicação: `http://localhost/exemplos-livro-ci/site-institucional/assets/css/home.css` .

No arquivo `application/views/home.php` , ele foi usado para inserir a URL completa para o arquivo `assets/css/home.css` , da seguinte forma: `base_url('assets/css/home.css')` .

Para utilizar as funções do *helper* **URL**, é preciso abrir o arquivo `application/config/autoload.php` , e inserir o nome do helper no array com os helpers a serem carregados. Dessa forma, ele será carregado automaticamente sempre que a aplicação for iniciada. Veja a seguir a linha que deve ser alterada:

```
$autoload['helper'] = array('url');
```

Criando a rota

Agora que o *controller* `Institucional` e a *view* `home` já foram criados, é hora de fazer com que, ao abrir o site, essa seja a página principal a ser carregada. Para fazer isso, abra o arquivo `routes.php` em `application/config` , e altere a linha que possui `$route['default_controller'] = 'welcome';` , conforme mostrado a seguir:

- **Antes:**

```
$route['default_controller'] = 'welcome';
```

- **Depois:**

```
$route['default_controller'] = 'Institucional';
```

Feito isso, ao acessar <http://localhost/exemplos-livro-ci/site-institucional> (ou o path corresponde no seu ambiente de desenvolvimento), você verá a home que acabou de ser criada, em vez de ver a página padrão da instalação do CI.



Figura 5.3: Resultado da construção da home

5.3 MONTANDO AS PÁGINAS SOBRE A EMPRESA E SERVIÇOS

As próximas duas páginas que serão montadas são de informações sobre a empresa e serviços prestados. Essas páginas vão precisar de dois métodos no *controller* Institucional, chamados respectivamente de Empresa e Servicos, duas *views* e configurações de rotas para acesso através de URLs específicas.

Criando os *controllers*

- **Controller Empresa**

```
public function Empresa()  
{  
    $this->load->view('empresa');  
}
```

- **Controller Servicos**

```
public function Servicos()  
{  
    $this->load->view('servicos');  
}
```

Mais uma vez, foram criados controllers com apenas uma linha, responsável por carregar a view com o conteúdo. Ainda neste capítulo, esses *controllers* serão atualizados para que possam enviar dados para as views.

5.4 CRIANDO E CONFIGURANDO AS ROTAS

Agora que os *controllers* já estão criados, você precisa criar as rotas corretas para acessar cada uma dessas páginas. As rotas vão fazer com que você possa determinar como será a URL que permitirá o acesso às páginas.

SOBRE AS ROTAS

São os caminhos das páginas de um site que podem ser configurados para executar um método de um *controller* específico, cujo nome é diferente do usado na URL.

No CodeIgniter, é possível acessar um método do *controller* pela URL usando a seguinte notação:
`http://url.com/class/method/variable` .

Exemplificando, dentro do site que está sendo criado, seria algo como:

```
http://localhost/exemplos-livro-ci/site-institucional/institucional/empresa
```

```
http://localhost/exemplos-livro-ci/site-institucional/institucional/servicos
```

Fazendo uso das rotas, você poderá utilizar a chamada que preferir na URL, tornando-a mais curta, amigável e condizendo com o conteúdo, o que é muito importante para a parte de SEO do site.

Veremos outros tipos de rotas ao longo do livro, mas se quiser se aprofundar mais nesse assunto, a documentação do CI é bastante completa.

Para alterar as rotas, acesse o arquivo `application/config/routes.php` :

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

$route['default_controller'] = 'Institucional';
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;
```

Repare que ele já traz algumas linhas preenchidas, sendo que a primeira rota já foi alterada no início do capítulo para acessar a página principal do site. Como você precisa criar as rotas para as páginas de serviço e informações sobre a empresa, basta adicionar duas novas linhas a esse arquivo.

```
$route['empresa'] = "Institucional/Empresa";  
$route['servicos'] = "Institucional/Servicos";
```

As rotas no CI são informadas em um array , no qual seu índice corresponde ao termo que será utilizado na URL. Então é importante que fique atento à escrita, não colocando caracteres especiais e espaços em branco, e evitando rotas com índices muito longos.

Feito isso, as rotas ainda não estarão funcionando, pois é necessário configurar o arquivo .htaccess .

Configurando o arquivo .htaccess

O arquivo .htaccess é responsável por diversas configurações do servidor Apache. Porém, neste momento, trabalharemos apenas com a configuração que permitirá o funcionamento das rotas com URLs amigáveis.

Na raiz do diretório do projeto, crie um arquivo chamado .htaccess , e nele coloque o código a seguir:

```
<IfModule mod_rewrite.c>  
RewriteEngine On  
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteCond %{REQUEST_FILENAME} !-d  
RewriteRule ^(.*)$ index.php/$1 [L]  
</IfModule>
```

Esse código só será executado caso o mod_rewrite esteja ativado (<IfModule mod_rewrite.c>) no Apache; se não estiver, as rotas não vão funcionar. Veja a seguir uma explicação sobre cada

linha desse arquivo:

- *Linha 2*: ativa a engine de reescrita das URLs;
- *Linhas 3 e 4*: se o arquivo com o nome especificado no navegador não existir, procede com a regra de reescrita da linha 5;
- *Linha 5*: regra de reescrita da URL, onde ele recupera o valor passado logo após o domínio.

.HTACCESS NÃO FUNCIONA EM SERVIDORES WINDOWS

É importante frisar que arquivos `.htaccess` não funcionam em servidores Windows que usam o IIS no lugar do Apache. Para isso, é necessário utilizar um arquivo chamado `web.config`. Veja a seguir como fica o arquivo `web.config` para o uso de rotas no CI:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <rewrite>
      <rules>
        <rule name="Imported Rule 1" stopProcessing="
true">
          <match url="^(.*)$" ignoreCase="false" />
          <conditions logicalGrouping="MatchAll">
            <add input="{REQUEST_FILENAME}" match
Type="IsFile" ignoreCase="false" negate="true" />
            <add input="{REQUEST_FILENAME}" match
Type="IsDirectory" ignoreCase="false" negate="true" />
          </conditions>
          <action type="Rewrite" url="index.php/{R:
1}" />
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
</configuration>
```

Veja no *Apêndice A* um tutorial sobre o processo de ativação do `mod_rewrite` em um servidor Linux.

Configurando as views

Com as rotas e *controllers* prontos, é hora de criar as *views* para exibir as informações sobre a empresa e os serviços prestados. Em `application/views`, crie dois arquivos: `empresa.php` e `servicos.php`. Como os dois arquivos terão partes em comum

(cabeçalho e rodapé), você vai criar também um diretório chamado `commons` dentro de `application/views` e, nesse diretório, criar outros dois arquivos: `header.php` e `footer.php`.

Para o arquivo `header.php`, você usará o seguinte conteúdo:

```
<!DOCTYPE html>
<html lang="pt_BR">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="Exercício de exemplo do capítulo 5 do livro Codeigniter Teorian na Prática">
    <meta name="author" content="Jonathan Lamim Antunes">

    <title>Site Institucional</title>

    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
    <link href="http://getbootstrap.com/assets/css/ie10-viewport-bug-workaround.css" rel="stylesheet">
    <link href="<?=base_url('assets/css/internas.css')?>" rel="stylesheet">

    <!--[if lt IE 9]><script src="http://getbootstrap.com/assets/js/ie8-responsive-file-warning.js"></script><![endif]-->
    <script src="http://getbootstrap.com/assets/js/ie-emulation-modes-warning.js"></script>

    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>
    <nav class="navbar navbar-default navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
```

```

        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
    </button>
    <a class="navbar-brand" href="#">LCI</a>
</div>
<div id="navbar" class="collapse navbar-collapse">
    <ul class="nav navbar-nav">
        <li><a href="#">Home</a></li>
        <li><a href="#">A Empresa</a></li>
        <li><a href="#">Serviços</a></li>
        <li><a href="#">Trabalhe Conosco</a></li>
        <li><a href="#">Fale Conosco</a></li>
    </ul>
</div>
</div>
</nav>

```

Esse vai ser o cabeçalho que aparecerá em todas as páginas do site. Ele contém o elemento principal para navegação, o menu.

E para o arquivo `footer.php` :

```

<footer class="footer">
    <div class="container">
        <p class="text-muted">&copy; Copyright 2016</p>
    </div>
</footer>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>
<script src="http://getbootstrap.com/assets/assets/js/ie10-viewport-bug-workaround.js"></script>

</body>
</html>

```

Assim como o cabeçalho, o rodapé aparece em todas as páginas do site, e esse arquivo é o responsável pela geração do seu conteúdo.

Feito isso, você já tem os arquivos de cabeçalho e rodapé que serão comuns em todas as páginas internas, o que reduzirá o volume de código HTML nas views e facilitará a manutenção do código.

DICA

Sempre que você tiver blocos de conteúdo que se repetem nas *views*, crie arquivos separados para eles, pois assim você facilita o processo de manutenção e otimiza o seu trabalho.

Para chamar um bloco de código dentro de uma *view*, você pode usar o método `$this->load->view()` diretamente na *view*, ou inserir o conteúdo dele em uma variável no *controller* e passar essa variável para a *view*. Veremos essa forma de uso ainda neste capítulo.

Para a página com informações sobre a empresa, o conteúdo será o seguinte:

```
<?php $this->load->view('commons/header'); ?>

<div class="container">
    <div class="page-header">
        <h1>A Empresa</h1>
    </div>
    <p class="lead">Você está criando a página com as informações da
    empresa</p>
    <p>
        Essa é uma página simples, contendo cabeçalho e rodapé, e o co
        ntéudo sendo um título e 2 parágrafos.
    </p>
</div>

<?php $this->load->view('commons/footer'); ?>
```

Veja que antes do conteúdo da *view* foi feita uma chamada para outra *view* (`$this->load->view('commons/header')`), que é o bloco de código que se repete (no caso, o cabeçalho) e, após o conteúdo, a mesma chamada, porém para o rodapé.

Uma outra alternativa, caso queira deixar o arquivo da *view* sem

essas chamadas, basta inseri-las no controller, que ficaria dessa forma:

```
...
public function Empresa()
{
    $this->load->view('commons/header');
    $this->load->view('empresa');
    $this->load->view('commons/footer');
}
...
```

Para a página de serviços, o conteúdo é o seguinte:

```
<?php $this->load->view('commons/header'); ?>

<div class="container">
    <div class="page-header">
        <h1>Serviços</h1>
    </div>
    <p class="lead">Esses são os serviços que a empresa oferece, e a
    página é composta por cabeçalho, rodapé e conteúdo tendo título e
    parágrafos.</p>
    <div class="row">
        <div class="col-lg-4">
            <h3>Serviço 1</h3>
            <p>Descrição do serviço</p>
            <p><a class="btn btn-primary" href="#" role="button">Detal
hes &raquo;</a></p>
        </div>
        <div class="col-lg-4">
            <h3>Serviço 2</h3>
            <p>Descrição do serviço</p>
            <p><a class="btn btn-primary" href="#" role="button">Detal
hes &raquo;</a></p>
        </div>
        <div class="col-lg-4">
            <h3>Serviço 3</h3>
            <p>Descrição do serviço</p>
            <p><a class="btn btn-primary" href="#" role="button">Detal
hes &raquo;</a></p>
        </div>
    </div>
    <div class="row">
        <div class="col-lg-4">
            <h3>Serviço 4</h3>
            <p>Descrição do serviço</p>
```

```

        <p><a class="btn btn-primary" href="#" role="button">Detal
hes &raquo;</a></p>
    </div>
    <div class="col-lg-4">
        <h3>Serviço 5</h3>
        <p>Descrição do serviço</p>
        <p><a class="btn btn-primary" href="#" role="button">Detal
hes &raquo;</a></p>
    </div>
    <div class="col-lg-4">
        <h3>Serviço 6</h3>
        <p>Descrição do serviço</p>
        <p><a class="btn btn-primary" href="#" role="button">Detal
hes &raquo;</a></p>
    </div>
</div>
</div>
<?php $this->load->view('commons/footer'); ?>

```

Essas duas novas páginas que acabaram de ser criadas possuem um código CSS bem específico para elas. O arquivo CSS usado para armazenar esse código é o `internas.css`, que deve ser armazenado em `assets/css`, e pode ser obtido na íntegra em <https://gist.github.com/jlamim/8668ebe94f4f3741a096>.

Otimizando o código do menu e inserindo a navegação

Se analisar o código escrito para as *views* até agora, verá que ainda existe um bloco de código que se repete, e que pode ser separado em um arquivo dentro de `commons`. Esse bloco contém os itens do menu, que são os mesmos tanto para *home* quanto para as páginas internas. A única diferença é no caso da *home*, no elemento `ul`, onde os itens que estão inseridos recebem uma classe diferente.

Para otimizar o código, crie um novo arquivo dentro do diretório `commons`, chamado `menu.php` com o conteúdo a seguir:

```

<ul class="nav masthead-nav">
    <li class="active"><a href="#">Home</a></li>
    <li><a href="#">A Empresa</a></li>
    <li><a href="#">Serviços</a></li>
    <li><a href="#">Trabalhe Conosco</a></li>

```

```

    <li><a href="#">Fale Conosco</a></li>
</ul>

```

Após criar o arquivo, altere o conteúdo da *view* `home.php` e `commons/header.php`, substituindo o menu pela chamada à *view* correspondente:

- `home.php`

```

...
<div class="inner">
    <h1 class="masthead-brand">LCI</h1>
    <nav>
        <?php $this->load->view('commons/menu'); ?>
    </nav>
</div>
...

```

- `commons/header.php`

```

...
<div id="navbar" class="collapse navbar-collapse">
    <?php $this->load->view('commons/menu'); ?>
</div>
...

```

Agora, volte ao arquivo `commons/menu.php`, pois vai ser preciso fazer uma checagem verificando se a página carregada é a `home` ou alguma outra, para que seja adicionada a classe correta ao `menu`.

```

<?php if($this->router->fetch_class() == 'Institucional' && $this->router->fetch_method() == 'index'){?>

<ul class="nav masthead-nav">

<?php } else {?>

<ul class="nav navbar-nav">

<?php } ?>

    <li><a href="#">Home</a></li>
    <li><a href="#">A Empresa</a></li>
    <li><a href="#">Serviços</a></li>

```

```

<li><a href="#">Trabalhe Conosco</a></li>
<li><a href="#">Fale Conosco</a></li>
</ul>

```

Na primeira linha, temos um `if` responsável por verificar se a classe chamada (`$this->router->class`) é a `Institucional`, e se o método é o `index`. Se for, então a página carregada é a *home* e o menu recebe as classes para ser formatado no padrão do layout da *home*. Se for outra classe ou outro método, então é uma página interna e recebe a classe de formatação para as páginas internas.

Feito isso, é hora de inserir os links no menu. Você usará novamente o método `base_url()`. Ainda existem duas páginas para serem criadas com suas respectivas rotas, mas isso não impede de montar os links do menu.

```

...
<li><a href="<?=base_url()?>">Home</a></li>
<li><a href="<?=base_url('empresa')?>">A Empresa</a></li>
<li><a href="<?=base_url('servicos')?>">Serviços</a></li>
<li><a href="<?=base_url('trabalhe-conosco')?>">Trabalhe Conosco</a></li>
<li><a href="<?=base_url('fale-conosco')?>">Fale Conosco</a></li>
...

```

Para ficar ainda melhor, precisamos deixar selecionado o item do menu relativo à página em que o usuário está. Então, vamos adicionar a checagem usando as propriedades vistas anteriormente:

```

$this->router->fetch_class()      e      $this->router->fetch_method() .

```

```

...
<li class="<?=( $this->router->fetch_class() == 'Institucional' &&
$this->router->fetch_method() == 'index' ) ? 'active' : null; ?>">
<a href="<?=base_url()?>">Home</a></li>
<li class="<?=( $this->router->fetch_class() == 'Institucional' &&
$this->router->fetch_method() == 'Empresa' ) ? 'active' : null; ?>"
><a href="<?=base_url('empresa')?>">A Empresa</a></li>
<li class="<?=( $this->router->fetch_class() == 'Institucional' &&
$this->router->fetch_method() == 'Servicos' ) ? 'active' : null; ?>"
"><a href="<?=base_url('servicos')?>">Serviços</a></li>
<li><a href="<?=base_url('trabalhe-conosco')?>">Trabalhe Conosco</a></li>

```

```
a></li>
<li><a href="<?=base_url('fale-conosco')?>">Fale Conosco</a></li>
...
```

5.5 PASSANDO DADOS DO CONTROLLER PARA A VIEW

A *view* em um projeto MVC serve para exibição das informações que são tratadas no *controller*, e no CodeIgniter não é diferente. Para exemplificar essa transferência de informações do *controller* para a *view*, implementaremos duas variáveis responsáveis por definir os cabeçalhos e descrição de cada página, aplicando assim algumas técnicas bem simples de SEO no site.

Abra o *controller* `Institucional.php` , e vá até o método `index()` , que é o responsável por carregar o conteúdo da home. Nele, adicione a variável `$data` , que é do tipo `array` , e no método `$this->load->view()` passe o segundo parâmetro, que será a variável criada. Veja o código a seguir:

```
public function index()
{
    $data['title'] = "LCI | Home";
    $data['description'] = "Exercício de exemplo do capítulo 5 do
livro CodeIgniter";

    $this->load->view('home',$data);
}
```

Esses dados (`title` e `description`) serão usados nas metatags, para que sejam exibidos quando o link do site for compartilhado em redes sociais e/ou na aba do browser em que a página estiver aberta.

Repita esse procedimento para os demais métodos (`Empresa` e `Servico`), colocando os seus respectivos títulos e descrições.

```
$data['title'] = "LCI | A Empresa";
$data['description'] = "Informações sobre a empresa";
```

```
$data['title'] = "LCI | Serviços";  
$data['description'] = "Informações sobre os serviços prestados";
```

ATENÇÃO

Anteriormente neste capítulo, foram apresentadas duas formas de carregar os blocos de conteúdo em uma *view*. Na página de informações da empresa, as *views* foram carregadas no *controller*, então você deverá adicionar a variável `$data` para a *view* header .

Já para a página de serviços, o bloco do cabeçalho foi inserido diretamente na *view* `servicos` , então não há a necessidade de passar a variável no método `$this->load->view()` dentro da *view*, pois automaticamente ele vai herdar os dados passados à *view* `servicos` .

Qualquer dado que for preciso passar do *controller* para a *view* deve ser armazenado em uma variável do tipo `Array` , e essa variável passada como segundo parâmetro do método `$this->load->view()` . Assim você poderá fazer a exibição dessas informações dentro da *view*.

5.6 COMPRIMINDO O HTML DE SAÍDA COM UM HOOK DO CI

Vou mostrar como utilizar um `hook` no CodeIgniter. Para esse exemplo, você vai minificar o HTML de saída das páginas do site que está criando. A minificação do HTML ajuda a deixar o carregamento da página mais rápido.

O primeiro passo é ativar os `hooks` . Então, vá até

application/config/config.php , e mude para TRUE o valor da variável \$config['enable_hooks'] .

- **Antes:**

```
$config['enable_hooks'] = FALSE;
```

- **Depois:**

```
$config['enable_hooks'] = TRUE;
```

O segundo passo é inserir a informação do hook de compressão em application/config/hooks.php . Por padrão, o arquivo vem vazio, somente com comentários, então basta adicionar o código a seguir:

```
$hook['display_override'][] = array(  
    'class' => '',  
    'function' => 'compress',  
    'filename' => 'compress.php',  
    'filepath' => 'hooks'  
);
```

Esse código é responsável por informar qual o hook deve ser executado (function), qual o arquivo (filename) e onde o arquivo com as instruções se encontra (filepath).

Para concluir, é hora de criar o arquivo compress.php dentro de applications/hooks , que é o diretório padrão.

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');  
function compress()  
{  
    ini_set("pcre.recursion_limit", "16777");  
    $CI =& get_instance();  
    $buffer = $CI->output->get_output();  
  
    $re = '%  
        (?>  
            [^\s ]\s*  
            | \s{2,}  
        )
```

```

(?=
  [^<]*+
  (?:
    <
      (?!/?(?:textarea|pre|script)\b)
      [^<]*+
    )*+
    (?:
      <
        (?:>textarea|pre|script)\b
      | \z
    )
  )
  %Six';

$new_buffer = preg_replace($re, " ", $buffer);

if ($new_buffer === null)
{
    $new_buffer = $buffer;
}

$CI->output->set_output($new_buffer);
$CI->output->_display();
}

```

Pronto, após salvar o arquivo `compress.php`, você só precisará abrir uma das páginas do exemplo que já foram criadas e dar um *refresh*. Logo em seguida, você visualizará o código-fonte, e verá que o código está todo escrito em apenas uma linha, e não em várias linhas como antes de ativar o `hook`.

O código utilizado para esse `hook` faz parte da Wiki oficial do CodeIgniter no GitHub, e pode ser visualizado com todos os seus comentários em <https://github.com/bcit-ci/CodeIgniter/wiki/Compress-HTML-output>.

5.7 CONCLUSÃO

Até aqui você criou as *views* das primeiras páginas do site, configurou as rotas e comprimiu o HTML de saída. No próximo capítulo, continuaremos desenvolvendo o site institucional, e você aprenderá a trabalhar com cache no CodeIgniter, criar as *views* com os formulários de contato e validá-lo, e fazer o envio de e-mails.

Dê uma relaxada, refresque a cabeça e volte com gás total no próximo capítulo.

Links úteis

- **Documentação oficial do CI sobre Controllers:**
https://codeigniter.com/user_guide/general/controllers.html
- **Documentação oficial do CI sobre o Helper URL:**
https://codeigniter.com/user_guide/helpers/url_helper.html
- **Documentação oficial do CI sobre Hooks:**
https://codeigniter.com/user_guide/general/hooks.html
- **Documentação oficial do CI sobre Sessions:**
https://codeigniter.com/user_guide/libraries/sessions.html
- **Documentação oficial do CI sobre Views:**
https://codeigniter.com/user_guide/general/views.html
- **Detalhes sobre Compress HTML Output:**
<https://github.com/bcit-ci/CodeIgniter/wiki/Compress-HTML-output>.

CRIANDO UM SITE INSTITUCIONAL — PARTE II

"Você tem tudo o que precisa para construir algo muito maior do que você mesmo." — Seth Godin

Vamos dar continuidade ao desenvolvimento do site institucional. Neste capítulo, concluiremos o site e você aprenderá a trabalhar com formulário, cache e a enviar e-mails usando o CodeIgniter.

6.1 CONFIGURANDO O CACHE PARA AS PÁGINAS DO SITE

Fazer uso de cache das páginas acelera o seu carregamento e melhora a experiência do usuário no que diz respeito ao tempo de carregamento do seu site. No CI, é muito simples configurar o cache, uma vez que as páginas são salvas completamente renderizadas, fazendo com que o carregamento seja muito similar ao de páginas estáticas.

No caso deste exemplo, a diferença de performance vai ser pequena, pois o conteúdo é todo estático, mas o modo de implementar é o mesmo tanto para conteúdo dinâmico quanto para conteúdo estático.

Como o cache do CodeIgniter funciona?

O cache é individual, por página, e você poderá definir por quanto tempo cada página deverá permanecer em cache sem que tenha o seu conteúdo atualizado. Ao ser carregada pela primeira vez, o arquivo de cache é gerado e salvo em `application/cache`. Já nos demais acessos à página, esse arquivo salvo será recuperado e exibido para o usuário. Caso ele tenha expirado, o arquivo será removido e atualizado antes de ser exibido para o usuário.

Fazendo a configuração

Você pode configurar o cache de cada uma das páginas diretamente no *controller*, ou então fazer uso da configuração global para a classe. Para configurar de forma individual para cada método, você deverá inserir a chamada `$this->output->cache($n)` em cada *controller* cuja view de saída você queira armazenar em cache.

O `$n` é o tempo em minutos que a página ficará armazenada em cache, até ter o conteúdo atualizado.

NOTA

Verifique se o diretório `application/cache` possui permissão de escrita, pois se ele tiver apenas permissão de leitura, os arquivos de cache não serão criados e o cache não vai funcionar.

```
...
public function index()
{
    $this->output->cache(1440); //Corresponde a 24 horas até o cac
    he ser atualizado
}
```

```

        $data['title'] = "LCI | Home";
        $data['description'] = "Exercício de exemplo do capítulo 5 do
livro CodeIgniter";

        $this->load->view('home',$data);
    }
    ...

```

Uma outra forma é configurar o cache global para todos os métodos de uma classe que possuem views como saída. Para isso, você vai criar um método `__construct()` na classe em que quer configurar o cache de forma global. No exemplo deste capítulo, faça isso na classe `Institucional`.

O método `__construct()` vai ser executado sempre que algum método dessa classe for acionado.

```

...
public function __construct()
{
    parent::__construct();
    $this->output->cache(1440);
}
...

```

ATENÇÃO

Se um método não possui uma *view* como saída, por exemplo, um método `private` usado para tratar algum dado no *controller*, que retorna um valor específico, ele não será armazenado em cache.

Se você alterar opções de configuração que afetem a saída (HTML), será necessário excluir manualmente os arquivos armazenados em cache para que essas mudanças sejam aplicadas às views.

Excluindo caches

Você pode excluir os caches manualmente, removendo os arquivos do diretório `application/cache`, mas também pode utilizar o método `delete_cache()`, que é capaz de remover todos os arquivos de cache e ele não será mais atualizado após expirar.

- **Removendo todos os arquivos:**

```
$this->output->delete_cache();
```

- **Removendo o arquivo de cache da página de serviços:**

```
$this->output->delete_cache('servicos');
```

6.2 CRIANDO AS PÁGINAS DE FALE CONOSCO E TRABALHE CONOSCO

Essas são páginas importantes em um site, pois é o canal de comunicação entre o usuário e a empresa.

Desenvolver formulários com CI é muito simples, pois você pode trabalhar de duas formas:

1. Usando campos de formulário em HTML;
2. Usando o *helper Form*, nativo do CI.

Para a página do Fale Conosco, você aprenderá a montar os formulários utilizando HTML, que é o mais tradicional. Já na página do Trabalhe Conosco, você utilizará o *helper* padrão do CI.

Montando os controllers

Crie dentro de `application/controllers` um *controller* chamado `Contato`, com a classe de mesmo nome. Adicione a ele os métodos `FaleConosco` e `TrabalheConosco`, cada um deles chamando sua respectiva *view*, `fale-conosco` e `trabalhe-`

conosco .

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Contato extends CI_Controller {

    public function FaleConosco()
    {
        $data['title'] = "LCI | Fale Conosco";
        $data['description'] = "Exercício de exemplo do capítulo 5
do livro CodeIgniter";

        $this->load->view('fale-conosco', $data);
    }

    public function TrabalheConosco()
    {
        $data['title'] = "LCI | Trabalhe Conosco";
        $data['description'] = "Exercício de exemplo do capítulo 5
do livro CodeIgniter";

        $this->load->view('trabalhe-conosco', $data);
    }
}

?>
```

Em seguida, crie os arquivos das *views* em `application/views` , mas não adicione o conteúdo delas ainda. Não se esqueça de configurar as rotas para essas duas páginas, pois os seus links já foram inseridos no menu, e também atualizar a *view* do menu com a verificação para marcá-lo quando a página estiver ativa.

- **application/config/routes.php**

```
$route['fale-conosco'] = "Contato/FaleConosco";
$route['trabalhe-conosco'] = "Contato/TrabalheConosco";
```

- **application/views/commons/menu.php**

...

```

<li class="<?=( $\$this->router->fetch\_class()$  == 'Inst
itucional' &&  $\$this->router->fetch\_method()$  == 'index'
) ? 'active' : null; ?>"><a href="<?= $\$base\_url()$ >">Ho
me</a></li>
<li class="<?=( $\$this->router->fetch\_class()$  == 'Inst
itucional' &&  $\$this->router->fetch\_method()$  == 'Empres
a') ? 'active' : null; ?>"><a href="<?= $\$base\_url('empre
sa')$ >">A Empresa</a></li>
<li class="<?=( $\$this->router->fetch\_class()$  == 'Inst
itucional' &&  $\$this->router->fetch\_method()$  == 'Servic
os') ? 'active' : null; ?>"><a href="<?= $\$base\_url('serv
icos')$ >">Serviços</a></li>
<li class="<?=( $\$this->router->fetch\_class()$  == 'Cont
ato' &&  $\$this->router->fetch\_method()$  == 'TrabalheCono
sco') ? 'active' : null; ?>"><a href="<?= $\$base\_url('tra
balhe-conosco')$ >">Trabalhe Conosco</a></li>
<li class="<?=( $\$this->router->fetch\_class()$  == 'Cont
ato' &&  $\$this->router->fetch\_method()$  == 'FaleConosco'
) ? 'active' : null; ?>"><a href="<?= $\$base\_url('fale-co
nosco')$ >">Fale Conosco</a></li>
...

```

A base para a criação das páginas já está pronta, então vamos criar a página do Fale Conosco.

6.3 CRIANDO A PÁGINA DO FALE CONOSCO

No início do nosso projeto, o cliente havia informado como essa página deveria ser:

"...o formulário deverá estar do lado esquerdo, e do lado direito informações de contato como telefones, e-mail, endereço e um mapa".

O código a seguir deve ser colocado no arquivo `application/views/fale-conosco.php`, para que o formulário de contato seja exibido na tela.

```

<?php $this->load->view('commons/header'); ?>
<div class="container">
    <div class="page-header">
        <h1>Fale Conosco</h1>
    </div>
    <div class="row">
        <div class="col-md-8">
            <form class="form-horizontal" method="POST" action="">
                <div class="form-group">
                    <label class="col-md-2 control-label" for="nome">Nome</l
abel>
                    <div class="col-md-8">
                        <input id="nome" name="nome" placeholder="Nome" class=
"form-control input-md" required="" type="text">
                    </div>
                </div>
                <div class="form-group">
                    <label class="col-md-2 control-label" for="email">Email<
/label>
                    <div class="col-md-8">
                        <input id="email" name="email" placeholder="Email" cla
ss="form-control input-md" required="" type="text">
                        <span class="help-block">Ex.: email@example.com</span>
                    </div>
                </div>
                <div class="form-group">
                    <label class="col-md-2 control-label" for="assunto">Assu
nto</label>
                    <div class="col-md-8">
                        <input id="assunto" name="assunto" placeholder="Assunt
o" class="form-control input-md" required="" type="text">
                    </div>
                </div>
                <div class="form-group">
                    <label class="col-md-2 control-label" for="mensagem">Men
sagem</label>
                    <div class="col-md-8">
                        <textarea class="form-control" id="mensagem" name="men
sagem" rows="10">mensagem</textarea>
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-10">
                        <input type="submit" value="Enviar" class="btn btn-def
ault pull-right"/>
                    </div>
                </div>
            </form>

```

```

</div>
<div class="col-md-4">
  <h4>Telefones</h4>
  <p>+55 99 9999-9999 | +55 88 8888-8888</p>
  <hr/>
  <h4>E-mail</h4>
  <p>contato@empresa.com.br</p>
  <hr/>
  <h4>Endereço</h4>
  <p>R. Quinze de Novembro - Praia da Costa, Vila Velha - ES <
/p>
  <hr/>
  <div class="embed-responsive embed-responsive-4by3">
    <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m
12!1m3!1d3741.1532158870327!2d-40.286650485399!3d-20.3352881863780
26!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0xb8163812c6b
305%3A0xe71db7e3d9c94285!2sR.+Quinze+de+Novembro+-+Praia+da+Costa%
2C+Vila+Velha+-+ES!5e0!3m2!1spt-BR!2sbr!4v1449523768427" width="60
0" height="450" frameborder="0" style="border:0" allowfullscreen><
/iframe>
  </div>
</div>
</div>
</div>
<?php $this->load->view('commons/footer'); ?>

```

A estrutura do formulário de contato está pronta. Agora é hora de começar a fazer a parte de validação do formulário e, consequentemente, o envio dos dados para o e-mail do responsável.

Validando o formulário de contato

Para validar o formulário, será utilizada a library *Form Validation*, nativa do CI. Também usaremos o helper `Form`, que vai permitir montar os formulários usando métodos nativos do CI.

Como somente as páginas do *controller* Contato vão fazer uso da library e do helper, vamos criar um método `__construct` na classe fazendo com que sejam carregados para todos os métodos:

```

...
function __construct(){
    parent::__construct();
}

```

```

    $this->load->library('form_validation');
    $this->load->helper('form');
}
...

```

Para iniciar o processo de validação do formulário, será necessário fazer algumas alterações no método `FaleConosco`, que foi criado anteriormente. Vamos adicionar nele um condicional `if` para testar se a validação ocorreu ou não.

```

public function FaleConosco()
{
    $data['title'] = "LCI | Fale Conosco";
    $data['description'] = "Exercício de exemplo do capítulo 5 do
livro CodeIgniter";

    if($this->form_validation->run() == FALSE){

    }else{

    }

    $this->load->view('fale-conosco',$data);
}

```

O `if` testa a condição de execução da validação. Se ela for `FALSE`, quer dizer que não foi executada ou que possui erros de validação, então será necessário recuperar os erros. Mas, antes de recuperar os erros, é preciso configurar as regras de validação, conhecidas no CI por `set_rules`.

Veja no código a seguir as regras que foram adicionadas para o formulário:

```

...
$this->form_validation->set_rules('nome', 'Nome', 'trim|required|min_length[3]');
$this->form_validation->set_rules('email', 'Email', 'trim|required|valid_email');
$this->form_validation->set_rules('assunto', 'Assunto', 'trim|required|min_length[5]');
$this->form_validation->set_rules('mensagem', 'Mensagem', 'trim|required|min_length[30]');

```

```

if($this->form_validation->run() == FALSE){
    $data['formErrors'] = validation_errors();
}else{
    $data['formErrors'] = null;
}
...

```

As `set_rules` foram setadas de forma individual para cada campo do formulário, e sua sintaxe é a seguinte:

```

$this->form_validation->set_rules(nome_do_campo, descricao_do_campo, regras_de_validacao_aninhadas);

```

- `nome_do_campo` : é o nome especificado no atributo `name` do `input` ;
- `descricao_do_campo` : é o nome do campo que será exibido na mensagem de erro;
- `regras_de_validacao_aninhadas` : são as regras que devem ser aplicadas em cada campo.

No capítulo 7. *Validando formulários*, você verá mais detalhes e aprofundará os conhecimentos sobre a `library form_validation` .

Com o código que foi escrito até o momento, o processo de validação do formulário já está funcionando. Ele está, inclusive, passando para a *view* as informações em caso de erro, passadas pela variável `$data['formErrors']` , que recebe os dados retornados de `validation_errors()` — método responsável por recuperar as mensagens logo após as regras de validação serem executadas em `$this->form_validation->run()` .

NOTA

Quando o usuário acessar a página com o formulário de contato, mesmo com os campos vazios, não será exibido nenhum erro, pois o próprio CodeIgniter identifica se o formulário foi ou não submetido. Somente após o usuário clicar em enviar e executar o `POST` do formulário é que o CI executará o processo de validação.

Após o formulário ser enviado e em caso de sucesso, é sempre bom exibir uma mensagem para o usuário. E como essa mensagem é temporária, você fará uso da `library Session`, mais uma `library` nativa do CI.

No método `__construct`, altere a forma como o método `$this->load->library()` é escrito, para que ele carregue as duas `libraries` em uma única chamada.

```
$this->load->library(array('form_validation','session'));
```

Repare a mudança na forma como a chamada do método `$this->load->library()` foi escrita. No início, estava sendo utilizada somente uma `library`, mas agora está sendo adicionada uma segunda. Para não ter de repetir a chamada, é possível inserir as `libraries` a serem carregadas através de um `array`. Assim, em uma única chamada do método, estão sendo passadas as duas `libraries` que precisam ser carregadas.

Agora já é possível fazer uso das `sessions` para armazenar informações. No formulário de contato, vamos usar `sessions` do tipo `flashdata`, que, ao recarregar o método ou mudar de página, são excluídas automaticamente.

Altere o método `FaleConosco` conforme exemplo a seguir:

```
...
if($this->form_validation->run() == FALSE){
    $data['formErrors'] = validation_errors();
}else{
    $this->session->set_flashdata('success_msg', 'Contato recebido
    com sucesso!');
    $data['formErrors'] = null;
}
...
```

Caso o formulário seja enviado com sucesso, está sendo criada uma session pelo método `set_flashdata`, que recebe dois parâmetros. O primeiro é o identificador da session, que será usado para recuperar essa informação na view, e o segundo é o valor da session, nesse caso a mensagem a ser exibida para o usuário. Se ocorrerem erros de validação, também será necessário exibir as mensagens, então altere a *view* `application/views/fale-conosco.php` para exibir as mensagens, complementando-a com o código a seguir:

```
...
<div class="row">
    <div class="col-md-8">
        <?php if($formErrors){?>
            <div class="alert alert-danger">
                <?=$formErrors?>
            </div>
        <?php }else{
            if($this->session->flashdata('success_msg')) {?>
                <div class="alert alert-success">
                    <?=$this->session->flashdata('success_msg')?>
                </div>
            <?php } } ?>
            <form class="form-horizontal" method="POST" action="<?=base_url('fale-conosco')?>">
        ...
```

Veja que foram usados dois `if`. O primeiro foi para verificar a ocorrência de erros e exibir as mensagens. Caso não exista mensagens de erro (o que pode indicar que o formulário não foi

submetido ou que foi preenchido corretamente), é feita a verificação da session, por meio do método `flashdata` .

A DIFERENÇA ENTRE `SET_FLASHDATA()` E `FLASHDATA()`

`set_flashdata()` é usado para setar o valor da session temporária, enquanto `flashdata()` é utilizado para recuperar o valor dessa session.

Para completar o processo de validação do formulário, falta somente o preenchimento dos campos com os dados informados pelo usuário caso algum erro seja encontrado durante a validação. Essa recuperação de dados é possível através do método `set_value()` , que será utilizado na *view*.

O uso do método `set_value()` é bem simples. O parâmetro que ele recebe é o nome do campo, o mesmo usado no atributo `name` do `input` .

No capítulo 9. *Gerenciando sessões com a library Session*, você verá mais detalhes e aprofundará os conhecimentos sobre a `library session` e o uso de sessões.

Abra a *view* e adicione o atributo `value` aos inputs, e como valor do atributo, chame o método `set_value()` . Para o `textarea` usado para redigir a mensagem, você vai usar diretamente o método, pois ele não tem atributo `value` .

```
<div class="form-group">
  <label class="col-md-2 control-label" for="nome">Nome</label>
  <div class="col-md-8">
```

```

        <input id="nome" name="nome" placeholder="Nome" class="form-control input-md" required="" type="text" value="<?=set_value('nome')?>">
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="email">Email</label>
    <div class="col-md-8">
        <input id="email" name="email" placeholder="Email" class="form-control input-md" required="" type="text" value="<?=set_value('email')?>">
        <span class="help-block">Ex.: email@example.com</span>
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="assunto">Assunto</label>
    <div class="col-md-8">
        <input id="assunto" name="assunto" placeholder="Assunto" class="form-control input-md" required="" type="text" value="<?=set_value('assunto')?>">
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="mensagem">Mensagem</label>
    <div class="col-md-8">
        <textarea class="form-control" id="mensagem" name="mensagem" rows="10"><?=set_value('mensagem')?></textarea>
    </div>
</div>

```

Finalmente, o formulário de contato está pronto, com todo o fluxo de validação funcionando. Mas não adianta muito se esses dados não forem enviados para alguém. Então, você aprenderá agora a enviar e-mails utilizando o CodeIgniter, e a concluir a montagem da página de contato.

6.4 ENVIANDO OS DADOS DO FORMULÁRIO DE CONTATO POR E-MAIL

Para enviar e-mails usando o CodeIgniter, não é necessário usar bibliotecas de terceiros, como por exemplo, o *PHP Mailer*. Porém, se preferir, você pode utilizar.

Ele já possui uma biblioteca nativa que faz o envio de e-mails, seja através da função `mail()` (padrão do PHP), seja usando os protocolos SMTP ou Sendmail.

O uso e configuração da biblioteca são extremamente fáceis e, com poucas linhas de código, você fará um envio de e-mail. Como no projeto que está sendo montado nesse capítulo você precisará disparar e-mails em dois formulários, vamos evitar código repetido e criar um método chamado `SendEmailToAdmin()` no *controller* Contato . Assim, quando precisar enviar o e-mail, você chamará esse método, sem a necessidade de reescrever toda a rotina de configuração, validação e envio.

```
...
private function SendEmailToAdmin($from, $fromName, $to, $toName,
$subject, $message, $reply = null, $replyName = null){
    $this->load->library('email');

    $config['charset'] = 'utf-8';
    $config['wordwrap'] = TRUE;
    $config['mailtype'] = 'html';
    $config['protocol'] = 'smtp';
    $config['smtp_host'] = 'smtp.seudominio.com.br';
    $config['smtp_user'] = 'user@seudominio.com.br';
    $config['smtp_pass'] = 'suasenha';
    $config['newline'] = '\r\n';

    $this->email->initialize($config);

    $this->email->from($from, $fromName);
    $this->email->to($to, $toName);

    if($reply)
        $this->email->reply_to($reply, $replyName);

    $this->email->subject($subject);
    $this->email->message($message);
```

```

        if($this->email->send())
            return true;
        else
            return false;
    }
    ...

```

A primeira operação a ser executada no método é o carregamento da library `Email` . Em seguida, foi criado o array `$config` , com as configurações para inicialização da biblioteca. Cada um desses parâmetros de configuração e todos os outros usados pelo CI podem ser vistos com seus detalhes no capítulo 8. *Enviando e-mails com a library Email*, onde há uma tabela com todas as variáveis de configuração da library.

Após definir as configurações, foi chamado o método `$this->email->initialize($config)` , passando como parâmetro para ele o array com as configurações. Depois, foram informados remetente (`from`) e destinatário (`to`) com seus e-mails e nome, respectivamente. O método `to()` permite enviar o e-mail para vários destinatários em uma única chamada. Para isso, basta separar os e-mails com uma vírgula (`,`).

```

$this->email->to("email@domain.com, email2@domain.com, email3@domain.com");

```

Em seguida, foi adicionada uma condição para verificar a variável `$reply` . Se ela foi passada, chama-se o método `reply_to()` , que adiciona o e-mail que deverá receber uma possível resposta às configurações de envio, caso ela seja enviada pelo destinatário.

Os próximos dois métodos executados são responsáveis por adicionar assunto (`subject`) e mensagem (`message`) ao e-mail, tornando-o completo para o envio, através do método `send()` .

Para fechar e enviar o e-mail, é utilizado um `if` combinado com o método `$this->email->send()` , que verifica se ele foi ou

não enviado. Em caso de envio, retorna `TRUE` ; caso contrário, retorna `FALSE` .

DICA

Se você estiver utilizando *"wordwrap"* e no corpo do email tiver um link longo que não pode ter quebra, você pode usar `{unwrap}{/unwrap}` para que a quebra seja ignorada para o bloco de texto entre as tags. Veja o exemplo a seguir:

```
{unwrap}http://example.com/a_long_link_that_should_not_be_wrapped.html{/unwrap}
```

Completando o processo de envio do e-mail de contato, agora é necessário chamar o método `SendEmailToAdmin()` quando as regras de validação do formulário forem atendidas. Faça a chamada desse método em `FaleConosco()` , passando os devidos parâmetros. Também será necessário completar e ajustar a lógica de retorno da mensagem de envio com sucesso, pois só podemos exibi-la em caso de sucesso no envio; caso contrário, devemos exibir uma mensagem de erro.

```
public function FaleConosco()
{
    $data['title'] = "LCI | Fale Conosco";
    $data['description'] = "Exercício de exemplo do capítulo 5 do livro CodeIgniter";

    $this->form_validation->set_rules('nome', 'Nome', 'trim|required|min_length[3]');
    $this->form_validation->set_rules('email', 'Email', 'trim|required|valid_email');
    $this->form_validation->set_rules('assunto', 'Assunto', 'trim|required|min_length[5]');
    $this->form_validation->set_rules('mensagem', 'Mensagem', 'trim|required|min_length[30]');

    if($this->form_validation->run() == FALSE){
```

```

        $data['formErrors'] = validation_errors();
    }else{
        $formData = $this->input->post();
        $emailStatus = $this->SendEmailToAdmin($formData['email'],
        $formData['nome'], "to@domain.com", "To Name", $formData['assunto'],
        $formData['mensagem'], $formData['email'], $formData['nome']);

        if($emailStatus){
            $this->session->set_flashdata('success_msg', 'Contato
recebido com sucesso!');
        }else{
            $data['formErrors'] = "Desculpe! Não foi possível envi
ar o seu contato. tente novamente mais tarde.";
        }
    }

    $this->load->view('fale-conosco', $data);
}

```

Logo no início do `else` foi criada a variável `$formData`, que recebe um array com os dados vindos do formulário. Esses dados foram recuperados utilizando o método `$this->input->post()`, que retorna todos os dados sempre que é feito um envio de formulário.

`$THIS->INPUT->POST()`

É um método muito importante quando se trabalha com formulários no CI. Além de recuperar todos os dados do formulário, é possível recuperar campos específicos, passando o nome do campo como parâmetro.

Por exemplo, se você quisesse recuperar apenas o campo e-mail do formulário de contato, bastaria chamar `$this->input->post('email')`.

Atenção: esse método não recupera informações de campos do tipo `file`.

O método `SendMailToAdmin()` passa o seu retorno (`TRUE` ou `FALSE`) para a variável `$emailStatus` , que é verificada para determinar qual mensagem deve ser exibida para o usuário.

No capítulo 8. *Enviando e-mails com a library Email*, você verá mais detalhes e aprofundará os conhecimentos sobre essa library.

Com isso, você concluiu a criação da página de contato e já pode montar a última página do site, que é a Trabalhe Conosco.

6.5 CRIANDO A PÁGINA DO TRABALHE CONSOCO

Essa página é muito semelhante à Fale Conosco, tanto no visual quanto no funcional. O *controller* e a rota para ela já foram criados, agora é hora de criar a *view*. Então abra `application/views/trabalhe-conosco.php` , e copie o código da *view* `fale-conosco.php` .

A base da página está pronta, mas como citei anteriormente, existem duas formas de trabalhar com formulários nas *views* usando o CI. Nesse formulário, não será utilizado HTML, e sim os métodos do helper *Form*. Vamos então substituir as tags HTML relativas ao formulário, como `form` , `input` , `textarea` e `label` , para os métodos do *helper Form*.

As tags `<form>` e `</form>` serão substituídas pelos métodos `form_open_multipart()` e `form_close()` , respectivamente, já que vamos fazer envio de arquivos e precisamos que o `enctype` do formulário seja `multipart/form-data` .

```

<!-- <form class="form-horizontal" method="POST" action="<?=base_u
rl('trabalhe-conosco')?>"> -->
<?= form_open_multipart(base_url('trabalhe-conosco'), array("class
" => "form-horizontal", "method"=>"POST")); ?>

...

<!-- </form> -->
<?= form_close(); ?>

```

Veja que, para o `base_url()` , foi passado como parâmetro 'trabalhe-conosco' em vez de 'fale-conosco'. Isso foi feito pois estamos trabalhando no formulário da tela 'Trabalhe Conosco', e vamos utilizar uma rota diferente para acesso e processamento dos dados a serem enviados.

O código comentado no exemplo deve ser removido do arquivo que você está editando.

O próximo passo é substituir as tags `input` pelo método `form_input()` .

```

<!-- <input id="nome" name="nome" placeholder="Nome" class="form-c
ontrol input-md" required="" type="text" value="<?=set_value('nome
')?>"> -->
<?= form_input(array("name"=>"nome", "id"=>"nome"), set_value('nome'
), array("class"=>"form-control input-md", "required"=>"", "type"=>"t
ext", "placeholder"=>"Nome")); ?>

```

Repita esse procedimento para os campos `email` , `assunto` e `mensagem` . Depois, substitua a tag `<textarea>` pelo método `form_textarea()` .

```

<!-- <textarea class="form-control" id="mensagem" name="mensagem"
rows="10"></textarea> -->
<?= form_textarea(array("name"=>"mensagem", "id"=>"mensagem"), set_v
alue('mensagem'), array("class"=>"form-control input-md", "required"
=>"", "type"=>"text", "placeholder"=>"Mensagem")); ?>

```

Concluídas as alterações, volte até o campo Assunto e mude para Telefone de Contato , alterando o nome do campo para telefone , e o label e o placeholder para Telefone de Contato .

```
<div class="form-group">
  <label class="col-md-2 control-label" for="telefone">Telefone
de Contato</label>
  <div class="col-md-8">
    <?= form_input(array("name"=>"telefone","id"=>"telefone"),
set_value('telefone'),array("class"=>"form-control input-md","requ
ired"=>"", "type"=>"text", "placeholder"=>"Telefone de Contato")); ?
  >
  </div>
</div>
```

Após o campo Mensagem , você inserirá o campo para upload do currículo. Será um campo do tipo file , e o método utilizado para esse campo será form_upload() .

```
...

<div class="form-group">
  <label class="col-md-2 control-label" for="curriculo">Currícul
o</label>
  <div class="col-md-8">
    <?= form_upload(array("name"=>"curriculo","id"=>"curriculo
"),set_value('curriculo'),array("class"=>"input-file", "required"=>
"")); ?>
  </div>
</div>
```

Feitas todas as alterações, o código HTML do formulário deve estar dessa forma:

```
...

<?= form_open_multipart(base_url('trabalhe-conosco'), array("class
" => "form-horizontal", "method"=>"POST")); ?>

  <div class="form-group">
    <label class="col-md-2 control-label" for="nome">Nome</lab
el>
    <div class="col-md-8">
```

```

        <?= form_input(array("name"=>"nome", "id"=>"nome"), set_
value('nome'), array("class"=>"form-control input-md", "required"=>"
", "type"=>"text", "placeholder"=>"Nome")); ?>
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="email">Email</l
abel>
    <div class="col-md-8">
        <?= form_input(array("name"=>"email", "id"=>"email"), se
t_value('email'), array("class"=>"form-control input-md", "required"
=>"", "type"=>"text", "placeholder"=>"Email")); ?>
        <span class="help-block">Ex.: email@example.com</span>
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="telefone">Telef
one de Contato</label>
    <div class="col-md-8">
        <?= form_input(array("name"=>"telefone", "id"=>"telefon
e"), set_value('telefone'), array("class"=>"form-control input-md", "
required"=>"", "type"=>"text", "placeholder"=>"Telefone de Contato")
); ?>
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="mensagem">Mensa
gem</label>
    <div class="col-md-8">
        <?= form_textarea(array("name"=>"mensagem", "id"=>"mens
agem"), set_value('mensagem'), array("class"=>"form-control input-md
", "required"=>"", "type"=>"text", "placeholder"=>"Mensagem")); ?>
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label" for="curriculo">Curr
iculo</label>
    <div class="col-md-8">
        <?= form_upload(array("name"=>"curriculo", "id"=>"curri
culo"), set_value('curriculo'), array("class"=>"input-file", "require
d"=>"")); ?>
    </div>
</div>

```

```

<div class="form-group">
    <div class="col-md-10">
        <?= form_submit(array("name"=>"Enviar","id"=>"enviar")
, "Enviar", array("class"=>"btn btn-default pull-right")); ?>
    </div>
</div>

<?=form_close();?>

...

```

Veja que os métodos usados para montagem dos campos são compostos por parâmetros que vão completar as informações do campo, como por exemplo, `placeholder` , `id` , `name` , `value` , entre outros. A base dos parâmetros é, na maioria dos casos, a seguinte:

- `form_open()` e `form_open_multipart()` :


```

form_open([$action = '[' , $attributes = '[' ,
$hidden = array()]]])

form_open_multipart([$action      =      '[' ,
$attributes = array()[ , $hidden = array()]]])

```

 - `$action` (string) – URL para execução do "action".
 - `$attributes` (array) – Atributos HTML.
 - `$hidden` (array) – Lista com campos do tipo hidden .
- `form_input()` , `form_upload()` , `form_textarea()` e `form_submit()` :


```

form_input([$data = '[' , $value = '[' , $extra
= ']]])

form_upload([$data = '[' , $value = '[' ,

```

```

$extra = ']]))

    form_textarea([$data = '[' , $value = '[' ,
$extra = ']]]])

    form_submit([$data = '[' , $value = '[' ,
$extra = ']]]])

```

- \$data (array) – Atributos do campo.
- \$value (string) – Valor do campo.
- \$extra (mixed) – Atributos extras, podendo ser um array ou string literal.

Com o formulário concluído, é hora de fazer algumas alterações no *controller* `TrabalheConosco` , validando o arquivo enviado e armazenando esse arquivo no servidor. O CodeIgniter possui uma biblioteca nativa muito boa para trabalhar com upload de arquivos, chamada *File Uploading*.

Para fazer o processo de validação do arquivo, crie um novo método chamado `UploadFile()` no controller `Contato` .

```

private function UploadFile($inputFileName)
{
    $this->load->library('upload');

    $path = "../curriculos";

    $config['upload_path'] = $path;
    $config['allowed_types'] = 'doc|docx|pdf|zip|rar';
    $config['max_size'] = '5120';
    $config['encrypt_name'] = TRUE;

    if (!is_dir($path))
        mkdir($path, 0777, $recursive = true);

    $this->upload->initialize($config);

    if (!$this->upload->do_upload($inputFileName)) {
        $data['error'] = true;
    }
}

```

```

        $data['message'] = $this->upload->display_errors();
    } else {
        $data['error'] = false;
        $data['fileData'] = $this->upload->data();
    }
    return $data;
}

```

Esse método vai fazer o processo de validação e upload do arquivo, armazenando-o no diretório `currículos`, na raiz do projeto, que será criado automaticamente caso não exista, como está nas linhas a seguir:

```

...

if (!is_dir($path))
    mkdir($path, 0777, $recursive = true);

...

```

O carregamento da biblioteca é feito através do já conhecido método `load->library('upload')`. Em seguida, é definido o *path* onde os arquivos serão armazenados, usando a variável `$path`.

A variável `$config` define parâmetros de configuração para o upload, nos quais são passados o *path*, os tipos de arquivo permitidos, o tamanho máximo do arquivo e a informação para criptografar o nome do arquivo, respectivamente.

O `$this->upload->initialize($config)` inicializa a biblioteca com as configurações definidas na variável `$config`, só então é feito o processo de validação do upload, usando `if (!$this->upload->do_upload($inputFileName))`. Se o upload realizado pelo método `do_upload()` não for bem sucedido, então `$data['error']` recebe o valor `TRUE`, e `$data["message"]` recebe as informações sobre o erro no upload retornadas pelo método `upload->display_errors()`. Caso o upload seja executado, então é retornado `FALSE` para o status do erro, e

`$data['fileData']` recebe as informações sobre o upload, retornadas pelo método `upload->data()` .

`DO_UPLOAD()`

O método `do_upload()` recebe como parâmetro o nome do campo tipo `file` usado no formulário. No exemplo, ele recebe uma variável `$inputFileName` , que é o parâmetro do método `UploadFile()` . Assim, sempre que precisar fazer um upload de arquivo, basta chamar esse método informando o nome do campo.

O retorno do método `upload->data()` é um array similar ao do exemplo a seguir, trazendo informações bem específicas sobre o arquivo. Essas informações podem variar conforme o tipo de arquivo.

Array

```
(
    [file_name]      => mypic.jpg
    [file_type]      => image/jpeg
    [file_path]      => /path/to/your/upload/
    [full_path]      => /path/to/your/upload/jpg.jpg
    [raw_name]       => mypic
    [orig_name]      => mypic.jpg
    [client_name]    => mypic.jpg
    [file_ext]       => .jpg
    [file_size]      => 22.2
    [is_image]       => 1
    [image_width]    => 800
    [image_height]   => 600
    [image_type]     => jpeg
    [image_size_str] => width="800" height="200"
)
```

DICA

Caso queira retornar somente um dos valores, basta informar o nome da chave como parâmetro do método. Por exemplo:

```
$this->upload->data('file_name')
```

E fechando o método, os dados são retornados para que possa haver um processamento sobre eles, a fim de informar ao usuário uma mensagem e enviar o e-mail com o arquivo em anexo.

No capítulo 10. *Upload, download e compressão de arquivos*, você verá mais detalhes e aprofundará os conhecimentos sobre a library `upload`.

A última etapa é atualizar o método `TrabalheConosco` para processar o upload, e o método `SendEmailToAdmin` para enviar um arquivo anexo.

```
public function TrabalheConosco()
{
    $data['title'] = "LCI | Trabalhe Conosco";
    $data['description'] = "Exercício de exemplo do capítulo 5 do
livro CodeIgniter";

    $this->form_validation->set_rules('nome', 'Nome', 'trim|required|
min_length[3]');
    $this->form_validation->set_rules('email', 'Email', 'trim|required|
valid_email');
    $this->form_validation->set_rules('assunto', 'Assunto', 'trim|
required|min_length[5]');
    $this->form_validation->set_rules('mensagem', 'Mensagem', 'trim|
required|min_length[30]');

    if($this->form_validation->run() == FALSE){
        $data['formErrors'] = validation_errors();
    }else{
```

```

        $uploadCurriculo = $this->UploadFile('curriculo');

        if($uploadCurriculo['error']){
            $data['formErrors'] = $uploadCurriculo['message'];
        }else{
            $formData = $this->input->post();
            $emailStatus = $this->SendEmailToAdmin($formData['email'],
            $formData['nome'], "to@domain.com", "To Name", $formData['assunto'],
            $formData['mensagem'], $formData['email'], $formData['nome'], $uploadCurriculo['fileData']['full_path']);

            if($emailStatus){
                $this->session->set_flashdata('success_msg', 'Contato recebido com sucesso!');
            }else{
                $data['formErrors'] = "Desculpe! Não foi possível enviar o seu contato. tente novamente mais tarde.";
            }
        }
    }

    $this->load->view('trabalhe-conosco', $data);
}

```

No método `TrabalheConosco`, foi feita a adição do método `UploadFile()`, passando o seu retorno para a variável `$uploadCurriculo`. Após isso, foi feita uma verificação para saber se o upload foi executado com sucesso ou não. Em caso de erro, eles são passados para a variável de erros e exibidos para o usuário. Mas se foi processado com sucesso, o fluxo segue e é feito o envio do e-mail com o arquivo anexado.

Para o envio do e-mail com anexo, foi adicionado um novo parâmetro (`$attach`) ao método `SendEmailToAdmin`, onde é passado o *path* do arquivo a ser anexado. Também foi feita uma checagem que, caso o arquivo foi informado, o método `attach()` é chamado para adicionar o anexo ao e-mail.

```

private function SendEmailToAdmin($from, $fromName, $to, $toName,
$subject, $message, $reply = null, $replyName = null, $attach = null)
{
    $this->load->library('email');

```

```

$config['charset'] = 'utf-8';
$config['wordwrap'] = TRUE;
$config['mailtype'] = 'html';
$config['protocol'] = 'smtp';
$config['smtp_host'] = 'smtp.seudominio.com.br';
$config['smtp_user'] = 'user@seudominio.com.br';
$config['smtp_pass'] = 'suasenha';
$config['newline'] = '\r\n';

$this->email->initialize($config);

$this->email->from($from, $fromName);
$this->email->to($to, $toName);

if($reply)
    $this->email->reply_to($reply, $replyName);

if($attach)
    $this->email->attach($$attach);

$this->email->subject($subject);
$this->email->message($message);

if($this->email->send())
    return true;
else
    return false;
}

```

6.6 CONCLUSÃO

Durante este capítulo você aprendeu a utilizar as libraries e helpers mais comuns do CodeIgniter, a fazer configurações na aplicação, a trabalhar com hooks, a usar sessões, e a criar controllers e views. Tudo isso através de um projeto simples. Se antes de ler o capítulo você não era capaz de criar um site básico usando PHP, agora você já é capaz de fazer isso.

Se durante a leitura e execução dos exemplos você teve alguma dúvida e ainda não conseguiu esclarecê-la, poste sua dúvida no fórum. Terei o maior prazer em continuar ajudando para a sua

evolução profissional.

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-5e6-criando-um-site-institucional>

Links úteis

- **Bootstrap:** <http://getbootstrap.com>
- **Documentação oficial do CI sobre o Helper Form:**
https://codeigniter.com/user_guide/helpers/form_helper.html
- **Documentação oficial do CI sobre a Library Email:**
https://codeigniter.com/user_guide/libraries/email.html
- **Documentação oficial do CI sobre a Library File Uploading:**
https://codeigniter.com/user_guide/libraries/file_uploading.html
- **Documentação oficial do CI sobre a Library Form Validation:**
https://codeigniter.com/user_guide/libraries/form_validation.html

VALIDANDO FORMULÁRIOS

"O homem mais pobre não é o homem sem dinheiro, é o homem sem sonhos." — Max L. Forman

Nos dois capítulos anteriores, quando foi criado o site institucional, foi implementada uma validação dos formulários de Contato e Trabalhe conosco. Entretanto, não foram ensinados detalhes mais específicos sobre o processo de validação.

Neste capítulo, veremos com mais detalhes sobre como validar os dados dos formulários usando a library *Form Validation*, uma library nativa do CodeIgniter. Ela possui, além de uma série de regras de validação, a possibilidade de criação de *callbacks* para validações mais específicas, como por exemplo, uma validação de CPF ou CNPJ, muito utilizada em alguns tipos de formulário de cadastro.

7.1 CARREGANDO A LIBRARY

Antes de qualquer operação de validação, é necessário carregar a library *Form Validation*, para que seus métodos fiquem disponíveis para uso.

**Carregamento da library no arquivo
config/autoload.php**

```
$autoload['libraries'] = array('form_validation');
```

Ao carregar a library *Form Validation* no `autoload` , seus métodos poderão ser utilizados em qualquer parte do projeto, sem a necessidade de carregar novamente a library através do método `$this->load->library()` .

Como a validação de formulários ocorre em páginas específicas e não em todas as páginas, é aconselhável carregá-la somente quando for utilizar, para que não sejam feitas cargas desnecessárias.

Carregando diretamente no método

```
$this->load->library('form_validation');
```

Essa é a maneira mais aconselhável para carregar essa library. Assim, seus métodos estarão disponíveis apenas dentro do contexto onde foi carregada. Por exemplo, se ela foi carregada no método `FaleConosco()` , ao acessar uma execução do método `TrabalheConosco()` , os métodos dessa library não estarão disponíveis, a não ser que ela seja carregada novamente.

No site institucional que você criou nos capítulos *Criando um site institucional — Parte I e II*, a library foi carregada no método `__construct()` do controller `Contato` , ficando disponível para todos os métodos desse controller e evitando a execução repetida de `$this->load->library('form_validation')` .

```
...
function __construct(){
    parent::__construct();
    $this->load->library('form_validation');
    $this->load->helper('form');
}
...
```

7.2 APLICANDO AS REGRAS DE VALIDAÇÃO

As regras de validação devem ser aplicadas no *controller*, utilizando o método `$this->form_validation->set_rules()`. Esse método deve ser executado para cada campo do formulário a ser validado, e ele é complementado por três parâmetros obrigatórios e um opcional:

- **1º parâmetro:** nome do campo, conforme colocado no atributo `name` da tag `input` do campo na view.

```
<input id="nome" name="nome" placeholder="Nome" class="form-control input-md" required="" type="text">
```

- **2º parâmetro:** nome ou pequeno texto para exibição na mensagem de erro que é retornada caso a validação retorne `FALSE`.
- **3º parâmetro:** regras de validação disponíveis na `library` e/ou regras personalizadas, criadas por você.
- **4º parâmetro (opcional):** mensagem de erro a ser exibida em substituição da mensagem padrão retornada pelo método.

Veja o exemplo com os 3 parâmetros obrigatórios:

```
$this->form_validation->set_rules('nome', 'Nome', 'trim|required|min_length[3]');
```

Agora o exemplo com o 4º parâmetro (opcional):

```
$this->form_validation->set_rules('nome', 'Nome', 'trim|required|min_length[3]', array('required' => 'Informe o seu nome, por favor.'));
```

Veja que o parâmetro opcional, que substitui a mensagem de erro padrão, é passado como um array, pois como podem ser aplicadas várias regras, cada uma delas possui sua própria mensagem. No exemplo anterior, temos três regras de validação:

- `trim` : remove espaços em branco antes e depois do

valor do campo;

- `required` : torna o preenchimento obrigatório;
- `min_length[3]` : faz com que a quantidade mínima de caracteres do campo seja 3 (três).

Como aplicamos a alteração da mensagem de erro apenas para a regra `required`, as demais regras aplicadas terão as mensagens padrões retornadas caso não sejam atendidas.

Mais à frente neste capítulo, você verá uma lista com as regras de validação da library *Form Validation*

Usando array para aplicar as regras de validação

Uma outra forma de aplicar as regras de validação é por meio de um `array`. Assim, você não precisa utilizar o método `$this->form_validation->set_rules()` de forma repetida.

No formulário de contato do site que foi criado temos regras de validação para quatro campos: nome, e-mail, assunto e mensagem. Veja no exemplo a seguir como ficaria a implementação dessas mesmas regras utilizando um `array`.

```
$rules = array(
    array(
        'field' => 'nome',
        'label' => 'Nome',
        'rules' => 'trim|required|min_length[3]'
    ),
    array(
        'field' => 'email',
        'label' => 'Email',
        'rules' => 'trim|required|valid_email'
    ),
    array(
        'field' => 'assunto',
        'label' => 'Assunto',
```

```

        'rules' => 'trim|required|min_length[5]'
    ),
    array(
        'field' => 'mensagem',
        'label' => 'Mensagem',
        'rules' => 'trim|required|min_length[30]'
    )
);

$this->form_validation->set_rules($rules);

```

Veja que temos um array composto, no qual cada posição do array principal é um novo array que representa um campo do formulário a ser validado. O array correspondente a cada campo possui como chave os parâmetros obrigatórios para a execução da validação: `field`, `label`, `rules` (respectivamente: nome do campo, texto a ser exibido junto da mensagem de erro e regras de validação).

Se fosse o caso de utilizar o quarto parâmetro, então este seria inserido logo após `rules` e também seria um array, pois como explicado anteriormente, ele é informado para cada uma das regras aplicadas ao campo.

Por fim, é chamado o método `$this->form_validation->set_rules($rules)` com o parâmetro `$rules`, que contém todas as regras de validação para o formulário. Esse código pode muito bem substituir o código que foi escrito durante o desenvolvimento do site institucional, conforme apresentado a seguir:

```

$this->form_validation->set_rules('nome', 'Nome', 'trim|required|min_length[3]');
$this->form_validation->set_rules('email', 'Email', 'trim|required|valid_email');
$this->form_validation->set_rules('assunto', 'Assunto', 'trim|required|min_length[5]');
$this->form_validation->set_rules('mensagem', 'Mensagem', 'trim|required|min_length[30]');

```

Regras de validação em cascata

Você reparou que as regras de validação até agora foram passadas em um único parâmetro e separadas por `|` ?

O CI permite que você determine as regras em cascata em vez de uma regra para cada chamada do método `$this->form_validation->set_rules()` . Além de ser possível informar as regras, como foi feito até o momento, elas também podem ser informadas como um `array` . Veja no exemplo a seguir como ficaria essa implementação, usando `array` em vez de `string` .

```
$this->form_validation->set_rules('nome', 'Nome', array('trim', 'required', 'min_length[3]'));
```

Tanto com `string` quanto com `array` a implementação é válida, então cabe a você escolher o que fica melhor na hora de escrever o seu código e deixá-lo mais legível.

7.3 REGRAS DE VALIDAÇÃO DA LIBRARY FORM VALIDATION

Até agora, você viu como aplicar as regras de validação através do método `$this->form_validation->set_rules()` . Além das regras que foram aplicadas no formulário de contato, existem várias outras disponíveis.

Veja a seguir quais são as regras de validação disponíveis no CodeIgniter:

`required` : retorna `FALSE` se o campo estiver vazio.

- **Parâmetro:** não possui

`matches` : retorna `FALSE` se o valor do campo não coincidir com o parâmetro.

- **Parâmetro:** possui
- **Exemplo:** `matches[form_item]`

`regex_match` : retorna `FALSE` se o valor do campo não coincidir com a expressão regular.

- **Parâmetro:** possui
- **Exemplo:** `regex_match[/regex/]`

`differs` : retorna `FALSE` se o valor do campo for igual ao parâmetro.

- **Parâmetro:** possui
- **Exemplo:** `differs[form_item]`

`is_unique` : retorna `FALSE` se o valor do campo não for único na tabela e campo informados como parâmetro. Essa regra exige que o *Query Builder* esteja habilitado.

- **Parâmetro:** possui
- **Exemplo:** `is_unique[table.field]`

`min_length` : retorna `FALSE` se o número de caracteres do campo for menor do que o parâmetro.

- **Parâmetro:** possui
- **Exemplo:** `min_length[3]`

`max_length` : retorna `FALSE` se o número de caracteres do campo for maior do que o parâmetro.

- **Parâmetro:** possui
- **Exemplo:** `max_length[12]`

`exact_length` : retorna `FALSE` se o número de caracteres do campo não for exatamente igual ao valor do parâmetro.

- **Parâmetro:** possui
- **Exemplo:** `exact_length[8]`

`greater_than` : retorna `FALSE` se o número de caracteres do campo for menor ou igual ao valor do parâmetro, ou não for numérico.

- **Parâmetro:** possui
- **Exemplo:** `greater_than[8]`

`greater_than_equal_to` : retorna `FALSE` se o número de caracteres do campo for menor ou igual ao valor do parâmetro, ou não for numérico.

- **Parâmetro:** possui
- **Exemplo:** `greater_than_equal_to[8]`

`less_than` : retorna `FALSE` se o número de caracteres do campo for maior ou igual ao valor do parâmetro, ou não for numérico.

- **Parâmetro:** possui
- **Exemplo:** `less_than[8]`

`less_than_equal_to` : retorna `FALSE` se o número de caracteres do campo for maior ou igual ao valor do parâmetro, ou não for numérico.

- **Parâmetro:** possui
- **Exemplo:** `less_than_equal_to[8]`

`in_list` : retorna `FALSE` se o valor do campo não estiver dentro da lista passada como parâmetro.

- **Parâmetro:** possui
- **Exemplo:** `in_list[red,blue,green]`

`alpha` : retorna `FALSE` se o valor do campo tiver algum caractere diferente de letra

- **Parâmetro:** não possui

`alpha_numeric` : retorna `FALSE` se o valor do campo tiver algum caractere diferente de letra ou número.

- **Parâmetro:** não possui

`alpha_numeric_spaces` : retorna `FALSE` se o valor do campo contiver algo diferente de letra, número ou espaço. Recomenda-se o uso de `trim` para remover os espaços no início e no final do valor do campo.

- **Parâmetro:** não possui

`alpha_dash` : retorna `FALSE` se o valor do campo for algo diferente de letras, números, sublinhados ou traços.

- **Parâmetro:** não possui

`numeric` : retorna `FALSE` se o valor do campo não for numérico.

- **Parâmetro:** não possui

`integer` : retorna `FALSE` se o valor do campo não for um inteiro.

- **Parâmetro:** não possui

`decimal` : retorna `FALSE` se o valor do campo não for um decimal.

- **Parâmetro:** não possui

`is_natural` : retorna `FALSE` se o valor do campo não for um

número natural: 0, 1, 2, 3 etc.

- **Parâmetro:** não possui

`is_natural_no_zero` : retorna `FALSE` se o valor do campo não for um número natural diferente de zero: 1, 2, 3 etc.

- **Parâmetro:** não possui

`valid_url` : retorna `FALSE` se a URL não for válida.

- **Parâmetro:** não possui

`valid_email` : retorna `FALSE` se o e-mail não for válido.

- **Parâmetro:** não possui

`valid_ip` : retorna `FALSE` se o IP não for válido. Aceita um parâmetro opcional `ipv4` ou `ipv6` para especificar o formato do IP.

- **Parâmetro:** não possui

`valid_base64` : retorna `FALSE` se o valor do campo não for um conjunto de caracteres Base64 válido.

- **Parâmetro:** não possui

7.4 CRIANDO SUAS PRÓPRIAS REGRAS DE VALIDAÇÃO

O CI permite que você crie suas próprias regras de validação para usar no método `set_rules()`. Você pode criar um método de validação dentro do *controller*, onde as validações estão sendo aplicadas, e chamá-lo em `set_rules()` usando o prefixo *callback_*.

Veja a seguir um exemplo de regra personalizada e como ela foi chamada dentro de `set_rules()` :

```
<?php

class Hash extends CI_Controller {

    public function index()
    {
        $this->load->library('form_validation');
        $this->form_validation->set_rules('hash', 'Hash', 'callback_hash_check');
        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('hash-form');
        }
        else
        {
            $this->load->view('hash-success');
        }
    }

    public function hash_check($str)
    {
        if ($str == 'A3@hbGF32mbN')
        {
            $this->form_validation->set_message('hash_check', 'O campo hash não corresponde a "A3@hbGF32mbN"');
            return FALSE;
        }
        else
        {
            return TRUE;
        }
    }
}
```

Veja que foi criado o método `hash_check($str)` , e não `callback_hash_check($str)` . O termo `callback_` é usado apenas para que o método `set_rules()` identifique a regra de validação personalizada.

No método criado, é feita a verificação da `string` digitada pelo usuário no campo do formulário. O método compara a `string`

digitada com uma string pré-estabelecida (A3@hbGF32mbN). Se elas forem idênticas, então o método retorna TRUE ; caso contrário, é setada a mensagem de erro para a regra de validação e retorna FALSE . O método usado para criar a mensagem de erro é o `$this->form_validation->set_message(nome_metodo, mensagem) .`

Uma outra forma bastante usada como método de validação é usando métodos já criados no *model*. Por exemplo, você precisa validar se o usuário informado no formulário de login existe e quer fazer isso como uma regra de validação em `set_rules()` . No *model*, você tem um método chamado `exist($user)` , então você usaria da seguinte forma em `set_rules()` :

```
$this->form_validation->set_rules(
    'username', 'Username',
    array(
        'required',
        array($this->users_model, 'exist')
    )
);
```

Usamos um array em vez de string para informar as regras de validação. Isso foi necessário pois, para usar um método do *model* como regra de validação, é preciso passar a informação como um array . Então, no array correspondente a essa regra temos primeiro a identificação do *model* `$this->users_model` e, em seguida, o nome do método do *model* a ser executado, que nesse caso é `exist()` .

7.5 CRIANDO MENSAGENS DE ERRO

As mensagens de erro são a forma de informar ao usuário o que ocorreu durante a validação do formulário. Por padrão, o CI retorna as mensagens em inglês, a menos que você altere as configurações para que todas as mensagens sejam em português, como foi

mostrado na criação do site institucional.

O local onde ficam as mensagens padrão é `system/language/english/form_validation_lang.php`. Para alterar, basta copiar o arquivo `form_validation_lang.php` para `application/language`, colocando-o dentro do diretório do idioma na qual você está alterando. Por exemplo, `application/language/english/form_validation_lang.php`.

Anteriormente, vimos como criar uma mensagem de erro para uma regra personalizada, dentro de um método do *controller*, usando `$this->form_validation->set_message(nome_metodo, mensagem)`. Vamos ver agora como criar mensagens personalizadas diretamente em `set_rules()`. Para isso, você precisará adicionar um 4º parâmetro, que é o responsável por determinar as mensagens.

Veja no exemplo a seguir quando alteramos a mensagem padrão da `regra2`.

```
$this->form_validation->set_rules('nome_campo', 'descricao_campo',  
'regra1|regra2|regra3',  
    array('regra2' => 'Mensagem de erro da regra2 para o campo  
    nome_campo')  
);
```

Se quiséssemos alterar as mensagens para outras regras na mesma validação, bastaria adicionar essas mensagens ao `array` do quarto parâmetro.

Agora que você já sabe como alterar as mensagens padrões das regras de validação e criar suas próprias mensagens de erros em regras personalizadas, vamos ver uma possibilidade interessante na criação das mensagens: o uso de tags para poder recuperar o nome do campo e o até mesmo o parâmetro passado para a regra.

Quando criamos a regra personalizada anteriormente neste capítulo, o código ficou assim:

```

...
public function hash_check($str)
{
    if ($str == 'A3@hbGF32mbN')
    {
        $this->form_validation->set_message('hash_check', 'O campo
hash não corresponde a "A3@hbGF32mbN"');
        return FALSE;
    }
    else
    {
        return TRUE;
    }
}
...

```

Veja que na mensagem de erro foi informado o nome do campo hash . Porém, isso está errado, pois a validação pode ser usada para outros campos, no caso de validações genéricas. Então, você precisará recuperar o nome do campo pela tag {field} , e o método set_message() passaria a ficar da seguinte forma:

```

$this->form_validation->set_message('hash_check', 'O campo {field}
não corresponde a "A3@hbGF32mbN"');

```

Ao utilizar a tag {field} , o CI automaticamente substitui essa tag pelo nome do campo onde a regra foi aplicada. Assim, você pode ter a mensagem personalizada com o nome do campo em uma regra de validação genérica, que poderá ser utilizada por diversos campos.

Uma outra tag que pode ser usada é {param} . Ela é substituída pelo valor informado como parâmetro da regra de validação.

Veja a seguir um exemplo de mensagem para regra min_length , em que a mensagem exibe o nome do campo e o valor mínimo de caracteres que deve ser digitado.

```

$this->form_validation->set_message('min_length', '{field} deve te
r pelo menos {param} caracteres.');
```

7.6 EXECUTANDO AS REGRAS DE

VALIDAÇÃO

Não basta somente adicionar as regras através do método `$this->form_validation->set_rules()` , é preciso executar a validação. Na maioria das vezes, o resultado dessa validação vai determinar o fluxo do sistema.

Para executar a validação, é usado o método `$this->form_validation->run()` , que retorna `FALSE` em caso de erros de validação, ou `TRUE` caso as validações estejam corretas.

Veja no fragmento de código a seguir como foi implementada essa verificação a partir do método `$this->form_validation->run()` .

```
public function index()
{
    $this->load->library('form_validation');
    $this->form_validation->set_rules('hash', 'Hash', 'callback_hash_check');
    //aqui fazemos a verificação do resultado das validações.
    if ($this->form_validation->run() == FALSE)
    {
        $this->load->view('hash-form');
    }
    else
    {
        $this->load->view('hash-success');
    }
}
```

Para fechar este capítulo, veremos como recuperar os dados que o usuário preencheu no formulário e exibiremos as mensagens de erro das regras de validação para ele.

7.7 RECUPERANDO OS DADOS DOS CAMPOS DO FORMULÁRIO

Recuperar os dados dos campos que o usuário preencheu é

muito simples. Basta você utilizar `set_value('nome_campo')` no atributo `value` dos inputs do formulário.

```
<input id="email" name="email" placeholder="Email" class="form-control input-md" required="" type="text" value="<?=set_value('email')?>">
```

7.8 EXIBINDO AS MENSAGENS DE ERRO

Para exibir as mensagens de erro, você tem duas possibilidades: agrupadas ou individuais.

Agrupadas

As mensagens de erros agrupadas são exibidas pelo método `validation_errors()`. Ele retorna todas as mensagens dos erros das regras aplicadas de uma única vez, em um único bloco. Você pode configurar essa exibição determinando quais serão os delimitadores, se serão uma `<div></div>`, ou uma lista ``. Isso pode ser feito de duas formas:

1. Como parâmetro de `validation_errors()`:

```
validation_errors('<div class="errors">', '</div>');
```

2. De forma global, com o método `set_error_delimiters()`:

```
$this->form_validation->set_error_delimiters('<div class="errors">', '</div>');
```

Individuais

Se, por acaso, você quiser exibir as mensagens de forma individual, próxima de cada campo com erro, você pode fazer usando o método `form_error()`.

```
form_error('nome_campo', 'delimitador_abertura', 'delimitador_fechamento');
```

O segundo e o terceiro parâmetros são opcionais. Se você usou `set_error_delimiters()` , então os delimitadores globais serão utilizados. Se você tiver informado delimitadores globais pelo método `set_error_delimiters()` e inseriu outros delimitadores em `validation_errors()` ou `form_error()` , os delimitadores globais serão sobrescritos na chamada dos respectivos métodos.

```
$this->form_validation->set_error_delimiters('<div class="errors">'
, '</div>');

validation_errors('<div class="errors">', '</div>'); // os delimit
adores globais serão ignorados e estes serão utilizados

form_error('nome_campo'); //utilizará os delimitadores globais
```

7.9 CONCLUSÃO

Chegamos ao final de mais um capítulo, e agora você já sabe como validar formulários usando a library *Form Validation* e seus recursos. Muito pode ser feito em termos de validação de formulário. Então, aproveite a possibilidade de criar regras personalizadas, e deixe os seus projetos ainda mais seguros e com dados ainda mais consistentes.

Código-fonte

Faça o download do código-fonte completo deste exemplo em <https://github.com/jlamim/livro-codeigniter/tree/master/CAP-07-validando-formularios>

Exemplo prático

Para reforçar o que você aprendeu neste capítulo, acesse o portal **Universidade CodeIgniter** pelo link a seguir, e desenvolva o exemplo prático do tutorial.

<http://www.universidadecodeigniter.com.br/validando->

Links úteis

- **Documentação oficial do CI sobre a library Form Validation:**
https://codeigniter.com/user_guide/libraries/form_validation.html

ENVIANDO E-MAILS COM A LIBRARY EMAIL

"A persistência é o caminho do êxito." — Charles Chaplin

Praticamente todo site ou sistema precisa ter, pelo menos, uma rotina para envio de e-mail. E com a library *Email* do CodeIgniter, o processo de criação dessa rotina é muito rápido.

8.1 ENVIANDO UM E-MAIL SIMPLES

Vamos começar vendo o código necessário para enviar um e-mail simples.

```
//Carrega a library
$this->load->library('email');

//Define o remetente. O primeiro parâmetro deve ser o e-mail e o s
egundo (opcional) o nome do remetente
$this->email->from('your@example.com', 'Your Name');

//Define o destinatário, e os parâmetros são os mesmos do método '
from'
$this->email->to('someone@example.com');

//Define quem receberá cópia do e-mail
$this->email->cc('another@another-example.com');

//Define quem receberá cópia oculta
$this->email->bcc('them@their-example.com');

//Define o assunto do e-mail
$this->email->subject('Email Test');
```

```
//Define o conteúdo da mensagem
$this->email->message('Testing the email class.');
```



```
//Faz o envio do e-mail. Retorna TRUE ou FALSE
$this->email->send();
```

Esse código faz o envio de um e-mail usando a configuração padrão da library *Email*. Não é necessário utilizar todos os métodos apresentados, pois com `from()` , `to()` , `subject()` , `message` e `send()` , você já executa o envio do e-mail.

Se você quiser verificar se o e-mail foi enviado ou não, pode aplicar um `if` , como no código a seguir:

```
if($this->email->send()){
    return TRUE;
}else{
    return FALSE;
}
```

Em caso de erro no envio, você pode recuperar informações usando o método `print_debugger()` , que retorna um log com informações sobre o envio. O método por padrão retorna todas as informações, mas você pode passar um `array` como parâmetro contendo o tipo de informação a ser retornada. Os valores possíveis para compor o `array` são: `headers` , `subject` , `body` .

```
$this->email->print_debugger(array('headers'));
```

Se no método `send()` você passar `FALSE` como parâmetro, independente do envio ocorrer ou não, o método `print_debugger()` não vai retornar nada.

8.2 ENVIANDO E-MAIL USANDO UMA VIEW COMO TEMPLATE DA MENSAGEM

Em sistemas que disparam vários e-mails, cada um com uma finalidade diferente, é mais eficiente você trabalhar com uma *view*

para cada template diferente de e-mail. Para enviar e-mail usando as *views* como conteúdo da mensagem, você só precisa alterar uma linha em relação ao código que foi mostrado anteriormente.

- **Antes:**

```
$this->email->message('Testing the email class.');
```

- **Depois:**

```
$mensagem = $this->load->view('nome_view', $dados, TRUE);  
$this->email->message($mensagem);
```

Dessa forma, usamos o método `$this->load->view()` para recuperar a *view* como se estivéssemos carregando o conteúdo de uma página. A grande diferença está no último parâmetro, passado como `TRUE`, pois ele define que, em vez de exibir a *view* na tela, vai retornar o conteúdo dela para a variável `$mensagem`, devidamente renderizada.

Tem dúvidas sobre o carregamento das *views*? Que tal refrescar sua memória e revisitar o capítulo 4. *Anatomia de uma view* para revisar o assunto?

8.3 ENVIANDO E-MAIL COM ANEXO

Para enviar um e-mail com arquivo anexo, basta executar o método `attach()` da library *Email* junto com os demais métodos já apresentados anteriormente neste capítulo. O método `attach()` aceita alguns parâmetros para possibilitar a configuração do envio do anexo. Veja a seguir alguns exemplos de uso do método:

- **Informando apenas o caminho do arquivo:**

```
$this->email->attach('/caminho/do/arquivo.jpg');
```

- **Informando uma URL como caminho do arquivo:**

```
$this->email->attach('http://www.arquivos.com/arquivo.jpg');
```

- **Customizando o nome do arquivo:**

```
$this->email->attach('arquivo.pdf', 'attachment', 'novo_nome.pdf');
```

- **Múltiplos arquivos:**

```
$this->email->attach('/caminho/do/arquivo.jpg');  
$this->email->attach('/caminho/do/arquivo2.jpg');  
$this->email->attach('/caminho/do/arquivo3.jpg');
```

Para múltiplos arquivos, você deve executar o método `attach()` para cada um deles, podendo usar qualquer uma das formas citadas.

8.4 ENVIO DE E-MAIL COM SMTP

Para enviar e-mails usando o protocolo SMTP, você precisará fazer uso dos parâmetros de configuração da library. O restante do código para envio permanece o mesmo, já apresentado anteriormente.

```
$this->load->library('email');  
  
$config['protocol'] = 'smtp';  
$config['smtp_host'] = 'smtp.domain.com';  
$config['smtp_user'] = 'user-smtp';  
$config['smtp_pass'] = 'smtp-pass';  
$config['smtp_port'] = 587;  
  
$this->email->initialize($config);
```

A variável `$config` do tipo `array` foi criada e recebe as informações necessárias do SMTP (host, usuário, senha e porta). O método `initialize($config)` é chamado para que as informações de configuração sejam passadas para a library.

A partir daí, é só proceder normalmente com a definição de remetente, destinatário e demais informações, como já foi mostrado neste capítulo.

8.5 PARÂMETROS DE CONFIGURAÇÃO

A library *Email* possui vários parâmetros para configuração. Veja a seguir a lista completa:

`useragent` : o "user agent"

- **Valor padrão:** CodeIgniter

`protocol` : o protocolo para envio do e-mail

- **Valor padrão:** mail
- **Opções:** mail, sendmail, ou smtp

`mailpath` : path para o servidor do Sendmail

- **Valor padrão:** /usr/sbin/sendmail

`smtp_host` : endereço do servidor SMTP

`smtp_user` : usuário do SMTP

`smtp_pass` : senha do SMTP

`smtp_port` : porta do SMTP

- **Valor padrão:** 25

`smtp_timeout` : timeout do SMTP (em segundos)

- **Valor padrão:** 5

`smtp_keepalive` : ativa conexão SMTP persistente

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

smtp_crypto : modo de encriptação do SMTP

- **Opções:** tls ou ssl

wordwrap : ativa quebra de palavra

- **Valor padrão:** TRUE
- **Opções:** TRUE ou FALSE

wrapchars : número de caracteres para a quebra

- **Valor padrão:** 76

mailtype : tipo de e-mail. Se o conteúdo for HTML, será enviado como uma página web. Fique atento a links e caminhos de imagens relativos, pois eles não vão funcionar.

- **Valor padrão:** text
- **Opções:** text ou html

charset : codificação de caracteres (utf-8, iso-8859-1 etc.)

- **Valor padrão:** \$config['charset']

validate : validação do e-mail

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

priority : prioridade do e-mail (1 = alta, 5 = baixa, 3 = normal)

- **Valor padrão:** 3
- **Opções:** 1, 2, 3, 4, 5

`crlf` : caractere para quebra de linha (use `"\r\n"` para atender à RFC 822)

- **Valor padrão:* `"\n"`
- **Opções:** `"\r\n"` ou `"\n"` ou `"\r"`

`newline` : caractere para quebra de linha (use `"\r\n"` para atender à RFC 822)

- **Valor padrão:** `"\n"`
- **Opções:** `"\r\n"` ou `"\n"` ou `"\r"`

`bcc_batch_mode` : ativa o modo BCC Batch

- **Valor padrão:** `FALSE`
- **Opções:** `TRUE` ou `FALSE`

`bcc_batch_size` : número de e-mail para BCC Batch

- **Valor padrão:** `200`

`dsn` : ativa notificações de mensagem para o servidor

- **Valor padrão:** `FALSE`
- **Opções:** `TRUE` ou `FALSE`

A aplicação desses parâmetros é muito simples. Veja no exemplo a seguir onde configuramos o envio do e-mail pelo protocolo Sendmail, definimos o charset e a quebra de palavras.

```
$config['protocol'] = 'sendmail';  
$config['mailpath'] = '/usr/sbin/sendmail';  
$config['charset'] = 'iso-8859-1';  
$config['wordwrap'] = TRUE;  
  
$this->email->initialize($config);
```

Foi criada a variável `$config`, que é um array e recebe como chave o parâmetro a ser configurado e o valor do parâmetro. Em

seguida, é executado o método `initialize($config)` , que passa as configurações para a library.

Uma outra alternativa para definir as configurações de envio e não precisar ficar repetindo essas configurações sempre que for criar uma rotina de envio de e-mail, é usar o arquivo de configuração. Para isso, você deve criar um arquivo `email.php` dentro de `application/config` , e adicionar ao seu conteúdo o array com as configurações, como foi feito no código anterior. Dessa forma você não precisará utilizar o método `initialize()` , pois, ao carregar a library, as configurações já serão passadas para ela automaticamente.

8.6 OUTROS MÉTODOS DA LIBRARY EMAIL

Além dos métodos `initialize` , `from` , `to` , `cc` , `bcc` , `subject` , `message` , `attach` e `send` , que foram vistos neste capítulo, a library ainda possui outros métodos:

- `reply_to($email,$nome)` : possui a mesma sintaxe do método `to` e serve para determinar qual e-mail receberá a resposta, caso o destinatário responda ao e-mail.
- `set_alt_message($altMessage)` : é uma mensagem alternativa para ser exibida no corpo do e-mail caso você tenha enviado um e-mail em formato HTML e o destinatário não tenha suporte a esse formato.
- `set_header($header, $value)` : adiciona informações ao cabeçalho do e-mail pelos parâmetros `$header` e `$value` .
- `clear($clear_attachments)` : limpa as variáveis evitando o disparo de novos e-mails com informações

de envios anteriores. O parâmetro `$clear_attachments` é um booleano que, se passado como `TRUE`, remove todos os anexos que tenham sido adicionados no envio anterior.

8.7 CONCLUSÃO

Mais um capítulo finalizado, e você vai se tornando um profissional melhor e mais capacitado do que já é no uso do CodeIgniter a cada página. Neste capítulo, você aprendeu detalhes sobre o envio de e-mails utilizando a library *Email*, nativa do CI.

Nos próximos capítulos, você aprenderá a trabalhar com sessões e a fazer uploads de arquivo usando o CI.

Bons estudos!

Exemplo prático

Para reforçar o que você aprendeu neste capítulo, acesse o portal **Universidade CodeIgniter** pelo link a seguir, e desenvolva o exemplo prático do tutorial.

<http://www.universidadecodeigniter.com.br/enviando-emails-com-a-library-nativa-do-codeigniter>

Links úteis

- **Documentação oficial do CI sobre a library Email:**
https://codeigniter.com/user_guide/libraries/email.html

GERENCIANDO SESSÕES COM A LIBRARY SESSION

"Qualquer colaborador vive de exemplo. Então, seja uma inspiração." — Geraldo Rufino

Trabalhar com sessões no CodeIgniter é algo fácil e de baixa complexidade, pois por meio de métodos objetivos e autodescritivos, você executa operações como armazenamento, atualização, remoção e limpeza dos dados em cache com poucas linhas de código. Ele ainda permite que você utilize para armazenamento dos dados de sessão a variável global `$_SESSION` (nativa do PHP), o Memcached, Redis, arquivos físicos e também banco de dados.

9.1 CONFIGURANDO A SESSÃO

Como o CI permite utilizar mais de um tipo de driver para armazenamento dos dados da sessão, existem parâmetros de configuração para definir qual será usado. Esses parâmetros de configuração ficam localizados em `application/config/config.php`.

Veja a seguir os parâmetros e suas respectivas funções:

`sess_driver` : driver utilizado para armazenamento da sessão.

- **Valor padrão:** files
- **Opções:** files/database/redis/memcached/custom

sess_cookie_name : nome utilizado para o cookie da sessão.

- **Valor padrão:** ci_session
- **Opções:** [A-Za-z_-]

sess_expiration : número de segundos para expiração da sessão. Para que a sessão não expire, informe zero (0) como valor.

- **Valor padrão:** 7200 (2 horas)
- **Opções:** Tempo em segundos (inteiro)

sess_save_path : caminho para armazenamento das informações, dependendo do driver usado.

- **Valor padrão:** NULL

sess_match_ip : validação ou não do IP do usuário na hora da leitura da sessão.

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

sess_time_to_update : frequência com que a sessão vai se regenerar e criar uma nova ID de sessão. Se definir o valor como 0 (zero), a regeneração será desativada.

- **Valor padrão:** 300
- **Opções:** Tempo em segundos (inteiro)

sess_regenerate_destroy : destrói a sessão mais antiga quando a sessão for regenerada.

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

9.2 CARREGANDO A LIBRARY E INICIALIZANDO A SESSÃO

Existem dois meios de se inicializar a sessão em uma aplicação com CodeIgniter. O primeiro é através do arquivo autoload, no qual você pode adicionar a library *Session* no array de libraries a ser carregado de forma automática. A segunda é usando o método `$this->load->library('session')` no *controller* de sua aplicação onde você precisará fazer uso das sessões.

Após carregar a library, você passará a utilizar `$this->session` para poder executar os métodos da library.

9.3 TRABALHANDO COM SESSÃO TEMPORÁRIA

No CI, uma sessão temporária - onde os dados serão passados do *controller* para a *view* e apagados após a sua exibição - é definida pelo método `$this->session->flashdata()` e suas variações.

Um exemplo de dado de sessão temporária é quando você recebe informações do usuário por meio de um formulário, valida e, em caso de sucesso, redireciona o usuário para outra página. Nessa outra página, você precisa exibir uma mensagem de que os dados foram processados com sucesso.

Veja a seguir cada um dos métodos para trabalhar com sessão temporária.

Armazenando dados com `set_flashdata()`

Para armazenar os dados e recuperá-los posteriormente, você utiliza o método `$this->session->set_flashdata()`, que possui dois parâmetros:

- `item` : nome do item da sessão que será usado para recuperá-la posteriormente.
- `valor` : valor do item da sessão.

```
$this->session->set_flashdata('item', 'valor');
```

O código anterior é o equivalente a `$_SESSION['item'] = valor` .

Recuperando os dados com `flashdata()`

Para recuperar os dados de uma sessão, ou de apenas um item da sessão, você utiliza o método `$this->session->flashdata()` . Ele possui um parâmetro opcional que, se especificado, retorna o valor do item da sessão que foi passado como parâmetro; caso contrário, retorna um `array` com todos os dados da sessão.

```
// retorna todos os dados da sessão
```

```
$this->session->flashdata();
```

```
// retorna somente o valor do item passado como parâmetro
```

```
$this->session->flashdata('item');
```

Se você passar como parâmetro o nome de um item errado, ou esse item não existir na sessão, o método retornará `NULL` .

Mantendo os dados de uma sessão temporária por um período maior de tempo

É possível manter os dados de uma sessão temporária por um período maior de tempo. Para isso, você usará o método `$this->session->keep_flashdata()` .

O método recebe como parâmetro o nome do item que você deseja que seja mantido por mais tempo, ou então um `array` com os itens que deverão ter o tempo de permanência na sessão ampliados.

```
$this->session->keep_flashdata('item');
```

```
$this->session->keep_flashdata(array('item', 'item2', 'item3'));
```

Determinando o tempo em que os dados da sessão estarão disponíveis

Por meio do método `$this->session->mark_as_temp()`, você pode determinar por quanto tempo os dados de uma sessão estarão disponíveis. Esse método recebe dois parâmetros:

- `item` : nome do item da sessão a ser mantido.
- `tempo` : tempo (em segundos) em que o item da sessão será mantido.

```
//mantém o item por 300 segundos (5 minutos)
```

```
$this->session->mark_as_temp('item', 300);
```

```
//mantém os itens por 300 segundos (5 minutos)
```

```
$this->session->mark_as_temp(array('item', 'item2'), 300);
```

Se você precisa que alguns itens tenham tempos de permanência diferentes, você pode passar um único parâmetro para o método, sendo esse parâmetro um array que contenha o item e o tempo que ele deve ser mantido.

```
// mantém item por 300 segundos e item 2 por 240 segundos
```

```
$this->session->mark_as_temp(array(  
    'item' => 300,  
    'item2' => 240  
));
```

Você ainda tem uma outra possibilidade, que é adicionar um novo item à sessão e passar o tempo de expiração dele pelo método `$this->session->set_tempdata()`.

```
$this->session->set_tempdata('item', 'valor', 300);
```

Caso você não informe o tempo para expiração do item da sessão, será utilizado o valor padrão, que é de 300 segundos.

Para listar os dados de uma sessão temporária, você pode

utilizar o método `$this->session->tempdata()` da mesma forma como usa o `$this->session->flashdata()`, com ou sem o parâmetro.

```
$this->session->tempdata('item');  
$this->session->tempdata();
```

Se quiser remover um item de uma sessão temporária antes do prazo de expiração, você pode usar o método `$this->session->unset_tempdata()`, informando como parâmetro o nome do item que deseja remover.

```
$this->session->unset_tempdata('item');
```

Limpando ou destruindo a sessão

Para limpar ou destruir uma sessão, em que todos os itens serão removidos, você usará o método `$this->session->sess_destroy()`.

9.4 TRABALHANDO COM SESSÃO PERMANENTE

As sessões permanentes no CodeIgniter são sessões utilizadas para armazenar dados que precisam ser recuperados em diversos momentos da aplicação. Um exemplo de uso de sessão permanente é no caso de login, no qual o usuário se loga e você pode armazenar um token para identificar qual usuário está logado, sem expor à sessão os seus dados, como e-mail e nome.

Na sessão permanente, você pode adicionar, alterar, remover e recuperar os dados da sessão, assim como na sessão temporária. Veja a seguir como funcionam essas operações.

Adicionando e atualizando dados

Para adicionar dados ou atualizar os dados de uma sessão permanente, você deve usar o método `$this->session->set_userdata()` , passando para ele dois parâmetros:

- `item` : nome do item a ser adicionado na sessão.
- `valor` : valor do item.

```
$this->session->set_userdata('item','valor');
```

Removendo dados

Para remover dados da sessão permanente, você deve utilizar o método `this->session->unset_userdata()` , passando para ele o parâmetro que define o item a ser removido.

```
$this->session->unset_userdata('item');
```

Recuperando os dados

Para recuperar os dados, você usa o método `$this->session->userdata()` . Se não passar nenhum parâmetro, ele retorna todos os dados da sessão permanente, e se quiser recuperar os dados de um único item da sessão, basta passar como parâmetro o nome desse item.

```
//retorna um array com todos os dados da sessão permanente
```

```
$this->session->userdata();
```

```
//retorna os dados do item passado como parâmetro
```

```
$this->session->userdata('item');
```

Verificando a existência de um item na sessão

Para verificar se determinado item existe na sessão, você deve utilizar o método `$this->session->has_userdata()` , passando como parâmetro o nome desse item. O método retornará `TRUE` ou `FALSE` .

```
$this->session->has_userdata('item');
```

9.5 ARMAZENANDO SESSÕES NO BANCO DE DADOS

Uma das possibilidades da library é armazenar a sessão no banco de dados, e a configuração para isso é bem simples. Primeiro, abra o arquivo `application/config/config.php` e defina o parâmetro `sess_driver` como `database`. Feito isso, você precisará criar a tabela para armazenar os dados da sessão.

Considerando que você está usando um banco de dados MySQL, precisará executar a seguinte instrução:

```
CREATE TABLE IF NOT EXISTS `ci_sessions` (  
  `id` varchar(40) NOT NULL,  
  `ip_address` varchar(45) NOT NULL,  
  `timestamp` int(10) unsigned DEFAULT 0 NOT NULL,  
  `data` blob NOT NULL,  
  KEY `ci_sessions_timestamp` (`timestamp`)  
);
```

Dependendo da configuração do parâmetro `sess_match_ip`, explicado anteriormente neste capítulo, você precisará criar uma chave primária. Para isso, execute as instruções a seguir conforme a configuração aplicada para `sess_match_ip`:

```
#Quando sess_match_ip = TRUE  
ALTER TABLE ci_sessions ADD PRIMARY KEY (id, ip_address);  
  
#Quando sess_match_ip = FALSE  
ALTER TABLE ci_sessions ADD PRIMARY KEY (id);  
  
#Remove a primary key criada anteriormente (execute quando sess_ma  
tch_ip mudar de valor)  
ALTER TABLE ci_sessions DROP PRIMARY KEY;
```

9.6 ARMAZENANDO SESSÕES EM ARQUIVOS FÍSICOS

Armazenar os dados de sessão em arquivos físicos também é

uma possibilidade. Para configurá-la, altere o parâmetro de configuração `sess_driver` em `application/config/config.php` para `files`. E no parâmetro `sess_save_path`, você deverá informar o caminho do diretório onde os arquivos serão gravados.

É importante lembrar de que esse diretório deve ter permissão `0700`, para que o CI possa criar os arquivos dentro dele e fazer a leitura, bloqueando essas ações para usuários diferentes do proprietário do diretório.

9.7 ARMAZENANDO SESSÕES COM REDIS

O Redis é um mecanismo de armazenamento normalmente utilizado para cache, e possui um alto desempenho. Uma desvantagem dele é que necessita da extensão `phpredis` instalada no servidor (ela não vem por padrão na instalação do PHP).

Veja no *Apêndice B* como instalar o Redis.

A configuração aplicada para uso do Redis é semelhante à configuração aplicada para o uso de arquivos físicos.

```
$config['sess_driver'] = 'redis';  
$config['sess_save_path'] = 'tcp://localhost:6379';
```

Para `sess_driver`, você informa `redis` e para `sess_save_path`, você informa o endereço para armazenamento dos dados

9.8 ARMAZENANDO SESSÕES COM MEMCACHED

Assim como o Redis, o Memcached é muito popular por seu desempenho. Ele é uma extensão do PHP distribuído por meio de PECL, e algumas distribuições Linux já disponibilizam como um pacote de fácil instalação.

A sua configuração é como a do Redis, na qual você informa o driver e o caminho para salvar os dados.

```
$config['sess_driver'] = 'memcached';  
$config['sess_save_path'] = 'localhost:11211';
```

IMPORTANTE

Uma garantia do Memcached é que, se você determinar um tempo para expiração de um valor na sessão, ele vai respeitar. São poucos casos em que ele ignora essa informação e expira os dados fora do tempo determinado. Mesmo acontecendo poucas vezes, é um fator a ser considerado, pois pode causar a perda de sessão e comprometer o funcionamento da aplicação.

9.9 CONCLUSÃO

Neste capítulo, você aprendeu a configurar e trabalhar com sessões no CodeIgniter, ampliando os conhecimentos e se preparando para o desenvolvimento do próximo projeto, no qual usaremos bastante.

No próximo capítulo, você aprenderá a trabalhar com upload e compressão de arquivos no CI, uma possibilidade muito interessante.

Exemplo prático

Para reforçar o que você aprendeu neste capítulo, acesse o portal **Universidade CodeIgniter** pelo link a seguir, e desenvolva o exemplo prático do tutorial.

<http://www.universidadecodeigniter.com.br/trabalhando-com-sessoes>

Links úteis

- **Documentação oficial do CI sobre a Library Session:**
https://codeigniter.com/user_guide/libraries/sessions.html
- **Dica de livro sobre Redis:**
<https://www.casadocodigo.com.br/products/livro-redis>

UPLOAD, DOWNLOAD E COMPRESSÃO DE ARQUIVOS

"Você não precisa fazer anotações, se for importante você se lembrará." — Steve Jobs

10.1 UPLOAD

O upload de arquivos no CodeIgniter é feito utilizando a library *File Upload*. Para realizar esse processo com sucesso, o seguinte fluxo deve ser aplicado:

1. Ter um formulário com `enctype="multipart/form-data"` para seleção do arquivo;
2. Ao submeter o formulário com o arquivo selecionado, o upload será executado do lado do servidor para o diretório especificado nas configurações do upload;
3. No processo de upload, a validação para saber se o upload foi executado ou não deve ser feita;
4. Por fim, devemos retornar para o usuário o feedback da operação, para que ele saiba se o upload foi executado ou não.

Inicializando a library

Antes de começar a utilizar os recursos de upload de arquivo do

CI, é preciso inicializar a library. Isso é feito usando o método `$this->load->library()` , conforme código:

```
$this->load->library('upload');
```

Configurando o upload

Para o upload ocorrer, não basta apenas inicializar a library e usar o método `$this->upload->do_upload()` . Existem alguns parâmetros que precisam ser informados para que a library saiba o que fazer.

Veja a seguir os parâmetros de configuração disponíveis para uso na library:

upload_path — O path para o diretório onde o arquivo será armazenado. Esse path pode ser relativo ou absoluto e precisa estar com permissão de escrita.

allowed_types — Os mime types de arquivos que poderão ser enviados. Pode utilizar a própria extensão do arquivo e, no caso de múltiplas extensões, separe-as com um traço vertical.

file_name — Nome para o arquivo | . Se informado, o arquivo será renomeado. Se informar o nome com extensão, o CI vai substituir a extensão original do arquivo; caso contrário, manterá a extensão original.

file_ext_tolower — Se `TRUE` , a extensão é convertida para letras minúsculas.

- **Valor padrão:** `FALSE`
- **Opções:** `TRUE` ou `FALSE`

overwrite — Se `TRUE` , o arquivo de mesmo nome no diretório será sobrescrito. Se `FALSE` , será adicionado um número ao arquivo que está sendo salvo.

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

`max_size` — Tamanho máximo do arquivo (em kb) que o arquivo pode ter. Para não limitar, basta passar o valor como zero (0).

- **Valor padrão:** 0

`max_width` — Largura máxima (em pixels) que a imagem pode ter. Para não limitar, basta passar o valor como zero (0).

- **Valor padrão:** 0

`max_height` — Altura máxima (em pixels) que a imagem pode ter. Para não limitar, basta passar o valor como zero (0).

- **Valor padrão:**
- **Opções:**

`min_width` — Largura mínima (em pixels) que a imagem pode ter. Para não limitar, basta passar o valor como zero (0).

- **Valor padrão:** 0

`min_height` — Altura mínima (em pixels) que a imagem pode ter. Para não limitar, basta passar o valor como zero (0).

- **Valor padrão:** 0

`max_filename` — Tamanho máximo que o nome do arquivo pode ter. Para não limitar, basta passar o valor como zero (0).

- **Valor padrão:** 0

`max_filename_increment` — Se `overwrite` estiver definido como `FALSE`, use esse parâmetro para determinar o máximo de

incremento (número) a ser adicionado ao nome do arquivo.

- **Valor padrão:** 100

`*encrypt_name` — Se `TRUE` , o nome do arquivo será convertido para uma string randômica encriptada.

- **Valor padrão:** `FALSE`
- **Opções:** `TRUE` ou `FALSE`

`remove_spaces` — Se `TRUE` , os espaços no nome do arquivo serão convertidos em underscore.

- **Valor padrão:** `TRUE`
- **Opções:** `TRUE` ou `FALSE`

`detect_mime` — Se `TRUE` , será feita a detecção do mime type do lado do servidor, evitando assim injeção de código malicioso. Não desative essa opção, pois pode afetar a segurança da aplicação.

- **Valor padrão:** `TRUE`
- **Opções:** `TRUE` ou `FALSE`

`mod_mime_fix` — Se `TRUE` , as múltiplas extensões no nome do arquivo serão seguidas de underscore. Não desative essa opção se o diretório de destino do arquivo for público, pois pode oferecer risco à segurança.

- **Valor padrão:** `TRUE`
- **Opções:** `TRUE` ou `FALSE`

Desses parâmetros listados, os mais utilizados são: `upload_path` , `allowed_types` , `max_size` , `max_width` e `max_height` .

```
$config['upload_path'] = './uploads/';  
$config['allowed_types'] = 'gif|jpg|png';  
$config['max_size'] = 100;
```

```
$config['max_width'] = 1024;  
$config['max_height'] = 768;
```

Existem duas formas de aplicar as configurações para a library. Se você está carregando a library pelo autoload, então você deve utilizar o método `$this->upload->initialize()` , passando como parâmetro para o método o array com as configurações:

```
$config['upload_path'] = './uploads/';  
$config['allowed_types'] = 'gif|jpg|png';  
$config['max_size'] = 100;  
$config['max_width'] = 1024;  
$config['max_height'] = 768;  
  
$this->upload->initialize($config);
```

Se não carregar a library pelo autoload, então você pode passar essas configurações no ato da carga da library:

```
$config['upload_path'] = './uploads/';  
$config['allowed_types'] = 'gif|jpg|png';  
$config['max_size'] = 100;  
$config['max_width'] = 1024;  
$config['max_height'] = 768;  
  
$this->load->library('upload',$config);
```

Se em algum momento você precisar realizar um novo upload que vá utilizar configurações diferentes, você pode chamar novamente o método `$this->upload->initialize()` , passando um novo array com as configurações desejadas.

Processando o upload

O processamento do upload é todo feito através do método `$this->upload->do_upload()` , e você só tem de validar o processo. O método `$this->upload->do_upload()` recebe como parâmetro o nome do campo `file` usado no formulário. Esse método retorna `TRUE` ou `FALSE` , para determinar se o upload foi ou não executado.

```
$this->upload->do_upload('input_file_name');
```

Só o fato de chamar o método conforme foi feito já abre o processamento do upload, mas você não consegue validar o processo. Veja a seguir um processo de validação usando `if`.

```
if($this->upload->do_upload('input_file_name')){
    $data = array('upload_data' => $this->upload->data());
    $this->load->view('upload_success', $data);
}else{
    $error = array('error' => $this->upload->display_errors());
    $this->load->view('upload_form', $error);
}
```

Nesse código, se o upload foi executado com sucesso, retornamos para a *view* os dados da imagem que foi enviada para o servidor pelo método `$this->upload->data()`. Se não foi feito o upload, então usamos o método `$this->upload->display_errors()` para poder recuperar os erros ocorridos e passar para a *view*.

O método `$this->upload->data()` retorna um array com várias informações sobre o arquivo que foi enviado para o servidor. Veja na tabela a seguir quais são essas informações:

Item	Descrição
file_name	Nome do arquivo que foi "upado", incluindo a extensão
file_type	MIME type do arquivo
file_path	Path do arquivo no servidor
full_path	Path do arquivo no servidor, incluindo o nome do arquivo e a extensão
raw_name	Nome do arquivo, sem extensão
orig_name	Nome original do arquivo. Só é utilizado caso a encriptação do nome do arquivo tenha sido ativada nas configurações
client_name	Nome do arquivo fornecido pelo "user agent" antes de qualquer modificação
file_ext	Extensão do arquivo, incluindo o ponto (.)
file_size	Tamanho do arquivo em kilobytes

is_image	Informa se o arquivo é ou não uma imagem (1 = sim, 0 = não)
image_width	Largura da imagem
image_height	Altura da imagem
image_type	Tipo do arquivo (a extensão sem o ponto)
image_size_str	String contendo largura e altura, geralmente usada para adicionar a tag <code>img</code>

O método `$this->upload->display_errors()` pode receber dois parâmetros, que identificam as tags HTML que envolverão os erros. Por exemplo, se for utilizado `$this->upload->display_errors('<p>', '</p>')`, os erros serão retornados dentro da tag `<p>...</p>`.

10.2 DOWNLOAD

O download de arquivos usando o CodeIgniter pode ser feito utilizando o helper *Download*. O carregamento do helper pode ser feito no autoload, ou então utilizando `$this->load->helper('download')`.

Para fazer o download do arquivo, você usará o método `force_download()`. O método recebe 3 parâmetros. O primeiro é o nome do arquivo que você deseja fazer o download, e o segundo, que é opcional, são os dados do arquivo.

Se você informar no primeiro parâmetro o *path* de um arquivo existente, como no exemplo a seguir, será feito o download desse arquivo.

```
force_download('path/do/arquivo.jpg');
```

Se você informar o nome de um arquivo e o conteúdo no segundo parâmetro, será feito o download desse arquivo com o conteúdo passado no segundo parâmetro.

```
$data = 'Conteúdo do arquivo';  
$name = 'arquivo.txt';  
force_download($name, $data);
```

- `$data` : é o texto do arquivo, que será inserido nele antes de liberar o download para o usuário.
- `$name` : é o nome do arquivo que eu quero que o usuário faça download (mas esse arquivo não existe, será criado em tempo de execução).

Para o terceiro parâmetro, que é o `mimetype` do arquivo, se você passar `TRUE`, ele usará o `mimetype` baseado na extensão do arquivo. Então, forçará o navegador a abrir primeiro o arquivo, caso ele seja capaz de interpretar esse tipo de arquivo que estiver passando. Por exemplo, se você está forçando o download de um arquivo PDF (`.pdf`), e passa o terceiro parâmetro como `TRUE`, em vez de abrir a caixa de diálogo de download, o browser vai renderizar o arquivo na tela.

```
force_download( './arquivos/arquivo.pdf', null, TRUE);
```

Nada mais do que isso é necessário para efetuar o download de arquivos no CI.

O método `force_download()` é muito útil para que você não precise disponibilizar o caminho real do arquivo. Você pode usar uma rota dinâmica que chama um *controller*, e nesse *controller* você recupera as informações e pega o *path* real do arquivo, chamando o método `force_download()` com esse *path* como parâmetro para executar o download.

10.3 COMPRESSÃO DE ARQUIVOS

A compressão de arquivos no CI é feita pela library `ZIP Encoding`. Para carregá-la, basta chamar `$this->load->library('zip')`, ou então carregá-la diretamente no autoloader.

Criando e compactando o arquivo

Para criar e compactar o arquivo, podemos utilizar o método `$this->zip->add_data()` , que recebe como parâmetros o nome e o conteúdo do arquivo. Veja no exemplo a seguir, em que criamos um arquivo TXT em tempo de execução, adicionando-o para o ZIP, e depois salvamos no servidor e acionamos o download do ZIP.

```
$name = 'meus_dados.txt';
$data = 'Esses são meus dados!';

$this->zip->add_data($name, $data);

// Cria o arquivo ZIP no servidor com o nome "meus_dados.zip"
$this->zip->archive('/path/meus_dados.zip');

// Libera o download dando outro nome para o arquivo (meus_dados_download.zip)
$this->zip->download('meus_dados_download.zip');
```

Lendo um arquivo e liberando download do arquivo compactado

Por meio do método `$this->zip->read_file()` , podemos ler um arquivo e adicioná-lo para um arquivo ZIP.

```
$path = '/path/photo.jpg';

$this->zip->read_file($path);

// Libera o download do arquivo com o nome de "minha_foto.zip"
$this->zip->download('minha_foto.zip');
```

O `$this->zip->read_file($path)` recebe como parâmetro o caminho do arquivo a ser compactado. Se um segundo parâmetro for passado e seu valor for `TRUE` , então ao compactar o arquivo, será mantida a estrutura de diretórios. Por exemplo:

```
$path = '/Dados/Fotos/profile.jpg';

$this->zip->read_file($path, TRUE);
```

```
// Libera o download do arquivo com o nome de "minha_foto.zip"
$this->zip->download('minha_foto.zip');
```

Ao descompactar o arquivo `minha_foto.zip`, em vez de ter somente o arquivo JPG, você terá a estrutura completa de diretório, conforme o *path* passado como parâmetro (que no caso desse código será `Dados/Fotos/profile.jpg`).

Existe ainda a possibilidade de compactar todo o *path* no ZIP, mas alterando a estrutura de diretórios. Para isso, basta enviar como segundo parâmetro o *path* que deseja usar na compactação.

```
$path = '/Dados/Fotos/profile.jpg';
$novopath = '/Fotos/Profile/social.jpg';

$this->zip->read_file($path, $novopath);

// Libera o download do arquivo com o nome de "minha_foto.zip"
$this->zip->download('minha_foto.zip');
```

Ao descompactar o arquivo, a foto estaria em `Fotos/Profile`, e não em `Dados/Fotos`, que é o *path* original da imagem.

Lendo um diretório e liberando download do arquivo compactado

Assim como podemos fazer com um arquivo específico, também podemos fazer com um diretório e compactar todos os seus arquivos de uma única vez. Para fazer isso, usamos o método `$this->zip->read_dir()`. Esse método pode receber até três parâmetros, sendo somente o primeiro obrigatório.

```
$path = '/path/do/diretorio/';

$this->zip->read_dir($path);

// Libera o download do arquivo com o nome de "backup.zip"
$this->zip->download('backup.zip');
```

No exemplo, `$this->zip->read_dir()` recebe como parâmetro o caminho do diretório a ser compactado. Todo o

conteúdo do diretório, seja ele outros arquivos e/ou diretórios, será compactado em um único arquivo ZIP, que vai ser chamado de `backup.zip` ao fazer o download com o método `$this->zip->download('backup.zip')` .

Por padrão, o método `$this->zip->read_dir()` adiciona no ZIP toda a estrutura de diretório do caminho passado. Mas caso você queira somente os arquivos dentro do ZIP, basta passar um segundo parâmetro, com o valor `FALSE` . Assim, ao compactar os arquivos, a árvore de diretórios será ignorada e somente os arquivos do último diretório passado no path serão compactados.

```
$path = '/path/do/diretorio/';

$this->zip->read_dir($path, FALSE);

// Libera o download do arquivo com o nome de "backup.zip"
$this->zip->download('backup.zip');
```

10.4 CONCLUSÃO

No decorrer deste capítulo, você aprendeu a utilizar os recursos de upload, download e compressão de arquivo. Todos esses recursos são muito úteis, pois vários tipos de sistema precisam enviar arquivos para o servidor, e disponibilizar download de arquivos para os usuários. E a compactação, associada ao download, torna o processo melhor.

Imagine um sistema no qual o volume de informações é grande e você precisa de uma opção de backup manual. Se juntarmos o que foi aprendido aqui com o que será aprendido no capítulo 14. *Trabalhando com banco de dados*, você poderá fazer um sistema de backup manual muito funcional.

Exemplo prático

Para reforçar o que você aprendeu neste capítulo, acesse o portal **Universidade CodeIgniter** pelos links a seguir, e desenvolva os exemplos práticos dos tutoriais.

- <http://www.universidadecodeigniter.com.br/upload-e-download-de-arquivos>
- <http://www.universidadecodeigniter.com.br/compressao-de-arquivos>

Links úteis

- **Documentação oficial do CI sobre a library File Uploading:**
https://codeigniter.com/user_guide/libraries/file_uploading.html
- **Documentação oficial do CI sobre o helper Download:**
https://codeigniter.com/user_guide/helpers/download_helper.html
- **Documentação oficial do CI sobre a library ZIP Encoding:**
https://codeigniter.com/user_guide/libraries/zip.html

IMPLEMENTANDO CAPTCHA NATIVO

"Conhecimento serve para encantar as pessoas, não para humilhá-las." — Mario Sérgio Cortella

"CAPTCHA é um acrônimo da expressão 'Completely Automated Public Turing test to tell Computers and Humans Apart' (Teste de Turing público completamente automatizado para diferenciação entre computadores e humanos): um teste de desafio cognitivo, utilizado como ferramenta antispam, desenvolvido de forma pioneira na universidade de Carnegie-Mellon. [...] Um tipo comum de CAPTCHA requer que o usuário identifique as letras de uma imagem distorcida, às vezes com a adição de uma sequência obscurecida das letras ou dos dígitos que apareça na tela" (Wikipédia - <https://pt.wikipedia.org/wiki/CAPTCHA>).

O CodeIgniter possui um helper próprio para implementação do CAPTCHA, permitindo que, com poucas linhas de código, esse recurso seja implementado de forma fácil. Neste capítulo, vamos implementar o CAPTCHA no formulário de contato do site institucional, que foi criado nos capítulos *Criando um site institucional — Parte I e II*.

Para iniciarmos o estudo do capítulo, você pode duplicar os arquivos do site institucional em um outro diretório, ou então implementar diretamente no diretório do projeto do site institucional.

O projeto original possuía apenas os formulários, sem nenhuma forma de evitar spam ou preenchimento e envio através de robôs, o que é muito comum na internet hoje. Esse capítulo vai servir para implementar um mecanismo de segurança, para ajudar a evitar spam e preenchimento por robôs.

11.1 CARREGANDO O HELPER

A primeira coisa a se fazer é carregar o helper *Captcha*. Como esse recurso será usado somente em formulários, não faremos sua carga no arquivo `application/config/autoload.php` , mas dentro do *controller* Contato .

O helper `captcha` possui as funções necessárias para a geração das imagens com uma sequência de caracteres que deverá ser digitada pelo usuário e validada no servidor no momento em que o formulário for submetido. Sem esse helper, não é possível fazer uso da funcionalidade nativa do CI.

Abra o arquivo `application/controllers/Contato.php` e, no método `__construct` , adicione a linha a seguir, antes da chave `}` que fecha o método:

```
$this->load->helper('captcha');
```

O método `__construct` ficará da seguinte forma:

```
function __construct()
{
    parent::__construct();
    $this->load->library(array('form_validation', 'session'));
    $this->load->helper('captcha');
}
```

Pronto, o helper já está sendo carregado e as funcionalidades dele estão disponíveis em qualquer método do *controller* Contato .

11.2 GERANDO O CAPTCHA

Para que o código fique mais organizado, vamos criar um método que vai gerar o CAPTCHA sempre que necessário. Assim, não será necessário ficar redefinindo configurações, basta chamar o método e este retornará o CAPTCHA criado.

Para que o CAPTCHA funcione corretamente, é necessário que a biblioteca GD do PHP esteja instalada. Veja o *Apêndice C* no final do livro para saber como verificar se a biblioteca GD está instalada e/ou como fazer a instalação.

Dentro do *controller* Contato , crie um novo método, chamado GenCaptcha() , que será o responsável pelo processo de geração da imagem.

```
private function GenCaptcha(){
    $vals = array(
        'img_path' => './captcha/',
        'img_url'  => base_url('captcha')
    );

    $cap = create_captcha($vals);
}
```

O caminho informado em `img_path` deve ser de um diretório existente. No caso desse exemplo, você deverá criar o diretório 'captcha' na raiz da aplicação, com permissão de leitura e escrita.

O código anterior possui inicialmente duas variáveis. A primeira é `$vals`, que recebe um `array` com as configurações para a geração do CAPTCHA. Dois valores foram passados no `array`, `img_path` e `img_url`, que são obrigatórios, os demais são opcionais e podem ser vistos a seguir:

`word` — Palavra que será exibida na imagem. Se uma palavra não for fornecida, a função gerará uma sequência aleatória de ASCII.

`img_path` — Caminho para o diretório onde são armazenadas as imagens geradas (parâmetro obrigatório).

`img_url` — URL para o diretório onde são armazenadas as imagens geradas (parâmetro obrigatório).

`font_path` — Caminho para o diretório onde a fonte a ser utilizada está armazenada.

`img_width` — Largura da imagem.

`img_height` — Altura da imagem gerada.

`expiration` — Quanto tempo uma imagem permanecerá no diretório `captcha` antes de ser excluída.

- **Valor padrão:** 7200

`word_length` — Quantidade de caracteres a serem exibidos na

imagem.

- **Valor padrão:** 8

`font_size` — Tamanho da fonte dos caracteres da imagem.

- **Valor padrão:** 16

`img_id` — ID da imagem.

- **Valor padrão:** Automático

`pool` — Cadeia de caracteres utilizados para geração da imagem.

- **Valor** **padrão:**
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHI
JKLMNOPQRSTUVWXYZ

`colors` — Array com as cores de background, border, text e grid da imagem. Essas cores devem ser passadas em formato RGB.

O diretório `captcha` passado como valor do parâmetro `img_path` precisa ter permissão de escrita; caso contrário, ocorrerá um erro durante a criação da imagem. Crie esse diretório na raiz do projeto.

A segunda variável é `$cap`, que recebe um array com as informações do CAPTCHA gerado através da função `create_captcha($vals)` do helper `Captcha`. Esse array contém as seguintes informações:

- `word` : palavra gerada na imagem;
- `time` : data e hora atual;
- `image` : tag `` para exibição do CAPTCHA.

11.3 ADICIONANDO O VALOR DO CAPTCHA À SESSÃO

Para que possamos comparar o texto digitado pelo usuário com o texto da imagem, é necessário que tenhamos o texto da imagem armazenado em algum lugar. Nesse caso, vamos utilizar a sessão para armazená-lo. Não precisaremos carregar a library *Session*, pois essa já é carregada no método `__construct` do *controller* *Contato*.

Vá até o método `GenCaptcha()` que você acabou de criar, e logo após `$cap = create_captcha($vals)`, adicione as linhas a seguir para que o valor do CAPTCHA seja armazenado em uma sessão. Chamaremos essa sessão de `user_captcha_value`.

```
$this->session->set_userdata('user_captcha_value', $cap['word']);
```

O método `GenCaptcha()` ficará dessa forma:

```
private function GenCaptcha(){
    $vals = array(
        'img_path' => './captcha/',
        'img_url'  => base_url('captcha')
    );

    $cap = create_captcha($vals);
    $this->session->set_userdata('user_captcha_value', $cap['word'])
};
}
```

11.4 EXIBINDO A IMAGEM NO FORMULÁRIO

A imagem do CAPTCHA já está sendo gerada, agora vamos exibi-la no formulário. Para que isso ocorra, ainda no método `GenCaptcha()`, adicione `return $cap['image'];` logo após passar o valor da imagem para a sessão. Assim, quando for executado o método, ele retornará a tag `` para exibição do CAPTCHA.

```

private function GenCaptcha(){
    $vals = array(
        'img_path' => './captcha/',
        'img_url'  => base_url('captcha')
    );

    $scap = create_captcha($vals);
    $this->session->set_userdata('user_captcha_value', $scap['word'])
);
    return $scap['image'];
}

```

Agora vá até o método `FaleConosco()` no *controller* `Contato` e adicione ao array `$data` a chave `captcha_image`, passando como valor o método `GenCaptcha()`. Assim, será possível recuperar essa imagem na *view*.

```

public function FaleConosco()
{
    $data['title'] = "LCI | Fale Conosco";
    $data['description'] = "Exercício de exemplo do capítulo 2 do
livro Codeigniter Da teoria à prática";

    $this->form_validation->set_rules('nome', 'Nome', 'trim|required|
min_length[3]');
    $this->form_validation->set_rules('email', 'Email', 'trim|required|
valid_email');
    $this->form_validation->set_rules('assunto', 'Assunto', 'trim|
required|min_length[5]');
    $this->form_validation->set_rules('mensagem', 'Mensagem', 'trim|
required|min_length[30]');

    if($this->form_validation->run() == FALSE){
        $data['formErrors'] = validation_errors();
    }else{
        $formData = $this->input->post();
        $emailStatus = $this->SendEmailToAdmin($formData['email'], $f
ormData['nome'], "to@domain.com", "To Name", $formData['assunto'], $
formData['mensagem'], $formData['email'], $formData['nome']);

        if($emailStatus){
            $this->session->set_flashdata('success_msg', 'Contato rece
bido com sucesso!');
        }else{
            $data['formErrors'] = "Desculpe! Não foi possível enviar o
seu contato. tente novamente mais tarde.";
        }
    }
}

```

```

    }

    $data['captcha_image'] = $this->GenCaptcha();
    $this->load->view('fale-conosco', $data);
}

```

Repare que a nova chave do array `$data` está no final do método, logo antes da renderização da `view`. Foi feito dessa forma para que o valor do CAPTCHA não seja alterado antes da validação do formulário.

Agora vamos editar `view` para poder exibir a imagem. Abra o arquivo `application/views/fale-conosco.php` e adicione o código abaixo antes do bloco `form-group` que contém o botão **Enviar**, assim será exibida imagem com o texto e o campo para o usuário digitar o texto da imagem.

```

<div class="form-group">
    <label class="col-md-2 control-label" for="assunto">Captcha</label>
    <div class="col-md-4">
        <input id="captcha" name="captcha" placeholder="Captcha" class="form-control input-md" required="" type="text" value="">
    </div>
    <div class="col-md-4">
        <?=$captcha_image?>
    </div>
</div>

```

11.5 VALIDANDO O CAPTCHA

Para validar o CAPTCHA e finalizar o processo de implementação, vamos usar uma função de callback, que possibilitará criar uma rotina específica de validação do CAPTCHA. Também inseriremos as regras de validação (`set_rules`) para o campo de confirmação do código, conforme ensinado no capítulo 7. *Validando formulários*, quando apresentei com mais detalhes o processo de validação de formulário.

Tem dúvidas sobre validação de formulários? Que tal refrescar sua memória e revisitar o capítulo *Validando formulários* para revisar o assunto?

Ainda no *controller* Contato , vá até o final do arquivo e crie um método chamado `captcha_check()` que recebe uma `string` como parâmetro. Essa `string` é o texto digitado pelo usuário no campo a ser validado, e que é comparada com o texto armazenado na sessão, que corresponde ao texto da imagem. Se o texto digitado for correspondente ao texto da sessão, então é retornado `TRUE` ; caso contrário, é retornado `FALSE` e uma mensagem personalizada é criada.

```
public function captcha_check($str)
{
    if ($str === $this->session->userdata('user_captcha_value'))
    {
        return TRUE;
    }
    else
    {
        $this->form_validation->set_message('captcha_check', 'O texto informado está incorreto.');
```

```
        return FALSE;
    }
}
```

Para executar essa função, é necessário adicionar a validação do campo `captcha` ao formulário. Então, vamos voltar ao método `FaleConosco()` e adicionar mais uma linha de validação do formulário, na qual informamos a função de `callback` que acabou de ser criada. Antes, eram 4 linhas de validação para os 4 campos do formulário, agora são 5 linhas, pois foi adicionado o campo `captcha` .

```
public function FaleConosco()
```

```

{
    $data['title'] = "LCI | Fale Conosco";
    $data['description'] = "Exercício de exemplo do capítulo 2 do
livro Codeigniter Da teoria à prática";

    $this->form_validation->set_rules('nome', 'Nome', 'trim|required|
min_length[3]');
    $this->form_validation->set_rules('email', 'Email', 'trim|required|
valid_email');
    $this->form_validation->set_rules('assunto', 'Assunto', 'trim|
required|min_length[5]');
    $this->form_validation->set_rules('mensagem', 'Mensagem', 'trim|
required|min_length[30]');
    $this->form_validation->set_rules('captcha', 'Captcha', 'trim|
required|callback_captcha_check');

    ...
}

```

Sempre que for utilizar uma função de callback, é necessário usar o prefixo `callback_` antes do nome da função, na hora de passá-la como parâmetro em `$this->form_validation->set_rules()` ;.

11.6 CONCLUSÃO

Neste capítulo, você viu como utilizar o captcha nativo do CI, de forma a diminuir o volume de spam enviado através dos formulários de contato. Além do CAPTCHA nativo, existem muitas outras possibilidades de CAPTCHA que você pode implementar. Dê uma pesquisada sobre eles, principalmente sobre reCAPTCHA, vai ser muito útil.

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-11-implementando-captcha-nativo-do-ci>

Links úteis

- **Documentação oficial sobre o helper Captcha:**
https://codeigniter.com/user_guide/helpers/captcha_helper.html

CRIANDO UM ENCURTADOR DE URLS — PARTE I

"Se os seus sonhos não te assustam, eles não são grandes o suficiente." — Ellen Johnson Sirleaf

Parabéns por ter chegado até aqui! Isso é sinal de que você está gostando do livro e quer aumentar ainda mais os seus conhecimentos.

Através da construção de um encurtador de URLs, você aprenderá os seguintes recursos com o CodeIgniter que ainda não foram aplicados de forma prática nos capítulos anteriores:

- Trabalhar com banco de dados;
- Criar *models*;
- Enviar e-mails.

Neste capítulo, você criará o *model* e *controller* da aplicação, e no capítulo a seguir criará as *views* e demais funcionalidades necessárias.

12.1 SOBRE O ENCURTADOR

O encurtador começará a ser desenvolvido neste capítulo com as

funcionalidades básicas, e depois ganhará novas funcionalidades ao longo de outros capítulos, onde abordaremos novas libraries e helpers.

A composição dessa primeira etapa de construção do encurtador será a seguinte:

- Home com o campo para redução
- Tela de cadastro do usuário com nome, e-mail e senha
- Tela de login
- Área do usuário com:
 - Lista das URLs encurtadas
 - Alteração de senha

Veja um exemplo de como ficará a home da primeira parte desse exemplo:

Encurtador de URLs

Home Minhas URLs Alterar Senha Sair

ENCURTE E COMPARTILHE

URL menor, TEXTO maior.

URL

Encurtar

Exemplo do capítulo 3 do livro CodeIgniter Teoria na Prática

Figura 12.1: Home do Encurtador de URLs

12.2 CRIANDO A ESTRUTURA DO PROJETO E O BANCO DE DADOS

A estrutura inicial do projeto mais uma vez poderá ser aproveitada do exemplo do capítulo 1. *Introdução ao CodeIgniter*,

onde você aprendeu a instalar o CI. Duplica o diretório `instalacao-ci` e renomeie-o para `short-urls`.

Você também pode fazer o download pelo GitHub, no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-01-base-para-projetos>

Após estar com a estrutura de arquivos preparada, você já pode começar a trabalhar no projeto. O primeiro passo é criar o banco de dados, pois é nele que serão armazenadas as informações das URLs encurtadas e dos usuários.

Para criar o banco de dados, você pode executar o comando a seguir no PhpMyAdmin, no terminal ou no software de gerenciamento de banco de dados de sua preferência.

INFORMAÇÃO

Para os exemplos do livro, serão usados bancos de dados MySQL, mas o CI suporta outros tipos de banco de dados, como Oracle e SQL Server, por exemplo.

Primeiro crie o banco de dados chamado `short_urls`:

```
CREATE DATABASE `short_urls`;
```

Agora crie a tabela `users`:

```
CREATE TABLE `users` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `name` varchar(255) DEFAULT '',  
  `email` varchar(255) NOT NULL DEFAULT '',
```

```
`passwd` varchar(255) NOT NULL DEFAULT '',
`created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

- `id` : chave primária da tabela, com valor inserido automaticamente (`AUTO_INCREMENT`);
- `name` : nome do usuário;
- `email` : e-mail do usuário;
- `passwd` : senha do usuário (criptografada);
- `created_at` : data de criação do registro, que é inserida automaticamente pelo MySQL através de `CURRENT_TIMESTAMP` .

Em seguida, crie a tabela `urls` :

```
CREATE TABLE `urls` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `code` varchar(10) NOT NULL,
  `address` text NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `user_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `user_id` (`user_id`),
  CONSTRAINT `user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (
    `id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

- `id` : chave primária da tabela, com valor inserido automaticamente (`AUTO_INCREMENT`);
- `user_id` : chave estrangeira para o `id` do usuário;
- `code` : código único para identificar a URL curta (por exemplo, `dom.ain/as12de98as`);
- `address` : URL original;
- `created_at` : data de criação do registro, que é

inserida automaticamente pelo MySQL através de `CURRENT_TIMESTAMP` .

Configurando o banco de dados

Para configurar a conexão ao banco de dados no CI, você precisará editar o arquivo `application/configs/database.php` . Nesse arquivo, você encontrará um trecho de código que é o seguinte:

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => '',
    'password' => '',
    'database' => '',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

Veja que a variável `$db` é do tipo `array` e já possui alguns valores para a chave `default` . Você pode criar outras chaves, com outras configurações, recurso muito usado para facilitar a conexão do banco de dados nos diferentes ambientes utilizados durante o desenvolvimento de um projeto (desenvolvimento, teste e

produção).

No próprio arquivo, já existem as informações explicando o tipo de informação de cada campo. Para esse exemplo, serão modificadas apenas `hostname` , `username` , `password` e `database` , de modo que os dados sejam os da sua conexão.

Se você estiver trabalhando com mais de uma chave para o array `$db` , precisará alterar o valor da variável `$active_group` , informando o nome da chave do array que contém os dados da conexão que deseja fazer. Poderá usar algum tipo de condicional para verificar qual é o ambiente, como é feito para `db_debug` , onde o valor é um booleano baseado no teste para saber se está no ambiente de produção.

Veja um exemplo de como ficaria o código caso já fosse ser configurada uma conexão para o ambiente de produção:

```
defined('BASEPATH') OR exit('No direct script access allowed');

$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => '',
    'password' => '',
    'database' => '',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
```

```

        'save_queries' => TRUE
    );

    $db['production'] = array(
        'dsn' => '',
        'hostname' => 'localhost',
        'username' => '',
        'password' => '',
        'database' => '',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => FALSE,
        'db_debug' => (ENVIRONMENT !== 'production'),
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'failover' => array(),
        'save_queries' => TRUE
    );

```

No capítulo 14. *Trabalhando com banco de dados*, você encontra conteúdo exclusivo sobre banco de dados no CI, inclusive detalhes mais técnicos sobre o arquivo de configuração da conexão.

Com o arquivo de configuração devidamente editado, é preciso alterar o arquivo `application/config/autoload.php` para que a biblioteca de banco de dados seja carregada de forma automática pelo CodeIgniter.

```

...
$autoload['libraries'] = array('database');
...

```

Aproveite o arquivo aberto e já adicione o carregamento

automático para o helper `URL` , que vai permitir que seja feito o uso de métodos como `base_url()` , por exemplo:

```
...
$autoload['helpers'] = array('url');
...
```

12.3 PREPARANDO AS ROTAS

As rotas são importantes para a navegação. Você já aprendeu como elas funcionam e onde as configurar. Abra o arquivo `application/config/routes.php` e configure as rotas para o projeto:

```
$route['default_controller'] = "Urls";
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;

$route['login'] = "User/Login";
$route['logout'] = "User/Logout";
$route['user'] = "User/Index";
$route['user/register'] = "User/Register";
$route['minhas-urls'] = "User/Urls";
$route['alterar-senha'] = "User/UpdatePassw";
$route['(:any)'] = "Urls/Go";
```

É extremamente importante que a rota `$route['(:any)'] = "Urls/Go"` seja a última da lista. Isso porque, como ela é uma rota dinâmica e aceita qualquer tipo de caractere, caso o usuário esteja acessando `www.doma.in/login` , por exemplo, essa URL será interceptada pela rota `(:any)` . E isso gerará erro, pois a primeira parte dela não é um código de URL curta válido.

DICA

Sempre que você tiver rotas desse tipo, que podem gerar interceptação em rotas estáticas, coloque-as sempre após as rotas que podem ser interceptadas.

Veja que nesse exemplo estamos usando uma outra possibilidade de escrita de rota. Em vez de passar uma string objetiva para a rota, estamos passando o *wildcard* (:any) , que quer dizer que qualquer valor que vier na URL logo após o domínio deverá ser tratado pelo método `Go()` do *controller* `urls` . É possível usar o *wildcard* (:num) para valores numéricos e até mesmo expressões regulares, como no exemplo a seguir:

```
$route['products/([a-z]+)/(\d+)'] = '$1/id_$2';
```

WILDCARD

O wildcard é uma espécie de curinga, que utilizamos aqui para representar uma expressão regular em vez de escrever sua notação.

Esse exemplo tem uma rota com três partes, sendo a segunda e a terceira expressões regulares. A segunda recebe somente letras e a terceira somente números. Veja que a definição do *controller* e do método para a rota também é diferente, pois está usando variáveis de identificação (`$1` e `id_$2`). Essas variáveis servem para recuperar os valores na ordem em que aparecem na URL, então, `$1` vai pegar o valor da segunda parte (`([a-z]+)`) e `$2` da terceira (`(\d+)`).

12.4 CRIANDO O MODEL URLS_MODEL

Os métodos contidos nesse *model* serão responsáveis por gerenciar as informações relacionadas às URLs, como gravar no banco de dados, gerar a URL curta e retornar a URL original relacionada a uma URL curta.

Crie o *model* em `application/models/Urls_model.php` utilizando o código a seguir:

```
<?php
if(!defined('BASEPATH')) exit('No direct script access allowed');

class Urls_model extends CI_Model{

}
```

Esse código ainda não faz nada de efetivo, mas já permite que o *model* seja carregado, pois ele tem a estrutura mínima necessária para a identificação da classe.

IMPORTANTE

- Todo *model* criado no CI deve ser estendido de `CI_Model` ;
- O nome do *model* deve sempre começar com letra maiúscula, assim como o nome do arquivo;
- Não é obrigatório, mas por uma questão de padrão do CI, é utilizado o sufixo `_model` logo após o nome do `model` .

A seguir, você verá cada método desse *model*. Adicione o código do método ao código escrito anteriormente para ir complementando a classe.

Método `__construct()`

O método construtor `__construct()` é chamado cada vez que um novo objeto da classe é criado. Então, nele são colocadas instruções de inicialização que possam ser necessárias aos objetos.

```
function __construct(){
    parent::__construct();
}
```

Método `GenerateUniqueCode()`

É o método que vai gerar o código único que representará a URL original. É usado um `do{...}while(condition)` para verificar se o código alfanumérico de 8 dígitos gerado já foi utilizado ou se está disponível. Enquanto o código gerado não for um código único, não haverá retorno — essa é a função do `do{...}while(condition)`.

```
function GenerateUniqueCode(){
    do{
        $code = random_string('alnum',8);
        $this->db->from('urls')->where('code',$code);
        $num = $this->db->count_all_results();
    }while($num >= 1);

    return $code;
}
```

Método `Save()`

É o método responsável por salvar no banco de dados as informações da URL que o usuário inseriu e retornar a URL curta. Esse método recebe como parâmetro um array com os dados a serem armazenados. Ele faz uma chamada ao método `GenerateUniqueCode()` para obter o código único de 8 dígitos que representará a URL e, em seguida, é executada a instrução SQL que vai armazenar no banco de dados o código gerado por `GenerateUniqueCode()` e o endereço completo da URL.

```
function Save($data){
    $data['code'] = $this->GenerateUniqueCode();
    $this->db->insert('urls',$data);

    if($this->db->insert_id()){
        return $data['code'];
    }else{
        return false;
    }
}
```

Para salvar os registros no banco de dados, utilizamos o método `$this->db->insert(table, data)` do Query Builder do CI. Ele recebe 2 parâmetros:

- `table` : nome da tabela;
- `data` : array com os dados a serem gravados.

```
$this->db->insert('urls',$data);
```

Ainda no `Save`, foi usado `$this->db->insert_id()`, que é um método capaz de retornar o ID da última instrução `INSERT` executada. Veja que, com base no retorno desse método, sabemos se a URL foi ou não gravada no banco de dados, e assim retornamos o código da URL curta.

\$THIS->DB

Sempre que precisar utilizar um método relacionado a banco de dados no CodeIgniter, exceto para a classe `Database Utility`, você deverá usar `$this->db`, pois é a indicação da instância e da classe onde os métodos estão localizados.

Veremos outros métodos para trabalhar com banco de dados ainda neste capítulo.

Método Fetch()

É o método que retornará os dados de uma URL a partir do código passado como parâmetro, que é o código único gerado pelo método `GenerateUniqueCode()` .

```
function Fetch($url_code){
    $this->db->select('*')->from('urls')->where('code',$url_code)->limit(1);
    $result = $this->db->get()->result();

    if($result){
        return $result[0]->address;
    }else{
        return false;
    }
}
```

Para localizar os dados relacionados ao código passado, foram utilizados 3 métodos do Query Builder do CI:

- `select()` : recebeu como parâmetro o `*` (asterisco), responsável por trazer todas as colunas da tabela no retorno;
- `from()` : recebe como parâmetro o nome da tabela onde será feito o `select` ;
- `where()` : recebe dois parâmetros, sendo o primeiro o nome da coluna e o segundo o valor dessa coluna.

No capítulo 14. *Trabalhando com banco de dados*, você encontra conteúdo exclusivo sobre banco de dados no CI, inclusive detalhes mais técnicos sobre os métodos da classe `Database` .

Os métodos foram concatenados, mas poderiam ter sido

executados de forma separada:

```
$this->db->select('*');  
$this->db->from('urls');  
$this->db->where('code',$url_code);
```

DICA

Quando executar um `select` ou qualquer outra operação com banco de dados, e quiser conferir a instrução SQL executada, basta utilizar `$this->db->last_query()` . Esse método retorna a última instrução SQL executada.

```
$this->db->select('*');  
$this->db->from('urls');  
$this->db->where('code',$url_code);  
  
echo $this->db->last_query();
```

Método GetAllByUser()

É o método responsável por retornar todas as URLs encurtadas por um determinado usuário. Toda vez que um usuário logado encurta uma URL, ao gravá-la no banco de dados é inserido o ID do usuário junto aos dados da URL, e por isso é possível recuperar somente as URLs encurtadas de um determinado usuário.

```
function GetAllByUser($user_id){  
    $this->db->select('*')->from('urls')->where('user_id',$user_id);  
    $result = $this->db->get()->result();  
    if($result){  
        return $result;  
    }else{  
        return false;  
    }  
}
```

Assim como no método `Fetch()` , foi executada uma instrução `SELECT` na qual o parâmetro de pesquisa é o `user_id` :

```
$this->db->select('*')->from('urls')->where('user_id',$user_id);
```

Com o *model* criado, é hora de trabalhar nos métodos do *controller*.

12.5 CRIANDO O MODEL USER_MODEL

Os métodos contidos nesse *model* serão responsáveis por gerenciar as informações relacionadas aos usuários.

Crie o *model* em `application/models/User_model.php` :

```
<?php
if(!defined('BASEPATH')) exit('No direct script access allowed');

class User_model extends CI_Model{

}
```

A seguir, você verá cada método desse *model*. Adicione o código do método ao código escrito anteriormente para ir complementando a classe.

O primeiro método criado nesse *model* foi o `__construct()` , que já foi explicado anteriormente.

Método Save()

É o método responsável por salvar os dados do usuário no banco de dados, permitindo que esse usuário faça login posteriormente no encurtador de URL.

```
function Save($data){
    $data['passw'] = password_hash($data['passw'], PASSWORD_DEFAULT);
;
    $this->db->insert('users',$data);
    $userID = $this->db->insert_id();
    if($userID){
        return $this->GetUser($userID);
    }else{
        return false;
    }
}
```

```
}  
}
```

O método `Save()` recebe como parâmetro um array , contendo os dados que deverão ser gravados na tabela `users` . Na primeira linha é gerado o hash da senha pela função `password_hash()` , nativa do PHP. Ela recebe dois parâmetros: o primeiro é a senha informada pelo usuário e recuperada no array na chave `passw` ; o segundo é a constante do algoritmo para gerar o hash da senha passada, que pode ser `PASSWORD_DEFAULT` ou `PASSWORD_BCRYPT` .

ATENÇÃO

O uso de `PASSWORD_BCRYPT` resultará em um hash com no máximo 72 caracteres.

Após gerar o hash e atualizar a chave `passw` no array, executamos o `INSERT` através do método `$this->db->insert()` , onde passamos o nome da tabela em que o `INSERT` deverá ser executado e os dados serem inseridos. Em seguida, recuperamos o ID do registro inserido através do método `$this->db->insert_id()` , alocando-o na variável `$userID` .

Feito isso, é checado se foi retornado um ID. Caso haja um ID, então retornamos os dados do usuário relacionado ao ID, que foram recuperados através do método `$this->getUser($userID)` , que veremos a seguir. Se não foi retornado nenhum ID, então o retorno do método será `FALSE` .

Método `getUser()`

Esse método é muito semelhante ao método `Fetch()` do *model*

`urls_model` . Ele recupera os dados do usuário cujo ID foi passado como parâmetro.

```
public function GetUser($id){
    $this->db->select('*')->from('users')->where('id',$id);
    $result = $this->db->get()->result();

    if($result){
        return $result[0];
    }else{
        return false;
    }
}
```

Se existe o usuário, então os dados são retornados pelo método; caso contrário, é retornado `FALSE` .

Método Update()

É o método responsável por atualizar os dados de um usuário no banco. No caso desse exemplo, somente a senha pode ser atualizada.

```
function Update($data){
    $data['passw'] = password_hash($data['passw'], PASSWORD_DEFAULT)
    ;
    $this->db->where('id', $this->session->userdata('id'));
    $this->db->update('users', $data);
}
```

Mais uma vez é gerado o hash da senha e, em seguida, é informada a cláusula `WHERE` , que vai determinar qual o registro será usado. Nesse caso, será o registro cujo `id` for igual ao `id` do usuário logado, que é recuperado pelo método `$this->session->userdata('id')` . Em seguida, é executado o `UPDATE` .

NOTA

No capítulo 14. *Trabalhando com banco de dados*, você encontra conteúdo exclusivo sobre banco de dados no CI, inclusive detalhes mais técnicos sobre o arquivo de configuração da conexão.

Método Login()

É o método que vai retornar os dados de um usuário que está tentando fazer login. Ele é praticamente igual ao método `GetUser()` . A diferença é que, em vez de usar o ID como parâmetro, usamos o email .

```
function Login($data){
    $this->db->select('*')->from('users')->where('email',$data['email']);
    $results = $this->db->get()->result();

    return $results;
}
```

É executado um `SELECT` na tabela `users` para obter os dados do usuário relacionado ao e-mail passado no parâmetro `$data['email']` . Independente de encontrar ou não o usuário, retornamos o resultado do `SELECT` . O processo de verificação do login ainda não acabou, ele vai ser concluído no *controller*, pois é necessário checar o hash da senha.

Mais adiante, criaremos o *controller* `User` e implementaremos o método para completar a verificação.

12.6 CRIANDO O CONTROLLER URLS

Crie um arquivo chamado `Urls.php` dentro do diretório `application/controllers` com o código a seguir:

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Urls extends CI_Controller {

}
```

IMPORTANTE

- Todo *controller* criado no CI deve ser estendido de `CI_Controller`.
- O nome do *controller* deve sempre começar com letra maiúscula, assim como o nome do arquivo.

Método `__construct()`

O método construtor `__construct()` é chamado cada vez que um novo objeto da classe é criado, então nele são colocadas instruções de inicialização que possam ser necessárias aos objetos.

```
...
function __construct(){
    parent::__construct();
    $this->load->helper('string');
    $this->load->library('form_validation');
}
...
```

Veja que logo, na primeira linha, é chamado o construtor da classe pai. Isso é feito porque construtores pais não são chamados de forma implícita quando um construtor é definido em sua classe filha.

Em seguida, foi carregado o helper `string` e a library

`form_validation` . Utilizamos o método `$this->load->helper()` e `$this->load->library()` , respectivamente, para poder executar essas cargas. Ambos os métodos fazem parte da classe `Loader` do CI, que além de `helper` e `library` , possui métodos como `$this->load->model()` , `$this->load->view()` , `$this->load->config()` , entre outros.

Poderíamos ter adicionado essas cargas dentro do arquivo `config/autoload.php` , eliminando o método construtor. Mas sempre que é adicionada uma classe para carregamento no `autoload`, ela passa a ser carregada em todas as requisições, e pode ser que um `helper` ou uma `library`, por exemplo, seja necessário em apenas uma página.

Método `Index()`

É o método chamado sempre que a `home` do encurtador é carregada. Esse método, além de carregar a `home`, é responsável por processar a geração da URL curta, quando o usuário aciona o botão **Encurtar**.

```
public function Index()
{
    $this->form_validation->set_rules('address', 'URL', 'required|min_length[5]|max_length[1000]|trim');
    if($this->form_validation->run() == FALSE){
        $data['error'] = validation_errors();
        $data['short_url'] = false;
    }else{
        $this->load->model('Urls_model');
        $data['address'] = $this->input->post('address');
        if($this->session->userdata('logged'))
            $data['user_id'] = $this->session->userdata('id');
        $res = $this->Urls_model->Save($data);
        if($res){
            $data['error'] = null;
            $data['short_url'] = base_url($res);
        }else{
            $data['error'] = "Não foi possível encurtar a URL.";
            $data['short_url'] = false;
        }
    }
}
```

```

    }

    }
    $this->load->view('home',$data);
}

```

Ele começa fazendo uso do método `$this->form_validation->set_rules()` , que permite validar os campos de um formulário e retornar os erros de preenchimento, evitando que os dados sejam gravados de forma incorreta no banco.

No caso do encurtador, temos apenas um campo para validar, que é o campo `address` , responsável por receber a URL a ser encurtada.

O primeiro parâmetro que ele recebe é o nome do campo (`input`) do formulário, o segundo é uma string que identificará o campo na mensagem de erro (geralmente, usa-se o nome do campo), e o terceiro parâmetro é a lista com os métodos de validação disponibilizados pela library `form_validation` .

Os métodos de validação são colocados como string, e separados por `|` (pipe). No encurtador, usamos os seguintes métodos:

- `required` : informa que o campo é obrigatório;
- `min_length[5]` : define um tamanho mínimo de 5 (cinco) caracteres para a string do campo;
- `max_length[1000]` : limita o tamanho máximo de caracteres a 1.000 (mil);
- `trim` : remove espaços em branco no início e fim da string recebida.

Após executar o método `set_rules()` , é hora de chamar um `if` com a execução do método `run()` , que faz de fato a validação do formulário. Se o retorno do método `run()` for `FALSE` , então ocorrerão erros de validação, e estes são recuperados pelo método `validation_errors()` . No encurtador, setamos

`$data['error']` com os dados de `validation_errors()` e `$data['short_url']` como `null`, e então carregamos a *view* `home`, passando a variável `$data` como parâmetro, para que os dados da variável possam ser recuperados na *view*.

Tem dúvidas sobre validação de formulários? Que tal refrescar sua memória e visitar o capítulo 7. *Validando formulários* para revisar o assunto?

Se não houver erros na validação do formulário, seguimos com o processo de gravação dos dados no banco. Primeiramente, carregamos o *model* `Urls_model` através do método `$this->load->model()`. Em seguida, populamos a variável `$data['address']` com o valor do campo preenchido pelo usuário, o que foi possível pelo método `$this->input->post()`.

Veja que o método `$this->input->post()` recebeu uma string como parâmetro, e essa string identifica o campo que será recuperado (nesse caso é o campo `address`). Tradicionalmente, em PHP, utiliza-se `$_POST`, mas o CodeIgniter já possui recursos próprios para recuperar os dados.

DICA

Para recuperar todos os campos de um formulário usando o método `$this->input->post()`, basta não passar nenhum parâmetro para ele.

Seguindo o processo, verificamos se existe algum usuário logado

por meio do método `$this->session->userdata('logged')` . Se for `TRUE` , quer dizer que há um usuário logado, e então recuperamos o id desse usuário com o método `$this->session->userdata('id')` .

Tem dúvidas sobre o uso de sessões? Que tal refrescar sua memória e visitar o capítulo 9. *Gerenciando sessões com a library Session* para revisar o assunto?

As informações de sessão são geradas no momento que o usuário se loga, e você verá esse processo mais adiante, ainda nesse projeto.

Após todas as verificações e validações, os dados são gravados no banco de dados por meio do método `$this->Urls_model->Save()` , que foi criado no *model* `Urls_model` . Como parâmetro, é passada a variável `$data` , que contém os dados necessários para gravação no banco.

Esse método retorna o código da URL caso esta tenha sido gravada, ou então retorna `FALSE` se não foi. Se ocorreu a gravação dos dados, então a variável `$data` é populada com informações de erro e a URL curta, conforme mostra o código a seguir:

```
...
if($res){
    $data['error'] = null;
    $data['short_url'] = base_url($res);
}else{
    $data['error'] = "Não foi possível encurtar a URL.";
    $data['short_url'] = false;
}
...
```

Método Go()

É o método responsável por redirecionar a URL curta para a URL original.

Pelo método `$this->uri->segment(1)` , obtemos o primeiro nó após a URL base. Caso não tenha sido passado o código da URL curta, o usuário é redirecionado para a home do encurtador. Se foi passado o código, então carregamos o *model* `Urls_model` e executamos o método `Fetch()` , que vai retornar a URL original para que seja feito o redirecionamento através do método `redirect()` .

```
public function Go()
{
    if (!$this->uri->segment(1)) {
        redirect(base_url());
    } else {
        $code = $this->uri->segment(1);
        $this->load->model('Urls_model');
        $result = $this->Urls_model->Fetch($code);
        if ($result) {
            redirect(prepare_url($result));
        } else {
            $data['error'] = "URL não localizada.";
            $data['short_url'] = null;
            $this->load->view('home', $data);
        }
    }
}
```

Veja que está sendo utilizado o método `prepare_url()` e ele está recebendo `$result` como parâmetro, que é a URL original associada ao código da URL curta que foi recuperado na URL. Esse método verifica se a URL possui `http://` , e caso não possua, ele adiciona.

Caso não seja encontrada nenhuma URL associada ao código, será exibida uma mensagem de erro ao carregar a home.



Figura 12.2: Mensagem de erro caso a URL não exista

12.7 CONCLUSÃO

Concluimos a primeira parte do encurtador de URLs, revimos conceitos aprendidos nos capítulos anteriores, praticamos mais sobre a criação de *models* e aprendemos a trabalhar com banco de dados. No próximo capítulo, daremos continuidade ao desenvolvimento do encurtador de URLs, e nos capítulos seguintes veremos de maneira mais aprofundada conceitos como: banco de dados e paginação de resultados - que será uma atualização para o encurtador de URLs.

Links úteis

- **Documentação oficial do CI sobre Models:**
https://codeigniter.com/user_guide/general/models.html
- **Documentação oficial do CI sobre Banco de Dados:**
https://codeigniter.com/user_guide/database/index.html

ml

CRIANDO UM ENCURTADOR DE URLS — PARTE II

"Quando você se compromete com sua visão, você vira profissional." — Gabriel Goffi

Até o momento, nós montamos o ambiente, configuramos o banco de dados, criamos as rotas, o *controller* `User`, e os *models* `User` e `Urls`.

Vamos continuar desenvolvendo o nosso encurtador de URLs, e neste capítulo vamos montar a parte dos usuários, com login, alteração de senha e relacionamento da URL encurtada com o usuário.

13.1 CRIANDO O CONTROLLER USER

Crie um arquivo chamado `User.php` dentro do diretório `application/controllers` com o código a seguir:

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class User extends CI_Controller {

}
```

Esse código torna possível o carregamento da classe, uma vez que sua estrutura mínima necessária está implementada. A seguir, veremos cada método do *controller* `User`, e você complementará esse código mostrado.

Método `__construct()`

O método construtor `__construct()` é chamado cada vez que um novo objeto da classe é criado, então nele são colocadas instruções de inicialização que possam ser necessárias aos objetos. Dessa forma, o model `User_model` e a library `form_validation` serão carregados automaticamente, sem a necessidade de ficar repetindo esse código dentro dos métodos da classe `User` que vão usar os recursos do model e da library.

```
...  
function __construct(){  
    parent::__construct();  
    $this->load->model('User_model');  
    $this->load->library('form_validation');  
}  
...
```

Método `Login()`

É o método que processa o login, autenticando o usuário no site e permitindo que ele possa ter um histórico das URLs que encurtou.

```
public function Login()  
{  
    $this->form_validation->set_rules('email', 'Email', 'required|min_  
length[1]|valid_email|trim');  
    $this->form_validation->set_rules('passw', 'Senha', 'required|min_  
length[6]|trim');  
    if($this->form_validation->run() == FALSE){  
        $data['error'] = validation_errors();  
    }else{  
        $dataLogin = $this->input->post();  
        $res = $this->User_model->Login($dataLogin);  
  
        if($res){
```

```

        foreach($res as $result){
            if (password_verify($dataLogin['passw'], $result->passw))
        {
            $data['error'] = null;
            $this->session->set_userdata('logged',true);
            $this->session->set_userdata('email',$result->email);
            $this->session->set_userdata('id',$result->id);
            redirect();
        }else{
            $data['error'] = "Senha incorreta.";
        }
    }

    }else{
        $data['error'] = "Usuário não cadastrado.";
    }
}

$this->load->view('login',$data);
}

```

Primeiramente, definimos as regras de validação do formulário para os campos email e passw, assim como já foi feito anteriormente ao validar a URL a ser encurtada. Em seguida, executamos a validação e, em caso de erro, passamos as informações para a variável `$data['error']`, e carregamos a view `login`. Se não houver erros, então recuperamos os dados do formulário pelo método `$this->input->post()`, e alocamos na variável `$dataLogin`.

Com os dados do formulário recuperados, então chamamos o método `$this->User_model->Login($dataLogin)`, que vai retornar a lista de usuários associados ao e-mail informado no formulário.

Se não retornar nenhum usuário, passamos uma informação para a variável `$data['error']`, dizendo que não há usuário cadastrado. Mas se retornar algum usuário, então executamos um loop `foreach` para comparar a senha digitada com a armazenada no banco.

Porém, não podemos fazer uma simples comparação, pois a senha foi armazenada no banco em forma de um hash. Então será necessário usar o método nativo do PHP `password_verify()`, cujo primeiro parâmetro é a senha digitada no formulário, e o segundo é a senha armazenada no banco de dados.

Se retornar `TRUE`, então registramos os dados de sessão do usuário e redirecionamos para a home. Caso contrário, adicionamos a mensagem de erro *"Senha incorreta"* à variável `$data['error']`, e depois carregamos a *view* de login.

Método Logout()

É o método que vai encerrar a sessão do usuário. A partir do logout, nenhuma URL encurtada será associada ao usuário até que este volte a estar logado.

```
public function Logout()
{
    $this->session->unset_userdata('logged');
    $this->session->unset_userdata('email');
    $this->session->unset_userdata('id');
    redirect();
}
```

Esse método simplesmente apaga os dados da sessão do usuário, e o redireciona para a home.

Método Register()

É o método executado para cadastro de um novo usuário. O fluxo dele é semelhante ao fluxo de criação de uma URL curta.

```
public function Register()
{
    $this->form_validation->set_rules('name', 'Nome', 'required|min_length[3]|trim');
    $this->form_validation->set_rules('email', 'Email', 'required|min_length[1]|valid_email|trim');
    $this->form_validation->set_rules('passw', 'Senha', 'required|min_
```

```

length[6]|trim');
if($this->form_validation->run() == FALSE){
    $data['error'] = validation_errors();
}else{
    $dataRegister = $this->input->post();
    $res = $this->User_model->Save($dataRegister);
    if($res){
        $data['error'] = null;
    }else{
        $data['error'] = "Não foi possível criar o usuário.";
    }
}

}
if($data['error'])
    $this->load->view('login',$data);
else{
    $this->session->set_userdata('logged',true);
    $this->session->set_userdata('email',$res->email);
    $this->session->set_userdata('id',$res->id);
    redirect();
}
}
}

```

Nesse método, as regras de validação são setadas e, em seguida, é executada a validação. Em caso de erro, informamos a mensagem para a variável `$data['error']`. Se tudo correu bem, então recuperamos os dados do formulário e executamos o método `$this->User_model->Save()` para salvar as informações no banco de dados.

Feito isso, é executada a verificação dos dados após o `INSERT` e, se não houver erros, então é feito o login automático do usuário, e este é redirecionado para a home.

Método UpdatePassw()

É o método que vai executar a mudança de senha do usuário.

```

public function UpdatePassw()
{
    $data['success'] = null;
    $data['error'] = null;
    $this->form_validation->set_rules('passw', 'Senha', 'required|min_

```

```
length[6]|trim');
    if($this->form_validation->run() == FALSE){
        $data['error'] = validation_errors();
    }else{
        $data = $this->input->post();
        $this->User_model->Update($data);
        $data['success'] = "Senha alterada com sucesso!";
        $data['error'] = null;
    }
    $data['user'] = $this->User_model->GetUser($this->session->userdata('id'));
    $this->load->view('alterar-senha',$data);
}
```

Funciona de modo semelhante ao `Register()` : é setada a regra de validação para a senha, e depois é feita a validação do formulário através do método `$this->form_validation->run()` . Em caso de erro, informamos a mensagem para a variável `$data['error']` .

Se tudo correu bem, atualizamos a informação no banco de dados, usando o método `$this->User_model->Update($data)` . Informamos a mensagem de sucesso na variável `$data['success']` , que será exibida para o usuário na *view*, e definimos `$data['erro']` como `null` , já que não houve erros durante o processo.

Método URLs()

É o método que listará todas as URLs encurtadas pelo usuário logado. Um método simples, com rotinas que já foram implementadas nos métodos anteriores.

```
public function URLs(){
    $this->load->model('Urls_model');
    $urls = $this->Urls_model->GetAllByUser($this->session->userdata('id'));
    $data['urls'] = $urls;
    $data['error'] = null;
    $data['short_url'] = false;
    $this->load->view('minhas-urls',$data);
}
```

Primeiro, carregamos o model `Urls_model` e, em seguida, recuperamos todas as URLs associadas ao usuário logado pelo método `$this->Urls_model->GetAllByUser()` . Feito isso, passamos os dados para a variável `$data` , em suas respectivas chaves:

- `urls` : recebe a lista com as URLs associadas ao usuário;
- `error` : mensagem de erro (caso exista);
- `short_url` : caso a URL tenha sido encurtada.

Finalizando, é carregada a *view* com as URLs.

DICA

É possível carregar os *models* com outros nomes, diferentes dos seus nomes originais. Para isso, basta você informar o nome que deseja como segundo parâmetro no método `$this->load->model('NomeOriginal_model', 'NovoNome')` .

13.2 CRIANDO AS VIEWS

Agora que os *models* e controllers já foram criados, precisamos criar as *views*. As *views* foram deixadas por último, pois elas são apenas a estrutura HTML das páginas com a exibição dos dados retornados pelos controllers, e nosso foco maior é o CodeIgniter e suas funcionalidades.

Nas *views*, nós teremos os formulários de interação, a exibição de dados como a lista de URLs encurtadas do usuário, a URL curta após o processamento da URL original e conteúdo de navegação, como os do menu.

Antes de começar a criá-las, crie um novo diretório dentro `application/views`. Este se chamará `commons`, assim como foi feito no projeto do site institucional.

Header

A *view* header é um fragmento do layout que se repete em todas as *views* do projeto, então ela será salva no diretório `application/views/commons` com o nome de `header.php`. Ela é composta basicamente pelas tags de cabeçalho do HTML, os links das folhas de estilo (arquivos CSS) e arquivos JavaScript. Além disso, ela contém o cabeçalho do encurtador com o menu.

```
<!DOCTYPE html>
<html lang="pt_BR">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <meta name="description" content="Encurtador de URLs">
    <meta name="author" content="Jonathan Lamim Antunes">

    <title>Encurtador de URLs</title>

    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
    <link href="http://getbootstrap.com/assets/css/ie10-viewport-bug-workaround.css" rel="stylesheet">
    <link href='https://fonts.googleapis.com/css?family=Roboto:300' rel='stylesheet' type='text/css'>
    <link href="<?=base_url('assets/css/layout.css')?>" rel="stylesheet">

    <!--[if lt IE 9]><script src="http://getbootstrap.com/assets/js/ie8-responsive-file-warning.js"></script><![endif]-->
    <script src="http://getbootstrap.com/assets/js/ie-emulation-modes-warning.js"></script>

    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.mi
```

```

n.js"></script>
<![endif]-->
</head>

<body
  <nav class="navbar navbar-default navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-
ta-toggle="collapse" data-target="#navbar" aria-expanded="false" a
ria-controls="navbar">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="<?=base_url()?">Encurtado
r de URLs</a>
      </div>

      <div id="navbar" class="collapse navbar-collapse">
        <ul class="nav navbar-nav pull-right">
          <li><a href="<?=base_url()?">Home</a></li>
          <?php if($this->session->userdata('logged')){?>
            <li><a href="<?=base_url('minhas-urls')?">Minhas UR
Ls</a></li>
            <li><a href="<?=base_url('alterar-senha')?">Alterar
Senha</a></li>
            <li><a href="<?=base_url('logout')?">Sair</a></li>
          <?php } else{ ?>
            <li><a href="<?=base_url('login')?">Login/Cadastro<
/a></li>
          <?php } ?>
        </ul>
      </div>

    </div>
  </nav>

```

As URLs dos links do menu foram definidas utilizando `base_url()` , para que o parâmetro passado seja concatenado à URL informada nas configurações, no arquivo `config/config.php` .

Veja que existe uma verificação para o caso do usuário estar logado. Se retornar `TRUE` , então exibe as opções: *Minhas URLs*,

Alterar Senha e Sair. Caso contrário, exibe *Login/Cadastro*.

Essa *view* será adicionada como primeira linha em todas as demais *views* criadas, exceto na *view* do rodapé.

Footer

Assim como a *view* header , essa se repete em todas as demais *views* do projeto. Ela contém apenas 5 linhas de código HTML que adicionam um texto ao rodapé do encurtador.

Crie o arquivo `footer.php` e salve-o em `application/views/commons` .

```
<footer class="footer">
  <div class="container">
    <p class="text-muted">Exemplo do capítulo 12 do livro <strong>
CodeIgniter Teoria na Prática</strong></p>
  </div>
</footer>
```

Home

A home é composta basicamente do campo para digitar a URL a ser encurtada, um botão para executar o processo, o cabeçalho e o rodapé.

Salve-a diretamente no diretório `application/views` com o nome de `home.php` .

```
<?php $this->load->view('commons/header')?>
<div class="container">
  <div class="page-header">
    <h1>Encurte e compartilhe</h1>
    <p>URL menor, TEXTO maior.</p>
  </div>
  <div class="row">
    <div class="col-md-8 col-md-offset-2">
      <?php if($error){?>
        <div class="alert alert-danger" role="alert"><?=$error?></di
v>
      <?php } ?>
```

```

        <form action="<?=base_url()?" method="POST">
            <label class="col-md-8 col-md-offset-2">
                <input type="text" class="form-control" placeholder="URL
" name="address"/>
            </label>
            <label class="col-md-2"><input type="submit" class="btn bt
n-success" value="Encurtar"/></label>
        </form>
    </div>
    <div class="col-md-8 col-md-offset-2 col-sm-12">
        <?php if($short_url){ ?>
            <p class="text-center"><a href="<?=$short_url?" target="_bl
ank"><?=$short_url?></a></p>
            <?php } ?>
        </div>
    </div>
</div>

<?php $this->load->view('commons/footer')?>

```

Antes do formulário com o campo para digitar a URL, temos um IF checando a variável \$error que é definida no *controller*. Só é exibido o bloco personalizado com o erro quando ele existir. No formulário, o parâmetro action é setado como base_url() , pois a rota para execução do processo de encurtar a URL é a mesma rota da home.

Logo após o formulário, temos mais um IF , que verifica se foi informada alguma URL curta através da variável \$short_url . Se existe, exibe; caso contrário, não.

As variáveis \$error e \$short_url são setadas no *controller* como \$data['error'] e \$data['short_url'] , e passadas para a *view* no método \$this->load->view('home', \$data) . Todas as chaves do array na variável \$data passam a ser chamadas como variáveis comuns na *view*, e não como array .

Veja na figura a seguir como ficará a home do encurtador:



Figura 13.1: Home

Login/Cadastro

Na *view* `Login`, temos os formulários de login e cadastro, um ao lado do outro. Nada de diferente foi aplicado a esta *view*, continuamos utilizando `base_url()` para definir as URLs do parâmetro `action` dos formulários. Só que, nesse caso, usamos `base_url('login')` para o formulário de login e `base_url('user/register')` para o formulário de cadastro.

Salve-a diretamente no diretório `application/views` com o nome de `login.php`.

```
<?php $this->load->view('commons/header')?>
<div class="container">
  <?php if($error){ ?>
    <div class="alert alert-danger" role="alert"><?=$error?></div>
  <?php } ?>
  <div class="col-md-4 col-md-offset-1">
    <h2 class="col-md-12">Login</h2>
    <form action="<?=base_url('login')?>" method="POST">
      <label class="col-md-12">
        <input type="text" class="form-control" placeholder="Email"
" name="email" required/>
      </label>
      <label class="col-md-12">
        <input type="password" class="form-control" placeholder="S
```

```

        enha" name="passw" required/>
        </label>
        <label class="col-md-12"><input type="submit" class="btn btn
-success" value="Entrar"/></label>
    </form>
</div>
<div class="col-md-4 col-md-offset-1">
    <h2 class="col-md-12">Cadastro-se</h2>
    <form action="<?=base_url('user/register')?>" method="POST">
        <label class="col-md-12">
            <input type="text" class="form-control" placeholder="Nome
(opcional)" name="name"/>
        </label>
        <label class="col-md-12">
            <input type="text" class="form-control" placeholder="Email
" name="email" required/>
        </label>
        <label class="col-md-12">
            <input type="password" class="form-control" placeholder="S
enha" name="passw" required/>
        </label>
        <label class="col-md-12"><input type="submit" class="btn btn
-success" value="Cadastrar"/></label>
    </form>
</div>
</div>
<?php $this->load->view('commons/footer')?>

```

Veja na figura a seguir como ficará a tela de login/cadastro do encurtador:

The image shows a web interface with a teal header bar. On the left, it says 'Encurtador de URLs'. On the right, there are links for 'Home' and 'Login/Cadastro'. Below the header, there are two main sections. The left section is titled 'LOGIN' and contains two input fields: 'Email' and 'Senha', followed by a blue button labeled 'Entrar'. The right section is titled 'CADASTRE-SE' and contains three input fields: 'Nome (opcional)', 'Email', and 'Senha', followed by a blue button labeled 'Cadastrar'.

Exemplo do capítulo 3 do livro CodeIgniter Teoria na Prática

Figura 13.2: Login/Cadastro

Alterar Senha

Assim como as *views* criadas anteriormente, ela é composta por formulário e condicionais para validação de erros e exibição de mensagens.

Salve-a diretamente no diretório `application/views` com o nome de `alterar-senha.php`.

```
<?php $this->load->view('commons/header')?>
<div class="container">
  <div class="col-md-4 col-md-offset-4">
    <h2 class="col-md-12">Alterar Senha</h2>
    <form action="<?=base_url('alterar-senha')?>" method="POST">
      <label class="col-md-12">
        <?=$user->email?>
      </label>
      <label class="col-md-12">
        <input type="password" class="form-control" placeholder="Nova Senha" name="passw" required/>
      </label>
    </form>
  </div>
</div>
```

```

        </label>
        <label class="col-md-12"><input type="submit" class="btn btn
-succes" value="Alterar"/></label>
    </form>
</div>
<div class="col-md-4 col-md-offset-4">
    <?php if($error){ ?>
    <div class="alert alert-danger" role="alert"><?=$error?></div>
    <?php } ?>

    <?php if($success){ ?>
    <div class="alert alert-success" role="alert"><?=$success?></d
iv>
    <?php } ?>
</div>
</div>

<?php $this->load->view('commons/footer')?>

```

Veja a seguir como ficará a tela de alteração de senha do encurtador:

Figura 13.3: Alterar Senha

Minhas URLs

Essa *view* é uma home com uma listagem de exibição das URLs encurtadas com o usuário. Salve-a diretamente no diretório `application/views` com o nome de `minhas-urls.php`.

```
<?php $this->load->view('commons/header')?>
<div class="container">
    <div class="page-header">
        <h1>Encurte e compartilhe</h1>
        <p>URL menor, TEXTO maior.</p>
    </div>
    <div class="row">
        <div class="col-md-8 col-md-offset-2">
            <?php if($error){ ?>
                <div class="alert alert-danger" role="alert"><?=$error?></div>
v>
            <?php } ?>

            <form action="<?=base_url()?>" method="POST">
                <label class="col-md-8 col-md-offset-2">
                    <input type="text" class="form-control" placeholder="URL
" name="address"/>
                </label>
                <label class="col-md-2"><input type="submit" class="btn bt
n-success" value="Encurtar"/></label>
            </form>
        </div>
        <div class="col-md-8 col-md-offset-2 col-sm-12">
            <?php if($short_url){ ?>
                <p class="text-center"><a href="<?=$short_url?>" target="_bl
ank"><?=$short_url?></a></p>
            <?php } ?>
        </div>
    </div>
    <div class="row">
        <div class="col-md-8 col-md-offset-2">
            <h3>Minhas URLs</h3>
            <?php if($urls){?>
                <table class="table">
                    <?php foreach($urls as $url){?>
                        <tr><td><?=$url->address?></td><td><a href="<?=base_url(
$url->code)?>" target="_blank"><?=base_url($url->code)?></a></td><
/tr>
                        <?php } ?>
                    </table>
                <?php }else{ ?>
```

```
<p>Nenhuma URL encurtada.</p>
<?php } ?>
</div>
</div>
</div>

<?php $this->load->view('commons/footer')?>
```

Temos um `IF` verificando a existência da lista de URLs. Caso exista, usamos um `foreach()` para fazer a interação entre os objetos e exibir as URLs encurtadas pelo usuário.

No capítulo 15. *Paginação de resultados*, vamos atualizar a exibição das URLs do usuário adicionando paginação dos resultados.

Veja a seguir como ficará a lista de URLs do usuário:

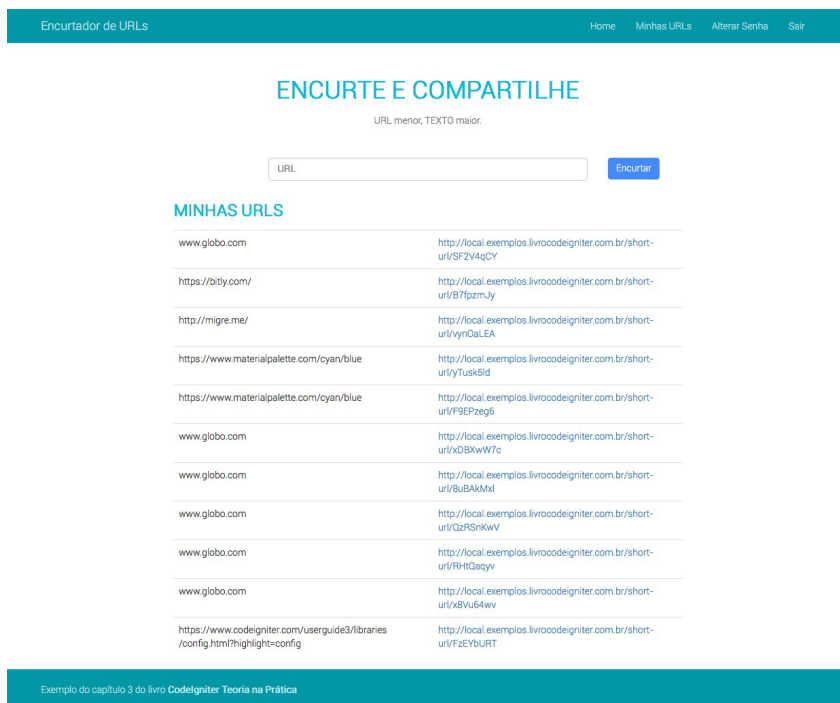


Figura 13.4: Minhas URLs

13.3 CONCLUSÃO

Chegamos ao final de mais um capítulo, e espero que ele tenha sido muito proveitoso para você. Vimos vários novos conceitos e recursos que são usados constantemente em projetos desenvolvidos com o CodeIgniter.

Daqui para a frente, veremos mais detalhes sobre os recursos que foram usados neste capítulo e você poderá ampliar ainda mais os seus conhecimentos.

Bons estudos!

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-12e13-criando-um-encurtador-de-url>

Links úteis

- **Documentação oficial do PHP sobre Password Hashing:**
http://php.net/manual/pt_BR/book.password.php

TRABALHANDO COM BANCO DE DADOS

"A maior habilidade de um líder é desenvolver habilidades extraordinárias em pessoas comuns." — Abraham Lincoln

Trabalhar com banco de dados no CodeIgniter é uma tarefa muito simples, pois ele disponibiliza nativamente vários recursos. Foi possível ver isso ao criar o encurtador de URLs nos dois capítulos anteriores. Durante este capítulo, veremos um pouco mais de detalhes sobre o trabalho com banco de dados no CI.

14.1 CONFIGURANDO UMA CONEXÃO COM O BANCO DE DADOS

Para configurar uma conexão com banco de dados no CI, você deverá editar o arquivo `application/config/database.php`. Veja a seguir como é o arquivo de configuração que já vem na estrutura de arquivos do CodeIgniter:

```
$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => '',
    'password' => '',
    'database' => '',
    'dbdriver' => 'mysqli',
```

```

'dbprefix' => '',
'pconnect' => FALSE,
'db_debug' => (ENVIRONMENT !== 'production'),
'cache_on' => FALSE,
'cachedir' => '',
'char_set' => 'utf8',
'dbcollat' => 'utf8_general_ci',
'swap_pre' => '',
'encrypt' => FALSE,
'compress' => FALSE,
'stricton' => FALSE,
'failover' => array(),
'save_queries' => TRUE
);

```

Esse arquivo é composto por três variáveis:

- `$active_group` : é o conjunto de configurações ativo para a conexão com o banco de dados;
- `$query_builder` : determina se vai carregar automaticamente a library responsável pelos métodos de consulta;
- `$db` : é um array multidimensional que armazena as configurações de cada um dos grupos de conexão com o banco de dados.

A variável `$db` vem configurada com um único grupo, por padrão, mas você pode adicionar quantos grupos quiser. Esses grupos podem ser usados para configurar as conexões de vários ambientes, por exemplo: desenvolvimento, homologação e produção. Para isso, basta recriar a estrutura padrão que já vem no arquivo, alterando o nome da chave no array `$db`.

Cada grupo de conexão criado possuirá suas próprias configurações, assim como o grupo `default`, apresentado no bloco de código anteriormente. Veja a seguir a definição de cada variável de configuração de um grupo:

- `dsn` : string de conexão com o banco de dados;

- `hostname` : nome do host de conexão;
- `username` : nome do usuário para autenticação;
- `password` : senha do usuário para autenticação;
- `database` : nome do banco de dados a ser conectado;
- `dbdriver` : driver de conexão (atualmente suporta: `cubrid` , `ibase` , `mssql` , `mysql` , `mysqli` , `oci8` , `odbc` , `pdo` , `postgre` , `sqlite` , `sqlite3` e `sqlsrv`);
- `dbprefix` : prefixo opcional que será adicionado ao nome da tabela no Query Builder;
- `pconnect` : determina se a conexão é persistente ou não (`TRUE` ou `FALSE`);
- `db_debug` : determina se os erros relacionados ao banco de dados serão exibidos ou não (`TRUE` ou `FALSE`);
- `cache_on` : ativa ou desativa o cache de consultas (`TRUE` ou `FALSE`);
- `cachedir` : path do diretório onde o cache será armazenado;
- `char_set` : o charset utilizado na comunicação com o banco de dados;
- `dbcollat` : o conjunto de caracteres usado na comunicação com o banco de dados;
- `swap_pre` : um prefixo da tabela padrão que deve ser trocado com o `dbprefix` ;
- `encrypt` : se deve ou não usar uma conexão criptografada. O `mysql` (descontinuado), `sqlsrv` e `pdo/sqlsrv` aceitam `TRUE` / `FALSE` , e os `mysqli` e `pdo/mysql` aceitam um array com os valores a seguir:
 - `ssl_key` - path do arquivo da chave privada

- `ssl_cert` - path do certificado público
- `ssl_ca` - path do certificado de autoridade
- `ssl_capath` - path do diretório que contém a chave no formato PEM
- `ssl_cipher` - lista de cifras autorizadas a serem usadas para a codificação, separadas por dois pontos (:)
- `ssl_verify` - (`mysqli` somente) verifica o certificado do servidor (`TRUE` ou `FALSE`)
- `compress` : (MySQL somente) se deve ou não usar compressão (`TRUE` ou `FALSE`);
- `stricton` : determina se será utilizado o "Strict Mode" para as instruções SQL (`TRUE` ou `FALSE`);
- `ssl_options` : usado para definir opções de SSL que podem ser usadas nas conexões;
- `failover` : array com outras conexões caso essa falhe;
- `save_queries` : salva as consultas (`TRUE` ou `FALSE`).

Quando `save_queries` é configurado como `TRUE` , o CI armazena as informações da consulta para fins de depuração. Com isso, é possível obter, por exemplo, a última instrução SQL executada, através do método `$this->db->last_query()` . Se `save_queries` for desativado, esses recursos passam a ficar indisponíveis.

14.2 INICIALIZANDO A LIBRARY DATABASE

Em uma aplicação que faz uso de banco de dados, na maior parte das operações é ideal que a inicialização da library *Database* seja feita no autoload. Para isso, você deverá abrir o arquivo `application/config/autoload.php`, e adicionar `database` no array de `libraries` a ser carregado.

```
$autoload['libraries'] = array('database');
```

Se você for usar a conexão com o banco de dados apenas em algumas operações isoladas, pode fazer a inicialização diretamente no `controller`, pelo método `$this->load->library('database')`.

Após inicializada a library *Database*, os seus métodos poderão ser chamados por meio do objeto `$this->db`.

14.3 EXECUTANDO CONSULTAS COM `$THIS->DB->QUERY()`

A library *Database* possui um método chamado `$this->db->query()`, que recebe como parâmetro uma `string` com a instrução SQL a ser executada.

```
$query = $this->db->query('SELECT * FROM users');
```

Com o mesmo método `$this->db->query()`, também é possível usar outros parâmetros, que vão conter as informações para complementar a `query`.

```
$sql = "SELECT * FROM users WHERE id = ? AND status = ? AND name = ?";  
$this->db->query($sql, array(3, 'active', 'Lamim'));
```

Nesse código, temos a cláusula `WHERE`, que utiliza três campos para filtrar os resultados, e cada um desses campos tem como valor associado o caractere `?` (interrogação). Ao chamar o método `$this->db->query()`, passando como primeiro parâmetro a

instrução SQL e como segundo um array contendo os valores para os filtros da query. Automaticamente a library associa os valores, substituindo as ? pelos valores do array , nas suas respectivas ordens.

Nesse caso, a instrução SQL seria:

```
SELECT * FROM users WHERE id = 3 AND status = "active" AND name = "Lamim"
```

O retorno do método `$this->db->query()` é um objeto no qual você terá vários outros métodos para trabalhar com os resultados da query executada. Esses métodos vão desde o total de resultados até a obtenção de um array com a lista de resultados.

\$THIS->DB

Sempre que precisar utilizar um método relacionado a banco de dados no CI, exceto para a classe Database Utility , você deverá usar `$this->db` , pois é a indicação da instância e da classe onde os métodos estão localizados.

Vejamos a seguir alguns dos métodos mais usados.

result()

Ele retorna um array de objetos, um array vazio, ou então um erro. Ele é muito utilizado em combinação com um `foreach()` .

```
$query = $this->db->query("SELECT * FROM users");

foreach ($query->result() as $row)
{
    echo $row->name;
    echo $row->status;
```

```
}
```

row()

Ele retorna uma única linha do resultado da consulta, e é sempre a primeira.

```
$query = $this->db->query("SELECT * FROM users");

$row = $query->row();

if (isset($row))
{
    echo $row->name;
    echo $row->status;
}
```

Se você quiser obter uma linha específica dentro do array , basta passar como parâmetro o número da linha.

```
$query = $this->db->query("SELECT * FROM users");

$row = $query->row(3);

if (isset($row))
{
    echo $row->name;
    echo $row->status;
}
```

num_rows()

Ele retorna a quantidade de registros encontrados na consulta.

```
$query = $this->db->query('SELECT * FROM users');

echo $query->num_rows();
```

14.4 QUERY HELPER

O Query Builder é um conjunto de métodos para obtenção de informações da query e do banco de dados. Vejamos alguns dos

métodos mais utilizados.

insert_id()

Ele retorna o ID da última instrução `INSERT` executada.

```
$this->db->insert_id()
```

affected_rows()

Ele retorna o número de linhas afetadas pela instrução executada.

```
$this->db->affected_rows();
```

last_query()

Ele retorna a string da última instrução SQL executada.

```
$this->db->last_query();
```

count_all()

Ele retorna o total de registros de uma tabela. Para isso, basta passar como parâmetro o nome da tabela.

```
$this->db->count_all('users');
```

14.5 QUERY BUILDER

É um conjunto de métodos nativos do CI que permite operações com banco de dados de forma mais simples, com menos código e sem a necessidade de escrever as instruções SQL manualmente. Vejamos os métodos mais usados.

get()

Executa a consulta e retorna o resultado. Pode ser usado para

recuperar todos os registros de uma tabela passando como parâmetro o nome da tabela.

```
$query = $this->db->get('users');  
//Instrução equivalente: SELECT * FROM users
```

Se passar mais dois parâmetros, você consegue limitar a quantidade de registros retornados.

```
$query = $this->db->get('users', 5, 10);  
//Equivalente a: SELECT * FROM users LIMIT 10, 5
```

get_where()

Idêntico ao método anterior, exceto que ele permite que você adicione uma cláusula `where` no segundo parâmetro em vez de usar o `$this->db->where`. O terceiro e quarto parâmetros podem ser utilizados para determinar os limites da quantidade de registros retornados.

```
$query = $this->db->get_where('users', array('id' => 3), 5, 10);  
//Equivalente a: SELECT * FROM users WHERE id = 3 LIMIT 10, 5
```

Os métodos apresentados a seguir podem ser usados de forma concatenada (veja o exemplo adiante). Mas para que fique mais fácil de você compreender o que cada um faz, vou listá-los de forma individual.

```
$this->db->select('*')->from('users')->where('id',3);
```

select()

Método que define quais serão os campos da(s) tabela(s) a serem retornados na consulta.

```
$this->db->select('*'); //Equivalente a: SELECT *  
$this->db->select('name, status'); //Equivalente a: SELECT name, status
```

from()

Método que define a tabela na qual a consulta será realizada.

```
$this->db->from('users');  
$this->db->select('name, status')->from('users');  
//Equivalente a: SELECT name, status FROM users
```

join()

Método utilizado para fazer consultas onde é necessário usar a instrução SQL JOIN, que junta registros de outras tabelas na mesma consulta. Ele recebe dois parâmetros, sendo que o primeiro é o nome da tabela e o segundo são os valores que se equivalem para retorno dos dados.

```
$this->db->select('*');  
$this->db->from('posts');  
$this->db->join('comments', 'comments.id = posts.id');  
  
// Equivalente a: SELECT * FROM posts JOIN comments ON comments.id  
= posts.id
```

Ele aceita ainda um terceiro parâmetro, que pode conter os valores `left` ou `right`, que fazer com que sejam executados, respectivamente, `LEFT JOIN` ou `RIGHT JOIN` em vez de um simples `JOIN`.

where()

Esse método permite que você defina cláusulas `where` usando um dos quatro métodos:

- **Chave/valor:** onde a comparação será sempre usando o comparador = (igual).

```
$this->db->where('name', 'Lamim'); // Equivalente a:  
WHERE name = "Lamim"
```

- **Chave/valor customizada:** onde você pode definir o operador de comparação a ser utilizado:

```
$this->db->where('name !=', 'Lamim'); // Equivalente a: WHERE name != "Lamim"
$this->db->where('id >', 3); // Equivalente a: WHERE id > 3
```

- **Array associativo:** onde você pode utilizar um array para definir as comparações.

```
$array = array('name' => 'Lamim', 'status' => 'active');
$this->db->where($array);
// Equivalente a: WHERE name = 'Lamim' AND status = 'active'
```

- **String customizada:** onde você pode customizar a string da cláusula where .

```
$where = "name='Lamim' OR status='active'";
$this->db->where($where); //Equivalente a: WHERE name='Lamim' OR status='active'
```

or_where()

É idêntico ao método anterior, mas adiciona um `or` à cláusula where .

```
$this->db->where('name !=', 'Lamim');
$this->db->or_where('id >', 3); // Equivalente a: WHERE name != 'Lamim' OR id > 3
```

like()

Este método permite gerar cláusulas `like` , muito úteis para fazer sistemas de buscas de blogs, por exemplo.

- **Chave/valor:**

```
$this->db->like('name', 'Lamim');
// Equivalente a: WHERE name LIKE '%Lamim%' ESCAPE '!'
```

- **Array associativo**

```

$array = array('name' => 'Lamim', 'status' => 'active');
$this->db->like($array);
// Equivalente a: WHERE name LIKE '%Lamim%' ESCAPE '!' AND status LIKE '%active%' ESCAPE '!'

```

or_like()

Idem ao método anterior, porém adicionando um `or` à cláusula `like`.

```

$this->db->like('name', 'Lamim')->or_like('status','active');
// Equivalente a: WHERE name LIKE '%Lamim%' ESCAPE '!' OR name LIKE '%Lamim%' ESCAPE '!'

```

not_like()

Idêntico ao `like()`, mas adiciona um `not` antes.

```

$this->db->not_like('name', 'Lamim');
// equivalente a: WHERE name NOT LIKE '%Lamim%' ESCAPE '!'

```

group_by()

Esse método adiciona uma cláusula `group by` à consulta.

```

$this->db->group_by("name"); // Equivalente a: GROUP BY name

```

Também pode ser passado um array como parâmetro:

```

$this->db->group_by(array('name','status')); // Equivalente a: GROUP BY name, status

```

order_by()

Esse método adiciona uma cláusula `order by` à consulta. Ele aceita dois parâmetros: o primeiro é o nome do campo da tabela para a ordenação, e o segundo é a direção do resultado (`asc`, `desc` ou `random`).

```

$this->db->order_by('name', 'DESC');
// Equivalente a: ORDER BY `name` DESC

```

O primeiro parâmetro também pode ser uma string com a cláusula `order by` já escrita:

```
$this->db->order_by('name DESC, id ASC');  
// Equivalente a: ORDER BY name DESC, id ASC
```

limit()

Esse método permite limitar o número de linhas retornadas pela consulta. Ele aceita até dois parâmetros: o primeiro é o número de linhas e o segundo é o deslocamento do resultado.

```
$this->db->limit(5); // Equivalente a: LIMIT 5  
$this->db->limit(5, 10); // Equivalente a: LIMIT 10, 5
```

count_all_results()

Esse método retorna o número de registros da consulta realizada. Você pode passar como parâmetro o nome da tabela, ou então usar de forma combinada com alguns dos métodos apresentados anteriormente.

```
$this->db->count_all_results('users'); // Retorna um inteiro, que  
equivale ao total de registros na tabela users
```

```
$this->db->like('name', 'Lamim');  
$this->db->from('users');  
$this->db->count_all_results(); // Retorna o total de registros da  
tabela users que possuem Lamim no nome
```

Esses são os métodos mais usados para lidar com banco de dados no CI. Todos os métodos listados podem ser concatenados para que se obtenham queries mais completas e, até mesmo, mais complexas.

Obtendo os resultados das queries executadas

Para obter os resultados de uma query, você pode combinar os métodos `get()` e `result()` e obter como retorno um array

com os resultados da consulta.

```
$this->db->select('name')
$this->db->from('users');
$this->db->like('name', 'Lamim');
$results = $this->db->get()->result();
```

14.6 CRUD

Vamos ver agora os outros três métodos que compõem um CRUD, já que um dos métodos nada mais é do que a combinação de todos os métodos mostrados anteriormente.

Recorde: CRUD é um acrônimo para *'create, read, update and delete'*

insert()

Esse método gera uma sequência de inserção com base nos dados fornecidos e executa a consulta. Você pode passar um array ou um objeto como segundo parâmetro, pois o primeiro deverá ser sempre o nome da tabela na qual o `insert` será executado.

```
$data = array(
    'name' => 'Antunes',
    'status' => 'inactive'
);

$this->db->insert('users', $data);
// Equivalente a: INSERT INTO users (name, status) VALUES ('Antunes', 'inactive')
```

update()

Esse método gera uma sequência de atualização com base nos dados fornecidos e executa a consulta. Você pode passar um array

ou um objeto como segundo parâmetro, pois o primeiro deverá ser sempre o nome da tabela na qual o `update` será executado.

```
$data = array(
    'name' => 'Jonathan Lamim',
    'status' => 'active'
);

$this->db->where('id', 3);
$this->db->update('users', $data);
// Equivalente a: UPDATE users SET name = 'Jonathan Lamim', status
= 'active' WHERE id = 3
```

delete()

Esse método gera e executa uma string para exclusão de registros do banco de dados. Ele pode receber os parâmetros da seguinte forma:

- **Nome da tabela:**

```
$this->db->where('id', 3);
$this->db->delete('users'); // Equivalente a: // DELETE FROM user
s WHERE id = 3
```

- **Nome da tabela / Array com identificador do registro a ser removido:**

```
$this->db->delete('users', array('id' => 3)); // Equivalente a: /
/ DELETE FROM users WHERE id = 3
```

Esses são os métodos mais utilizados quando se trabalha com banco de dados no CodeIgniter. Você pode usar outras libraries para operações com banco de dados em vez de usar a library padrão do CI, só depende de você.

14.7 CONCLUSÃO

Neste capítulo, você viu com mais detalhes alguns dos métodos mais utilizados no CI para trabalhar com banco de dados. Nos

capítulos *Criando um encurtador de URLs — Parte I e II*, já haviam sido passadas algumas informações, mas de forma superficial, apenas para que fosse possível compreender a aplicação dos códigos.

Exemplo prático

Para reforçar o que você aprendeu neste capítulo, acesse o portal **Universidade CodeIgniter** pelo link a seguir, e desenvolva o exemplo prático do tutorial.

<http://www.universidadecodeigniter.com.br/criando-um-crud-com-codeigniter/>

Links úteis

- **Documentação oficial do CI sobre banco de dados:**
https://codeigniter.com/user_guide/database/index.html

PAGINAÇÃO DE RESULTADOS

"Você pode criar qualquer coisa: basta escrever." — C. S. Lewis

A paginação de resultados é um recurso importante em projetos nos quais temos um volume grande de dados que precisam ser exibidos para o usuário. Com a paginação, você pode reduzir o tempo de processamento da página, uma vez que não serão carregados todos os registros de uma única vez.

15.1 INTRODUÇÃO À LIBRARY PAGINATION

O CodeIgniter possui uma library chamada *Pagination*, que permite implementar a paginação de resultados com poucas linhas de código, de forma bem simples e 100% customizável. Veja no exemplo a seguir que, com somente 6 linhas de código, já é possível fazer a implementação da paginação.

```
$this->load->library('pagination');

$config['base_url'] = 'http://www.livrocodeigniter.com.br/';
$config['total_rows'] = 200;
$config['per_page'] = 20;

$this->pagination->initialize($config);

echo $this->pagination->create_links();
```

Esse código será explicado ao longo do capítulo.

Inicializando a library

Para inicializar a library *Pagination*, você pode utilizar tanto o arquivo `application/config/autoload.php` quanto o método `$this->load->library('pagination')` .

Se você vai usar a paginação em *views* isoladas, recomendo inicializar a library dentro do *controller* que chamará a *view*. Mas se o volume de uso for grande, então é melhor inicializar no *autoload*, para não correr o risco de esquecer de fazer a chamada de inicialização no *controller*.

Parâmetros de configuração

Toda a configuração da paginação é feita por meio de um array passado ao método `$this->pagination->initialize($config)` , que vai setar os parâmetros na library. As configurações vão desde a URL base da aplicação até a customização dos links da paginação.

Veja a seguir a lista dos parâmetros e qual a função de cada um:

- `base_url` : é a URL base que será utilizada nos links da paginação;
- `total_rows` : é o número total de registros a serem paginados;
- `per_page` : é o número de registros a serem exibidos em cada página;
- `uri_segment` : é a posição na URL em que se encontra a informação da página em que o usuário está;
- `num_links` : é a quantidade de páginas a serem exibidas na paginação, antes e depois da página atual do usuário;
- `use_page_numbers` : define se na URL será exibida a posição da página ou a posição inicial da contagem de

registros (TRUE ou FALSE);

- `page_query_string` : define se a paginação será apresentada na URL por meio de variáveis explícitas ou URL amigável (TRUE ou FALSE);
- `reuse_query_string` : define se a query string utilizada será reaproveitada (TRUE ou FALSE);
- `prefix` : é um prefixo personalizado a ser adicionado à URL que virá antes da definição da página em que o usuário está;
- `suffix` : é um sufixo personalizado a ser adicionado à URL que virá depois da definição da página em que o usuário está;
- `full_tag_open` e `full_tag_close` : definem, respectivamente, qual tag vai abrir e fechar o bloco que vai conter os links da paginação;
- `first_link` : é o texto ou caractere que representará o link para voltar à primeira página de resultados;
- `first_tag_open` e `first_tag_close` : são as tags que envolverão o primeiro link da paginação;
- `first_url` : uma URL alternativa para ser utilizada no primeiro link;
- `last_link` : uma URL alternativa para ser utilizada no último link;
- `last_tag_open` e `last_tag_close` : são as tags que envolverão o último link da paginação;
- `next_link` : é o texto ou caractere que representará o link para avançar para a próxima página de resultados;
- `next_tag_open` e `next_tag_close` : são as tags que envolverão o link para avançar para a próxima da paginação;
- `prev_link` : é o texto ou caractere que representará o link para avançar para a página de resultados anterior;
- `prev_tag_open` e `prev_tag_close` : são as tags que

envolverão o link para voltar para a página anterior da paginação;

- `cur_tag_open` e `cur_tag_close` : são as tags que envolverão o link da página atual da paginação;
- `num_tag_open` e `num_tag_close` : são as tags que envolverão os links numéricos da paginação;
- `display_pages` : exibe ou não os links numéricos da paginação (`TRUE` ou `FALSE`);
- `attributes` : determina atributos extras para os links da paginação, como por exemplo, adicionar uma classe
`$config['attributes'] = array('class' => 'myclass') .`

Os métodos que compõem a library

Essa library é composta por apenas dois métodos:

- `initialize` : responsável por carregar o array de configuração para a library;
- `create_links` : responsável por criar a estrutura HTML da paginação a partir das configurações definidas e setadas com o método `initialize()` .

O método `create_links()` retorna uma string com a estrutura HTML da paginação, logo essa string pode ser armazenada em uma variável e esta enviada para a *view*, ou então ser utilizado diretamente na *view* acompanhado de um `echo` . Veja novamente o exemplo apresentado no início do capítulo:

```
$this->load->library('pagination');

$config['base_url'] = 'http://www.livrocodeigniter.com.br/';
$config['total_rows'] = 200;
$config['per_page'] = 20;

$this->pagination->initialize($config);
```

```
echo $this->pagination->create_links();
```

15.2 IMPLEMENTANDO A PAGINAÇÃO NO ENCURTADOR DE URL

Vamos usar o encurtador de URL desenvolvido nos capítulos anteriores para aplicar os conceitos práticos e teóricos sobre a paginação de resultados no CodeIgniter. Você pode fazer uma cópia do diretório do projeto para fazer a implementação da paginação, ou fazer essas implementações diretamente no diretório criado no capítulo 12. *Criando um encurtador de URLs — Parte I.*

Atualizando a view

Abra a *view* `application/views/minhas-urls.php`, e localize o bloco `<div class="row">...</div>` que está antes de `$this->load->view('commons/footer')`.

```
<div class="row">
    <div class="col-md-8 col-md-offset-2">
        <h3>Minhas URLs</h3>
        <?php if($urls){?>
            <table class="table">
                <?php foreach($urls as $url){?>
                    <tr><td><?=$url->address?></td><td><a href="<?=$base_url($url->code)?>" target="_blank"><?=$base_url($url->code)?></a></td></tr>
                <?php } ?>
            </table>
            <?php }else{ ?>
                <p>Nenhuma URL encurtada.</p>
            <?php } ?>
        </div>
    </div>
```

Logo após fechar a tag `<table>`, adicione a chamada para os links de paginação (`<?=$pagination?>`):

```
<div class="row">
    <div class="col-md-8 col-md-offset-2">
        <h3>Minhas URLs</h3>
```

```

        <?php if($urls){?>
        <table class="table">
        <?php foreach($urls as $url){?>
            <tr><td><?=$url->address?></td><td><a href="<?=base_url($url->code)?>" target="_blank"><?=base_url($url->code)?></a></td></tr>
        <?php } ?>
        </table>
        <div class="clearfix"></div>
        <?=$pagination?>
        <?php }else{ ?>
            <p>Nenhuma URL encurtada.</p>
        <?php } ?>
    </div>
</div>

```

A *view* está pronta para exibir a paginação dos resultados. Agora vamos implementar a rotina de paginação no *controller* e no *model*.

Atualizando o model `Urls_model`

A paginação de resultados vai exibir um conjunto limitado de registros, e não todos de uma única vez. Sendo assim, não podemos continuar usando o método `$this->Urls_model->GetAllByUser()` para obter a lista de URLs.

Vamos criar um novo método no *model* `Urls_model` chamado `GetAllByPage()`, que vai receber três parâmetros:

- `$user_id` : determina de qual usuário devem ser as URLs;
- `$limit` : número de registros a serem retornados pela consulta;
- `$offset` : a 'página' de registros a ser retornada.

Abra o *model* `Urls_model.php` e crie o novo método com o código a seguir:

```

function GetAllByPage($user_id, $limit, $offset){
    $this->db->select('*')->from('urls')->where('user_id', $user_id)->limit($limit, $offset);
}

```

```

$result = $this->db->get()->result();
if($result){
    return $result;
}else{
    return false;
}
}

```

A estrutura do método é como a do método `GetAllByUser()`, porém, ele adiciona o `limit` na instrução SQL da primeira linha, que retornará os registros. Em seguida, ele recupera os registros a serem exibidos pelo método `$this->db->get()->result()` e faz a verificação dos dados para retornar, ou os dados, ou `FALSE` caso a consulta não tenha retornado nenhum registro.

Com o *model* atualizado, é hora de atualizar o *controller*.

Atualizando o controller User

Abra o *controller* `User.php` e localize o método `URLs()`. É nesse método que vamos implementar toda a rotina de paginação, para que através do método `create_links()` seja possível exibir na tela a lista de páginas.

```

public function URLs(){
    $this->load->model('Urls_model');
    $urls = $this->Urls_model->GetAllByUser($this->session->userdata
('id'));
    $data['urls'] = $urls;
    $data['error'] = null;
    $data['short_url'] = false;
    $this->load->view('minhas-urls',$data);
}

```

Atualmente, o código do método `URLs()` apenas carrega a lista completa de URLs encurtadas pelo usuário e retorna para a *view*, que exibe essa lista na tela.

Vamos alterar o código para que, em vez de exibir todas as URLs, exiba somente cinco em cada página. E também, conforme o

usuário for navegando pela paginação, vamos carregando os demais registros na sequência.

Para isso, vamos criar uma variável chamada `$config`, que vai ser do tipo `array`, para armazenar as configurações a serem usadas na library *Pagination*. Em seguida, vamos chamar o método `$this->pagination->initialize($config)` para que ele defina as configurações.

Na sequência, rodamos um `if` para verificar qual a página o usuário está, e em seguida vamos chamar o método `$this->pagination->create_links()` que retornará os links da paginação para a variável `$data['pagination']`. Também vamos substituir o método `$this->Urls_model->GetAllByUser()` pelo método `$this->Urls_model->GetAllByPage()`.

```
public function URLs(){
    $this->load->model('Urls_model');

    $config['base_url'] = base_url('minhas-urls');
    $config['total_rows'] = $this->db->select('*')->from('urls')->where('user_id',$this->session->userdata('id'))->count_all_results()
;
    $config['per_page'] = 5;
    $config['uri_segment'] = 2;
    $config['num_links'] = 5;
    $config['use_page_numbers'] = TRUE;
    $config['full_tag_open'] = "<nav><ul class='pagination'>";
    $config['full_tag_close'] = "<ul></nav>";
    $config['first_link'] = "Primeira";
    $config['first_tag_open'] = "<li>";
    $config['first_tag_close'] = "</li>";
    $config['last_link'] = "Última";
    $config['last_tag_open'] = "<li>";
    $config['last_tag_close'] = "</li>";
    $config['next_link'] = "Próxima";
    $config['next_tag_open'] = "<li>";
    $config['next_tag_close'] = "</li>";
    $config['prev_link'] = "Anterior";
    $config['prev_tag_open'] = "<li>";
    $config['prev_tag_close'] = "</li>";
    $config['cur_tag_open'] = "<li class='active'><a href='#'>";
    $config['cur_tag_close'] = "</a></li>";
```

```

$config['num_tag_open'] = "<li>";
$config['num_tag_close'] = "</li>";

$this->pagination->initialize($config);

if($this->uri->segment(2))
    $offset = ($this->uri->segment(2) - 1) * $config['per_page'];
else
    $offset = 0;

$urls = $this->Urls_model->GetAllByPage($this->session->userdata
('id'), $config['per_page'], $offset);
$data['urls'] = $urls;
$data['error'] = null;
$data['short_url'] = false;
$data['pagination'] = $this->pagination->create_links();
$this->load->view('minhas-urls', $data);
}

```

Nas configurações da library, nós utilizamos praticamente todas as variáveis de configuração, personalizando assim a paginação conforme o estilo do *Bootstrap*, framework usado para estilizar o layout.

Agora, quando o usuário acessar a lista de URLs que ele encurtou, não verá todos os registros, mas os cinco primeiros e os links da paginação para navegar pelos demais.

Para finalizar, vamos atualizar as rotas.

Atualizando as rotas

Para que seja possível manter a paginação com URLs amigáveis, vamos atualizar o arquivo `application/config/routes.php`, adicionando uma rota que permitirá visualizar os registros por meio da paginação.

```
$route['minhas-urls/(:num)'] = "User/Urls/$1";
```

Essa nova rota adiciona um parâmetro à URL, que é o parâmetro que define qual é a posição da paginação. Com isso,

concluímos a aplicação da paginação de resultados no encurtador de URLs.

15.3 CONCLUSÃO

Neste capítulo você aprendeu a implementar a paginação de resultados utilizando a library *Pagination*, nativa do CI. No próximo capítulo você aprenderá a utilizar o template parser, para reduzir o volume de código PHP nas suas views.

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-15-paginacao-de-resultados>

Links úteis

- **Documentação oficial sobre a library *Pagination*:**
http://www.codeigniter.com/user_guide/libraries/pagination.html

USANDO TEMPLATE PARSER

"Se alguém lhe oferece uma oportunidade incrível, mas você não tem certeza de que pode executar, diga sim - e aprenda como fazer depois." — Richard Branson

Nem todo programador front-end tem conhecimento de PHP. Logo, uma *view* com muito código PHP fica complexa para que ele dê manutenção. O template parser ajuda a reduzir o volume de código PHP na *view*, tornando-a mais legível para um desenvolvedor front-end que não está familiarizado com a linguagem.

Através de alguns blocos de exemplos, você vai aprender neste capítulo como usar o template parser nativo do CodeIgniter. Para isso, inicie um novo diretório de exemplo, como foi feito nos capítulos anteriores.

Tem dúvidas sobre a criação do diretório base para os exemplos? Que tal refrescar sua memória e revisar o capítulo 1. *Introdução ao CodeIgniter* para revisar o assunto?

16.1 INTRODUÇÃO

O template parser substitui parte do código PHP na *view* por pseudovariáveis dentro de `{variável}` (chaves). Veja a seguir o código de duas views com a mesma estrutura: a primeira usa código PHP e a segunda somente as pseudovariáveis do template parser.

- **Sem pseudovariáveis**

```
<html>
  <head>
    <title><?=$blog_title?></title>
  </head>
  <body>
    <h3><?=blog_heading?></h3>

    <?php foreach($blog_entries as $entries) ?>
      <h5><?=$entries->title?></h5>
      <p><?=$entries->body?></p>
    <?php endforeach; ?>

  </body>
</html>
```

- **Com pseudovariáveis**

```
<html>
  <head>
    <title>{blog_title}</title>
  </head>
  <body>
    <h3>{blog_heading}</h3>

    {blog_entries}
      <h5>{title}</h5>
      <p>{body}</p>
    {/blog_entries}

  </body>
</html>
```

Comparando os dois códigos, o segundo está bem mais legível e *clean* do que o primeiro, que mistura código PHP com código HTML. Porém, ambos vão gerar o mesmo resultado final na tela do usuário.

Mesmo usando template parser, haverá situações em que será necessário usar algum código PHP na *view*, pois nem todo tipo de lógica pode ser feito usando pseudovariáveis. Não há problema nisso, mas é bom usar o mínimo possível, já que a *view* é o local no qual deve estar apenas o código HTML. Toda a parte de tratamento dos dados deve ser feita no *controller* e no *model*, por uma questão de boas práticas.

16.2 INICIALIZANDO A LIBRARY

A inicialização da library *Template Parser* é feita da mesma forma como as outras libraries. Ou seja, pode ser feita tanto no autoloader quanto no *controller*, antes de seu uso.

Geralmente, usa-se template parser para o projeto em um todo, e não somente em algumas *views*. Então, dentro desse cenário, é ideal que a inicialização seja feita no arquivo `application/config/autoload.php`.

```
$autoload['libraries'] = array('parser');
```

16.3 APLICANDO O TEMPLATE PARSER NA VIEW

Abra a *view* `welcome_messenger.php` da base padrão que estamos utilizando para o exemplo, e altere as linhas de código conforme mostrado a seguir:

```
<title>Welcome to CodeIgniter</title> -> <title>
{page_title}</title>
```

```
<h1>Welcome to CodeIgniter!</h1> -> <h1>
{content_title}</h1>
```

Remova o conteúdo que está dentro de `<div id="body">...`

</div> , e coloque o seguinte conteúdo:

```
{list_entries}
<p>- <strong>{name}</strong> - {email}</p>
{/list_entries}
```

O conteúdo adicionado ao bloco `<div id="body">...</div>` vai exibir uma lista de nomes e e-mails.

Ao substituir o conteúdo de texto da *view* pelas pseudovariáveis entre chaves (`{ }`), você está fazendo com que o conteúdo a ser exibido dependa de valores definidos no *controller*. Esses valores serão enviados para a *view* conforme o fluxo de operação do MVC. Ou seja, o conteúdo da *view* passa a ser dinâmico, mas sem o uso de *short tags* e variáveis do PHP.

Com a *view* reescrita, é hora de atualizar o *controller*.

16.4 CHAMANDO O TEMPLATE PARSER NO CONTROLLER

Quando utilizamos o template parser, devemos chamar os seus métodos para exibição da *view*, e não o método padrão que usamos até o momento: `$this->load->view()` . Isso é necessário pois o `$this->load->view()` não é capaz de substituir as pseudovariáveis pelos seus valores reais.

Abra o *controller* `welcome.php` e substitua o `$this->load->view('welcome_message');` por `$this->parser->parse('welcome_message');` . Em seguida, abra no browser para ver o resultado.

Você verá que as pseudovariáveis foram impressas na tela, mesmo tendo chamado o método `$this->parser->parse('welcome_message');` para fazer a renderização da *view*. Isso aconteceu porque os valores das pseudovariáveis não foram

definidos. Sempre que uma pseudovariável não tiver seu valor definido, ela será exibida dentro das chaves.

Se a exibição de erros estiver habilitada, serão exibidos alguns *warnings* por conta da falta das pseudovariáveis.

Para resolver isso, vamos criar uma variável do tipo `array`, contendo as informações que precisamos exibir na *view*, e passar como segundo parâmetro para o método `$this->parser->parse()`.

```
public function index()
{
    $data = array(
        'page_title' => 'Usando template parser',
        'content_title' => 'Nomes e Emails',
        'list_entries' => array(
            array('name' => 'User 1', 'email' => 'user@mail.com'),
            array('name' => 'User 2', 'email' => 'user_2@mail.com'),
            array('name' => 'User 3', 'email' => 'user_3@mail.com'),
            array('name' => 'User 4', 'email' => 'user_4@mail.com')
        )
    );
    $this->parser->parse('welcome_message', $data);
}
```

Atualize a janela do browser e veja que agora estão sendo exibidos os valores no lugar das pseudovariáveis. Veja que, para `list_entries`, foi associado um novo `array`, contendo a lista de nomes e e-mails a serem exibidos.

Em uma aplicação que faz uso de banco de dados para obter listas de registros, é possível associar os dados retornados pela consulta diretamente à variável, e utilizar o nome dos campos como as pseudovariáveis da *view*. Seria algo como, por exemplo:

```
$query = $this->db->query("SELECT * FROM users");
$data = array(
```

```
'page_title' => 'Usando template parser',  
'content_title' => 'Nomes e Emails',  
'list_entries' => $query->result_array()  
)  
);
```

Se você reparar bem entre o exemplo mostrado na introdução deste capítulo, temos uma pseudovariável `{blog_entries}` que substitui o `foreach` do PHP. No nosso exemplo, também usamos uma para renderização da lista, substituindo assim o `foreach` (a pseudovariável usada foi `{list_entries}...{/list_entries}`).

- Se você chamar uma pseudovariável na *view*, mas ela não tiver sido passada no `array`, a renderização será da pseudovariável e não do conteúdo;
- Se você passar um valor no `array`, mas não chamar a pseudovariável na *view*, nenhum erro ocorrerá.

16.5 USANDO O TEMPLATE PARSE PARA UMA STRING

Existe possibilidade de você utilizar o template parser para uma string dentro do *controller*, e carregar a *view* também usando o template parser. Para isso ser possível, você utiliza o método `$this->parser->parse_string()`, que é capaz de renderizar uma string sem exibir para o usuário após a conclusão do processo.

Volte ao *controler* `welcome.php` e, no método `index()`, antes de renderizar a *view*, adicione o código a seguir. Com ele, vamos renderizar uma outra lista de nomes e e-mails, mas desta vez sem executar o loop de renderização na *view*. Faremos isso dentro do *controller*, e enviaremos para a *view* somente a string com o

conteúdo completo da lista, inclusive o HTML.

```
public function index()
{
    $data = array(
        'page_title' => 'Usando template parser',
        'content_title' => 'Nomes e Emails',
        'list_entries' => array(
            array('name' => 'User 1', 'email' => 'user@mail.com'),
            array('name' => 'User 2', 'email' => 'user_2@mail.com'),
            array('name' => 'User 3', 'email' => 'user_3@mail.com'),
            array('name' => 'User 4', 'email' => 'user_4@mail.com')
        )
    );

    $nouser_list_template = "<li>{name} - {email}</li>";
    $nusers = array(
        array('name' => 'No User 1', 'email' => 'no_user@mail.com'),
        array('name' => 'No User 2', 'email' => 'no_user_2@mail.com'),
        array('name' => 'No User 3', 'email' => 'no_user_3@mail.com'),
        array('name' => 'No User 4', 'email' => 'no_user_4@mail.com')
    );

    $base_list = "<ul>";
    foreach ($nusers as $user)
    {
        $base_list .= $this->parser->parse_string($nouser_list_templat
e, $user, TRUE);
    }
    $base_list .= "</ul>";

    $data["no_users"] = $base_list;
    $data["no_users_title"] = 'Não Usuários';

    $this->parser->parse('welcome_message', $data);
}
```

Foi criada uma variável `$nouser_list_template` que recebe como conteúdo uma string contendo uma pseudovariável, uma variável `$nusers` que recebe um array com a lista dos nomes e e-mails, e um `foreach` que faz a interação na lista de nomes e e-mails da variável `$nusers`.

Dentro do `foreach`, é chamado o método `$this->parser->parse_string($nouser_list_template, $user, TRUE)` com

três parâmetros:

- **Primeiro:** o template a ser renderizado (que nesse caso é uma string e não um arquivo físico do diretório `application/views`).
- **Segundo:** as informações do usuário para a substituição das pseudovariáveis no template passado no primeiro parâmetro.
- **Terceiro (opcional):** foi setado como `TRUE` para que o conteúdo do template, depois de renderizado, fosse retornado para a variável em vez de exibir diretamente na tela. Se tivesse sido setado como `FALSE` , seria impresso na tela o resultado da renderização.

Na *view*, atualize o conteúdo do bloco `<div id="body">...</div>` , conforme o código:

```
{list_entries}
<p>- <strong>{name}</strong> - {email}</p>
{/list_entries}

<hr/>

<h2>{no_users_title}</h2>

{no_users}
```

Com essa atualização, passaremos a mostrar uma nova lista. Mas, dessa vez, em vez de executar na *view* o loop para exibir os itens da lista, fazemos o loop no *controller* e mandamos para a lista o conteúdo pronto para ser exibido. Atualize a janela do browser para verificar o resultado.

Não há mistério em se trabalhar com template parser, pois ele nada mais é do que uma forma mais amigável de escrever as variáveis na *view*.

16.6 CONCLUSÃO

Neste capítulo, você viu uma forma simples de reduzir o volume de código PHP na view por meio do uso das pseudovariáveis do template parser. Aqui foi apresentado o parser nativo do CI, mas existem vários outros (como o Smart, por exemplo) disponíveis e compatíveis com PHP e CodeIgniter.

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-16-usando-template-parser>

Links úteis

- **Documentação oficial sobre a library Template Parsers:**
http://www.codeigniter.com/user_guide/libraries/parser.html

MANIPULANDO IMAGENS

"Se você quer chegar aonde maioria não chega, faça o que a maioria não faz." — Bill Gates

Neste capítulo, você vai aprender a utilizar cinco recursos de manipulação de imagem que o CodeIgniter disponibiliza nativamente. Esses recursos são:

- Redimensionamento
- Thumbnail
- Rotação
- Crop
- Marca d'água

Por meio de um exemplo prático, no qual faremos uso dos conhecimentos de upload de arquivos adquiridos no capítulo 10. *Upload, download e compressão de arquivos*, você criará uma aplicação que implementará os cinco recursos listados anteriormente.

Para iniciar o desenvolvimento, faça o download do código-fonte da estrutura inicial no link: <https://github.com/jlamim/livro-codeigniter/tree/master/CAP-17-manipulando-imagens-arquivos-base>

No diretório, você encontrará somente os arquivos da aplicação. O diretório `system` foi omitido, pois no início do livro já montamos uma estrutura para que não fosse necessário ter o diretório do CI em todos os exemplos. Caso tenha dúvidas quanto a essa estrutura, dê uma revisada no capítulo 1. *Introdução ao CodeIgniter*.

A estrutura é composta pelo *controller* `Base.php`, pela *view* `home.php` e pelo cabeçalho `views/commons/header.php` e rodapé `views/commons/footer.php` do exemplo. A *home* já possui o formulário inicial com o campo para upload de imagem e o botão para processar o formulário. A partir desses arquivos, será feita a implementação das funcionalidades de manipulação de imagens.

17.1 BIBLIOTECAS NATIVAS DO PHP SUPORTADAS

A manipulação de imagens no CI tem suporte a três grandes bibliotecas de manipulação de imagens do PHP: GD/GD2, NetPBM e ImageMagick. Essas bibliotecas fazem parte da compilação padrão do PHP, então se você utiliza sempre sua versão mais recente, não terá problemas durante o uso desses recursos.

Dessas 3 bibliotecas, a mais usada é a GD/GD2, e vamos utilizá-las durante este capítulo. Caso tenha problemas com a biblioteca GD, vá até o *Apêndice C* no final do livro, e veja como verificar se a biblioteca está instalada, ativa e também como instalar.

17.2 A LIBRARY IMAGE MANIPULATION

A library *Image Manipulation* é a biblioteca nativa do CI para manipulação de imagens. Ela possui recursos como:

- Redimensionamento
- Thumbnail
- Rotação
- Marca d'água

Para utilizar os recursos dessa library, você precisa inicializá-la. Para esse exemplo, faremos isso no arquivo `application/config/autoload.php` , adicionando `image_lib` ao array de `libraries`.

```
...
$autoload['libraries'] = array('image_lib');
...
```

Após inicializar a library, os seus métodos estarão disponíveis pelo objeto `$this->image_lib` .

17.3 CONFIGURANDO O UPLOAD DE IMAGEM

Como a intenção desse exemplo é trabalhar um cenário próximo do real, vamos implementar o upload de arquivos. Para isso, precisamos fazer uso da library *Upload*.

Tem dúvidas sobre o upload de arquivos? Que tal refrescar sua memória e visitar o capítulo 10. *Upload, download e compressão de arquivos* para revisar o assunto?

No arquivo `application/config/autoload.php` , adicione também a library *Upload* para que ela possa ser inicializada junto

com a aplicação.

```
...
$autoload['libraries'] = array('image_lib','upload');
...
```

Como vamos utilizar configurações padrões para o upload, usaremos um arquivo de configuração para armazená-las. Crie o arquivo `upload.php` no diretório `application/config`, e adicione nele o código adiante. Esse código nada mais é do que o array com as configurações do upload, que no capítulo 10. *Upload, download e compressão de arquivos* foram definidas diretamente em um método no *controller*.

```
<?php
$config['upload_path'] = './uploads/';
$config['allowed_types'] = 'gif|jpg|png';
$config['max_size']      = '4096';
$config['max_width']     = '2048';
$config['max_height']    = '1536';
?>
```

Essas configurações são respectivamente: o diretório onde serão armazenadas as imagens, os tipos de imagens permitidas, o tamanho máximo do arquivo (4Mb), e a largura e a altura da imagem (em pixels).

Com esse arquivo de configuração criado, ao ser inicializada a library, o CodeIgniter automaticamente passa para ela esses valores, sem a necessidade de utilizar o método `$this->upload->initialize($config)` ou `$this->load->library('upload', $config)`.

Como vamos fazer a manipulação dos arquivos dentro do diretório `uploads`, é necessário que ele tenha permissão de escrita.

17.4 PROCESSANDO O UPLOAD

O primeiro passo para a manipulação de imagens é a imagem a ser manipulada estar no servidor. Então, vamos montar a rotina de upload da imagem, para que possamos executar a sua manipulação.

Abra o *controller* `Base.php` , e crie um novo método chamado `Upload()` . Esse método será o responsável por executar todo o processo de upload, inclusive a sua validação. O método faz inicialmente o upload da imagem através do método `$this->upload->do_upload()` , checando o status do processo. Se processou com sucesso, ele emite uma mensagem de sucesso; caso contrário, emite a mensagem relacionada ao erro ocorrido.

```
public function Upload()
{
    if (!$this->upload->do_upload('image'))
    {
        $data['info'] = $this->upload->display_errors();
    }
    else
    {
        $data['info'] = "Imagem enviada com sucesso!";
        $data['info_upload'] = $this->upload->data();
    }
    $this->load->view('home', $data);
}
```

Agora que o método está pronto, vá até a *view* `home.php` , e adicione a rota para o upload, que nesse caso será `base/upload` . Essa rota será adicionada no atributo `action` do formulário, combinada com a função `base_url()` .

```
...
<form action="php=base_url('base/upload')?" method="POST" enctype=
"multipart/form-data">
    <div class="form-group">
        <label>Selecione uma imagem</label>
        <input type="file" name="image" />
    </div>
    <div class="form-group">
        <input type="submit" class="btn btn-success pull-right" value=
```

```
"Processar" />
</div>
</form>
...
```

Após atualizar a *view*, faça um teste e veja se o upload está sendo processado corretamente. Se estiver, podemos prosseguir; caso contrário, verifique se a codificação está correta e se a biblioteca GD está instalada e ativada.

17.5 CRIANDO UM THUMBNAIL DA IMAGEM ORIGINAL

Para a criação do thumbnail, vamos criar um novo método no *controller* Base , que vai se chamar `GenThumbnail()` . Esse método receberá como parâmetro um `array` , contendo as seguintes informações:

- *Path* da imagem original;
- Largura da imagem;
- Altura da imagem;
- *Aspect ratio* (manter a proporção da imagem ao gerar o thumbnail).

Criando o método `GenThumbnail` no controller

O método `GenThumbnail()` fará o processo de criação do thumbnail, retornando um `array` contendo:

- O status da operação;
- A mensagem (`null` em caso de sucesso).

```
...
private function GenThumbnail($config)
{
    $config['image_library'] = 'gd2';
    $config['create_thumb'] = TRUE;
    $config['new_image'] = "./uploads/thumbs/";
}
```

```

$this->image_lib->initialize($config);

if (!$this->image_lib->resize())
{
    $data['message'] = $this->image_lib->display_errors();
    $data['status'] = false;
}else{
    $data['message'] = null;
    $data['status'] = true;
}
$this->image_lib->clear();
return $data;
}
...

```

Além de receber um array como parâmetro, o método `GenThumbnail()` adiciona novos elementos a esse array . Os elementos adicionados são:

- `$config['image_library'] = 'gd2';` : define a biblioteca de manipulação de imagens utilizada, que nesse caso é a GD2;
- `$config['create_thumb'] = TRUE;` : determina que é para criar um thumbnail em vez de redimensionar a imagem original, já que o método `resize()` que será usado tem dupla funcionalidade;
- `$config['new_image'] = './uploads/thumbs/';` : define o caminho onde o thumbnail será salvo. Se não fosse informado, ele seria salvo no mesmo diretório da imagem original.

O diretório `thumbs` não é criado automaticamente, você precisará criá-lo dentro de `uploads` (lembrando de dar permissão de escrita). Caso contrário, o procedimento de criação do thumbnail retornará um erro relacionado à falta de permissão de escrita, ou até mesmo diretório inexistente.

Usamos o método `$this->image_lib->initialize($config);` para poder aplicar as configurações à biblioteca, uma vez que ela foi inicializada automaticamente por meio do arquivo `autoload.php`. Em seguida, temos o `if` que verifica se a criação do thumbnail ocorreu ou não, e então são geradas e retornadas as informações da operação.

Repare que, antes de retornar os dados, é chamado o método `$this->image_lib->clear()`, responsável por limpar as configurações aplicadas à *library Image Manipulation*. Assim, quando for utilizar outro método dela, não correrá o risco de herdar os dados do uso anterior.

Criando a estrutura da view

Feito isso, vamos voltar ao arquivo `views/home.php` para adicionar um checkbox no qual será possível escolher se o thumbnail será criado ou não. Abra o arquivo `views/home.php` e atualize a estrutura do formulário, inserindo o checkbox antes do botão **Processar**, conforme o código:

```
...
<form action="<?=base_url('base/upload')?>" method="POST" enctype=
"multipart/form-data">
  <div class="form-group">
    <label>Selecione uma imagem</label>
    <input type="file" name="image" />
  </div>
```

```

<div class="checkbox">
  <label>
    <input type="checkbox" name="thumbnail"> Criar thumbnail
  </label>
</div>
<div class="form-group">
  <input type="submit" class="btn btn-success pull-right" value=
"Processar" />
</div>
</form>
...

```

Atualizando o método Upload do controller

Com a *view* e o método `GenThumbnail()` prontos, é hora de atualizar o método `Upload()` no *controller* `Base`. Vamos adicionar ao método o código que vai verificar se é ou não para gerar o thumbnail. Se for para gerar o thumbnail, então chamamos o método `GenThumbnail()`; caso contrário, não fazemos nada relacionado ao thumbnail, deixando o fluxo seguir normalmente.

```

public function Upload()
{
  if (!$this->upload->do_upload('image'))
  {
    $data['info'] = $this->upload->display_errors();
  }
  else
  {
    $data['info'] = "Imagem processada com sucesso!";
    $data['info_upload'] = $this->upload->data();

    if($this->input->post('thumbnail'))
    {
      $configThumbnail['source_image'] = $data['info_upload']['f
ull_path'];
      $configThumbnail['maintain_ratio'] = TRUE;
      $configThumbnail['width'] = 75;
      $configThumbnail['height'] = 50;

      $thumbnail = $this->GenThumbnail($configThumbnail);

      if(!$thumbnail['status'])
      {
        $data['info'] .= "<br/>Não foi possível gerar o thumbnail

```

```

devido ao(s) erro(s) abaixo:<br />";
    $data['info'] .= $thumbnail['message'];
}
else
{
    $data['info_upload']['thumb_path'] = $data['info_upload']['
'file_path']."/thumbs/".$data['info_upload']['raw_name']."_thumb".
$data['info_upload']['file_ext'];
}
}
}

$this->load->view('home', $data);
}

```

Logo após inserir os dados do upload no array (`$data['info_upload'] = $this->upload->data();`), fazemos a verificação para saber se o thumbnail deve ser gerado. Com `$this->input->post('thumbnail')` , pegamos o valor do campo *Criar thumbnail*, que foi inserido no formulário anteriormente. Se o campo foi marcado, então o processo de geração do thumbnail é iniciado. A primeira parte dele é a definição das configurações a partir de um array :

- `$configThumbnail['source_image'] = $data['info_upload']['full_path'];` : define o *path* da imagem original;
- `$configThumbnail['maintain_ratio'] = TRUE;` : define que a proporção da imagem original deverá ser mantida ao gerar o thumbnail;
- `$configThumbnail['width'] = 75;` : define a largura do thumbnail;
- `$configThumbnail['height'] = 50;` : define a altura do thumbnail.

Se `$configThumbnail['maintain_ratio']` for definido como `TRUE` , será considerado o menor valor entre `$configThumbnail['width']` e `$configThumbnail['height']` para definir o tamanho do thumbnail, ficando o outro proporcional.

Após configurar os parâmetros, o método `GenThumbnail()` é chamado, retornando os dados para a variável `$thumbnail` . Após a execução do método `GenThumbnail()` , é verificado o status da operação. Se for `FALSE` , então `$data['info']` recebe uma atualização na mensagem a ser exibida para o usuário. Se for `TRUE` , é adicionada uma nova informação ao array `$data['info_upload']` , sendo essa informação o path do thumbnail.

Como o método `resize()` (nativo do CI) não retorna os dados da imagem processada como acontece com o upload, usaremos as informações do upload para poder definir o *path* do thumbnail. Por padrão, o CI adiciona sufixo `_thumb` antes da extensão da imagem original ao criar o thumbnail, então não fica complicado montar o *path* com os dados que são retornados pelo upload.

Veja no código a seguir que concatenamos `file_path` , `raw_name` , `_thumb` e `file_ext` , gerando assim o *path* do thumbnail.

```
$data['info_upload']['thumb_path'] = $data['info_upload']['file_path']. "/thumbs/" . $data['info_upload']['raw_name'] . "_thumb" . $data['info_upload']['file_ext'];
```

Após concatenar as informações e passá-las para o array de dados, o fluxo segue normalmente, e a *view* é chamada para exibir as

informações ao usuário.

17.6 REDIMENSIONANDO UMA IMAGEM

Para o redimensionamento de imagem, vamos utilizar uma rotina parecida com a do método `GenThumbnail()` , pois tanto a criação do thumbnail quanto o redimensionamento de imagens usam o mesmo método nativo da library *Image Manipulation*, o `resize()` . A diferença é que, para o redimensionamento, devemos definir `$config['create_thumb']` como `FALSE` . Além disso, salvaremos o arquivo redimensionado em um outro diretório, para que fique mais fácil visualizar e comparar as imagens geradas com as originais.

Do mesmo modo como criou o diretório `thumbs` , crie o diretório `resized` .

Criando o método `ResizeImage` no controller

No *controller* `Base` , crie o método `ResizeImage()` , conforme o código:

```
private function ResizeImage($config)
{
    $config['image_library'] = 'gd2';
    $config['create_thumb'] = FALSE;
    $config['new_image'] = "../uploads/resized/";

    $this->image_lib->initialize($config);

    if (!$this->image_lib->resize())
    {
        $data['message'] = $this->image_lib->display_errors();
        $data['status'] = false;
    }else{
        $data['message'] = null;
    }
}
```

```

    $data['status'] = true;
}
$this->image_lib->clear();
return $data;
}

```

O método `ResizeImage()` vai processar o redimensionamento da imagem. Para isso, é preciso definir os valores padrões para configurar a library e, em seguida, inicializá-la e checar através de um `if` se o método `resize()` foi executado com sucesso, definindo assim as mensagens a serem retornadas.

Poderia ter usado um único método tanto para criar o thumbnail quanto para redimensionar a imagem. Isso seria o ideal, já que os códigos são praticamente iguais. Mas, por questões didáticas, estão sendo usados dois métodos. Sinta-se desafiado a refatorar o código, e juntar `GenThumbnail()` e `ResizeImage()` em um único método.

Atualizando a estrutura da view

Vá até a *view* `home.php` e adicione mais três campos no formulário. Esses campos serão utilizados para definir a largura e a altura para o redimensionamento da imagem, e definir se a proporção em relação à imagem original deverá ser aplicada. Veja a seguir o código da estrutura do formulário após inserir os dois campos:

```

...
<form action="php echo base_url('base/upload'); ?" method="POST" enctype=
"multipart/form-data">
    <div class="form-group">
        <label>Selecione uma imagem</label>
        <input type="file" name="image" />
    </div>
    <div class="checkbox">

```

```

        <label>
            <input type="checkbox" name="thumbnail"> Criar thumbnail
        </label>
    </div>
    <div class="form-group">
        <label>Largura da Imagem após redimensionar (em pixels)</label>
    >
        <input type="number" name="width" class="form-control"/>
    </div>
    <div class="form-group">
        <label>Altura da Imagem após redimensionar (em pixels)</label>
        <input type="number" name="height" class="form-control" />
    </div>
    <div class="checkbox">
        <label>
            <input type="checkbox" name="ratio"> Manter proporção
        </label>
    </div>
    <div class="form-group">
        <input type="submit" class="btn btn-success pull-right" value=
"Processar" />
    </div>
</form>
...

```

Com o formulário preparado e o método de redimensionamento criado, é hora de implementar a rotina de verificação dentro do método `Upload()`, assim como fizemos quando foi implementada a criação de thumbnail. Agora, a verificação será a partir dos campos criados para largura e altura. Se pelo menos um dos dois campos (largura ou altura) tiver sido preenchido, então o redimensionamento deverá ser aplicado.

Atualizando o método Upload no controller

Vá até o método `Upload()` no *controller* Base e faça as alterações conforme o código a seguir:

```

public function Upload()
{
    if (!$this->upload->do_upload('image'))
    {
        $data['info'] = $this->upload->display_errors();
    }
}

```

```

else
{
    $data['info'] = "Imagem processada com sucesso!";
    $data['info_upload'] = $this->upload->data();

    if($this->input->post('thumbnail'))
    {
        $configThumbnail['source_image'] = $data['info_upload']['full_path'];
        $configThumbnail['maintain_ratio'] = TRUE;
        $configThumbnail['width'] = 75;
        $configThumbnail['height'] = 50;

        $thumbnail = $this->GenThumbnail($configThumbnail);

        if(!$thumbnail['status'])
        {
            $data['info'] .= "<br/>Não foi possível gerar o thumbnail devido ao(s) erro(s) abaixo:<br />";
            $data['info'] .= $thumbnail['message'];
        }
        else
        {
            $data['info_upload']['thumb_path'] = $data['info_upload']['file_path']. "/thumbs/". $data['info_upload']['raw_name']. "_thumb". $data['info_upload']['file_ext'];
        }
    }

    if($this->input->post('width') || $this->input->post('height'))
    {
        $configResize['source_image'] = $data['info_upload']['full_path'];
        $configResize['maintain_ratio'] = ($this->input->post('ratio')) ? TRUE : FALSE;
        $configResize['width'] = ($this->input->post('width')) ? $this->input->post('width') : null;
        $configResize['height'] = ($this->input->post('height')) ? $this->input->post('height') : null;

        $resize = $this->ResizeImage($configResize);

        if(!$resize['status'])
        {
            $data['info'] .= "<br/>Não foi possível redimensionar a imagem devido ao(s) erro(s) abaixo:<br />";
            $data['info'] .= $resize['message'];
        }
    }
}

```

```

    }
    else
    {
        $data['info_upload']['thumb_path'] = $data['info_upload']['file_path'].
        "/resized/".$data['info_upload']['raw_name'].$data['info_upload']['file_ext'];
    }
}
}

$this->load->view('home', $data);
}

```

Logo após o `if` usado para checar se o thumbnail seria criado, foi inserido um novo `if`, agora para verificar se a largura ou a altura foram informados. Em seguida, os valores dos parâmetros de configuração foram definidos no array `$configResize`. Fique atento aos valores de `maintain_ratio`, `width` e `height`, pois eles foram passados a partir de uma verificação condicional usando o operador ternário do PHP.

No caso de `maintain_ratio`, se o campo `ratio` foi marcado pelo usuário, então retorna `TRUE` para o parâmetro; caso contrário, retorna `FALSE`. Da mesma forma para os campos `width` e `height`, nos quais, no caso de preenchimento, é retornado o valor preenchido; caso contrário, retorna `null`.

A sequência do código é a mesma que foi usada na verificação do thumbnail, checando o status e definindo a mensagem para depois exibir a *view*.

17.7 O MÉTODO RESIZE()

Para as duas funcionalidades criadas até o momento, foi utilizado o método `resize()`. Este faz parte da library *Image Manipulation*, que permite o trabalho com imagens, fazendo uso de bibliotecas nativas do PHP, como por exemplo, a GD.

O método `resize()` é usado tanto para o redimensionamento de uma imagem quanto para a criação de uma nova em tamanho diferente, tendo seu uso mais aplicado para a criação de thumbnails. Ele possui uma série de parâmetros de configuração que são passados diretamente à library. Nos códigos anteriores, aplicamos essas configurações através de `$this->image_lib->initialize($config)`, já que a library foi inicializada por meio do arquivo `autoload.php`.

Na documentação da library, existe uma tabela contendo todos os parâmetros de configuração aceitos por ela, em cada um de seus métodos. Veja no link https://codeigniter.com/user_guide/libraries/image_lib.html. A seguir, explicarei os mais utilizados para o método `resize()`:

- `source_image` : é o *path* da imagem original.
- `new_image` : não possui valor padrão e aceita como valor o *path* onde a imagem gerada deve ser gravada. Se não for informada e `create_thumb` for `FALSE`, então a imagem original é redimensionada; mas se `create_thumb` for `TRUE`, então cria o thumbnail no mesmo diretório da imagem original.
- `width` : não possui valor padrão, aceita qualquer número inteiro e serve para determinar a largura da imagem.
- `height` : não possui valor padrão, aceita qualquer número inteiro e serve para determinar a altura da imagem.
- `create_thumb` : seu valor padrão é `FALSE`. Ele aceita `TRUE` ou `FALSE`, e serve para definir se a imagem será somente redimensionada ou se será criada uma nova.
- `maintain_ratio` : seu valor padrão é `TRUE`. Ele aceita `TRUE` ou `FALSE`, e serve para definir se o tamanho da nova imagem deverá ser proporcional ao tamanho da

original ou seguir os valores de `width` e `height` especificados.

17.8 ROTACIONANDO UMA IMAGEM

O processo de rotação não tem um uso tão frequente, mas em casos onde é necessário um editor de imagens com funcionalidades mais variadas, com certeza essa funcionalidade será utilizada. Fazer a rotação de uma imagem no CI é muito fácil, pois a library *Image Manipulation* possui um método específico para essa operação, o `$this->image_lib->rotate()` .

O método `$this->image_lib->rotate()` possui cinco opções para rotação da imagem. Veja quais são elas:

- `90` : gira a imagem 90° graus no sentido horário;
- `180` : gira a imagem 180° graus no sentido horário;
- `270` : gira a imagem 270° graus no sentido horário;
- `hor` : gira a imagem horizontalmente;
- `vrt` : gira a imagem verticalmente.

A rotação a ser aplicada deve ser passada para o método por meio de um array de configuração, como já foi feito com o método `$this->image_lib->resize()` . A chave para esse valor é `rotation_angle` .

Atualizando a estrutura da view

Vamos aplicar a rotação no exemplo que está sendo desenvolvido neste capítulo. Abra a `view home.php` , e adicione no formulário um select box, com as opções listadas anteriormente, para que o usuário possa escolher qual a rotação aplicar. O fragmento de código a seguir é exatamente o ponto onde esse select box deverá ser adicionado, logo antes do botão **Processar**.

```

...
<div class="form-group">
  <label>Girar Imagem?</label>
  <select name="rotation" class="form-control">
    <option value="">Não girar</option>
    <option value="90">90 graus</option>
    <option value="180">180 graus</option>
    <option value="270">270 graus</option>
    <option value="hor">Na Horizontal</option>
    <option value="vrt">Na vertical</option>
  </select>
</div>
<div class="form-group">
  <input type="submit" class="btn btn-success pull-right" value="P
rocessar" />
</div>
...

```

Criando o método RotateImage no controller

Com a opção adicionada na *view*, abra o *controller* Base.php para criarmos a rotina de rotação da imagem. Em Base.php, crie um método chamado RotateImage(), que vai ser muito semelhante ao ResizeImage(). Porém, a imagem rotacionada será criada no diretório rotated (você deverá criá-lo assim como criou os diretórios thumbs e resized). Também, é preciso mudar o método de resize para rotate e remover \$config['create_thumb'] = TRUE; , uma vez que esse parâmetro só é válido para o método resize() .

Veja a seguir como ficará o código do método RotateImage() :

```

private function RotateImage($config)
{
    $config['image_library'] = 'gd2';
    $config['new_image'] = "./uploads/rotated/";

    $this->image_lib->initialize($config);

    if (!$this->image_lib->rotate())
    {
        $data['message'] = $this->image_lib->display_errors();
        $data['status'] = false;
    }
}

```

```

    }else{
        $data['message'] = null;
        $data['status'] = true;
    }
    $this->image_lib->clear();
    return $data;
}

```

Atualizando o método Upload no controller

Com o método criado, é hora de executá-lo no processamento da imagem, após verificar se o usuário quer ou não rotacioná-la. Vá até o método `upload()` e adicione o código adiante logo após o fechamento do `if` que verifica se é para redimensionar a imagem. Esse código vai verificar se o campo `rotation` no formulário possui um dos cinco valores aceitos pelo método `rotate()`, e então processa a rotação, chamando o método `RotateImage()`.

```

...
if($this->input->post('rotation'))
{
    $configRotate['source_image'] = $data['info_upload']['full_path'];
    $configRotate['rotation_angle'] = $this->input->post('rotation');
    ;

    $rotate = $this->RotateImage($configRotate);

    if(!$rotate['status'])
    {
        $data['info'] .= "<br/>Não foi possível redimensionar a imagem devido ao(s) erro(s) abaixo:<br />";
        $data['info'] .= $rotate['message'];
    }
    else
    {
        $data['info_upload']['thumb_path'] = $data['info_upload']['file_path']."/rotated/".$data['info_upload']['raw_name'].$data['info_upload']['file_ext'];
    }
}
...

```

Até o momento, foram criadas quatro rotinas:

- **Upload:** processa o upload da imagem para o servidor para que seja possível realizar a manipulação da imagem;
- **Thumbnail:** cria um thumbnail a partir da imagem original enviada pelo usuário;
- **Redimensionamento:** redimensiona a imagem conforme o tamanho passado pelo usuário, e mantém ou não a proporção;
- **Rotação:** gira a imagem em três graus diferentes, ou então na horizontal ou vertical.

Antes de passar para a funcionalidade de recorte de imagem, vamos ver algumas particularidades sobre o método nativo `rotate()`.

17.9 O MÉTODO ROTATE()

Esse método faz parte da library *Image Manipulation*, que permite o trabalho com imagens, fazendo uso de bibliotecas nativas do PHP, como por exemplo, a GD. Tem por finalidade fazer o processo de rotação das imagens.

Ele possui uma série de parâmetros de configuração que são passados diretamente à library. Nos códigos anteriores aplicamos essas configurações por meio de `$this->image_lib->initialize($config)`, já que a library foi inicializada pelo arquivo `autoload.php`.

Na documentação da library, existe uma tabela contendo todos os parâmetros de configuração aceitos por ela, em cada um de seus métodos. Veja no link https://codeigniter.com/user_guide/libraries/image_lib.html. A seguir, explicarei os mais utilizados para o método `rotate()`:

- `source_image` : é o *path* da imagem original.
- `new_image` : não possui valor padrão e aceita como valor o *path* onde a imagem gerada deve ser gravada. Se não for informada e `create_thumb` for `FALSE` , então a imagem original é redimensionada; mas se `create_thumb` for `TRUE` , então cria o thumbnail no mesmo diretório da imagem original.
- `rotation_angle` : não possui valor padrão, aceita `90` , `180` , `270` , `vrt` , `hor` e serve para determinar o ângulo ou a posição da rotação da imagem.

17.10 RECORTANDO UMA IMAGEM

É muito comum vermos sistemas de upload de imagem que possibilitam fazer um corte dela, para que se adapte a um tamanho específico dentro do layout em que será aplicada. A library *Image Manipulation* possui o método `crop()` , que faz o corte de uma imagem a partir de coordenadas dos eixos *X* e *Y* e do tamanho desse corte, tudo passado como parâmetro para o método através de um array , como já foi usado nos métodos anteriores.

O ponto de corte de uma imagem é definido por meio dos eixos *X* e *Y* e do tamanho da imagem a ser gerada após o corte. Assim, se você tem uma imagem de 800x600 pixels, por exemplo, e quer fazer um corte central nela, esse corte terá o tamanho de 400x300 pixels. Para isso, você vai posicionar os eixos *X* e *Y* em 200 e 150 pixels, respectivamente, e definir que a largura será 400 pixels e a altura será 300 pixels. Com essas definições, bastará chamar o método `crop()` .

No exemplo que será criado agora, vamos definir pontos e tamanho fixos para o corte da imagem. Mas a forma mais utilizada de **CROP** é a combinação do processamento em *server-side* com alguma ferramenta visual para o *client-side*. Assim, o usuário pode

arrastar e redimensionar um bloco sobre a imagem para definir o ponto de corte que ele quer.

Atualizando a estrutura da view

Abra a `view` `home.php` novamente e adicione um checkbox que determinará se é ou não para gerar uma imagem recortada, assim como foi feito com a geração de thumbnail. Você pode adicionar esse checkbox logo antes do botão **Processar**.

```
...
<div class="checkbox">
  <input type="checkbox" name="crop"> Recortar imagem?
</div>
<div class="form-group">
  <input type="submit" class="btn btn-success pull-right" value="P
rocessar" />
</div>
...
```

Criando o método CropImage no controller

Agora que já adicionou a opção na view, vá até o controller `Base.php` e crie um método chamado `CropImage()`. Esse método será praticamente uma cópia do método `GenThumbnail()`, mas com as configurações de eixos *X* e *Y* setadas para 50, e o tamanho da imagem para 300x300 e sem a criação de thumbnail.

Lembre-se de criar o diretório *cropped* com permissão de leitura e escrita, para que as imagens recortadas sejam armazenadas corretamente nele.

```
private function CropImage($config)
{
```

```

$config['image_library'] = 'gd2';
$config['new_image'] = "./uploads/cropped/";
$config['width'] = 300;
$config['height'] = 300;
$config['x_axis'] = 50;
$config['y_axis'] = 50;

$this->image_lib->initialize($config);

if (!$this->image_lib->crop())
{
    $data['message'] = $this->image_lib->display_errors();
    $data['status'] = false;
}else{
    $data['message'] = null;
    $data['status'] = true;
}
$this->image_lib->clear();
return $data;
}

```

Qualquer tamanho de imagem que for utilizada com essa codificação, vai ser recortada em uma imagem de 300x300 pixel, a partir de 50 pixels da borda, tanto do topo quanto da lateral esquerda.

Atualizando o método Upload no controller

Após concluir a edição do método `CropImage()`, vá até o método `Upload()` e adicione a verificação para saber se deve ou não recortar a imagem. É o mesmo procedimento feito para o thumbnail, mas agora chamando o método `CropImage()`.

```

...
if($this->input->post('crop'))
{
    $configCrop['source_image'] = $data['info_upload']['full_path'];
};

$crop = $this->CropImage($configCrop);

if(!$crop['status'])
{
    $data['info'] .= "<br/>Não foi possível recortar a imagem devi

```

```

do ao(s) erro(s) abaixo:<br />;
    $data['info'] .= $crop['message'];
}
else
{
    $data['info_upload']['thumb_path'] = $data['info_upload']['file_path']. "/cropped/". $data['info_upload']['raw_name']. $data['info_upload']['file_ext'];
}
}
...

```

O crop da forma como foi montado aqui não é tão funcional, pois não permitirá o usuário determinar o tamanho do corte e da sua posição, correndo o risco de recortar uma parte da imagem que não deveria ser recortada. Como já foi falado anteriormente, é mais interessante combinar a funcionalidade de posicionamento da máscara de corte no layout da aplicação, para que o usuário possa definir essas informações.

Veja no link a seguir um exemplo completo de *cropping* usando CodeIgniter e jQuery.

UNIVERSIDADE CODEIGNITER - Recortando imagens com jCrop e CodeIgniter, em <http://www.universidadecodeigniter.com.br/recortando-imagens-com-jcrop-e-codeigniter>.

17.11 O MÉTODO CROP()

Esse método faz parte da library *Image Manipulation*, que permite o trabalho com imagens, fazendo uso de bibliotecas nativas do PHP, como por exemplo, a GD. Ele tem por finalidade fazer o processo de recorte das imagens.

Ele possui uma série de parâmetros de configuração que são passados diretamente à library,. Nos códigos anteriores aplicamos essas configurações através de `$this->image_lib->initialize($config)` , já que a library foi inicializada pelo arquivo `autoload.php` .

Na documentação da library, existe uma tabela contendo todos os parâmetros de configuração aceitos por ela, em cada um de seus métodos. Veja no link https://codeigniter.com/user_guide/libraries/image_lib.html. A seguir, explicarei os mais usados para o método `crop()` :

- `source_image` : é o *path* da imagem original.
- `new_image` : não possui valor padrão e aceita como valor o *path* onde a imagem gerada deve ser gravada. Se não for informada e `create_thumb` for `FALSE` , então a imagem original é redimensionada; mas se `create_thumb` for `TRUE` , então cria o thumbnail no mesmo diretório da imagem original.
- `width` : não possui valor padrão, aceita qualquer número inteiro e serve para determinar a largura da imagem.
- `height` : não possui valor padrão, aceita qualquer número inteiro e serve para determinar a altura da imagem.
- `x_axis` : não possui valor padrão, aceita inteiro e serve para definir o posicionamento do ponto de corte da imagem no eixo X.
- `y_axis` : não possui valor padrão, aceita inteiro e serve para definir o posicionamento do ponto de corte da imagem no eixo Y.

17.12 INSERINDO MARCA D'ÁGUA NA

IMAGEM

Inserir marca d'água em imagens é uma rotina comum em sites que querem assinar as imagens divulgadas por eles, principalmente sites de fotógrafos. Fazer esse processo no CodeIgniter é simples. Para isso, é utilizado o método `$this->image_lib->watermark()`.

Atualizando a estrutura da view

Abra a view `home.php` e adicione um novo checkbox para o usuário escolher se quer ou não inserir a marca d'água. Pode adicionar o checkbox como sendo a última opção, antes do botão **Processar**.

```
...
<div class="checkbox">
    <input type="checkbox" name="watermark"> Inserir marca d'água
</div>
<div class="form-group">
    <input type="submit" class="btn btn-success pull-right" value="P
rocessar" />
</div>
...
```

Criando o método ApplyWatermark no controller

No *controller* `Base.php`, crie um novo método chamado `ApplyWatermark()`, onde vamos executar o processo de aplicação da marca d'água. Esse método seguirá a lógica dos métodos anteriores, nos quais são setadas as configurações, executado o método nativo `watermark()` e retornado o status da operação.

```
...
private function ApplyWatermark($config){
    $config['new_image'] = './uploads/watermark/';
    $config['wm_type'] = 'overlay';
    $config['wm_overlay_path'] = './assets/images/watermark.png';
}
```

```

$config['wm_opacity'] = '50';

$this->image_lib->initialize($config);

if (!$this->image_lib->watermark())
{
    $data['message'] = $this->image_lib->display_errors();
    $data['status'] = false;
}else{
    $data['message'] = null;
    $data['status'] = true;
}
$this->image_lib->clear();
return $data;
}
...

```

Feito isso, é hora de ir até o método `Upload()` e adicionar a verificação para execução do método.

Atualizando o método Upload no controller

A rotina se repete mais uma vez. Por meio de um `if`, é feita a verificação do checkbox e, caso este esteja marcado, executa o método `ApplyWatermark()`.

```

...
if($this->input->post('watermark'))
{
    $configWM['source_image'] = $data['info_upload']['full_path'];

    $wm = $this->ApplyWatermark($configWM);

    if(!$wm['status'])
    {
        $data['info'] .= "<br/>Não foi possível aplicar a marca d'água na imagem devido ao(s) erro(s) abaixo:<br />";
        $data['info'] .= $wm['message'];
    }
    else
    {
        $data['info_upload']['wm_path'] = $data['info_upload']['file_path']. "/watermark/". $data['info_upload']['raw_name']. $data['info_upload']['file_ext'];
    }
}
}

```

...

Pronto! Agora que você já sabe como utilizar a library *Image Manipulation*, está pronto para fazer com que esse conhecimento evolua e você crie rotinas de tratamento de imagem ainda mais interessantes.

17.13 O MÉTODO WATERMARK()

Esse método faz parte da library *Image Manipulation*, que permite o trabalho com imagens, fazendo uso de bibliotecas nativas do PHP, como por exemplo, a GD. Ele tem por finalidade fazer o processo de aplicação de marca d'água nas imagens.

Ele possui uma série de parâmetros de configuração que são passados diretamente à library. Nos códigos anteriores, aplicamos essas configurações por meio de `$this->image_lib->initialize($config)` , já que a library foi inicializada pelo arquivo `autoload.php` .

Na documentação da library, existe uma tabela contendo todos os parâmetros de configuração aceitos por ela, em cada um de seus métodos. Veja no link https://codeigniter.com/user_guide/libraries/image_lib.html. A seguir, explicarei os mais usados para o método `watermark()` :

```
$config['new_image'] = './uploads/watermark/";  
$config['wm_type'] = 'overlay';  
$config['wm_overlay_path'] = './assets/images/watermark.png';  
$config['wm_opacity'] = '50';
```

- `source_image` : é o *path* da imagem original.
- `new_image` : não possui valor padrão e aceita como valor o *path* onde a imagem gerada deve ser gravada. Se não for informada e `create_thumb` for `FALSE` , então a imagem original é redimensionada; mas se `create_thumb` for `TRUE` , então cria o thumbnail no

mesmo diretório da imagem original.

- `wm_type` : o valor padrão é `text` , mas aceita também `image` , e serve para determinar o tipo de marca d'água a ser adicionada.
- `wm_overlay_path` : não possui valor padrão, aceita uma string e serve para determinar o *path* da imagem a ser inserida como marca d'água.
- `wm_opacity` : valor padrão é 50, aceita inteiros entre 1 e 100, e serve para definir a opacidade da imagem da marca d'água.

17.14 CONCLUSÃO

Neste capítulo, você aprendeu a usar os recursos de manipulação de imagens do CodeIgniter. Esses recursos são muito úteis quando você precisa desenvolver projetos que envolvem upload de imagem, e que o usuário precisa de ferramentas para a edição delas.

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-17-manipulando-imagens>

Links úteis

- **Documentação oficial sobre a library Image Manipulation:**
http://www.codeigniter.com/user_guide/libraries/image_lib.html

TRABALHANDO COM COMPOSER

"O sucesso é ir de fracasso em fracasso sem perder o entusiasmo."
— Winston Churchill

O Composer é uma ferramenta de gerenciamento de dependências para o PHP. Com poucas linhas de configuração, você define quais as bibliotecas precisa utilizar em seu projeto e o Composer se encarrega de baixá-las.

Vamos criar um pequeno projeto para aplicar o uso do Composer junto com o CodeIgniter. Assim como foi feito em capítulos anteriores, duplique o diretório `instalacao-ci`, e o renomeie para `trabalhando-com-composer`.

Ao abrir o diretório raiz da instalação, você encontrará um arquivo chamado `composer.json`. Esse arquivo é onde serão definidas quais bibliotecas devem ser importadas para dentro do projeto, como uma dependência.

```
{
  "description": "The CodeIgniter framework",
  "name": "codeigniter/framework",
  "type": "project",
  "homepage": "http://codeigniter.com",
  "license": "MIT",
  "support": {
    "forum": "http://forum.codeigniter.com/",
    "wiki": "https://github.com/bcit-ci/CodeIgniter/wiki",
    "irc": "irc://irc.freenode.net/codeigniter",
```

```

        "source": "https://github.com/bcit-ci/CodeIgniter"
    },
    "require": {
        "php": ">=5.2.4"
    },
    "require-dev": {
        "mikey179/vfsStream": "1.1.*"
    }
}

```

Repare que o arquivo `composer.json` já vem preenchido com várias informações que você pode atualizar com os dados do projeto, além de trazer algumas dependências configuradas, como a versão do PHP e a library `vfsStream`. Das informações trazidas no arquivo `composer.json`, somente a chave `require` é obrigatória. Na maioria dos projetos que utilizam o Composer, não tem mais nada a não ser as informações dessa chave.

18.1 ADICIONANDO, ATUALIZANDO E REMOVENDO DEPENDÊNCIAS

Vamos adicionar as dependências à library `PHPMailer`, uma biblioteca para envio de e-mails que pode ser usada em substituição à library de envio de e-mails nativa do CI (<https://github.com/PHPMailer/PHPMailer>).

```

{
    "description": "CodeIgniter Teoria na Prática",
    "name": "codeigniter/framework",
    "type": "example",
    "homepage": "http://livrocodeigniter.com.br",
    "license": "MIT",
    "support": {
        "forum": "http://forum.codeigniter.com/",
        "wiki": "https://github.com/bcit-ci/CodeIgniter/wiki",
        "irc": "irc://irc.freenode.net/codeigniter",
        "source": "https://github.com/bcit-ci/CodeIgniter"
    },
    "require": {
        "php": ">=5.2.4",
        "phpmailer/phpmailer": "~5.2"
    }
}

```

```
}  
}
```

Feito isso, você precisa abrir o arquivo `config/config.php` e ativar o autoload do Composer. Ele vai estar setado como `FALSE`, mude para `vendor/autoload.php`.

```
$config['composer_autoload'] = 'vendor/autoload.php';
```

Agora abra o terminal (UNIX) ou o prompt de comando (Windows) e, estando no diretório raiz do projeto, execute o seguinte comando:

```
composer install
```

Esse é o comando que fará a instalação das dependências. Ele pode demorar um pouco, pois será feito o download de cada uma das dependências informadas. Veja na figura a seguir o resultado da execução do comando:

```
MacBook-Pro-de-Jonathan:CAP-18-trabalhando-com-composer jlamim$ composer install  
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
- Installing phpmailer/phpmailer (v5.2.14)  
  Downloading: 100%  
  
phpmailer/phpmailer suggests installing league/oauth2-client (Needed for XOAUTH2 authentication)  
phpmailer/phpmailer suggests installing league/oauth2-google (Needed for Gmail XOAUTH2)  
Writing lock file  
Generating autoload files
```

Figura 18.1: Processo de instalação das dependências utilizando o terminal do Mac OS X

Ao terminar o processo, veja que foi criado o diretório `vendor` na raiz do projeto e, dentro dele, estão todas as dependências instaladas. Se precisar atualizar as dependências após ter instalado, basta editar o arquivo `composer.json` e, em seguida, executar o comando `composer update`.

Caso queira remover uma dependência, basta executar o comando `remove`, como no código a seguir:

```
composer remove phpmailer/phpmailer --update-with-dependencies
```

18.2 TESTANDO AS DEPENDÊNCIAS INSTALADAS

Ajustando o controller

Abra o arquivo `application/controllers/welcome.php`, e apague todo o conteúdo do método `index()`, pois vamos escrever uma rotina nova. A nova rotina será um envio de e-mail usando a library `PHPMailer`, que acabou de ser adicionada ao projeto pelo `Composer`.

O código a seguir carrega a library *PHPMailer*, define as variáveis de configuração para o envio do e-mail e verifica se este foi enviado, exibindo na tela a mensagem. Em caso de erro, exibe a descrição do erro ocorrido.

Substitua os valores de `setFrom` e `addAddress` por dados válidos para que você possa receber o e-mail.

```
public function index()
{
    $mail = new PHPMailer;

    $mail->setFrom('from@example.com', 'Mailer');
    $mail->addAddress('joe@example.net', 'Joe User');

    $mail->isHTML(true);

    $mail->Subject = 'Livro CodeIgniter';
    $mail->Body    = 'Estou estudando o <b>capítulo 18!</b>';

    if(!$mail->send()) {
        echo 'Email não enviado.';
        echo 'Erro: ' . $mail->ErrorInfo;
    } else {
        echo 'Email enviado';
    }
}
```

```
}  
}
```

18.3 CONCLUSÃO

Neste capítulo, você aprendeu a utilizar o Composer para gerenciar dependências em seu projeto, instalando a library *PHPMailer*. No próximo, você aprenderá recursos do CodeIgniter que vão otimizar o seu trabalho e poupar tempo durante a escrita de código.

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-18-trabalhando-com-composer>

Links úteis

- Site oficial do Composer: <https://getcomposer.org/>

POUPANDO TEMPO DE DESENVOLVIMENTO COM FUNCIONALIDADES NATIVAS DO CODEIGNITER

"Quando você aprende a lidar com sua mente, todo o resto fica fácil..." — Gabriel Goffi

Neste capítulo, veremos algumas funcionalidades do CodeIgniter que vão poupar um bom tempo de trabalho escrevendo código, o que vai permitir focar em outros pontos do projeto, como novas features, refatoramento do código para obter um código melhor e outras coisas.

19.1 TRABALHANDO COM URLS

O CI possui um helper chamado *URL*, e ele disponibiliza uma série de funções para lidar com URLs. Para utilizar esse helper, você deve carregá-lo no autoload, ou então no *controller*, usando `$this->load->helper('url')`.

A seguir, veremos algumas das principais funções que o helper *URL* disponibiliza.

site_url()

Essa função pode receber dois parâmetros, sendo o primeiro uma rota para alguma página do site, e o segundo (que é opcional) os valores `http` ou `https`. O resultado dessa função é a concatenação da URL definida na configuração com a rota passada e o protocolo, caso esse tenha sido informado.

Se você utilizá-la sempre que for adicionar um link na aplicação, e de repente precisar mudar o domínio, bastará fazer a atualização do domínio no arquivo `application/config/config.php`, que todos os links passarão a apontar para o novo domínio.

Veja o exemplo a seguir. Nele a variável `$link_https` recebe o retorno da função, que gerará o link para uma página que lista os produtos, forçando o link para o protocolo HTTPS. E a variável `$link_http` vai gerar o link para a mesma página de produtos, mas sem o segundo parâmetro, usando o domínio conforme estiver especificado no arquivo de configuração.

```
$link_https = site_url('produtos', 'https');  
$link_http = site_url('produtos');
```

Existe uma outra função que faz a mesma coisa, pois é um *alias* de `site_url()`, a função `base_url()`. Qualquer uma das duas que você usar gerará o mesmo resultado final.

current_url()

Essa função retorna a URL atual. Sempre que chamada, o seu retorno será a string contendo a URL da página em que o usuário está. Se, por exemplo, o usuário estiver acessando `http://www.domain.com/user/edit`, essa será a string retornada pela função.

```
current_url();
```

uri_string()

Essa função retorna o conteúdo da URL sem o domínio, ou seja, ignorando a URL passada no arquivo de configuração. Chamar essa função ao acessar `http://www.livrocodeigniter.com.br/user/edit` fará com que seja retornado `user/edit` apenas.

```
uri_string();
```

url_title()

Função muito importante, pois ela é responsável por converter uma string em formato de URL, substituindo os espaços em branco por `-` (traço), ou por outro caractere passado como parâmetro. Ela aceita três parâmetros, sendo o primeiro (obrigatório) a string a ser convertida, o segundo (opcional) o caractere usado para substituir os espaços em branco (por padrão, é utilizado o traço), e o terceiro (opcional) para definir se a string retornada deverá estar totalmente em letras minúsculas e seu valor é um booleano (`TRUE` ou `FALSE`).

Ele é muito útil quando você precisa montar uma URL para leitura de uma matéria, ou visualização de detalhes de um produto, por exemplo. Veja no exemplo adiante no qual vamos montar uma URL para exibição de um post, e teremos o título do post compondo a URL.

Usaremos não só `url_title()` , mas também `site_url()` , para obtermos a URL completa. Como resultado, teremos `http://www.livrocodeigniter.com.br/review/livro-codeigniter` .

```
$url_title = url_title('Livro CodeIgniter', '-', TRUE);  
$link = site_url('review/'.$url_title);
```

Nem sempre a string utilizada será composta apenas por caracteres simples, pode ser que ela tenha caracteres especiais. Para mantermos a URL bem definida, podemos usar em conjunto a função `convert_accented_characters()`, que faz parte do helper *Text* e veremos mais adiante ainda neste capítulo. Essa função substitui os caracteres especiais da string por caracteres simples.

O exemplo a seguir é como o apresentado anteriormente, mudando apenas o texto. O resultado será

```
http://www.livrocodeigniter.com.br/review/livro-codeigniter-teoria-na-pratica .
```

```
$url_title = url_title(convert_accented_characters('livro CodeIgniter'), '-', TRUE);  
$link = site_url('review/'.$url_title);
```

prep_url()

Essa função trata uma URL, verificando se ela possui o protocolo HTTP definido. Caso não haja, ela retorna a URL passada como parâmetro já com o HTTP.

O exemplo seguinte retornará

```
http://www.livrocodeigniter.com.br .
```

```
prep_url('www.livrocodeigniter.com.br');
```

redirect()

Essa função é muito útil e deve ser explorada o máximo possível quando for necessário redirecionar o usuário dentro da aplicação. Ela aceita três parâmetros, sendo o primeiro a rota para onde o usuário será redirecionado, o segundo (opcional) é o método de redirecionamento (`auto` , `location` or `refresh`), e o terceiro (opcional) o código de resposta HTTP.

Veja um exemplo em que, caso o usuário não esteja logado, redirecionamos para a tela de login e, se estiver logado, enviamos para a página de estatísticas:

```
if ($logged_in == FALSE){  
    redirect('/login');  
}  
  
redirect('/estatisticas', 'location', 301);
```

Essa função interrompe qualquer execução de código que esteja depois dela. No caso desse exemplo, se o usuário não estiver logado, ele é redirecionado para o login. Então, o próximo redirect é ignorado, pois a execução do código foi interrompida.

19.2 TRABALHANDO COM TEXTOS

O CI possui um helper chamado *Text*. Ele disponibiliza uma série de funções para lidar com textos. Para utilizar esse helper, você deve carregá-lo no autoload, ou então no *controller*, usando `$this->load->helper('text')`.

A seguir, veremos algumas das principais funções que o helper *Text* disponibiliza.

Limitador de palavras e caracteres

A função `word_limiter()` é a responsável por retornar o texto passado como parâmetro com um número limitado de palavras. Ela aceita três parâmetros. O primeiro é a string com o texto a ser limitado, o segundo (opcional) é o número máximo de palavras que o texto retornado deve conter, e o terceiro (opcional) é o caractere a ser exibido logo após a última palavra.

Se você só especificar o primeiro parâmetro, o limite de palavras será 100, que é o valor padrão dessa função.

Vamos usar a frase *"Eu sou capaz de utilizar o CodeIgniter de maneira produtiva, pois eu estou estudando e aplicando esse conhecimento agora."* para ilustrar o funcionamento da função. Limitaremos essa frase em 8 palavras. Após executar a função `word_limiter()` , a frase será *"Eu sou capaz de utilizar o CodeIgniter de"*.

```
$texto = "Eu sou capaz de utilizar o CodeIgniter de maneira produt  
iva, pois eu estou estudando e aplicando esse conhecimento agora."  
;  
$texto_limitado = word_limiter($texto, 8);
```

Do mesmo modo como podemos limitar o número de palavras de um texto, também podemos limitar o número de caracteres, da mesma maneira, com as mesmas variáveis. Entretanto, fazemos isso usando uma função com outro nome, que é a `character_limiter()` .

Vamos repetir o código mostrado anteriormente, com a mesma frase, limitando a 25 caracteres. Após executar a função `character_limiter()` , a frase será *"Eu sou capaz de utilizar o Code"*.

```
$texto = "Eu sou capaz de utilizar o CodeIgniter de maneira produt  
iva, pois eu estou estudando e aplicando esse conhecimento agora."  
;  
$texto_limitado = character_limiter($texto, 25);
```

Removendo caracteres especiais

Remover caracteres especiais é algo de que frequentemente precisamos em sistemas, principalmente quando é necessário gerar conteúdo para montar uma URL, conforme foi exemplificado anteriormente neste capítulo. Para isso, é usada a função `convert_accented_characters()` , que recebe como parâmetro o texto a ser tratado.

Ao executar essa função, os caracteres especiais do texto são substituídos por caracteres simples. Por exemplo, um "é" (letra e com acento agudo) passará a ser simplesmente "e" (letra e sem acento).

No exemplo adiante, vamos usar a frase *"É fácil aprender a usar o CodeIgniter"*, e após aplicar a função, ela será *"E facil aprender a usar o CodeIgniter"*.

```
$texto = "É fácil aprender a usar o CodeIgniter";  
$texto_sem_caracteres_especiais = convert_accented_characters($texto);
```

Censurando palavras

Mais uma função interessante e útil é a `word_censor()`, capaz de localizar um conjunto de palavras em uma `string` e censurá-las, substituindo por outra palavra ou conjunto de caracteres.

A função `word_censor()` aceita três parâmetros. O primeiro é a `string` com o texto a ser verificado, a segunda é o `array` com a lista de palavras a serem censuradas, e o terceiro (opcional) é a palavra ou caracteres que vão substituir as palavras localizadas no texto.

Vejamos um exemplo em que vamos censurar as palavras *"fácil"* e *"CodeIgniter"* na frase *"É fácil aprender a usar o CodeIgniter"*. Vamos substituir as palavras censuradas por `####`. Após executar a função, a frase ficará da seguinte forma: *"É #### aprender a usar o ####"*.

```
$texto = "É fácil aprender a usar o CodeIgniter";  
$censurar = array('fácil', 'CodeIgniter');  
$text_censurado = word_censor($texto, $censurar);
```

A variável `$censurar` é um `array`, e contém as palavras a serem censuradas (`fácil` e `CodeIgniter`). Ela é passada como segundo parâmetro para o método `word_censor()`, e assim temos

o retorno da string com a censura aplicada:

É ##### aprender a usar o #####

19.3 TRABALHANDO COM STRINGS

Algumas operações com strings são recorrentes em muitos sistemas. Para isso, o CI possui o helper *String*. Para utilizar esse helper, você deve carregá-lo no autoload, ou então no controller, usando `$this->load->helper('string')`.

Strings randômicas

Gerar strings randômicas é uma funcionalidade usada em vários sistemas, como por exemplo, para gerar tokens de acesso. No CodeIgniter, você pode usar a função `random_string()` para gerar strings randômicas.

Essa função aceita dois parâmetros. O primeiro é o tipo de string a ser gerada (`alpha` , `alnum` , `basic` , `numeric` , `nozero` , `md5` , `sha1`), e o segundo (opcional) é a quantidade de caracteres dessa string.

Para gerar uma string alfanumérica de 16 caracteres, você deve usar `random_string('alpha',10)`. Os tipos `alpha` e `alnum` geram strings numéricas e alfanuméricas, respectivamente, com caracteres maiúsculos e minúsculos misturados, além dos números, no caso de alfanuméricas.

O tipo `md5` tem o tamanho fixo de 32 caracteres, e o `sha1` o limite de 40 caracteres. Se você passar valores maiores que isso no segundo parâmetro, ele será ignorado e o padrão do tipo respeitado.

19.4 MAPEANDO DIRETÓRIOS

Quando um sistema precisa trabalhar com imagens e arquivos, estes são armazenados em diretórios no servidor. Para acessá-los posteriormente, é preciso mapear esses diretórios.

Para facilitar esse processo, o CI possui o helper *Directory* que, por meio da função `directory_map()`, faz o mapeamento completo de um diretório, passado como parâmetro para a função, que aceita três parâmetros. O primeiro é o *path* do diretório que deverá ser mapeado, o segundo (opcional) é para informar se o mapeamento será recursivo (`0` = recursivo, `1` = diretório atual), e o terceiro (opcional) é um booleano (`TRUE` ou `FALSE`) para informar se diretórios ocultos devem ou não ser ignorados.

Para utilizar esse helper, você deve carregá-lo no autoload, ou então no *controller*, usando `$this->load->helper('directory')`.

Vamos exemplificar o mapeamento de diretório com base na estrutura do CI. Executaremos o mapeamento para o diretório `application/languages` de forma recursiva. O retorno será um array com o mapa do diretório.

```
$map = directory_map('./application/languages/', FALSE, TRUE);
```

O array retornado conterá o mapeamento do diretório passado e de todos os diretórios dentro dele. Veja a seguir:

```
array(3) {
    ["portuguese/"]=> array(14) {
        [0]=> string(11) "db_lang.php"
        [1]=> string(15) "number_lang.php"
        [2]=> string(19) "pagination_lang.php"
        [3]=> string(10) "index.html"
        [4]=> string(14) "email_lang.php"
        [5]=> string(24) "form_validation_lang.php"
        [6]=> string(18) "migration_lang.php"
        [7]=> string(17) "calendar_lang.php"
```

```

        [8]=> string(12) "ftp_lang.php"
        [9]=> string(15) "upload_lang.php"
        [10]=> string(18) "unit_test_lang.php"
        [11]=> string(13) "date_lang.php"
        [12]=> string(15) "imglib_lang.php"
        [13]=> string(17) "profiler_lang.php"
    }
    [0]=> string(10) "index.html"
    ["english/"]=> array(1) {
        [0]=> string(10) "index.html"
    }
}

```

Se executássemos a função sem recursividade, apenas para o diretório atual \$map = directory_map('./application/languages/', TRUE, TRUE); , o retorno seria somente relacionado ao diretório principal, deixando de fora o conteúdo dos diretórios dentro do principal, passado como parâmetro.

```

array(3) {
    [0]=> string(11) "portuguese/"
    [1]=> string(10) "index.html"
    [2]=> string(8) "english/"
}

```

19.5 CONCLUSÃO

Neste capítulo, você aprendeu algumas funcionalidades nativas do CodeIgniter que podem ajudar nas tarefas diárias, poupando tempo e agilizando o processo de desenvolvimento. No próximo capítulo, você aprenderá um processo muito importante, que é a migração entre versões do CI. Faremos o processo de migração da versão 2.x para a 3.x.

Links úteis

- **Documentação oficial do CI sobre o helper URL:**
https://codeigniter.com/user_guide/helpers/url_helper.

[html](#)

- **Documentação oficial do CI sobre o helper Text:**
https://codeigniter.com/user_guide/helpers/text_helper.html
- **Documentação oficial do CI sobre o helper String:**
https://codeigniter.com/user_guide/helpers/string_helper.html
- **Documentação oficial do CI sobre o helper Directory:**
https://codeigniter.com/user_guide/helpers/directory_helper.html

MIGRANDO UM PROJETO DA VERSÃO 2.X PARA A 3.X

"Um homem inteligente sai de um buraco em que um homem sábio não cairia." — Provérbio Árabe

O CodeIgniter vem ganhando atualizações com certa frequência desde que seu desenvolvimento foi assumido pelo *British Columbia Institute of Technology*, e muitas mudanças foram feitas da versão 2.x para a 3.x. Mas fique calmo, você não vai precisar reescrever todo o código de seus projetos ao migrar para uma versão mais nova.

Antes de começar o processo de migração, faça um backup do projeto.

20.1 ATUALIZE O DIRETÓRIO SYSTEM

Como já foi falado, os arquivos do *core* do CI ficam no diretório `system`. Sendo assim, o que você precisa fazer é remover o diretório `system` atual, que corresponde à versão 2.x, e copiar para o diretório do projeto correspondente à versão 3.x.

Caso você tenha feito algum tipo de alteração em qualquer

arquivo do diretório `system` - essa prática não é recomendada -, copie esse arquivo para um local seguro, e depois compare com os novos arquivos inseridos para aplicar as modificações, pois pode ser que o arquivo que você alterou tenha sofrido mudanças nessa nova versão.

Não copie o diretório `system` da versão 3.x sobre o diretório da versão 2.x, pois isso pode causar bugs.

Após substituir o diretório `system`, você deverá substituir o arquivo `index.php` que fica na raiz do projeto.

20.2 ATUALIZE O NOME DAS CLASSES

No CodeIgniter 3.x, o nome de todas as classes (libraries, drivers, controllers e models) deve começar com letra maiúscula.

- **2.x:** `application/libraries/email.php`
- **3.x:** `application/libraries/Email.php`

Se você precisar estender alguma classe, logicamente vai precisar utilizar o prefixo `MY_`, e também deverá colocar o nome da classe com a primeira letra maiúscula.

- **2.x:** `application/libraries/MY_email.php`
- **3.x:** `application/libraries/MY_Email.php`

Fique atento, pois a primeira letra maiúscula só deve ser aplicada a classes. Diretórios, arquivos de configuração, helpers e outros permanecem com o nome todo em letras minúsculas.

20.3 ATUALIZE O ARQUIVO CONFIG/MIMES.PHP

Na nova versão, foram adicionados novos mime-types. Então, substitua o arquivo `config/mimes.php` pelo da nova versão.

20.4 ATUALIZE O ARQUIVO CONFIG/AUTOLOAD.PHP

O arquivo `config/autoload.php` sofreu uma pequena alteração, mas em vez de sobrescrevê-lo, você simplesmente deve remover a seguinte linha:

```
$autoload['core']
```

O autoload do core foi descontinuado na versão 1.4.1, e foi totalmente removido na versão 3.0.

20.5 MOVER AS ALTERAÇÕES DA CLASSE LOG OU EXTENSÕES

A classe `Log` passou a ser considerada como uma classe do *core* do CI, então foi movida para o diretório `system/core`. Se você estendeu essa classe, então precisa mudar a localização do arquivo de extensão.

- **2.x:** `application/libraries/Log.php` ->

`application/core/Log.php`

- **3.x:** `application/libraries/MY_Log.php` ->
 `application/core/MY_Log.php`

20.6 ATUALIZAÇÃO PARA AS NOVAS FEATURES DA LIBRARY SESSION

No CI 3.x, a `library Session` foi totalmente reescrita, e ganhou novas features e atualizações. Ela passou a utilizar drivers de armazenamento em vez de usar somente `COOKIES` criptografados.

Algumas opções de configuração foram removidas e outras adicionadas. Você pode ver todos esses detalhes na documentação oficial sobre *Sessions*. O link encontra-se no final do capítulo.

20.7 ATUALIZE O ARQUIVO CONFIG/DATABASE.PHP

Na versão 3.x, o *Active Record* foi renomeado para *Query Builder*. Então, no arquivo `config/database.php`, você deve renomear a variável `$active_record` para `$query_builder`.

```
$active_group = 'default';  
// $active_record = TRUE;  
$query_builder = TRUE;
```

20.8 SUBSTITUA OS TEMPLATES DE ERRO

Templates de erro no CI 3.x são considerados como *views*, e devem ser movidos para o diretório `application/views/errors`.

Também foi adicionado suporte para templates de erro em formato texto, utilizado para exibição em linha de comando. Com isso, é preciso separar esses templates. Então, dentro do diretório

`application/views/errors` , crie o diretório `html` e o diretório `cli` , assim você separará os templates nos diferentes formatos renderizados pelo CodeIgniter.

20.9 ATUALIZE O ARQUIVO CONFIG/ROUTES.PHP

Rotas utilizando :any

O CI sempre possibilitou o uso da representação `:any` , possibilitando obter qualquer tipo de caractere de um segmento da URI. Essa representação nada mais era do que a expressão regular `.+` , que acabava gerando um bug, pois combinava a `/` (barra) na identificação do conteúdo, sendo que ela é o delimitador de segmento da URI.

Para evitar bugs, `:any` agora representa a expressão regular `[^/]+` , que recupera todo o conteúdo menos a `/` (barra). Se você quer recuperar o conteúdo incluindo a barra, utilize `(.+)` . Mas se quer recuperar tudo, exceto a barra, use `(:any)` .

Diretórios, default_controller e 404_error

Na versão 2.x, havia uma falha ao informar a rota para o `$route['default_controller']` e `$route['404_override']` . Isso possibilitou que alguns usuários usassem a estrutura como `diretório/controller` em vez de `controller/method` .

No CodeIgniter 3.x, a rota agora é aplicada por diretório. Ou seja, se o seu controller estiver em um diretório dentro de `controllers` , esse deve ser informado.

Vamos exemplificar. Ao acessar o domínio `www.doma.in` , o controller padrão é o `Default` , que está na raiz do diretório

controllers .

```
$route['default_controller'] = 'Default';
```

Se o domínio fosse `www.doma.in/admin` , que corresponde à tela inicial de uma área para gerenciamento de conteúdo, o CI procuraria pelo controller `Admin` dentro de `application/controller` , e não encontrando retornaria um erro 404.

Para esse tipo de situação, é necessário criar rotas específicas, como já foi visto no decorrer do livro. Essa situação teria a seguinte rota:

```
$route['admin'] = "admin/Default";
```

20.10 FUNÇÕES E MÉTODOS COM MUDANÇA DO VALOR RETORNADO

Alguns métodos e funções tiveram o retorno alterado de `FALSE` para `NULL` quando o valor a ser retornado não existir. Veja quais são:

- Funções comuns
 - `config_item()`
- Classe Config
 - `config->item()`
 - `config->slash_item()`
- Classe Input
 - `input->get()`
 - `input->post()`
 - `input->get_post()`
 - `input->cookie()`
 - `input->server()`
 - `input->input_stream()`

- `input->get_request_header()`
- Classe Session
 - `session->userdata()`
 - `session->flashdata()`
- Classe URI
 - `uri->segment()`
 - `uri->rsegment()`
- Helper Array
 - `element()`
 - `elements()`

20.11 USO DO FILTRO DE XSS

Muitas funções, principalmente para tratamento de campos de formulário, utilizam esse filtro. Até a versão 3.x, o valor passado como parâmetro para ele era `FALSE`. Porém, agora foi alterado para `NULL`, e é informado de maneira dinâmica em `config/config.php`, pela variável `$config['global_xss_filtering']`.

Se você passa o valor booleano para o parâmetro `$xss_filter` de forma manual, ou sempre utilizou a configuração no arquivo `config/config.php`, não se preocupe, não precisará alterar nada.

Veja as funções que utilizam o filtro:

- **Input Library**
 - `input->get()`
 - `input->post()`
 - `input->get_post()`
 - `input->cookie()`
 - `input->server()`
 - `input->input_stream()`
- **Cookie Helper**

- `get_cookie()`

`$_GET` , `$_POST` , `$_COOKIE` e `$_SERVER` não são mais sobrescritos pelo método de filtragem quando este está ativado.

Filtro XSS em dados recuperados da URL

Outro ponto em que filtro XSS era usado e agora não é mais é no tratamento de conteúdo recuperado das URLs com os métodos da biblioteca *URI*. Junto com `$config['permitted_uri_chars']` , geravam uma proteção adicional, mas também geravam alguns bugs, e acabou sendo removida no CodeIgniter 3.0.

Para tratar conteúdo vindo da URL, você pode usar `$this->security->xss_clean()` antes de executar qualquer operação com o dado recuperado.

Filtro XSS na validação de formulário

Na validação de formulários era utilizada a regra `xss_clean` , porém ela validava dados de entrada, quando deveria validar os dados de saída. Como era uma prática comum entre os desenvolvedores que usavam CI, ela foi removida das regras de validação na versão 3.0, para que o seu uso fosse feito da maneira correta.

Use `$this->security->xss_clean()` para tratar os dados do formulário após aplicar a validação com `form_validation` , ou então crie uma regra de validação específica em que faça a aplicação do filtro XSS.

20.12 USO DE GET_POST()

O método `get_post()` da classe *Input* recupera os dados de uma requisição, buscando primeiro por `$_POST` e, em seguida, por `$_GET`, o que não combinava com o nome do método. Na versão 3.x, foi feita uma alteração, na qual primeiro é feita a busca por dados em `$_GET`, e depois em `$_POST`. Também foi adicionado um novo método, o `post_get()`, que faz a busca pelos dados de maneira inversa a `get_post()`.

20.13 ATUALIZAÇÃO DE MENSAGENS DO FORM_VALIDATION

Duas mudanças ocorreram que podem afetar projetos feitos em CI 2.x, e que usam arquivo de tradução para mensagens de validação de formulário.

A primeira mudança é no nome da chave no array `$lang`. Foi adicionado o prefixo `form_validation_`:

- **Antes:**

```
$lang['rule'] = ...
```

- **Depois:**

```
$lang['form_validation_rule'] = ...
```

A segunda mudança foi na estrutura da mensagem, na qual foi alterada a forma como a pseudovariável é escrita:

- **Antes:** `'The **%s** field does not match the %s field.'`
- **Depois:** `'The **{field}** field does not match the {param} field.'`

20.14 MUDANÇAS MENORES

Algumas outras mudanças, menores que as anteriores, foram aplicadas na versão 3.x e merecem alguma atenção durante o processo de migração.

directory_map()

A função `directory_map()`, responsável por mapear um diretório e retornar o seu conteúdo em um array, agora mostra uma barra à direita do nome do diretório, facilitando assim a sua identificação.

Método `drop_table` atualizado

O método foi atualizado, ganhando um segundo parâmetro e permitindo que seja adicionada a condicional `IF EXISTS`. Por padrão, esse parâmetro vem setado como `FALSE` no método.

```
// DROP TABLE `table_name`  
$this->dbforge->drop_table('table_name');  
  
// DROP TABLE IF EXISTS `table_name`  
$this->dbforge->drop_table('table_name', TRUE);
```

Biblioteca de envio de e-mails atualizada

A biblioteca de envio de e-mails nativa do CI foi atualizada, e agora o método `$this->email->send()` possui um parâmetro que, se setado como `TRUE`, limpa todos os demais parâmetros usados no envio de e-mails. Essa mudança faz com que automaticamente, ao enviar um e-mail com sucesso, os dados usados sejam apagados, permitindo o envio de um novo e-mail sem a necessidade de reinicializar o método.

```
if ($this->email->send(FALSE))  
{  
    // Parameters won't be cleared  
}
```

Não deixe o parâmetro de configuração `base_url` vazio

Se o parâmetro `$base_url` é deixado vazio, o CI assume para ele a URL base que está sendo acessada. Essa detecção automática tem implicações na segurança da aplicação, e não é confiável.

No CI 3.x, se você deixar o parâmetro vazio, não será retornado o nome do host, e sim o IP do servidor. Isso pode gerar alguns bugs no seu sistema se você depender de verificações da URL.

Se você precisar dar permissão a múltiplos domínios, usar HTTP ou HTTPS, você pode criar uma regra em `config/config.php` para solucionar o problema:

```
$allowed_domains = array('domain1.com', 'domain2.com');
$default_domain = 'domain1.com';

if (in_array($_SERVER['HTTP_HOST'], $allowed_domains, TRUE))
{
    $domain = $_SERVER['HTTP_HOST'];
}
else
{
    $domain = $default_domain;
}

if ( ! empty($_SERVER['HTTPS']))
{
    $config['base_url'] = 'https://'.$domain;
}
else
{
    $config['base_url'] = 'http://'.$domain;
}
```

20.15 ATENÇÃO COM AS FUNCIONALIDADES DESCONTINUADAS

Uma série de funcionalidades foram descontinuadas. Veja a seguir quais foram e como proceder com a alteração.

Biblioteca SHA1

A biblioteca foi removida, assim como o método `sha1()` da biblioteca *Encrypt*. Se você fazia uso desse método, substitua-o pelo método nativo do PHP, o `sha1()`.

Constante EXT

Ficou sem suporte nativo no PHP 4, e o CodeIgniter 3.0 removeu o suporte também. Use apenas `.php` para o nome dos arquivos.

Helper Smiley

Esse helper foi descontinuado, pois era uma feature do *EllisLab's ExpressionEngine*, e o CI não é mais mantido pela Ellislab.

Biblioteca Encrypt

Após vários problemas reportados em relação a vulnerabilidades, a biblioteca foi descontinuada e uma nova foi adicionada, a *Encryption*.

Essa nova biblioteca necessita da extensão *MCrypt*, ou PHP 5.3.3, e a extensão *OpenSSL*. Com essa biblioteca, temos de fato funções criptográficas.

A biblioteca *Encrypt* foi descontinuada, mas não removida do CI 3.x. Assim, ainda há compatibilidade com projetos que utilizam a biblioteca.

Biblioteca Cart

Essa biblioteca possibilitava a criação de carrinhos de compra de forma simples, e como era muito específica para o CI, decidiram por descontinuí-la. No CodeIgniter 3.1+, ela será totalmente removida.

Drivers mysql, sqlite, mssql e pdo/dblib

A extensão `mysql` foi descontinuada no PHP 5.5, e o CI assumiu isso na versão 3.x, substituindo pela extensão `mysqli`. Se você usa o driver de banco de dados MySQL, altere as configurações para que passe a utilizar `mysqli` ou `pdo/mysql`.

Os demais drivers dependem de extensões do PHP, e algumas não existem mais desde o PHP 5.3. Nas próximas versões do CI, esses drivers serão descontinuados, mas ainda não há uma definição exata em qual versão isso acontecerá. Recomendo, então, que você passe a usar os drivers mais avançados, como o `sqlite3`, `sqlsrv` ou `pdo/sqlsrv`.

do_hash()

A função `do_hash()` passou a ser um *alias* da função nativa do PHP `hash()`, e será removida do helper *Security* no CI 3.1+.

read_file()

A função `read_file()` passou a ser um *alias* da função nativa do PHP `file_get_contents()`, e será removida do helper *File* no CI 3.1+.

repeater()

A função `repeater()` passou a ser um *alias* da função nativa do PHP `repeater()`, e será removida no CI 3.1+.

trim_slashes()

A função `trim_slashes()` passou a ser um *alias* da função nativa do PHP `trim()` , e será removida no CI 3.1+.

form_prep()

A função `form_prep()` passou a ser um *alias* da função nativa do PHP `html_escape()` , e será removida futuramente.

Helper Email

Duas funções foram convertidas em *alias*es de funções nativas do PHP:

- `valid_email()` -> `filter_var()`
- `send_mail()` -> `mail()`

Esse helper está sendo descontinuado e será removido no CI 3.1+.

standard_date()

Devido a disponibilidade de constantes nativas do PHP, que combinadas com a função `date()` geram o mesmo resultado que `standard_date()` , ela está sendo descontinuada. Veja a seguir um exemplo de como substituir a função nativa do CI pela função nativa do PHP, obtendo o mesmo resultado:

- **Antes:**

```
standard_date(); // parâmetros padrões de standard_date('DATE_RFC822', now());
```

- **Depois:**

```
date(DATE_RFC822, now());
```

- **Antes:**

```
standard_date('DATE_ATOM', $time);
```

- **Depois:**

```
date(DATE_ATOM, $time);
```

Essa função será removida no CI 3.1.

nbs() e br()

Essas funções são *aliases* da função nativa `str_repeat()`, utilizada com ` ` e `
`, respectivamente. Por terem se tornado aliases de funções nativas do PHP, foram descontinuadas e serão removidas no CI 3.1+.

Configuração `anchor_class` da biblioteca `Pagination`

A biblioteca de paginação dá suporte a adição de atributos HTML para os links por meio do do parâmetro `attributes` nas configurações. Com isso, usar o parâmetro `attributes` nas configurações separado de `anchor_class` é sem sentido.

Assim, `anchor_class` foi descontinuado e será removido no CI 3.1+.

`random_string()`

A função utiliza `random` e `encrypt` para gerar as strings. Porém, isso acaba sendo um *alias* para `md5` e `sha1`, respectivamente, o que fez com que fosse descontinuada e removida no CI 3.1+.

`url_title()` com dash e underscore

Ao usar `url_title()`, você pode passar como parâmetro os valores `dash` e `underscore` como separadores de palavras, mas ela aceita qualquer caractere que você passar como parâmetro.

Sendo assim, o uso de `dash` e `underscore` foi descontinuado e será removido no CI 3.1+.

all_userdata()

O método `all_userdata()` da biblioteca *Session* retorna todos os dados da sessão do usuário que foram armazenados usando `$this->session->set_userdata()`. Porém, o método `$this->session->userdata()` faz a mesma coisa se não for passado nenhum parâmetro para ele, o que gera uma redundância entre esses métodos.

Por isso, `all_userdata()` foi descontinuado e será removido no CI 3.1+.

add_column() com cláusula AFTER

Caso você esteja utilizando o terceiro parâmetro, ao adicionar um campo na tabela usando o método `add_column()`, você precisará alterar o seu código.

O terceiro parâmetro foi descontinuado e será removido no CI 3.1+. Para que ele continue funcionando, você precisa fazer a alteração conforme o código a seguir:

- **Antes:**

```
$field = array(
    'new_field' => array('type' => 'TEXT')
);

$this->dbforge->add_column('table_name', $field, 'another_field');
```

- **Depois:**

```
$field = array(
    'new_field' => array('type' => 'TEXT', 'after' => 'another_field')
);
```

```
);
```

```
$this->dbforge->add_column('table_name', $field);
```

Em vez passar o parâmetro para o método `add_column()`, você deverá passá-lo como uma chave no array com as informações do campo.

Essa alteração é apenas para bancos de dados MySQL e CUBRID, os demais vão ignorar essa cláusula.

Métodos de roteamento `fetch_directory()`, `fetch_class()` e `fetch_method()`

Esses métodos fazem o mesmo que `CI_Router::$directory`, `CI_Router::$class` e `CI_Router::$method`, respectivamente, por isso foram descontinuados.

Você pode substituir as chamadas tradicionais por essas:

```
fetch_directory() -> $this->router->directory  
fetch_class() -> $this->router->class;  
fetch_method() -> $this->router->method;
```

20.16 CONCLUSÃO

Neste capítulo, você aprendeu como fazer a migração de um projeto do CI 2.x para o CI 3.x. É um processo um pouco trabalhoso, mas é necessário, pois manter o framework atualizado é muito importante. Sempre existem correções de segurança e melhorias nas últimas versões, em relação às versões anteriores.

Links úteis

- **Documentação oficial do CI sobre a Library Session:**
http://www.codeigniter.com/user_guide/libraries/sessions.html
- **Lista completa com toda a documentação sobre upgrade de versões:**
http://www.codeigniter.com/user_guide/installation/upgrading.html
- **Documentação oficial sobre migração entre a versão 2.x e 3.x:**
http://www.codeigniter.com/user_guide/installation/upgrading_300.html

MANTENDO A ESTRUTURA DE BANCO DE DADOS ATUALIZADA COM MIGRATIONS

"O fracasso é somente a oportunidade de começar de novo, de forma mais inteligente." — Henry Ford

Em algum momento durante o desenvolvimento ou pós-desenvolvimento de uma aplicação, pode ser necessário fazer mudanças na estrutura do banco de dados, alterando a estrutura original. Quando precisamos fazer isso com arquivos de código-fonte, podemos usar ferramentas de controle de versão (como o Git, por exemplo), para manter os arquivos atualizados para todos os envolvidos.

Para fazer isso com banco de dados, no CodeIgniter temos as *Migrations*, que são uma forma conveniente para que você possa alterar seu banco de dados de forma estruturada e organizada.

21.1 AJUSTANDO AS CONFIGURAÇÕES

As configurações das migrations são feitas em `application/config/migration.php`. Nesse arquivo são setadas as informações necessárias para as migrações, conforme a lista a

seguir:

`migration_enabled` — Ativa ou desativa as migrations.

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

`migration_path` — Caminho para o diretório onde se encontram os arquivos com a atualização do schema.

- **Valor padrão:** APPPATH.'migrations/'

`migration_version` — Versão atual do banco de dados utilizada.

- **Valor padrão:** 0

`migration_table` — Nome da tabela usada para armazenar o número da versão do schema.

- **Valor padrão:** migrations

`migration_auto_latest` — Ativa ou desativa a execução automática das migrations.

- **Valor padrão:** FALSE
- **Opções:** TRUE ou FALSE

`migration_type` — Tipo de identificador usado nos nomes dos arquivos.

- **Valor padrão:** timestamp
- **Opções:** timestamp ou sequential

21.2 A LÓGICA DE EXECUÇÃO DAS MIGRATIONS

Cada migration é composta por dois métodos, `up()` e `down()`, que são executados para upgrade e downgrade, respectivamente. Quando você cria a primeira migration, e atualiza as configurações para que ela seja executada, o CI automaticamente cria uma tabela adicional, chamada `migrations`. Essa tabela armazena um único registro, que é a última migration executada. Essa informação serve como referência para o CodeIgniter definir se vai ser executado um upgrade ou downgrade.

Ao criar uma outra migration e atualizar as configurações, informando que essa é a que deve ser executada, o CI vai verificar pelo parâmetro `migration_type` se a nova migration é posterior ou anterior à que está armazenada na tabela `migrations`. Se for anterior, então ele vai executar o método `down()` da migrate salva no banco de dados, e depois vai atualizar a informação da migrate para a que foi informada no parâmetro `migration_version`. Mas se ela for posterior, então ele executa o método `up()` da nova migrate e atualiza a informação no banco de dados.

21.3 PROJETO PRÁTICO

Vamos criar um novo projeto para aplicar o uso de migrations. Para isso, faça uma cópia do diretório `instalacao-ci` para o diretório onde está armazenando os arquivos do projeto.

Esse diretório possui as configurações iniciais necessárias para esse novo exemplo. Então renomeie-o para `exemplo-migrations`, e teste o acesso para ver se abrirá corretamente a tela padrão do CI.

Estrutura inicial do banco de dados

Esse exemplo ilustrará a atualização do banco de dados já existente. Para isso, execute a instrução SQL a seguir para criar a tabela necessária para iniciar o exemplo:

```
CREATE TABLE `livro_cap_21`.`posts` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `title` VARCHAR(255) NOT NULL,  
  `description` LONGTEXT NOT NULL,  
  `datetime` DATETIME NOT NULL,  
  PRIMARY KEY (`id`) );
```

Não se esqueça de atualizar o arquivo `application/config/database.php` com as instruções de conexão do banco de dados.

Tem dúvidas sobre a configuração da conexão com banco de dados? Que tal refrescar sua memória e revisitar o capítulo 14. *Trabalhando com banco de dados* para revisar o assunto?

Configurando a execução automática de migrations

Por meio das configurações feitas em `application/config/migration.php`, é possível fazer com que a atualização do schema do banco de dados seja feita sem a necessidade de executar o código manual em algum *controller*.

As configurações necessárias para que isso funcione são as que veremos a seguir.

Adicione o carregamento da library *Migration* ao autoload, em `application/config/autoload.php` :

```
$autoload['libraries'] = array('migration');
```

Em seguida, atualize o arquivo `application/config/migration.php`, conforme mostrado a seguir:

```
$config['migration_enabled'] = TRUE;  
$config['migration_auto_latest'] = TRUE;  
$config['migration_version'] = '20150217164137';
```

Com os parâmetros `migration_enabled` e `migration_auto_latest` setados como `TRUE`, e a library *Migration* sendo carregada automaticamente no `autoload`, o processo de atualização do banco de dados, seja para upgrade ou downgrade, acontecerá automaticamente ao carregar a aplicação.

O parâmetro `migration_version` pode ser setado de duas formas, dependendo do que foi passado para o parâmetro `migration_type`:

- `$config['migration_type'] = 'sequencial'`

Se `migration_type` for `sequencial`, o nome do arquivo deve ser um número inteiro sequencial (1, 2, 3, 4 etc.). Por exemplo: `001_add_comments.php`.

- `$config['migration_type'] = 'timestamp'`

Se `migration_type` for `'timestamp'`, o nome do arquivo deve ser no formato **YYYYMMDDHHIISS**. Por exemplo: `20150217164137_add_comments.php`

O uso do tipo `timestamp` faz com que seja mais fácil saber de quando exatamente é a alteração a ser executada, pois ele traz a data como identificação do nome do arquivo. Já o tipo `sequencial` não dá essa possibilidade, mas torna possível saber quantas alterações no banco foram feitas.

Você deve fazer a escolha conforme o que achar mais viável para a aplicação, e não alterar esse valor posteriormente. Isso porque pode afetar a lógica de execução dos upgrades e downgrades, tendo em vista que o CI utiliza esse parâmetro como base para definição de qual método deve executar.

Criando o arquivo com a atualização do banco de dados

O arquivo com as instruções sobre a migração deve ser criado em `application/migrations/`, e o seu nome deve ter o prefixo conforme o parâmetro `migration_type`, informado nas configurações.

Tomando por base a configuração apresentada anteriormente, o nome do arquivo ficaria da seguinte forma: `20150217164137_add_comments.php`.

Após salvar o arquivo, adicione o conteúdo:

```
<?php

defined('BASEPATH') OR exit('No direct script access allowed');

class Migration_Add_comments extends CI_Migration {

    public function up()
    {
        $this->dbforge->add_field(array(
            'id' => array(
```

```

        'type' => 'INT',
        'constraint' => 10,
        'unsigned' => TRUE,
        'auto_increment' => TRUE
    ),
    'post_id' => array(
        'type' => 'INT',
        'constraint' => 5
    ),
    'name' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
    ),
    'comment' => array(
        'type' => 'MEDIUMTEXT',
        'null' => FALSE,
    ),
    'date' => array(
        'type' => 'DATETIME',
        'null' => FALSE,
    ),
));
$this->dbforge->add_key('id', TRUE);
$this->dbforge->create_table('comments');
}

public function down()
{
    $this->dbforge->drop_table('comments');
}
}

```

Veja que o nome da classe é `Migration_Add_comments` . Ou seja, o nome do arquivo sem o prefixo `timestamp` , mas com `Migration` . Além disso, ela é estendida da classe `CI_Migration` para que possa herdar as funcionalidades.

Dentro da classe, temos os métodos `up` e `down` , respectivamente upgrade e downgrade do schema. Eles serão executados conforme a requisição, ficando essa verificação e execução por conta do CodeIgniter.

No método `up` , usamos `$this->dbforge->add_field()` para informar os campos que devem ser adicionados à tabela

`comments` , criada pelo método `$this->dbforge->create_table()` . O método `$this->dbforge->add_field()` recebe como parâmetro um array multidimensional, contendo todos os campos da tabela que deverá ser criada ao executar a migration.

Além dos campos, deve ser criada também uma chave na tabela. Isso é feito com o método `$this->dbforge->add_key()` , que recebe dois parâmetros. O primeiro é o nome da chave, e o segundo é um booleano que determina se ela deve ou não ser a chave primária.

No método `down` , é executado apenas a remoção da tabela `comments` , pelo método `$this->dbforge->drop_table()` , que recebe como parâmetro o nome da tabela a ser removida. A partir desse momento, o CI já está gerenciando automaticamente as atualizações no banco de dados a partir das migrations.

Assim, qualquer coisa que você precise modificar na estrutura, basta seguir a sequência de passos:

1. Atualizar o parâmetro `migration_version` nas configurações, para que o CI saiba qual arquivo de atualização executar;
2. Criar o arquivo com as instruções de atualização no diretório `application/migrations` .

Configurando a execução manual de migrations

Para que as migrations sejam executadas manualmente, é preciso desativar a execução automática, setando `FALSE` para o parâmetro `migration_auto_latest` nas configurações. Ao setar `FALSE` para esse parâmetro, você precisará executar a migração por meio de uma chamada de método em algum de seus *controllers*.

Abra o arquivo `application/controllers/welcome.php` , e atualize o código do método `index()` conforme o código a seguir:

```
public function index()
{
    if ($this->migration->current() === FALSE)
    {
        show_error($this->migration->error_string());
    }else{
        $this->load->view('welcome_message');
    }
}
```

Essa alteração faz com que a atualização do banco de dados seja executada quando o método `index()` do controller `welcome` for executado. Se a migration for executada com sucesso, a tela padrão do CI será carregada. Caso contrário, será exibida uma mensagem com o erro ocorrido, que é obtido usando o método `$this->migration->error_string()` .

Para testar a execução manual de uma migration, altere o parâmetro de configuração `migration_auto_latest` , conforme explicado anteriormente. Em seguida, altere `migration_version` para `20150217174212` . Feito isso, crie um novo arquivo de migration chamado
`20150217174212_add_comments_field_email.php` .

Coloque nesse arquivo o seguinte código:

```
defined('BASEPATH') OR exit('No direct script access allowed');

class Migration_Add_comments_field_email extends CI_Migration {

    public function up()
    {
        $this->dbforge->add_column('comments', array(
            'email' => array(
                'type' => 'VARCHAR',
                'constraint' => '100',
            )
        ));
    }
}
```

```

    }

    public function down()
    {
        $this->dbforge->drop_column('comments', 'email');
    }
}

```

Nessa migration, vamos adicionar uma coluna chamada `email` à tabela `comments`, que foi criada na atualização anterior. No método `up`, temos a chamada do método `$this->dbforge->add_column()`, que recebe como parâmetros o nome da tabela onde a coluna deve ser criada e o array com as informações da coluna, respectivamente.

Já no método `down()`, que será executado em caso de downgrade, é executado o método `$this->dbforge->drop_column()`, que remove a coluna criada no método `up()`. Ele recebe o nome da tabela e o nome da coluna, respectivamente, como parâmetros.

O método `down()` de toda migration deve ser sempre o oposto das ações do método `up()`. Na primeira migration criada, o método `up()` criava uma nova tabela, logo o método `down()` removia a tabela criada. Na segunda migration, o método `up()` adicionava uma coluna à tabela `comments`, enquanto no método `down()` essa coluna é removida.

Agora quando o método `index()` do *controller* `welcome` for acionado, será criada a coluna `email` na tabela `comments`.

21.4 CONCLUSÃO

Neste capítulo, você aprendeu a usar as migrations, um recurso

muito útil para manter a estrutura de banco de dados sempre atualizada no projeto, sem a necessidade de ter de ficar enviando arquivos `.sql` para toda a equipe, e correndo o risco de alguém esquecer de atualizar (ou até você mesmo).

Código-fonte

Faça o download do código-fonte completo desse exemplo no link:

<https://github.com/jlamim/livro-codeigniter/tree/master/CAP-21-mantendo-a-estrutura-do-banco-de-dados-atualizada-com-migrations>

Links úteis

- **Documentação oficial sobre Migrations:**
http://www.codeigniter.com/user_guide/libraries/migration.html

APÊNDICE A

"O seu tempo é limitado, não desperdice isso vivendo a vida de outra pessoa." — Steve Jobs

22.1 COMO ATIVAR O MOD_REWRITE NO APACHE EM UM SERVIDOR LINUX

Vamos ver neste apêndice um breve tutorial sobre como ativar o `mod_rewrite` em servidor Linux por meio de acesso SSH. Caso você não tenha permissão para acesso SSH ao servidor, deverá entrar em contato com o serviço de hospedagem e solicitar, ou o acesso, ou a ativação do `mod_rewrite`.

O `mod_rewrite` é um módulo do servidor Apache que permite fazer a reescrita de URLs, tornando possível termos URLs amigáveis, como por exemplo: `www.livrocodeigniter.com.br/contato` em vez de `www.livrocodeigniter.com.br/index.php?page=contato`.

Sem esse módulo instalado e ativado, você não consegue trabalhar com URLs amigáveis.

Passo 1 — Ativar o `mod_rewrite`

Acesse o servidor via SSH e digite o seguinte comando no terminal para habilitar o `mod_rewrite`:

```
sudo a2enmod rewrite
```

Passo 2 — Editar o arquivo de configuração

Abra o arquivo de configuração do Apache com o comando:

```
sudo gedit /etc/apache2/sites-available/default
```

Localize no arquivo a entrada `AllowOverride None`, alterando o seu valor para `AllowOverride All`, e salve o arquivo.

Passo 3 — Reiniciar o Apache

Para que o `mod_rewrite` funcione após as mudanças, você deve reiniciar o Apache com o comando:

```
sudo /etc/init.d/apache2 restart
```

IMPORTANTE

Caso você aplique essas configurações e o `mod_rewrite` não funcione, verifique se o serviço de hospedagem disponibiliza algum tipo de documentação (blog, wiki etc.) e busque informações sobre o tema. Se ele não tiver esses canais, entre em contato com o suporte.

22.2 LINKS ÚTEIS

- **Documentação oficial do Apache sobre o `mod_rewrite`:**
http://httpd.apache.org/docs/current/mod/mod_rewrite.html

APÊNDICE B

"Investir em conhecimento rende sempre os melhores juros." — Benjamin Franklin

No capítulo 9, *Gerenciando sessões com a library Session*, quando falamos sobre o gerenciamento de sessões, mostrei que é possível fazer o armazenamento de dados de sessão utilizando o Redis. Como é possível que você não tenha o Redis instalado no seu computador, neste apêndice vamos ver um tutorial bem rápido sobre como instalá-lo no Windows, no Linux e no Mac OS X.

Redis significa *REmote DIctionary Server*. É um banco de dados não relacional, também conhecido por NOSQL, criado por Salvatore Sanfiippo.

23.1 INSTALANDO O REDIS

Vamos ver a seguir os passos para instalar o Redis em ambiente Linux, Mac OS X e Windows.

Instalando o Redis no Linux e no Mac OS X

Para instalar o Redis no Linux, abra o terminal e execute a sequência de comandos a seguir:

```
wget http://download.redis.io/releases/redis-3.0.7.tar.gz
tar xzf redis-3.0.7.tar.gz
cd redis-3.0.7
make
```

No Mac OS X, o modo de instalação é parecido, mas em vez de usar o comando `wget`, você deverá utilizar o comando `curl`:

```
curl http://download.redis.io/releases/redis-3.0.7.tar.gz
tar xzf redis-3.0.7.tar.gz
cd redis-3.0.7
make
```

Instalando o Redis no Windows

Ainda não há suporte oficial para o Windows, mas existe uma versão experimental que pode ser utilizada. O uso dessa versão em ambiente de produção não é recomendado.

O download para instalação no Windows pode ser feito em <https://github.com/MSOpenTech/redis/releases/download/win-2.8.2400/Redis-x64-2.8.2400.zip>

Após o download, descompacte os arquivos e, em seguida, execute o arquivo `redis-server`. Depois, execute o arquivo `redis-cli`, sem fechar a janela aberta no passo anterior.

Para verificar se o Redis está sendo mesmo executado, na janela aberta ao executar o arquivo `redis-cli`, digite `ping`. Se o retorno for `PONG`, é porque o Redis está sendo executado corretamente.

23.2 SAIBA MAIS SOBRE O REDIS

Para saber mais sobre o Redis, dê uma olhada na sua documentação, em <http://redis.io/>.

Se quiser aprofundar os conhecimentos sobre Redis, a Casa do Código tem um livro sobre o assunto:

- **Armazenando dados com Redis:**
<https://www.casadocodigo.com.br/products/livro-redis>.

APÊNDICE C

24.1 BIBLIOTECA GD

A biblioteca GD é uma das bibliotecas responsáveis por tornar possível o trabalho com imagens no PHP. Com ela, você pode criar, editar e mesclar imagens, entre outras operações.

Verificando se a biblioteca GD está instalada

Para saber se a biblioteca GD está instalada, você deve criar um arquivo `.php` (pode chamar o arquivo de `test.php`) na raiz do servidor com o código a seguir. Esse código é responsável por imprimir na tela as informações do servidor Apache do PHP, os módulos instalados, banco de dados e outras informações.

```
<?php phpinfo(); ?>
```

Localize as informações sobre a biblioteca GD, que deverão aparecer de forma semelhante à figura a seguir:

→ gd

GD Support	enabled
GD Version	bundled (2.1.0 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
FreeType Version	2.4.12
T1Lib Support	enabled
GIF Read Support	enabled
GIF Create Support	enabled
JPEG Support	enabled
libJPEG Version	8
PNG Support	enabled
libPNG Version	1.6.6
WBMP Support	enabled
XBM Support	enabled

Directive	Local Value	Master Value
gd.jpeg_ignore_warning	0	0

gettext

GetText Support	enabled
-----------------	---------

hash

hash support	enabled
Hashing Engines	md2 md4 md5 sha1 sha224 sha256 sha384 sha512 ripemd128 ripemd160 ripemd256 ripemd320 whirlpool tiger128,3 tiger160,3 tiger192,3 tiger128,4 tiger160,4 tiger192,4 snfru snfru256 gost adler32 crc32 crc32b

Figura 24.1: Informações sobre a biblioteca GD

Caso a biblioteca GD esteja instalada, mas *GD Support* esteja marcado como *disabled* (veja a primeira linha da tabela de informações da figura anterior, que está marcado como *enabled*, ou seja, está ativada), você deverá checar o arquivo `php.ini`, pois pode ser que a linha que ativa a biblioteca esteja comentada.

No arquivo `php.ini`, localize a linha apresentada a seguir, e remova o `;` (ponto e vírgula). Assim, você estará descomentando a linha. Em seguida, salve o arquivo e reinicie o servidor Apache com o comando `$ /etc/init.d/apache2 restart`.

```
;extension=php_gd2.dll
```

Instalando a biblioteca GD no Linux

Para instalar a biblioteca GD no Ubuntu, você deverá executar o comando:

```
$ apt-get install php5-gd
```

Ou o comando:

```
$ sudo apt-get install php5-gd
```

Depois, deverá reiniciar o servidor Apache para que as configurações sejam aplicadas. O comando a seguir reinicia o Apache:

```
$ /etc/init.d/apache2 restart
```

24.2 LINKS ÚTEIS

- **Documentação oficial do PHP sobre a biblioteca GD e manipulação de imagens:**
<http://php.net/manual/en/book.image.php>

CONCLUSÃO

Parabéns por ter chegado ao final do livro, e por ter se dedicado e aprendido bastante sobre o CodeIgniter! Esse framework PHP lhe permitirá, daqui para a frente, desenvolver seus projetos de maneira mais rápida e eficiente.

Durante os capítulos deste livro, mais os 3 apêndices nos quais falamos sobre *mod_rewrite*, *Redis* e *biblioteca GD*, você pôde conhecer o CodeIgniter mais de perto, desenvolver duas aplicações, e ainda aprender a utilizar diversos recursos nativos dele por meio das libraries, dos helpers e dos exemplos apresentados.

Com o conhecimento adquirido durante o processo de leitura e prática, agora é a hora de você aprimorar esse conhecimento e fazê-lo evoluir, para que assim sua produtividade possa ser ainda maior. Evolua não somente nos conhecimentos do framework, mas do PHP, que é a sua linguagem base.

Não se esqueça de que a documentação, tanto da linguagem quanto do framework, é a sua melhor amiga, em todos os momentos. Antes de qualquer busca na internet ou em fóruns de discussão, dê uma pesquisada na documentação.

Espero que tanto o conteúdo deste livro quanto o do portal **Universidade CodeIgniter** possam ser de grande utilidade na sua caminhada profissional. Deixo aqui o meu sincero agradecimento por ter adquirido esta obra, e minha alegria por você ter decidido

aprimorar e/ou ampliar os seus conhecimentos.

25.1 LINKS ÚTEIS

Praticamente não temos referências bibliográficas em português sobre CodeIgniter, e isso me motivou a produzir este livro. Mas também me motivou a criar o **Universidade CodeIgniter**, um portal com tutoriais e dicas de CodeIgniter, totalmente em português, voltado para todos os níveis de desenvolvedores, desde o iniciante que quer aprender como funciona o framework até os mais avançados, com dicas e tutoriais dos mais diversos tipos.

A seguir, estão os links dos projetos relacionados a este livro e ao framework CodeIgniter, para os quais estou trabalhando diariamente na produção de conteúdo:

- **Site do livro:** <http://www.livrocodeigniter.com.br>
- **Códigos-fonte dos exemplos e projetos do livro:**
<http://github.com/jlamim/livro-codeigniter>
- **Portal Universidade CodeIgniter:**
<http://www.universidadecodeigniter.com.br>
- **GitHub Universidade CodeIgniter:**
<http://www.github.com/universidadecodeigniter>
- **Facebook:**
<http://www.facebook.com/UniversidadeCodeIgniter>