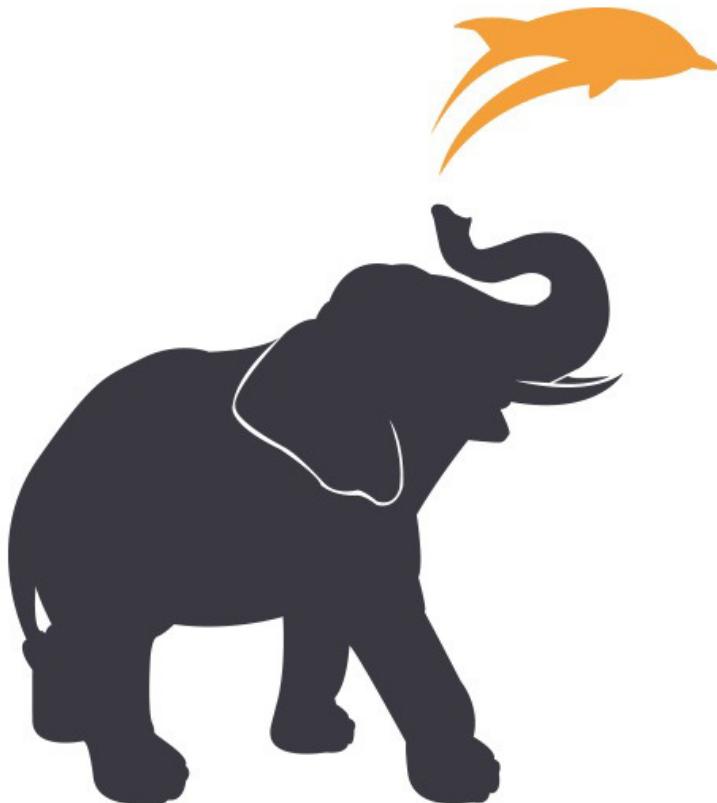


Desenvolvimento web com **PHP e MySQL**

Edição atualizada



Casa do
Código

EVALDO JUNIOR BENTO

© 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

[2017]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

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

www.casadocodigo.com.br

SOBRE O GRUPO CAELUM

Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código faz parte do Grupo Caelum, um grupo focado na educação e ensino de tecnologia, design e negócios.

Se você gosta de aprender, convidamos você a conhecer a Alura (www.alura.com.br), que é o braço de cursos online do Grupo. Acesse o site deles e veja as centenas de cursos disponíveis para você fazer da sua casa também, no seu computador. Muitos instrutores da Alura são também autores aqui da Casa do Código.

O mesmo vale para os cursos da Caelum (www.caelum.com.br), que é o lado de cursos presenciais, onde você pode aprender junto dos instrutores em tempo real e usando toda a infraestrutura fornecida pela empresa. Veja também as opções disponíveis lá.

ISBN

Impresso e PDF: 978-85-66250-30-5

EPUB: 978-85-66250-77-0

Você pode discutir sobre este livro no Fórum da Casa do Código: <http://forum.casadocodigo.com.br/>.

Caso você deseje submeter alguma errata ou sugestão, acesse <http://erratas.casadocodigo.com.br>.

AGRADECIMENTOS

A vida não é uma sequência de acontecimentos aleatórios. Muita coisa teve de acontecer em uma certa ordem para que este livro fosse possível, desde uma longínqua oportunidade de fazer o primeiro curso de informática, passando por curso técnico, faculdade, grupos de estudo, palestras, até as oportunidades de trabalhar com pessoas que me fizeram evoluir. Por isso, agradeço a Deus por ter me dado as ferramentas e o discernimento necessários para encontrar os caminhos que me trouxeram até este livro.

Este livro não seria uma realidade sem o apoio da minha amada esposa *Cássia Luz*. Quando conversamos sobre a oportunidade de eu escrever um livro, ela disse "**você tem de escrever um livro!**". E isso me motivou bastante a encarar essa tarefa tão difícil. Obrigado, Cassinha! (E obrigado também pela ideia da ilustração da capa, ficou demais!)

Agradeço também aos meus pais, Silvana e Evaldo, meus irmãos, Jenner e Antonio Paulo, e à sra. Creuza e ao sr. Mário, tios da Cássia, que me desculparam por não comparecer aos almoços e jantares em família porque estava escrevendo só mais algumas páginas.

Na verdade, eu sempre quis escrever um livro sobre programação e até cheguei começar alguns rascunhos, mas acabei deixando todos de lado. Até que um dia o Caio Ribeiro Pereira, que estava terminando seu livro de Node.js, me perguntou se eu não queria escrever um livro de Python e me apresentou ao Paulo

Silveira da Caelum/Casa do Código. Depois disso, trocamos alguns e-mails e comecei a escrever este livro de PHP e MySQL, finalmente colocando no "papel", o que pensei em fazer por muito tempo. Obrigado, Caio! E obrigado, Paulo!

Agradeço também aos leitores da primeira edição que enviaram diversas mensagens através da lista de discussão, e que me permitiram refinar e melhorar o conteúdo. Fica aqui um agradecimento especial ao Genivaldo Santos, que tem ajudado a todos respondendo dúvidas e indicando soluções na lista de discussão.

Não posso esquecer do pessoal do GCCSD (Grupo de Compartilhamento do Conhecimento Santos Dumont) que foi bastante incentivador quando eu ainda dava passos pequenos em TI e, ainda hoje, é uma fonte de ideias e discussões sem igual.

SOBRE O AUTOR

Evaldo Junior Bento trabalha com TI desde 2004. É desenvolvedor web com foco em boas práticas e padrões de desenvolvimento utilizando PHP como sua principal linguagem de desenvolvimento desde 2008.

Foi professor universitário por quatro anos, e ministrou disciplinas relacionadas a desenvolvimento de software e sistemas operacionais. É palestrante em eventos relacionados a software livre e desenvolvimento de software no Brasil e na Europa.

Possui formação em Processamento de Dados pela Fatec e Pós Graduação em Gestão Estratégica de TI. Mantém um blog sobre desenvolvimento e tecnologia em português (<http://blog.evaldojunior.com.br/>) e em inglês (<http://blog.evaldojunior.com/>), como também projetos em software livre no GitHub, em <https://github.com/InFog>.

PREFÁCIO

Muita gente que pensa em Web lembra logo de HTML, CSS e JavaScript. Claro, são as linguagens fundamentais dos navegadores. Mas elas só contam metade da história. Muita coisa precisa acontecer do outro lado da conexão, nos servidores. E, se você quer programar seriamente na Web, vai precisar dominar uma linguagem de programação que rode no servidor.

Nesse cenário, o PHP é a melhor escolha se você quer evoluir suas páginas estáticas HTML para páginas dinâmicas e sistemas complexos. Há muitas opções de linguagens de programação e de bancos de dados. Entretanto, a dupla PHP e MySQL é uma das mais importantes no mercado Web atual, com aplicações em sites de conteúdo, comércio eletrônico e até sistemas grandes, como Facebook.

Este livro é obrigatório para quem quer começar com PHP e MySQL. O Evaldo é um excelente autor com grande experiência, e conseguiu chegar em um livro fácil de ler e acompanhar, com uma escrita dinâmica e vibrante. Segue o modelo de livros objetivos da Casa do Código, e fará você entrar nesse mercado rapidamente e com bastante conhecimento.

Uma boa leitura, bons estudos e bem-vindo ao imenso e importante mundo do PHP.

Sérgio Lopes

— Instrutor e desenvolvedor na Caelum, autor do livro *A Web*

Mobile, também da editora Casa do Código

<http://sergiolopes.org>

Sumário

1 Introdução	1
1.1 Ganhando a web 2.0 com páginas dinâmicas	1
1.2 O navegador e a exibição de páginas web estáticas	2
1.3 E como funciona uma página dinâmica?	3
1.4 E onde entra o PHP e o MySQL?	3
1.5 Mas, por que escolher PHP e MySQL?	4
1.6 O que vou precisar para acompanhar este livro?	6
1.7 Sobre este livro	7
2 Instalando o PHP	9
2.1 PHP no Windows	10
2.2 PHP no GNU/Linux	14
2.3 PHP no Mac OS X	16
2.4 Após a instalação	20
2.5 Versões do PHP	22
3 O primeiro programa em PHP	24
3.1 Requisição, processamento e resposta	27
3.2 A minha página está mostrando a hora errada!	28

Sumário	Casa do Código
3.3 Resumo	30
3.4 Desafios	30
4 Construindo um calendário com PHP	32
4.1 Definindo nosso calendário	32
4.2 Começando o calendário	34
4.3 Usando funções	36
4.4 Pausa para os arrays	39
4.5 De volta ao calendário	43
4.6 Entendendo e se entendendo com os erros	47
4.7 Finalizando o calendário	50
4.8 Resumo	51
4.9 Desafios	52
5 Entrada de dados com formulário	53
5.1 O formulário de cadastro de tarefas	54
5.2 Entrada de dados	57
5.3 Acessando os dados da URL	59
5.4 Sessões no PHP	65
5.5 Resumo	71
5.6 Desafios	71
6 Tratamento de diferentes campos de formulários	73
6.1 Organizando o código em arquivos separados	74
6.2 Adicionando mais informações às tarefas	77
6.3 Resumo	81
6.4 Desafios	82
7 Acessando e usando um banco de dados	84

Casa do Código	Sumário
7.1 O banco de dados MySQL	85
7.2 PHPMyAdmin, administrando o banco de dados	86
7.3 Criando o banco de dados	90
7.4 Criando a tabela	91
7.5 Cadastrando e lendo os dados de uma tabela	95
7.6 Filtrando os resultados do SELECT	98
7.7 Resumo	99
7.8 Desafios	100
8 Integrando PHP com MySQL	101
8.1 PHP e MySQL	101
8.2 Conectando o PHP ao MySQL	102
8.3 Buscando dados no banco	108
8.4 Cadastrando as tarefas no banco	114
8.5 Cadastrando o prazo das atividades	120
8.6 Outra maneira de formatar as datas	126
8.7 Marcando uma tarefa como concluída	128
8.8 Resumo	131
8.9 Desafios	132
9 Edição e remoção de registros	133
9.1 Edição de tarefas	133
9.2 Operador ternário	145
9.3 Novidade no PHP 7: O operador "??"	146
9.4 De volta à edição de tarefas	147
9.5 Remoção de tarefas	152
9.6 Evitando o problema com a atualização de página	155
9.7 Resumo	156

9.8 Desafios	156
10 Validação de formulários	158
10.1 Validação na lista de tarefas	158
10.2 Entrada de dados usando POST	159
10.3 Validando o nome da tarefa	163
10.4 Adicionando o aviso de erro	167
10.5 Validando a data digitada	172
10.6 Expressões regulares	174
10.7 Validando o formulário de edição de tarefas	182
10.8 Resumo	185
10.9 Desafios	186
11 Upload de arquivos	188
11.1 Anexos para a lista de tarefas	188
11.2 Mudanças no banco de dados	190
11.3 Página com os detalhes das tarefas	192
11.4 O formulário para cadastrar anexos	195
11.5 Recebendo arquivos pelo PHP	197
11.6 Gravando os dados do anexo no banco dados	203
11.7 Exibindo os anexos	204
11.8 Removendo anexos	207
11.9 Resumo	210
11.10 Desafios	210
12 Lembretes de tarefas por e-mail	212
12.1 Definindo o e-mail de aviso	213
12.2 Unificando a configuração da aplicação com constantes	214

Casa do Código	Sumário
12.3 Adicionando a opção de aviso por e-mail	218
12.4 A função enviar_email()	220
12.5 As funções e seus parâmetros opcionais	222
12.6 Escrevendo o corpo do e-mail usando um arquivo com o template	224
12.7 Instalando uma biblioteca para enviar e-mails	228
12.8 Finalizando a função enviar_email()	229
12.9 Gravando os erros do envio de e-mails	235
12.10 Resumo	238
12.11 Desafios	239
13 Hospedagem de aplicações PHP	241
13.1 Sua aplicação para o mundo!	241
13.2 Escolhendo um servidor para hospedagem	242
13.3 Hospedagem com a Hostinger	243
13.4 Configurando a aplicação para a Hostinger	252
13.5 Enviando a aplicação para a Hostinger	252
13.6 Hospedagem no Jelastic da Locaweb	256
13.7 Configurando a aplicação para o Jelastic	263
13.8 Enviando a aplicação para o Jelastic	264
13.9 Resumo	267
13.10 Desafios	267
14 Programando com Orientação a Objetos	269
14.1 A classe Tarefa	273
14.2 O repositório de tarefas	285
14.3 Usando as novas classes na aplicação	291
14.4 Usando as classes no envio do e-mail	298

Sumário	Casa do Código
14.5 Criando a classe para os anexos	301
14.6 Usando as classes na exibição e edição das tarefas	306
14.7 Resumo	313
14.8 Desafios	314
15 Protegendo a aplicação contra SQL Injection e XSS	316
15.1 Proteção contra SQL Injection	317
15.2 Protegendo-se contra SQL Injection usando MySQLi	319
15.3 Exibindo e editando campos com aspas	323
15.4 Proteção contra XSS	324
15.5 Resumo	332
15.6 Desafios	333
16 Conhecendo o PDO	334
16.1 Exceptions, try e catch	337
16.2 Substituindo o MySQLi pelo PDO	339
16.3 Prepared Statements	342
16.4 Usando o PDO no restante da aplicação	346
16.5 Resumo	353
16.6 Desafios	354
17 Introdução ao MVC	355
17.1 O front-controller e os controllers	358
17.2 Organizando as views	363
17.3 Organizando as models	364
17.4 Ligando as pontas	365
17.5 Organizando o restante dos arquivos	370
17.6 Frameworks	373

17.7 Resumo	375
17.8 Desafios	376
18 Apenas o começo	377
18.1 Onde posso buscar mais informações?	379

Versão: 20.8.24

CAPÍTULO 1

INTRODUÇÃO

1.1 GANHANDO A WEB 2.0 COM PÁGINAS DINÂMICAS

Imagine a internet na qual você pode apenas consumir conteúdos, como se fosse um jornal, uma revista, ou ainda, um programa na televisão. Chato, né?

Mas quando estudamos as linguagens da web, como HTML e CSS, é isso que aprendemos. Usando apenas HTML, podemos construir sites que são como revistas e servem apenas para leitura, sem permitir interação com os internautas.

O segredo da chamada **web 2.0** é a capacidade de interação entre as pessoas e os serviços online. E para que esta interação seja possível, é necessário que os sites sejam capazes de receber informações dos internautas e também de exibir conteúdos personalizados para cada um dos usuários, como em uma rede social, por exemplo, ou de mudar seu conteúdo automaticamente, como um site de notícias, sem que o desenvolvedor precise criar um novo HTML para isso.

Estes dois tipos de sites são chamados de estático e dinâmico, respectivamente. Ambos são importantes para web como um todo,

cada um com suas vantagens e desvantagens para diferentes usos.

1.2 O NAVEGADOR E A EXIBIÇÃO DE PÁGINAS WEB ESTÁTICAS

Você já parou para pensar em tudo o que acontece quando você digita um endereço em seu navegador web? Tanto para páginas estáticas quanto para páginas dinâmicas, a história toda é mais ou menos assim:

1. O navegador pesquisa em qual servidor o site está hospedado.
2. O navegador vai até o servidor que responde no endereço solicitado e requisita o conteúdo. Algo como "Olá, servidor, eu gostaria do conteúdo para o site www.casadocodigo.com.br".
3. O servidor verifica se o endereço existe e se a página também existe em seu sistema de arquivos, e então retorna o arquivo para o navegador. Esse arquivo é, em geral, uma página em HTML.
4. Após receber o arquivo HTML, o navegador começa o trabalho de renderização, que é tipo desenhar os elementos da página nos locais corretos, para exibir a página para o usuário. É neste momento que o navegador também requisita arquivos de estilos (CSS), imagens e outros arquivos necessários para a formatação e exibição da página.

Quando se desenvolve páginas estáticas, este é basicamente todo o processo necessário para que o navegador exiba a página para o usuário. Chamamos de **estáticas** as páginas web que não mudam seu conteúdo, mesmo em uma nova requisição ao

servidor. Mais uma vez, é como pegar aquela sua revista ou livro guardado na gaveta. O conteúdo continua sempre o mesmo.

1.3 E COMO FUNCIONA UMA PÁGINA DINÂMICA?

O processo para páginas dinâmicas é muito parecido com o das páginas estáticas. A diferença é que a página será processada no servidor antes de ser enviada para o usuário. Este processamento no servidor é usado para alterar dinamicamente o conteúdo de uma página, seja ele HTML, CSS, imagens ou outros formatos.

Pense, por exemplo, em um site de um jornal. Em geral, este tipo de site contém algumas áreas destinadas às notícias de destaque, outras para notícias gerais e ainda outras para outros fins, como publicidade.

Quando o navegador solicita a página para o servidor, este montará o conteúdo antes de enviar para o navegador. Este conteúdo pode ser conseguido de algumas fontes. A mais comum é um banco de dados, onde, neste caso, as notícias ficam armazenadas para serem exibidas nas páginas quando estas forem solicitadas.

O resultado ainda será uma página HTML, mas ela poderá ser diferente quando o usuário fizer uma nova requisição ao servidor. Isso porque, neste caso, podem existir novas notícias para serem exibidas.

1.4 E ONDE ENTRA O PHP E O MYSQL?

PHP é uma ferramenta que possibilita o pré-processamento de páginas HTML. Dessa forma, o PHP consegue alterar o conteúdo de uma página, antes de enviá-la para o navegador, ou mesmo criar uma página nova a cada requisição. Além disso, ele também permite capturar entradas de dados do usuário, como formulários e outras formas de interação.

Já o MySQL é o banco de dados onde guardamos informações em estruturas no estilo de tabelas, sendo que cada linha da tabela é um novo registro. É em bancos como o MySQL que os sites de notícias, redes sociais etc. guardam suas informações para que depois sejam recuperadas e exibidas nas páginas.

A dupla PHP e MySQL se conhece há muitos anos e trabalha bem em equipe, sendo a principal responsável pelo conteúdo dinâmico na web, desde portais de notícia e conteúdos a lojas online, blogs e redes sociais.

1.5 MAS, POR QUE ESCOLHER PHP E MYSQL?

Quando fui professor universitário, ao final de uma aula de introdução ao PHP, um aluno veio até mim e perguntou o motivo de as empresas escolherem PHP e MySQL para desenvolver seus sites e até mesmo seus sistemas mais complexos. Ele me disse também que existem linguagens consideradas superiores ao PHP e bancos de dados que são tidos como melhores e mais eficientes que o MySQL.

Responder a esta questão não é fácil, pois existem diversos motivos para escolher esta ou aquela tecnologia. No caso da dupla PHP e MySQL, alguns motivos são:

- PHP nasceu para a web, e sua integração com servidores web dos mais diversos é simples. Isso facilita a manutenção de servidores e diminui as barreiras de entradas no mercado para novos produtos e serviços.
- PHP tem uma curva de aprendizado suave, comparada a outras linguagens. Isso possibilita que muitas pessoas aprendam o básico e consigam com isso testar suas ideias em projetos simples, sem muitos custos.
- PHP abstrai muitas das rotinas mais comuns da comunicação entre clientes (navegadores) e servidores, deixando assim o desenvolvedor se focar na lógica do que está construindo.
- PHP e MySQL são tecnologias livres, abertas e gratuitas. Mais uma vez, derrubando as barreiras dos custos de entrada no mercado para novas empresas, por exemplo.
- É fácil encontrar serviços de hospedagem que oferecem PHP e MySQL, dos mais simples e acessíveis aos mais complexos e caros.
- Serviços de hospedagem PHP e MySQL são mais baratos que serviços semelhantes para outras tecnologias, o que facilita bastante a entrada de novos desenvolvedores e reduz custos até para as grandes empresas.
- MySQL é leve e rápido, mesmo para quantidades razoavelmente grandes de dados.
- Como muitas pessoas começam a programar para web com PHP e MySQL, é natural que elas continuem com

PHP e MySQL quando evoluem na área e em suas carreiras.

1.6 O QUE VOU PRECISAR PARA ACOMPANHAR ESTE LIVRO?

Para desenvolver softwares, são necessárias algumas ferramentas. Neste livro, farei uso e indicarei apenas ferramentas em software livre e/ou código aberto que os leitores poderão instalar em seus computadores. Claro que os leitores mais experientes podem usar as ferramentas que já conhecem e com as quais se sintam mais confortáveis, apenas se certificando de fazer as devidas adaptações quando necessário.

No geral, tudo o que será necessário é um computador com o ambiente WEB com PHP e MySQL instalados e configurados, um editor de arquivos de texto e um navegador WEB para testar as páginas que serão criadas.

Uma dica importante para quem busca aprender uma nova linguagem de programação, ou mesmo a primeira linguagem de programação, é reservar tempo para estudar e praticar bastante. Se você conseguir separar um certo tempo por dia e realmente se dedicar à leitura e à prática dos exercícios propostos, rapidamente se sentirá mais confortável com PHP e com o ambiente WEB. Isso lhe dará conceitos gerais para desenvolvimento de páginas dinâmicas, até mesmo usando outras linguagens.

Ou seja, um dos requisitos para o estudo será mesmo o tempo. E quanto mais tempo você conseguir dedicar aos estudos, mais conseguirá absorver novos conhecimentos e mais rápido

conseguirá desenvolver suas aplicações.

1.7 SOBRE ESTE LIVRO

A ideia central deste livro é oferecer a oportunidade de o leitor começar a desenvolver suas primeiras páginas dinâmicas utilizando a linguagem PHP associada ao banco de dados MySQL. O livro apresenta uma experiência de aprendizado que pode (e deve) ser aplicada não somente ao PHP, mas também a quaisquer outras tecnologias para desenvolvimento de aplicações, sejam elas web ou não.

Durante os capítulos, os exemplos são construídos aos poucos e alguns erros são encorajados, além de haver algumas reescritas e melhorias em códigos que já funcionam. Porém, estes podem sofrer por não utilizarem técnicas que simplificam a lógica e garantem maior facilidade para futuras alterações.

Este livro não é um guia de referência para PHP e MySQL e, assim sendo, não apresenta listas de funções e bibliotecas disponíveis para estas tecnologias. O foco aqui é realmente um processo de aprendizado por meio da construção gradual de aplicações e assimilação dos conceitos.

Estudantes de cursos relacionados a desenvolvimento de sistemas, curiosos estudando programação para web e hobistas podem se beneficiar grandemente do conteúdo deste livro. Porém, desenvolvedores mais avançados que desejam apenas um guia de referência para tirar aquela dúvida sobre uma função ou outra da linguagem podem não encontrar benefícios nestas páginas.

Web designers com experiência em HTML e CSS que desejam

aprender a desenvolver para back-end também podem se beneficiar bastante do conteúdo deste livro. Mesmo sendo focado no iniciante, o livro busca trazer conteúdo atualizado com as práticas mais recentes do PHP e seu ambiente.

Sempre que estiver com alguma dúvida, não deixe de perguntar no fórum da Casa do Código. Ele está em <http://forum.casadocodigo.com.br/>. O fórum é um espaço aberto para todos os leitores do livro trocarem ideias sobre os exercícios e até sobre outros tópicos relacionados ao PHP e às tecnologias contidas aqui.

O fórum também pode ser usado para discutir outros tópicos relacionados ao PHP em geral, não se limitando ao contexto do livro, além de outras tecnologias tratadas em outros livros da Casa do Código. No fórum, todos podem perguntar e responder, então não deixe de dar uma passadinha por lá para tirar as suas dúvidas e também para ajudar os demais participantes/estudantes.

Os exemplos de código usados neste livro podem ser encontrados no GitHub em <https://github.com/InFog/phpmysql>.

Agora, acomode-se na cadeira e bons estudos!

CAPÍTULO 2

INSTALANDO O PHP

O PHP é uma linguagem que pode ser usada para várias tarefas diferentes, não apenas para gerar páginas web. O PHP pode, por exemplo, ser utilizado para escrever scripts em linha de comando, automatizar tarefas e, até mesmo, escrever aplicações desktop utilizando o PHP-GTK, ou fazer um emulador de Gameboy (<https://github.com/gabrielcouto/php-terminal-gameboy-emulator>).

Neste livro, o foco do uso do PHP é a construção de aplicações web. Por isso será necessário fazer a instalação do PHP em conjunto com um servidor web e também um banco de dados.

Existem diversas maneiras de se instalar o PHP e as demais ferramentas necessárias para acompanhar este livro, e também para trabalhar com PHP no dia a dia. Uma maneira é instalar cada ferramenta separadamente e depois fazer as configurações para que elas trabalhem em conjunto. Isso nos daria um trabalho enorme antes mesmo de começarmos a aprender a linguagem.

Outra maneira mais simples é utilizando um pacote tudo-em-um com as ferramentas já configuradas. Um desses pacotes é o XAMPP. Ele contém tudo o que é preciso para começar a programar em PHP sem ter de se preocupar em como instalar e

configurar um servidor web e um banco de dados.

2.1 PHP NO WINDOWS

Para instalar o XAMPP no Windows, acesse a página de downloads no site do XAMPP em https://www.apachefriends.org/pt_br/download.html. Depois, clique na opção Download da versão 7.x, como na figura a seguir:

The screenshot shows the Apache Friends website's download section. At the top, there are links for Apache Friends, Download, Extensões, Hosting, Comunidade, Sobre, a search bar, and a language switcher set to PT BR. Below this, a large heading says "Download". A text box states: "O XAMPP é uma distribuição Apache fácil de instalar contendo PHP, MariaDB e Perl. Basta fazer o download e iniciar o instalador. É assim tão fácil." To the right, a sidebar titled "Documentação/Perguntas Freqüentes" contains text about documentation and links to Linux, Windows, and OS X FAQ. Below the sidebar, a section titled "Add-ons and Themes" is visible. The main content area displays three download options for XAMPP 7.0.1:

Versão	O que está incluído?	Soma de verificação	Tamanho
5.5.30 / PHP 5.5.30	O que está incluído?	md5 sha1	Download (32 bit) 106 Mb
5.6.15 / PHP 5.6.15	O que está incluído?	md5 sha1	Download (32 bit) 108 Mb
7.0.1 / PHP 7.0.1	O que está incluído?	md5 sha1	Download (32 bit) 116 Mb

Figura 2.1: Página de download do XAMPP com link para o XAMPP para Windows

O instalador tem um pouco mais de 100MB. Aproveite para tomar um café enquanto o download é finalizado. Após seu término, abra o instalador:

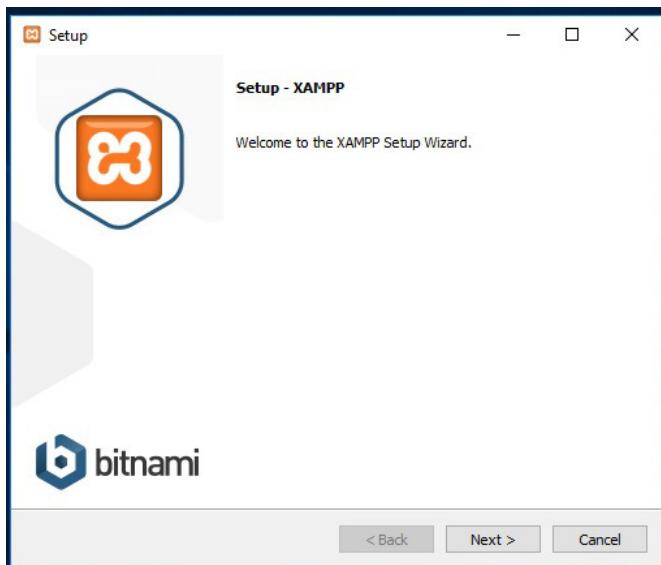


Figura 2.2: Instalador do XAMPP

Na tela de seleção, podemos desmarcar as opções *FileZilla*, *Tomcat*, *Mercury* e *Perl*, pois não vamos usar essas ferramentas no decorrer do livro.

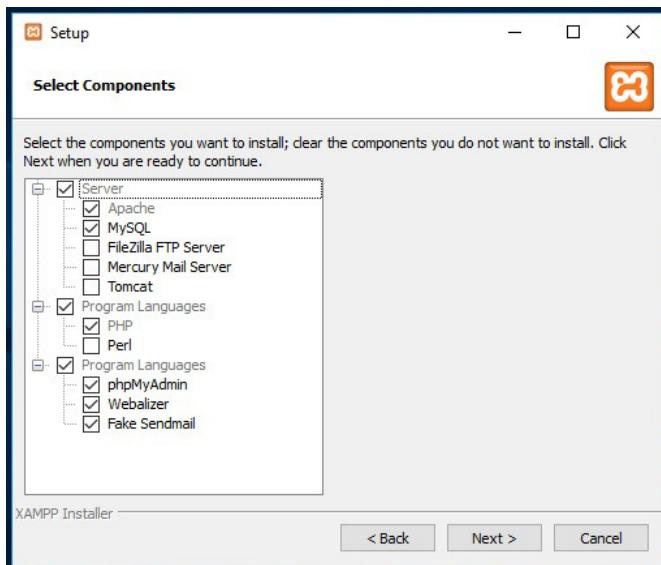


Figura 2.3: Removendo componentes não necessários no instalador

O restante do instalador pode ser completado usando apenas o botão Próximo ou Next . Depois de instalar, clique em Finalizar ou Finish , deixando a opção de abrir o painel do XAMPP marcada:



Figura 2.4: Finalizando o a instalação do XAMPP

No painel que abrir, clique no botão `start` apenas do serviço chamado **Apache**:

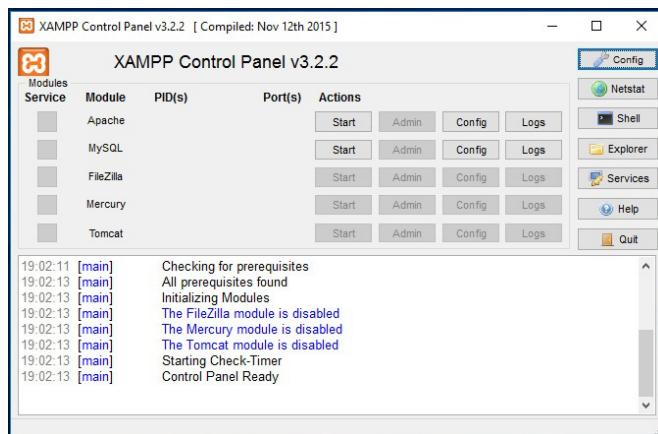


Figura 2.5: Painel do XAMPP, use a opção de iniciar o Apache

O Apache é o servidor web que será responsável por entregar as páginas quando o navegador solicitar. Pode ser que seja necessário liberar o Apache no **firewall** do Windows:



Figura 2.6: Liberar o Apache no Firewall do Windows

O XAMPP é instalado por padrão na pasta `C:\xampp\`.

2.2 PHP NO GNU/LINUX

Se você utiliza GNU/Linux, use o gerenciador de pacotes da sua distribuição favorita para instalar o PHP. É bem fácil usar o **apt-get** no Debian e no Ubuntu, ou o **yum** nos derivados do Red Hat e do Fedora. Apenas procure pelos pacotes do PHP e do Apache, e faça a instalação.

Para instalar o PHP no Debian/Ubuntu, use o comando a seguir. Isso instalará o PHP versão 5, Apache e o MySQL e deixará tudo pronto para usar.

```
sudo aptitude install php5 php5-mysql apache2  
libapache2-mod-php5 mysql-server
```

Caso esse comando dê algum erro, é por que a versão da distribuição que você está usando provavelmente oferece o PHP versão 7. Neste caso, o comando para instalação é o seguinte:

```
sudo aptitude install php7.0 php7.0-mysql apache2  
libapache2-mod-php7.0 mysql-server
```

Apenas lembre-se de digitar os comandos anteriores em apenas uma linha. Caso ainda encontre algum erro, faça a pesquisa pelo nome dos pacotes:

```
aptitude search php
```

Ou tente usar o **apt-cache** para pesquisar e o **apt-get** para instalar os pacotes:

```
apt-cache search php
```

```
apt-get install php7.0 php7.0-mysql apache2  
libapache2-mod-php7.0 mysql-server
```

Durante o processo de instalação, alguns diálogos serão abertos requisitando uma senha para o usuário root do MySQL, que é o usuário administrador. Insira a sua senha e continue o processo de instalação:

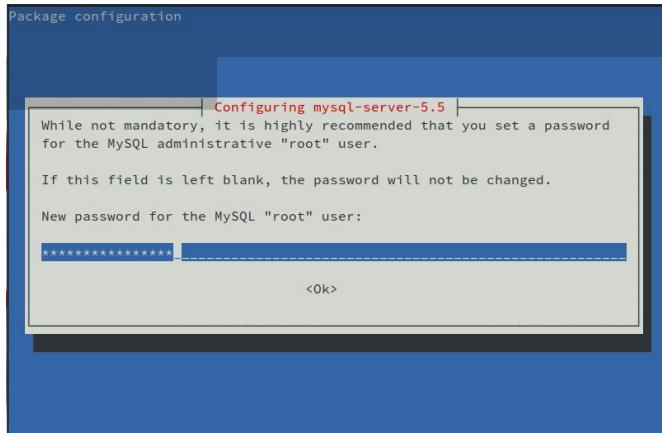


Figura 2.7: Senha do usuário administrador do MySQL

Para iniciar o Apache, utilize o seguinte comando:

```
sudo service apache2 start
```

A pasta padrão no Debian/Ubuntu para os arquivos servidos pelo Apache é `/var/www/` .

2.3 PHP NO MAC OS X

O PHP vem instalado no Mac OS X, mas também é possível usar um pacote como o XAMPP para ter um ambiente com o Apache e o MySQL. Para instalar o XAMPP no Mac OS X, vá até a página de download já citada e procure a opção para Mac OS X na lista.

Baixe o pacote DMG e siga as instruções de instalação que são bem parecidas com as do Windows:

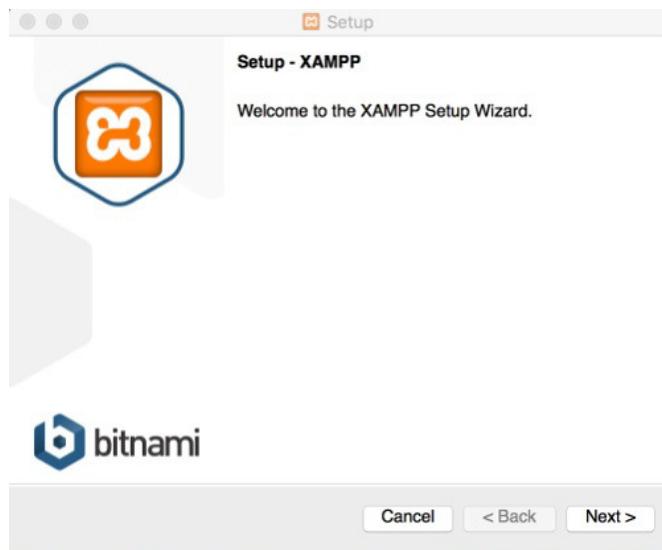


Figura 2.8: Instalação do XAMPP no Mac OS X

Deixe todas as opções selecionadas na seleção de componentes:

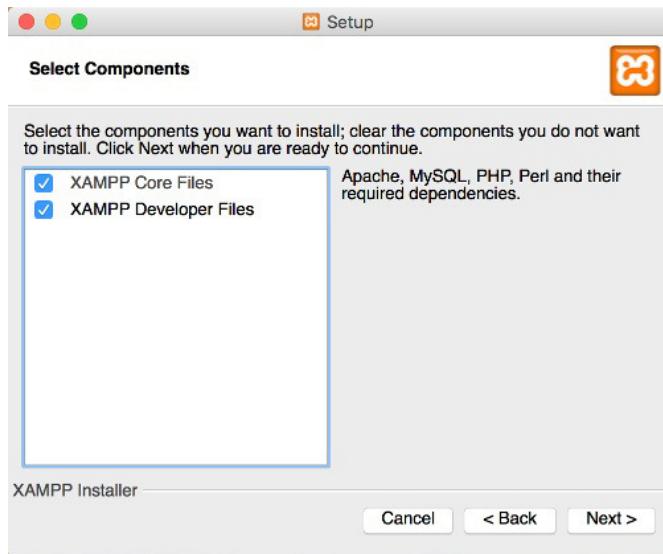


Figura 2.9: Seleção dos componentes do XAMPP para instalar

Ao final da instalação, deixe a opção de iniciar o XAMPP marcada e clique em `Finish`.

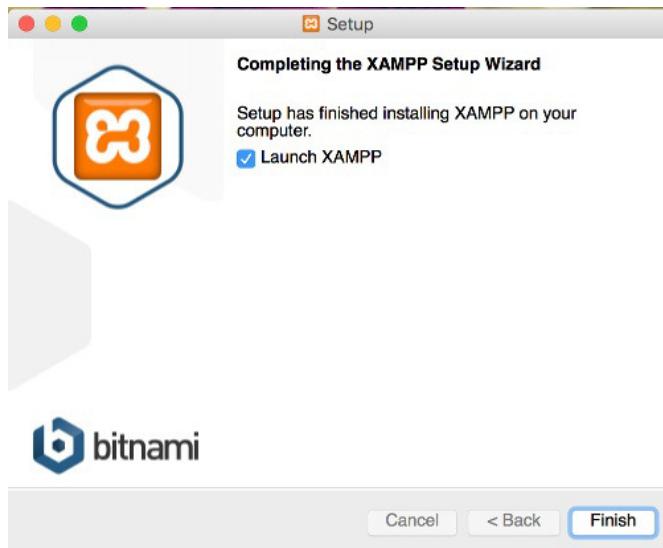


Figura 2.10: Finalização da instalação

O painel do XAMPP será iniciado:



Figura 2.11: Painel do XAMPP

Clique na aba Manage Servers ou o equivalente em português, e verifique se o serviço do Apache está em execução:

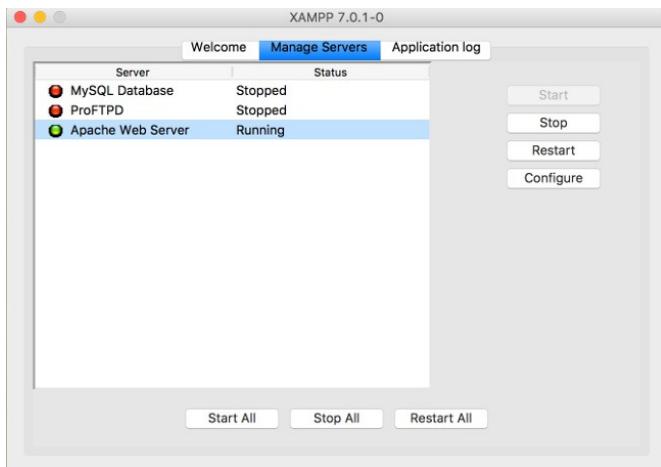


Figura 2.12: Painel do XAMPP mostrando o Apache em execução

O XAMPP é instalado por padrão em Applications/XAMPP .

2.4 APÓS A INSTALAÇÃO

Após a instalação do XAMPP no seu sistema operacional, abra seu navegador e acesse o endereço **localhost**. Você deverá ver uma página de boas-vindas do XAMPP, ou uma página dizendo que o Apache está funcionando:



Welcome to XAMPP for Windows 7.0.1

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our [Forums](#), adding yourself to the [Mailing List](#), and liking us on [Facebook](#), following our exploits on [Twitter](#), or adding us to your [Google+](#) circles.

Figura 2.13: Homepage do XAMPP instalado localmente

O QUE É O LOCALHOST?

Acabamos de acessar o endereço **localhost** pelo navegador. Este é o endereço local do computador — ele corresponde, por padrão, ao próprio computador. Após instalar o XAMPP, um servidor web também será instalado no computador, o Apache, por isso será possível acessá-lo por meio do navegador.

MEU XAMPP NÃO TEM MySQL, MAS SIM MARIA DB

O MariaDB é um projeto derivado do MySQL criado pelos próprios desenvolvedores do MySQL após a aquisição deste pela Oracle. O MariaDB mantém a compatibilidade com o MySQL.

Para a maioria dos casos, podemos simplesmente pensar nele como se fosse o MySQL. Neste livro, os termos MySQL e MariaDB são usados com o mesmo significado e, nas opções de conexão do PHP, tudo é feito como se fosse o MySQL, mesmo se o banco usado for o MariaDB.

No futuro, pode ser que estes dois bancos de dados passem a ter diferenças que não nos permitam utilizar um como se fosse o outro. Mas na época da escrita deste livro, essas tecnologias ainda são muito semelhantes.

2.5 VERSÕES DO PHP

O PHP é uma linguagem em constante evolução e, às vezes, acontecem mudanças importantes e que acabam por não manter funcionalidades antigas. O mais recente lançamento do PHP, durante a escrita deste livro, é a versão 7.

Enquanto a versão 7 é um passo importante para a linguagem e sua comunidade, esta ainda não é uma versão amplamente distribuída. Isso pode gerar problemas de compatibilidade quando o desenvolvedor opta por utilizar os mais recentes avanços da

linguagem, e depois precisa hospedar o código em um servidor que não suporta as novas funcionalidades.

O código neste livro tem como foco manter a compatibilidade com o PHP 5 e 7. Desta forma, é possível hospedar tanto em servidores mais atualizados quanto nos mais defasados.

Claro que isso dificulta a adição de funcionalidades mais recentes e muito interessantes do PHP 7. Mas, *hey*, este é um livro para iniciantes, certo? Então, vamos focar em dominar a linguagem principal, para depois poder partir para os novos avanços. ;)

De qualquer forma, em algumas partes deste livro serão demonstrados trechos de código que precisam do PHP 7 para funcionar. Nestes casos sempre há indicações deste fato e, no geral, esse tipo de código é utilizado apenas para demonstrar como o PHP vem crescendo e melhorando a cada nova versão.

CAPÍTULO 3

O PRIMEIRO PROGRAMA EM PHP

Após a instalação do ambiente de desenvolvimento, chegou o momento de escrever o primeiro programa em PHP! Para isso, será necessário usar um editor de textos, como: o Bloco de Notas no Windows, o Gedit ou o Kate no Linux, ou o editor de textos no Mac OS X (usando a opção de formatar em texto ascii).

QUAL EDITOR DE TEXTOS DEVO USAR?

Atenção: não utilize editores/processadores de textos como o Word ou o Writer, pois estes softwares gravam várias informações além do texto nos arquivos. Existem diversos editores de texto que podem ser usados para editar seus códigos em PHP. No Windows, o mais simples deles é o bloco de notas, mas existem outras opções que deixam o código colorido e até ajudam completando nomes de funções do PHP.

A minha recomendação é o **Notepad++**, um ótimo editor de textos que permite escrever em diversas linguagens de programação, inclusive PHP e HTML. Se você ainda não usa

um editor de textos para seus códigos, ou está começando agora, eu recomendo usá-lo.

Para baixar o Notepad++, acesse <http://notepad-plus-plus.org/>, e clique na opção Downloads . Na próxima página, clique no link Notepad++ Installer . O instalador é simples, basta usar as opções padrão.

Se você usa Linux, recomendo o Gedit ou o Kate, que podem ser instalados via gerenciador de pacotes, como aptitude , apt-get , yum etc. No Mac OS X, você também pode usar ambos, basta pesquisar pelos instaladores.

Uma outra boa opção para todas as plataformas é o Atom, desenvolvido pelo GitHub, que pode ser baixado em <https://atom.io/>.

O nosso primeiro programa vai apenas exibir algumas informações referentes à data e hora, para entendermos como funciona o processamento realizado pelo PHP e a exibição da página feita pelo navegador.

Vamos lá. Abra o seu editor de textos e digite o seguinte código:

```
<?php  
echo "Hoje é dia " . date('d/m/Y');
```

Repare que o código começa com um sinal de menor que, uma interrogação e a palavra php . Esta é a **tag** de abertura do PHP.

Agora, salve o arquivo em C:\xampp\htdocs\programa.php ,

se você estiver no Windows, e acesse através do navegador o endereço `http://localhost/programa.php` . Você deverá ver uma página assim:

Hoje é dia 10/08/2016

ONDE SALVAR OS ARQUIVOS NO LINUX E NO MAC OS X?

Se você usa Debian ou Ubuntu, a pasta para salvar o arquivo é `/var/www` . Lembre-se de dar permissão para seu usuário criar arquivos nesta pasta, algo como `sudo chmod a+w /var/www` .

Usuários do Mac OS X que instalaram o XAMPP devem salvar os arquivos em `/Applications/XAMPP/htdocs/` .

O QUE É A PASTA HTDOCS DO XAMPP?

Esta pasta é a raiz, a primeira pasta, do servidor Apache. É a partir dela que podemos acessar nossos arquivos dentro do Apache. Por exemplo, um arquivo chamado `pagina.php` dentro da pasta `htdocs` poderá ser acessado através do endereço `http://localhost/pagina.php` .

Agora, se você criar uma pasta nova dentro de `htdocs` , por exemplo, a pasta `site` e, dentro dela um arquivo `pagina.php` , o acesso será feito através do endereço `http://localhost/site/pagina.php` .

É claro que a data exibida no seu navegador será diferente, pois ela será gerada automaticamente pelo PHP. Vamos alterar um pouco este código para termos uma melhor visualização de que as mudanças estão realmente acontecendo quando carregamos a página no navegador. Altere o arquivo `programa.php`, colocando o código a seguir:

```
<?php  
echo "Hoje é dia " . date('d/m/Y');  
echo " agora são " . date('H:i:s');
```

Atualize a página no navegador. Para isso, use a opção de recarregar ou aperte o botão F5. Percebeu o que aconteceu? Agora ele também exibe a hora, os minutos e os segundos da hora da requisição.

Experimente atualizar a página algumas vezes. Veja que, a cada requisição, a página é alterada. Mas, o que acontece se você não atualizar mais a página no navegador?

NADA!

Isso mesmo, nada acontecerá. E isso nos leva a pensar que a página deveria ser atualizada a cada segundo, entretanto, não é bem isso que acontece.

3.1 REQUISIÇÃO, PROCESSAMENTO E RESPOSTA

O que acontece é o seguinte: quando o navegador pede uma página nova, ou atualiza uma sendo exibida — que para o servidor é como pedir uma página nova —, acontece uma **requisição**. O

servidor web, ou servidor HTTP (no nosso caso, o Apache), recebe esta requisição e percebe que o arquivo solicitado tem a extensão `.php`. Quando isso acontece, o Apache primeiro manda o arquivo para o PHP para que este faça o **processamento** do arquivo.

É nesta hora que o PHP preenche a página pegando a data e a hora do sistema operacional. Depois de processar tudo, o PHP devolve o resultado para o Apache, que, por sua vez, entrega uma **resposta** para o navegador.

O navegador, então, exibe a página HTML e não faz novas requisições ao servidor enquanto o usuário ou um script não pedir que uma nova requisição seja feita. É por isso que a página não muda mais, pois, para isso, é necessário que uma nova requisição seja feita.

O QUE SIGNIFICA PHP?

Aliás, você sabia que PHP significa *PHP Hypertext Preprocessor*? Ou, em português: PHP Pré-processador de Hipertexto, sendo que hipertexto é o HTML. Legal, né? Ele pré-processa um arquivo antes de o servidor web mandar a resposta para o navegador.

3.2 A MINHA PÁGINA ESTÁ MOSTRANDO A HORA ERRADA!

A sua página poderá mostrar a hora errada, ou você poderá receber uma mensagem de erro do PHP dizendo que o `timezone`

não foi configurado corretamente. Isso acontece pois o PHP do XAMPP vem configurado com a hora de outro país, e o PHP no Debian vem sem um `timezone` padrão configurado.

Para corrigir isso, basta editar o arquivo de configuração do PHP dentro da instalação do XAMPP. O arquivo é o `php.ini` e, no XAMPP do Windows, ele fica em `c:\xampp\php\php.ini`. No XAMPP do Mac OS X, ele fica em `Applications/XAMPP/php/php.ini`, e no Debian, ele fica em `/etc/php5/apache2/php.ini` ou `/etc/php/7.0/apache2/php.ini`, dependendo da versão do PHP que você instalou.

Neste arquivo, procure pela linha que começa com `date.timezone = .` Na minha configuração, ele veio assim:

```
date.timezone = Europe/Berlin
```

Veja que ele está com o horário de Berlim! Para fazer a correção, basta alterar para um horário brasileiro. No meu caso, ficou assim:

```
date.timezone = America/Sao_Paulo
```

Se você não estiver no horário de São Paulo (Brasília), você poderá pesquisar o seu `timezone` na lista de `timezones` da América, que fica neste endereço:
<http://php.net/manual/en/timezones.america.php>.

Após alterar e salvar o arquivo, será necessário reiniciar o Apache. Use o painel do XAMPP para parar e iniciar novamente o Apache. Se você usa Debian/Ubuntu, use o comando `sudo service apache2 restart`.

3.3 RESUMO

O que fizemos até agora é bem simples, mas já nos dá algumas informações importantes sobre o PHP. Primeiro, você deve ter percebido que um programa em PHP começa com `<?php`. É este pequeno trecho que **diz a partir daqui, o PHP deverá processar os códigos**, por isso este trecho é chamado de **tag de abertura do PHP**.

Outra coisa que podemos perceber é que a instrução `echo` é usada para **imprimir** informações no arquivo que será enviado para o navegador. Este arquivo também nos mostra que, assim como em outras linguagens, em PHP uma linha de instruções termina com `;` (ponto e vírgula).

Um último aprendizado deste arquivo é a função `date()`, que recebe um formato de data para formatar a data atual do computador. Usamos algumas opções de formatação como `Y` para exibir o ano com quatro dígitos, e `H` para exibir a hora no formato 24 horas.

3.4 DESAFIOS

Ao final de cada capítulo, há alguns desafios sugeridos como forma de praticar o que foi visto até aquele momento e até forçar um pouco mais. Dessa forma, você pode pesquisar mais sobre PHP e resolver novos problemas.

Agora que já conseguimos exibir a data e a hora, tente fazer os seguintes desafios:

1. Na função `date()`, experimente mudar o `Y` para `y`. O

que acontece?

2. Você consegue exibir a hora no formato de 12 horas, *A.M.* e *P.M.*?
3. E se você tivesse que exibir o dia da semana em formato numérico, como 1 para segunda, 2 para terça etc.? Como faria?
4. Exiba quantos dias faltam para o próximo sábado. Por exemplo, se hoje for quarta, então faltam 3 dias para sábado.
5. Exiba também o nome do mês atual.

DICA PARA OS DESAFIOS

Procure no manual do PHP a função `date()`. Eu não deixarei o link para este manual aqui, tente usar algum site de busca para encontrá-la.

CAPÍTULO 4

CONSTRUINDO UM CALENDÁRIO COM PHP

Certo, nosso primeiro programa apenas exibe a data e a hora do processamento realizado pelo PHP. Isso nos mostra como funciona o processo de requisição ao servidor, e processamento e resposta ao navegador.

Agora vamos criar uma página que exibe um calendário para entender como funciona a geração de HTML através do PHP. Veremos também como escrever e chamar funções e alguns laços de repetição e condicionais.

Neste capítulo, conheceremos mais sobre o PHP, por isso leia com atenção e faça e até refaça os exercícios, pois a prática é fundamental para compreender melhor algumas partes do PHP. Fique de olho nas novidades que serão apresentadas e não tenha receio de tentar mais de uma vez para fixar bem os conceitos aprendidos.

4.1 DEFININDO NOSSO CALENDÁRIO

Nossa página exibirá um calendário, logo, usaremos tabelas HTML que serão montadas pelo PHP a cada nova requisição.

Dessa forma, teremos sempre o calendário gerado pelo PHP, sem ter de criar a página manualmente.

E por falar em HTML, como estão seus conhecimentos sobre esse assunto? Talvez agora seja uma boa hora de desenferrujar e praticar. Mesmo usando bastante PHP, a nossa meta ainda é gerar HTML para que o navegador possa renderizar.

Não é necessário ser mestre do HTML e CSS, mas um conhecimento básico será bastante útil para compreender como o PHP é utilizado para gerar HTML. Se julgar necessário, faça uma revisão sobre o funcionamento das principais tags do HTML, como cabeçalho, título, listas, tabelas e formulários.

Não vamos realmente utilizar muito CSS neste livro, apenas um arquivo para dar estilo ao projeto principal que será apresentado no próximo capítulo. Porém, você pode alterar o CSS se desejar uma aparência diferente para seu projeto.

Ao final deste capítulo, você terá desenvolvido um programa PHP que vai gerar um calendário parecido com o seguinte:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 4.1: Exemplo de como ficará o calendário

Mais uma vez, perceba que o calendário é apenas uma tabela

HTML. Aliás, repare também que você poderá adicionar esta tabela em outros programas em PHP que você esteja desenvolvendo! Isso é bem legal, seu primeiro componente reutilizável em PHP.

Vamos em frente. Neste capítulo, você também aprenderá novos comandos e conceitos do PHP e de programação em geral.

4.2 COMEÇANDO O CALENDÁRIO

Vamos começar definindo a primeira parte do calendário. Crie um novo arquivo chamado `calendario.php`, e salve-o em `c:\xampp\htdocs` ou no diretório equivalente em seu sistema operacional.

Lembre-se de que um arquivo salvo nesta pasta poderá ser acessado usando o `localhost` através do navegador.

Pois bem, vamos ao código:

```
<table border="1">
  <tr>
    <th>Dom</th>
    <th>Seg</th>
    <th>Ter</th>
    <th>Qua</th>
    <th>Qui</th>
    <th>Sex</th>
    <th>Sáb</th>
  </tr>
</table>
```

Pronto, bem simples. Acesse o endereço `localhost/calendario.php` no seu navegador preferido e você verá uma saída parecida com esta:

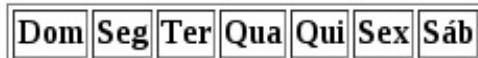


Figura 4.2: Cabeçalho do calendário

Mas, espera um pouco. Isso é HTML! Não iríamos escrever PHP? Então, este é mais um atributo interessante do PHP: **podemos escrever HTML dentro de arquivos PHP**, e isso é o que diferencia o PHP de praticamente qualquer outra linguagem de programação, pois PHP nasceu para fazer esta integração. Não é por menos que PHP é a linguagem de programação mais usada na internet, certo?

Veja este outro exemplo de integração entre HTML e PHP:

```
<h1><?php echo "Título dentro do H1"; ?></h1>
```

Perceba que o conteúdo da tag `h1` é **Título dentro do H1**, e este conteúdo foi adicionado usando o `echo` do PHP. Note também que iniciamos o PHP com `<?php`, assim como no primeiro programa feito no capítulo anterior, mas desta vez também fechamos o bloco do PHP com `?>`. Deste forma, temos ambas as **tags** de abertura e de fechamento do PHP.

Após o `?>`, ou **tag** de fechamento do PHP, podemos voltar a colocar código HTML, pois o interpretador do PHP não tentará ler os trechos fora das tags de abertura e fechamento do PHP como código PHP. Isso facilita bastante as coisas.

Podemos iniciar e fechar o PHP diversas vezes dentro de uma estrutura HTML, mas devemos nomear o arquivo como `.php`, pois desta forma o Apache vai identificar que o arquivo é `.php` e o enviará para o interpretador do PHP.

Vamos a um exemplo deste tipo de funcionamento. Crie um novo arquivo chamado `hoje.php` dentro da pasta raiz do seu Apache, e nele coloque o seguinte código:

```
<html>
    <head>
        <title>Dia <?php echo date('d/m/Y'); ?></title>
    </head>
    <body>
        <h1>
            Estamos em <?php echo date('Y'); ?> e hoje é dia
            <?php echo date('d/m'); ?>
        </h1>
        <p>
            Esta página foi gerada às <?php echo date('H'); ?>
            horas e <?php echo date('i'); ?> minutos.
        </p>
    </body>
<html>
```

Agora acesse `localhost/hoje.php` no seu navegador e você deverá ver uma página com as informações atualizadas de data e hora, assim como fizemos no primeiro programa. Você pode usar a função de ver o código-fonte da página no navegador, em geral, clicando com o direito e usando a opção **exibir código-fonte** ou algo similar, para ver que todas as tags PHP foram substituídas pelos valores de data e hora no momento do processamento.

Veja que realmente não sobra nada de PHP no resultado final, apenas HTML. Isso acontece devido ao PHP fazer o processamento de seus trechos de código e substituí-los pelos resultados de funções que exibem informações, como a função `echo()`.

4.3 USANDO FUNÇÕES

Bem, voltando ao assunto do calendário, vamos adicionar uma função ao arquivo `calendario.php` para desenhar uma nova linha na tabela. Uma linha deve conter sete colunas, uma para cada um dos sete dias da semana:

```
<?php
    function linha()
    {
        return "
            <tr>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
            </tr>
        ";
    }
?>

<table>
    ...
</table>
```

Adicionamos a função antes da tabela, dentro de tags de abertura e fechamento do PHP. Agora vamos adicionar algumas chamadas à função `linha()` para escrever as linhas do calendário:

```
<table border="1">
    <tr>
        <th>Dom</th>
        <th>Seg</th>
        <th>Ter</th>
        <th>Qua</th>
        <th>Qui</th>
        <th>Sex</th>
        <th>Sáb</th>
    </tr>
```

```

<?php echo linha(); ?>
</table>

```

Adicionamos cinco chamadas à função `linha()`, imprimindo o seu resultado usando `echo`. Quando acessarmos `localhost/calendario.php`, será renderizada uma tabela parecida com esta:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb

Figura 4.3: Calendário ainda sem os dias

Agora vamos adicionar uma nova função para desenhar o calendário. Esta função será chamada de `calendario()` e deverá ser adicionada logo após a função `linha()`:

```

<?php
    function linha()
    {
        // ...
    }

    function calendario()
    {
        $calendario = '';
        $dia = 1;
        $semana = [];
        while ($dia <= 31) {
            array_push($semana, $dia);

            if (count($semana) == 7) {
                $calendario .= linha($semana);
                $semana = [];
            }
        }
    }

```

```
        }
        $dia++;
    }

    return $calendario;
}
?>
```

Aqui temos bastante coisa nova. Pela primeira vez, estamos usando variáveis no PHP. Repare que as variáveis sempre começam com um cifrão (\$). Esta é uma regra do PHP: nomes de variáveis sempre iniciam com cifrão seguido de uma letra ou um underline .

Sendo assim, as variáveis \$calendario , \$dia , \$semana , \$pessoa e \$_nome são válidas para o PHP. Porém, as variáveis \$1 , \$-nome e !\$nome são inválidas e gerarão um erro no interpretador, caso sejam executadas.

Neste trecho, também estamos usando um tipo de dados do PHP chamado de **array** para a variável \$semana . Um **array** contém uma lista de dados e pode ser composto por strings, números, outros arrays e outros tipos de dados do PHP que veremos no decorrer do livro.

4.4 PAUSA PARA OS ARRAYS

Arrays são uma parte bastante importante do desenvolvimento em PHP. Como este tipo de variável é bastante flexível, aceitando diversos tipos de dados e simples de manipular, diversos softwares em PHP o usam para os mais variados fins, como configurações, coleções, listas etc.

Para iniciar um novo array, é possível utilizar a sintaxe `array()` ou a nova sintaxe dos colchetes, que é mais simples e elegante. Veja no exemplo a seguir duas formas de se iniciar um novo array vazio:

```
<?php  
    $dias = array();  
    $meses = [];  
?>
```

O resultado será o mesmo para ambas as variáveis, mas atualmente o padrão para os projetos PHP é a utilização da forma dos colchetes. Por isso esta é a forma que foi usada neste livro.

Mesmo assim, se você se sente mais confortável usando a forma `array()`, pode utilizá-la sem problemas. Apenas lembre-se de, quando estiver trabalhando em um projeto, sozinho(a) ou em equipe, usar sempre a mesma forma, para seu código ficar mais uniforme. E se a equipe já tiver algum padrão de preferência quando você se juntar à ela, tente se adaptar.

Algo bastante legal dos arrays é que podemos criar um novo array já com alguns valores. Com isso, não é necessário ter uma linha de código para declarar o array e mais linhas para adicionar os valores:

```
<?php  
    $dias = array('Segunda', 'Terça', 'Quarta', 'Quinta');  
    $meses = ['Janeiro', 'Fevereiro', 'Março', 'Abril'];  
?>
```

Em ambos os casos, estamos criando apenas os valores que o array contém, e o PHP se encarregará de criar as chaves para acessar os valores, que serão numéricas. Veja um exemplo:

```
<?php
```

```
$meses = ['Janeiro', 'Fevereiro', 'Março', 'Abril'];  
echo $meses[1];  
?>
```

Neste caso, será exibido **Fevereiro**, pois as chaves começam em 0. Então, Janeiro terá o índice 0, Fevereiro o índice 1, Março o índice 2 e Abril o índice 3.

Após criar um array, você também pode inserir valores nele. Existem duas formas de se fazer isso: uma usando a função `array_push()` e outra usando a sintaxe dos colchetes. Veja um exemplo onde as duas formas funcionam da mesma maneira:

```
<?php  
$dias = [];  
$meses = [];  
  
$dias[] = 'Segunda';  
$dias[] = 'Terça';  
  
array_push($meses, 'Janeiro');  
array_push($meses, 'Fevereiro');  
?>
```

O resultado nos dois casos será um array com dois valores string. Essas formas de inserir dados em um array são bastante similares, com a diferença que `array_push()` vai consumir mais recursos da máquina — de forma mínima, mas mais recursos.

Agora vamos ao mais interessante, as diferenças. Veja o exemplo:

```
<?php  
  
$pessoa = [];  
$alunos = [];  
  
$pessoa['nome'] = 'Linus';
```

```
$pessoa['sistema'] = 'Linux';
$pessoa['linguagem'] = 'C';

array_push($alunos, 'Maria', 'Joana', 'Paulo', 'Pedro');

echo $pessoa['nome'];
echo $alunos[1];
?>
```

Perceba que, ao utilizar a forma dos colchetes, é possível nomear as chaves dos valores. Dessa forma, podemos ter não apenas chaves numéricas, mas também strings. Assim fica bem simples de montar variáveis contendo diversas informações, como nesse caso, um array com os dados de uma pessoa.

Já o `array_push()` nos permite adicionar diversos itens de uma vez só. Porém, não é possível escolher as chaves, pois elas serão numeradas automaticamente pelo PHP.

Existe ainda uma outra forma de criar arrays já atribuindo chaves e valores. Veja no exemplo:

```
<?php

$pessoa = [
    'nome' => 'Linus',
    'sistema' => 'Linux',
    'linguagem' => 'C',
];

echo $pessoa['nome'];
echo $pessoa['sistema'];
echo $pessoa['linguagem'];
?>
```

Neste caso, estamos criando o array `$pessoa` já com os índices `nome`, `sistema` e `linguagem`. Após sua criação, ainda é possível modificar os valores dos índices existentes, ou mesmo criar mais índices:

```

<?php

$pessoa = [
    'nome' => 'Linus',
    'sistema' => 'Linux',
];
$pessoa['nome'] = 'Linus Torvalds';
$pessoa['linguagem'] = 'C';
?>

```

Legal, né? Tenha em mente que os arrays são realmente bastante usados no mundo PHP. Tente se acostumar com as duas formas de criar arrays e também com as diferentes formas de manipulá-los, pois você pode trabalhar em projetos usando tanto uma forma quanto a outra.

4.5 DE VOLTA AO CALENDÁRIO

Agora que já temos mais informações sobre como os arrays funcionam, vamos voltar ao exemplo:

```

<?php
    function linha()
    {
        // ...
    }

    function calendario()
    {
        $calendario = '';
        $dia = 1;
        $semana = [];
        while ($dia <= 31) {
            array_push($semana, $dia);

            if (count($semana) == 7) {
                $calendario .= linha($semana);
                $semana = [];
            }
        }
    }

```

```
        $dia++;
    }

    return $calendario;
}
?>
```

Reparou no uso da instrução `while`? Viu que esta instrução é praticamente igual ao `while` de linguagens como C? O `while` é um laço e executará o conteúdo do bloco entre as chaves enquanto a condição entre parênteses for verdadeira.

Outra instrução bem parecida com outras linguagens é o `if`. Ela verifica se uma condição é verdadeira e, em caso positivo, executa o conteúdo do bloco entre as chaves.

Para desenhar o calendário, iniciamos no dia primeiro e usamos o `while` para fazer um laço que se repetirá até o dia 31. O array `$semana` é usado para guardar os dias da semana, e garantimos que ele não terá mais que sete dias usando o `if`.

A função `array_push()` adiciona mais um valor em nosso array, como já vimos nos exemplos dos arrays. Dentro do `if`, criamos uma linha usando a função `linha()` passando o array `$semana` como parâmetro. Em seguida, limpamos o array `$semana` atribuindo para ele um novo array vazio, onde podemos colocar os dias da próxima semana. Então, mais uma vez, neste caso "limpar" o array significa atribuir um novo array vazio à variável.

Ah, uma nova função apresentada foi a `count()`. Seu funcionamento é fácil de deduzir, certo? Ela conta a quantidade de itens do nosso array `$semana`.

Outra novidade é a variável `$calendario`, que é uma string. Ela é iniciada vazia e, a cada chamada da função `linha()`, recebe um pouco mais de texto. Isso é feito através da concatenação de strings. Veja um exemplo da concatenação:

```
<?php  
    $texto = 'Estou';  
  
    $texto = $texto . 'estudando PHP';  
  
    echo $texto;  
?>
```

Concatenação de strings

Em PHP, usamos o ponto (`.`) para fazer a concatenação de strings, isso é, juntar uma string ao final da outra. No exemplo anterior, a string `estudando PHP` será adicionada ao conteúdo já existente na variável `$texto`. Também podemos utilizar a forma reduzida, que é usando o `.=`, como na variável `$calendario` do exemplo. Usar o `.=` é a mesma coisa de usar `$variavel = $variavel . 'algo a mais'`. Veja o exemplo:

```
<?php  
    $texto = 'Estou';  
  
    $texto .= 'estudando PHP';  
  
    echo $texto;  
?>
```

Ufa, este foi mais um pequeno desvio, mas vamos voltar ao exemplo do calendário. Repare que a nossa função `linha()` foi chamada com um parâmetro, que é um array com os dias da semana. Então, precisamos alterar a função `linha()` para receber este array e usar seus valores. Altere-a para ficar como a seguinte:

```

<?php

function linha($semana)
{
    return "
        <tr>
        <td>{$semana[0]}</td>
        <td>{$semana[1]}</td>
        <td>{$semana[2]}</td>
        <td>{$semana[3]}</td>
        <td>{$semana[4]}</td>
        <td>{$semana[5]}</td>
        <td>{$semana[6]}</td>
        </tr>
    ";
}

```

Veja que agora ela recebe um parâmetro chamado \$semana e o conteúdo de \$semana é exibido usando os colchetes para acessar cada um dos itens. **Atenção:** um array inicia suas chaves no número zero! Por isso, exibimos do zero ao seis.

Agora precisamos fazer mais uma alteração para exibir o calendário que temos até agora. Na tabela, retire aquelas cinco chamadas para a função linha() e troque por uma chamada para a função calendario(). Não esqueça de exibir o resultado usando echo , pois a função calendário retorna uma string:

```


| Dom | Seg | Ter | Qua | Qui | Sex | Sáb |
|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|


```

Agora, acesse `localhost/calendario.php`, ou atualize a página (caso já esteja aberta), e você verá um resultado parecido com este:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

Figura 4.4: Calendário, mas sem todos os dias do mês

Já está parecido com um calendário! Mas ele está exibindo apenas até o dia 28! Isso está acontecendo por estarmos verificando se o número de itens na variável `$semana` é igual a 7. O problema é que este número não vai chegar a sete na última etapa, já que de 28 para 31 são apenas 3 dias de diferença.

4.6 ENTENDENDO E SE ENTENDENDO COM OS ERROS

Uma possível solução seria chamar a função `linha()` após o laço `while` passando o array `semana` com menos dias. Vamos fazer isso para ver o que acontece, pois, de certa forma, uma boa parte do trabalho em desenvolvimento é feito com tentativas, erros e acertos. Altere seu código para fazer esta chamada:

```
<?php  
  
function calendario()  
{
```

```

$calendario = '';
$dia = 1;
$semana = [];
while ($dia <= 31) {
    array_push($semana, $dia);

    if (count($semana) == 7) {
        $calendario .= linha($semana);
        $semana = [];
    }

    $dia++;
}
$calendario .= linha($semana);

return $calendario;
}

?>

```

Atualize a página. Veja que os dias estão sendo exibidos, mas perceba a quantidade de erros que apareceram!

Notice: Undefined offset: 3 in **calendario.php** on line **9**

Notice: Undefined offset: 4 in **calendario.php** on line **10**

Notice: Undefined offset: 5 in **calendario.php** on line **11**

Notice: Undefined offset: 6 in **calendario.php** on line **12**

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 4.5: Calendário completo, mas com alguns erros

Veja que o erro informado é `Undefined offset: 3`. Isso

quer dizer que o PHP está tentando acessar o índice 3 no array \$semana e não está encontrando-o. O mesmo acontece com os índices 4, 5 e 6, por isso esse monte de erros apareceu.

A IMPORTÂNCIA DAS MENSAGENS DE ERRO

É comum que programadores novatos ignorem as mensagens de erro da linguagem. Isso é um problema, pois quando alguma falha acontece, a linguagem faz o melhor possível para indicar o que aconteceu. Quando topar com um erro no PHP, tente ler a mensagem e compreender o que houve. Em geral, ele diz a linha do erro e explica bem o problema.

Saber ler e entender as mensagens de erro é como saber ler e entender o painel de um carro, por exemplo. O painel indica que o tanque está vazio, mas se o motorista não sabe reconhecer isso, o carro vai parar por falta de combustível e o motorista ficará tentando ligar de novo, trocar uma peça ou outra, trocar um pneu etc. Ou seja, vai fazer de tudo para tentar corrigir o problema, mas sem perceber que o próprio carro já está indicando o que há de errado.

Não tenha medo de copiar a mensagem de erro e fazer buscas na internet, pois é bem provável que outros já passaram pelo mesmo problema e até postaram soluções e/ou explicações em sites, blogs e fóruns.

Meu PHP não mostrou os erros!

Pode acontecer de você não enxergar os erros gerados, pois seu

PHP pode estar configurado para não exibi-los. Esta é uma configuração do PHP, assim como aquela do *timezone* que fizemos no exemplo do início do livro. Para fazer com que o PHP exiba os erros, altere no arquivo `php.ini` a linha: `display_errors = off` para `display_errors = on`.

Depois disso, reinicie o seu servidor web (parar e iniciar o XAMPP novamente, ou outro servidor que você esteja usando). Após reiniciar, os erros serão exibidos.

Lembre-se de sempre deixar os erros ativos durante o desenvolvimento de sites e aplicativos. Eles são os melhores amigos dos profissionais de desenvolvimento.

4.7 FINALIZANDO O CALENDÁRIO

Vamos mudar um pouco o script para não gerar erros e para exibir corretamente o nosso calendário. Altere apenas a função `linha()` para testar se os índices existem antes de exibi-los.

Para isso, vamos usar um laço `for`, que é bem parecido com o `for` de outras linguagens. Ele recebe um valor inicial para uma variável, um controle que diz até quando executar o laço e também um incrementador para a variável que controla o laço.

Dentro do laço, vamos usar a função `array_key_exists()`, que verifica se um índice em um array existe. O código deverá ficar assim:

```
<?php  
  
function linha($semana)  
{  
    $linha = '<tr>';
```

```

for ($i = 0; $i <= 6; $i++) {
    if (array_key_exists($i, $semana)) {
        $linha .= "<td>{$semana[$i]}</td>";
    } else {
        $linha .= "<td></td>";
    }
}
$linha .= '<tr>';

return $linha;
}

```

Agora, execute novamente o arquivo `calendario.php` e você terá um resultado bem parecido com este:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 4.6: Calendário final, sem erros e com todos os dias

4.8 RESUMO

Neste capítulo, foram introduzidas algumas características e funcionalidades interessantes do PHP, como a opção de usar HTML e PHP no mesmo arquivo, e o uso de funções para centralizar blocos que podem ser repetidos no fluxo do programa.

Também foram tratados os laços `for` e `while`, além do uso da condicional `if` e da função `array_key_exists()`, que verifica se um índice em um array foi definido. Neste capítulo, também vimos como funcionam os arrays em PHP, e a

importância de ler e entender as mensagens de erro.

4.9 DESAFIOS

Hora de alguns desafios! Sugiro copiar o calendário atual para um novo arquivo para fazer os testes e desafios propostos, algo como `calendario_dia_negrito.php` para o segundo desafio.

Não esqueça que este arquivo deve ser acessado em http://localhost/calendario_dia_negrito.php.

1. Faça uma página que exiba a hora e a frase "Bom dia", "Boa tarde" ou "Boa noite", de acordo com a hora. Use a condicional `if` e a função `date()`.
2. Faça com que o calendário exiba o dia atual em negrito, usando a função `date()`.
3. Adicione uma linha no topo do calendário para os dias da semana, começando pelo domingo: *Dom, Seg, Ter, Qua, Qui, Sex, Sáb.*
4. Exiba os domingos em vermelho e os sábados em negrito. Pesquise um pouco sobre cores em CSS.
5. Faça o calendário começar em um dia que não seja um domingo.
6. E um calendário do ano todo? Será que é muito complexo?

CAPÍTULO 5

ENTRADA DE DADOS COM FORMULÁRIO

No capítulo anterior, foram apresentadas várias funcionalidades do PHP e também sua sintaxe para blocos usada nas funções e em instruções, como `if`, `while` e `for`. Também vimos os laços para repetição de trechos usando `while` e `for`.

Quem já desenvolveu em alguma linguagem deve ter percebido que a sintaxe do PHP é muito parecida com as sintaxes usadas em diversas linguagens de programação, principalmente C e suas derivadas

Agora vamos a um novo projeto: a construção de uma lista de tarefas. Este é o projeto que nos acompanhará até o final deste livro, e cada capítulo será uma continuação do capítulo anterior, adicionando novas funcionalidades e melhorando o código conforme novas funcionalidades da linguagem são apresentadas.

O projeto da lista de tarefas será usado para aprender os principais conceitos da programação para a web, como mais requisições, envio de dados, manipulação de banco de dados, envio de arquivos e até mesmo envio de e-mails. Sendo assim, é importante ler cada capítulo com atenção e praticar os exercícios

até você entender os conteúdos.

Definindo a lista de tarefas

Nosso projeto será composto por apenas algumas páginas, como a lista de tarefas e um formulário para adicioná-las e editá-las. Durante este projeto, será necessário receber dados do usuário, no caso, as informações das tarefas.

É muito provável que você já tenha usado sites que pedem algum tipo de informação, ou mesmo sistemas online nos quais é necessário cadastrar vários tipos de dados. Pois é exatamente isso o que faremos — exibiremos um formulário perguntando algumas informações para o usuário e, quando as informações forem inseridas, vamos validá-las, guardá-las e exibi-las.

5.1 O FORMULÁRIO DE CADASTRO DE TAREFAS

Trabalhar com páginas web exige conhecimentos em HTML. E quando digo conhecimentos em HTML, estou dizendo que é necessário conhecer a estrutura do HTML e não ter problemas em digitar as tags e criar páginas usando apenas um editor de textos para escrever seu HTML.

Se você já conhece um pouco (ou bastante) de web e usa editores nos quais você praticamente desenha a página, como o Dreamweaver e similares, recomendo novamente que use apenas um editor de texto para reproduzir os exercícios deste livro. Assim fica mais fácil de entender o que está acontecendo e por que acontece desta ou daquela maneira.

Após entender a escrita do HTML e como integrar o PHP para criar páginas dinâmicas, você pode decidir se voltará a usar um editor de páginas que gera HTML para depois você preencher determinados trechos com PHP, ou se vai continuar com o poder em suas mãos e vai editar as páginas usando apenas HTML e CSS.

Acredito que, depois de entender e dominar como tudo funciona, você não vai querer voltar para os editores que geram HTML, pois, em geral, eles geram código focado no browser, e não exatamente em ajudar um ser humano a ler e entender como tudo se encaixa. Mas essa é a minha opinião. ;)

Enfim, vamos ao formulário para o cadastro das tarefas. Crie uma pasta nova chamada `tarefas` dentro da pasta `htdocs` do XAMPP (ou onde for a pasta raiz do Apache no seu sistema operacional). Nela, crie um arquivo chamado `tarefas.php`. É nele que criaremos o formulário para as tarefas.

Inicie o arquivo com o seguinte conteúdo:

```
<html>
    <head>
        <title>Gerenciador de Tarefas</title>
    </head>
    <body>
        <h1>Gerenciador de Tarefas</h1>

        <!-- Aqui irá o restante do código -->

    </body>
</html>
```

Nosso projeto começará simples e depois adicionaremos mais funcionalidades conforme a necessidade, o que reflete bastante a realidade dos projetos de desenvolvimento de software. Por isso, vamos começar pelo mais importante, o nome da tarefa a ser

realizada.

O código a seguir cria um formulário para inserir os dados das tarefas, e ele deve ser adicionado após a tag H1 no arquivo `tarefas.php`:

```
<form>
    <fieldset>
        <legend>Nova tarefa</legend>
        <label>
            Tarefa:
            <input type="text" name="nome" />
        </label>
        <input type="submit" value="Cadastrar" />
    </fieldset>
</form>
```

Não tenha medo de digitar o código HTML. Lembre-se de treinar bastante, assim fica na memória muscular da sua mão, e seus dedos vão saber o que digitar mesmo quando seu cérebro não estiver prestando atenção (sério mesmo, acredite em mim, digite bastante HTML). Então, vá em frente e adicione o código do formulário na página.

Agora acesse esta nova página no endereço: <http://localhost/tarefas/tarefas.php>. Seu formulário vai ficar bem parecido com este:

Gerenciador de Tarefas

The screenshot shows a web page with a title 'Gerenciador de Tarefas'. Below it is a form with a legend 'Nova tarefa'. The form contains a label 'Tarefa:' followed by a text input field and a button labeled 'Cadastrar'.

Figura 5.1: Formulário inicial para o cadastro de tarefas

Perceba que agora estamos colocando o nome da pasta/diretório dentro da raiz do Apache, por isso o endereço fica `tarefas/tarefas.php`.

5.2 ENTRADA DE DADOS

Certo, agora temos um formulário sendo exibido, mas, como será que o PHP consegue pegar os dados que são informados nele? Vamos fazer um experimento: no nome da tarefa, digite **Estudar PHP** e clique no botão **Cadastrar**.

O que aconteceu? Aparentemente, nada, mas o texto sumiu da caixa de texto. E outra coisa muito interessante também aconteceu: o endereço do nosso script mudou! Sim, veja que agora existe uma interrogação no endereço, seguida por um texto que tem muito a nos dizer. Veja como ficou o endereço: <http://localhost/tarefas/tarefas.php?nome=Estudar+PHP>.

Dá para tirarmos algumas conclusões aqui, certo? O nome do `input` no formulário é `nome` e, neste endereço, temos um trecho `nome=`. Ou seja, o que digitamos no campo foi **enviado** através do endereço, por meio da **URL** (vou chamar o endereço de **URL** daqui para a frente).

Na primeira vez que chamamos a página seu endereço, URL, não tinha o trecho com o nome da tarefa, mas quando digitamos o nome e enviamos o formulário, este trecho passou a aparecer depois do símbolo de interrogação (`?`). Ou seja, um formulário serve para **enviarmos os dados para algum lugar**. Isso é um conhecimento muito importante, pois a internet não seria o que é se as pessoas não pudessem enviar dados.

ENTRADA DE DADOS NA WEB

Se você já programou algum software para desktop, ou scripts para a linha de comando, já deve ter lidado com entrada de dados do usuário. Neste tipo de software, em geral basta parar o processamento e pedir para o usuário informar um valor, e então seguir em frente usando o valor fornecido.

Na web, as coisas são um pouco diferentes. O PHP não pode ser parado para perguntar algo para o usuário, então os dados já devem existir para o PHP quando o script começar a rodar. Lembre-se de que o PHP é executado no servidor e envia o resultado do processamento para o servidor web que, em seguida, envia uma resposta para o navegador.

Tudo isso é confuso? Na verdade, é só uma maneira diferente de encarar a entrada de dados. Veja a nossa URL, por exemplo, ela tem o trecho `nome=Estudar+PHP`. Este trecho serve para inserir uma informação para o PHP, no caso, o nome da tarefa. Sendo assim, quando o script `tarefa.php` começar a executar, ele já terá esta informação.

Por isso, na web devemos sempre pensar na entrada de dados usando os formulários que vão enviar os dados para a **próxima** página. Por enquanto, a nossa próxima página é a mesma página, mas veremos como mudar isso mais adiante no livro, quando tivermos que mandar os dados para diferentes páginas.

Uma outra maneira de pensar sobre a entrada de dados e que pode ajudar bastante no seu entendimento é fazer uma

analogia com os formulários de papel. Quando você vai fazer um cadastro em uma loja, por exemplo, você preenche todos os seus dados em um formulário e entrega o formulário para alguém. Com PHP (web em geral), também é assim: você preenche um formulário com os dados e envia para o servidor processar. É como se o PHP fosse esse alguém que recebe o formulário.

Legal, já sabemos que o nome da tarefa foi enviado através da URL. Agora a questão é: como acessar esse dado dentro do PHP?

5.3 ACESSANDO OS DADOS DA URL

PHP nasceu para a web e se dá muito bem com ela. Por isso, em PHP é super simples pegar os dados fornecidos pelo usuário via formulários e URLs.

Em PHP, existem algumas variáveis chamadas de **superglobais**, isso significa que estas variáveis existem em todos os escopos de seus programas PHP. Ou seja, você pode acessá-las dentro de funções, dentro de laços, dentro de classes; enfim, em qualquer lugar do seu código, elas estarão disponíveis por padrão.

Para o nosso projeto, vamos usar a superglobal `$_GET`. Lembra dos arrays que foram usados para fazer o calendário? `$_GET` também é um array e o mais interessante é que o PHP pega o que está na URL e coloca no array `$_GET` para usarmos!

Vamos fazer um teste exibindo o nome digitado logo após o nosso formulário. Para isso, após o fechamento do formulário, ou

seja, depois da tag `</form>`, adicione o seguinte código:

```
...
</form>
<?php
    if (array_key_exists('nome', $_GET)) {
        echo "Nome informado: " . $_GET['nome'];
    }
?>
...
...
```

As reticências são apenas ilustrativas, claro. Alias, de agora em diante, vou colocar reticências várias vezes para ilustrar trechos que deverão ser incluídos dentro de códigos já existentes.

Neste trecho, estamos usando a função `array_key_exists()` que verifica se o índice `nome` existe no array `$_GET`. Caso o índice exista, nós o exibimos usando o `echo`.

Em PHP, assim como em outras linguagens, é necessário definir uma variável ou um índice de um array antes de tentar usá-lo. A função `array_key_exists()` é interessante para nos ajudar a verificar se um índice em um array existe, e assim evitamos tentar acessar um índice não definido, o que gera um erro.

Uma outra maneira de verificar se um índice em um array existe é usando a função `isset()` do PHP. Esta função consegue verificar se um índice em um array existe, e também se uma variável foi definida ou não. Veja um exemplo:

```
<?php
$tarefa = [
    'nome' => 'Comprar Cebolas',
    'prioridade' => 'urgente',
];

```

```
if (isset($tarefa['prioridade'])) {  
    echo 'A tarefa tem uma prioridade definida';  
} else {  
    echo 'A tarefa NÃO tem uma prioridade definida';  
}  
  
if (isset($pessoa)) {  
    echo 'A variável $pessoa foi definida';  
} else {  
    echo 'A variável $pessoa NÃO foi definida';  
}
```

Perceba que a variável `$pessoa` não foi criada no código antes do `if` que verifica se ela existe. Por isso, a frase que diz que ela não foi definida será exibida.

Utilize a função `isset()` para verificar a existência de variáveis, e a função `array_key_exists()` para verificar se índices em arrays existem.

Voltando ao exemplo, atualize sua página, em <http://localhost/tarefas/tarefas.php>, mantendo a URL com o nome, ou digite no formulário novamente e envie os dados. Sua página deverá ficar assim:

Gerenciador de Tarefas

Nova tarefa

Tarefa: Cadastrar

Nome informado: Estudar PHP

Figura 5.2: Exibindo os dados usando `$_GET`

Legal, agora que sabemos acessar o valor que foi passado por meio da URL usando `$_GET`, podemos guardar este valor em um array que será a nossa lista de tarefas. Para isso, troque o código que acabamos de fazer para exibir o nome informado por um que pegue o valor em `$_GET` e guarde em um array chamado `$lista_tarefas`:

```
...
</form>

<?php
    $lista_tarefas = [];

    if (array_key_exists('nome', $_GET)) {
        $lista_tarefas[] = $_GET['nome'];
    }
?>
...
```

Este código é bem simples. Apenas verificamos se existe o índice **nome** dentro de `$_GET` e, caso exista, criamos um novo índice numérico em `$lista_tarefas` usando a sintaxe de colchetes vazios.

Agora é necessário exibir a lista de tarefas. Ainda no mesmo arquivo, antes da tag `</body>`, vamos adicionar uma tabela HTML para listar as atividades. Dentro desta tabela, usaremos o `foreach` para pegar cada uma das tarefas que está no array `$lista_tarefas` e exibir como linhas da tabela:

```
...
<table>
    <tr>
        <th>Tarefas</th>
    </tr>

    <?php foreach ($lista_tarefas as $tarefa) : ?>
```

```
<tr>
    <td><?php echo $tarefa; ?></td>
</tr>
<?php endforeach; ?>

</table>
</body>
...
```

Repassando o código, foi criada uma tabela com apenas uma coluna, na qual o cabeçalho é a palavra **Tarefas** e as linhas serão os nomes das tarefas.

Aqui existe um laço novo do PHP. Já vimos o `while`, então agora apresento o `foreach`. Este laço serve para passar por todos os índices de um array, atribuindo cada índice a uma variável que escolhemos, no caso, a variável `$tarefa`.

Uma outra novidade aqui é que não foram usadas as chaves para criar o bloco do `foreach`, assim como foram usadas chaves para o bloco do `while`. Na verdade, o `foreach` também pode ser usado com as chaves, sem problemas, mas usando dois pontos para abrir o bloco e a palavra `endforeach` fica mais legível em templates HTML.

Lembre-se de que o nosso código com `while` estava apenas dentro de um grande código PHP. Neste caso estamos colocando pequenos pedaços de PHP dentro do HTML. Sendo assim, fica mais legível fazer o uso desta sintaxe. Como exemplo, veja como fica o mesmo bloco usando as chaves:

```
<?php foreach ($lista_tarefas as $tarefa) { ?>

<tr>
    <td><?php echo $tarefa; ?></td>
</tr>
```

```
<?php } ?>
```

Não parece ter muita diferença em um trecho pequeno desses, mas perceba que `<?php } ?>` é uma linha bem genérica, que pode estar fechando um `if`, um `while`, um `foreach` ou mesmo uma função e outros blocos PHP, enquanto `<?php endforeach; ?>` é mais expressivo. Por isso, recomendo usar esta forma.

Aliás, você pode experimentar outros blocos com esta sintaxe, como o `if` e `endif`, `while` e `endwhile` e `for` e `endfor`. Mas prefira esta sintaxe para os templates HTML, deixando a sintaxe de chaves para os arquivos e blocos que contenham apenas PHP.

Bem, voltando ao assunto da lista de tarefas, ao enviar um nome de tarefa, a sua página deverá ficar assim:

Gerenciador de Tarefas

A screenshot of a web application interface. At the top, there is a text input field labeled "Nova tarefa" containing the placeholder text "Tarefa:". To the right of the input field is a button labeled "Cadastrar".

Tarefas
Estudar PHP

Figura 5.3: Lista de tarefas com uma tarefa

Como estamos deixando tudo na mesma página, fica fácil adicionar uma nova tarefa. Então vamos adicionar uma nova tarefa usando o formulário que está antes da tabela, no endereço <http://localhost/tarefas/tarefas.php>. Aqui eu digitei **Estudar HTML** e usei o botão cadastrar, o resultado foi este:

Gerenciador de Tarefas

Nova tarefa

Tarefa: Cadastrar

Tarefas
Estudar HTML

Figura 5.4: Lista de tarefas com uma tarefa

Apenas a tarefa nova é listada! Onde está a tarefa antiga?

POR QUE A PRIMEIRA TAREFA SUMIU?

PHP trabalha principalmente com web e, neste caso, o que acontece a cada nova requisição que fazemos, seja pedindo uma página ou enviando dados, é que o PHP interpreta tudo de novo e devolve apenas HTML para o navegador. Ou seja, ele não lembra do que aconteceu na última requisição feita!

A cada nova requisição, o PHP processa tudo de novo e não guarda as variáveis para outros acessos. Isso é um problema para a nossa aplicação, já que precisamos manter a lista das nossas tarefas. Mas calma, nem tudo está perdido! O PHP tem um recurso que nos auxiliará a solucionar isso.

5.4 SESSÕES NO PHP

O recurso que nos auxiliará a manter os dados entre as requisições são as sessões. Uma sessão serve para mantermos uma

variável especial que existirá em todas as nossas requisições. Lembra da superglobal `$_GET`? As sessões são tão fáceis de usar quanto a `$_GET`, basta usar a superglobal `$_SESSION`.

A grande diferença é que usamos a `$_GET` para ler informações, e usaremos a `$_SESSION` para escrever e ler informações.

O uso da `$_SESSION` exige só um esforço adicional, que é chamar a função `session_start()` no começo do nosso programa. Para isso, vamos adicionar a função antes da abertura do HTML:

```
<?php session_start(); ?>
<html>
...

```

ATENÇÃO

A função `session_start()` deve ser invocada antes de qualquer envio de dados para o navegador, isso inclui qualquer `echo` ou HTML fora do escopo do PHP.

Depois disso, precisamos alterar o uso da lista `$lista_tarefas` para pegar os dados de `$_SESSION`, caso esses dados existam. Vamos mudar o `if` que verifica se existem dados em `$_GET`. Também adicionaremos um novo `if` após a criação do array `$lista_tarefas` para preenchê-lo com os dados da `$_SESSION`, quando necessário:

```
<?php
// ...

```

```
if (array_key_exists('nome', $_GET)) {
    $_SESSION['lista_tarefas'][] = $_GET['nome'];
}

$lista_tarefas = [];

if (array_key_exists('lista_tarefas', $_SESSION)) {
    $lista_tarefas = $_SESSION['lista_tarefas'];
}
// ...
?>
```

Agora, ao cadastrar algumas tarefas, o PHP vai manter os dados entre as requisições! Veja como a sua lista deve ficar:

Gerenciador de Tarefas

Nova tarefa

Tarefa: Cadastrar

Tarefas

Estudar HTML

Estudar PHP

Estudar Javascript

Figura 5.5: A lista de tarefas usando `$_SESSION` para manter os dados

COMO FUNCIONAM AS SESSÕES NO PHP?

Se o PHP não consegue guardar os dados entre as requisições, como ele faz para manter as sessões?

Ele utiliza um cookie que, por padrão, é chamado de **PHPSESSID**. Procure como acessar os cookies no seu navegador, e você achará um com esse nome para a URL de desenvolvimento usada, localhost .

Nesse cookie, o PHP guardará apenas uma chave que define a identificação da sessão, mas não os dados da sessão. Estes ficam guardados no servidor e o PHP vai usar a chave guardada no cookie para recuperar as informações a cada requisição.

É como ir em um médico que você já foi antes. Você não leva o seu histórico médico, você leva apenas o número do seu prontuário e, usando esse número, eles vão buscar em um arquivo (físico ou digital) as suas informações.

Legal! Temos uma lista de tarefas já bem funcional! Claro que agora precisamos de mais funcionalidades, como editar e excluir uma tarefa e adicionar mais informações. Porém, já conseguimos pegar vários conceitos da programação para web usando PHP.

Agora dá até para brincar um pouco com o visual da aplicação, usando umas linhas de CSS. O meu ficou assim:

Gerenciador de Tarefas

The screenshot shows a web application titled "Gerenciador de Tarefas". At the top, there is a form with a label "Nova tarefa" and a text input field labeled "Tarefa:". Below the input field is a "Cadastrar" button. Underneath the form, there is a section titled "Tarefas" containing two items: "Estudar HTML" and "Estudar PHP".

Figura 5.6: A lista usando um pouco de CSS

Se quiser pegar este CSS, basta baixar aqui:
<https://gist.github.com/InFog/6860949>.

Agora faça duas experiências: abra outro navegador e acesse o endereço da lista de tarefas em `http://localhost/tarefas/tarefas.php`. O que acontece? As tarefas aparecem? Não? Mas elas não estão na sessão?

UM POUCO MAIS SOBRE AS SESSÕES NO PHP

Se colocamos os dados na sessão, por que eles não aparecem quando usamos outro navegador?

Lembra de que para o PHP saber qual usuário é o dono de uma sessão, ele guarda algumas informações nos cookies do navegador? Pois então, para cada usuário (ou para cada navegador) acessando a página, o PHP gerará um número de identificação único para guardar no cookie **PHPSESSID**. Então, quando usamos um outro navegador, temos um outro número de sessão e, por isso, uma nova lista de tarefas.

Nós também podemos usar os cookies para guardar informações que serão mantidas entre as requisições. Para isso, basta usar uma outra superglobal do PHP chamada `$_COOKIE` e a função `setcookie()`. Esta superglobal também é um array, assim como a `$_SESSION`.

A outra experiência é atualizar a página após cadastrar uma tarefa. Faça isso usando F5 ou outra opção de atualização do navegador. O que acontece? A última tarefa cadastrada se repete! E continua se repetindo após as atualizações da página!

Veremos mais para a frente como resolver estas duas situações, já que queremos que a lista fique sempre disponível, independente do navegador que a acesse. Por isso vamos usar um banco de dados.

Também não queremos que, por engano, a mesma tarefa seja

adicionada diversas vezes ao atualizar a página. Por isso, no capítulo de edição de tarefas, apresento uma técnica para evitar este tipo de situação.

5.5 RESUMO

Neste capítulo, iniciamos o desenvolvimento de uma lista de tarefas. Ela ainda é bem simples e contém poucos dados, mas já estamos trabalhando com formulários e entrada de dados na web utilizando a superglobal `$_GET` do PHP. Também estamos manipulando as sessões usando a superglobal `$_SESSION`.

Superglobais são variáveis do PHP que estão disponíveis em qualquer ponto da aplicação. No caso das superglobais `$_GET` e `$_SESSION`, os valores são guardados em **arrays**.

IMPORTANTE

Sempre que quiser usar sessões, será necessário usar a função `session_start()`. Sem isso, a sessão simplesmente não funciona.

E lembre-se de que `session_start()` deve ser chamada antes de qualquer envio de dados para o navegador, seja um `echo`, um HTML ou qualquer outra forma de exibir dados. Ou seja, coloque `session_start()` sempre na primeira linha. Fica a dica :D

5.6 DESAFIOS

Muito bem, é hora de alguns desafios:

1. Usando os mesmos conceitos que vimos até agora, monte uma lista de contatos na qual devem ser cadastrados o nome, o telefone e o e-mail de cada contato. Continue usando as sessões para manter os dados. Uma forma simples de resolver este desafio é copiando o arquivo `tarefas.php` para `contatos.php`, mudar alguns nomes e adicionar os campos necessários.
2. Crie uma cópia do projeto até agora (pois vamos continuar nos próximos capítulos) e altere para usar a superglobal `$_COOKIE` em vez de usar as sessões. Para adicionar um cookie, use a função `setcookie()` que recebe o nome do cookie e um texto com seu valor. Para pegar um cookie já definido, use a superglobal `$_COOKIE`.

Esse será um pouco mais complicado, pois é necessário armazenar textos nos cookies, não arrays. Procure por ajuda na internet ou veja o que os outros leitores já falaram sobre isso na nossa lista de discussão. O manual do PHP também pode ajudar muito neste caso.

3. Depois de alterar a aplicação para usar cookies no lugar de sessões, tente abrir os cookies nas opções do navegador e veja se seus dados aparecem lá. Se eles aparecerem, pense sobre o que isso significa e se é seguro armazenar dados desta forma, ou quais tipos de dados podem ser armazenados assim e quais não podem.

CAPÍTULO 6

TRATAMENTO DE DIFERENTES CAMPOS DE FORMULÁRIOS

No capítulo anterior, construímos a base para a nossa lista de tarefas. Agora vamos adicionar mais informações e funcionalidades ao projeto.

Antes de começar a adicionar mais informações e funcionalidade, podemos parar para analisar o nosso cenário atual e a estrutura do projeto que estamos criando, assim podemos então decidir se continuamos com o projeto como está ou se devemos fazer algo para melhorar.

Esta é uma etapa importante no desenvolvimento de uma aplicação, pois torna o trabalho à frente mais simples ou, pelo menos, ajuda a prevenir as dificuldades e os problemas de se trabalhar em um projeto onde as diferentes partes e responsabilidades não estão bem separadas.

Às vezes, é melhor alterar a base da aplicação ainda no começo para poder evoluir de forma mais eficiente do que insistir em um código não muito bom para poupar um pouco de tempo no começo.

Não se preocupe de não conseguir fazer esse tipo de análise logo no começo, pois a experiência ajuda bastante neste tipo de decisão. Quando mais projetos você desenvolver, mais vai ter experiência para evitar tropeços futuros em novos projetos.

6.1 ORGANIZANDO O CÓDIGO EM ARQUIVOS SEPARADOS

O que temos até o momento é uma lista de tarefas que tem apenas o nome da tarefa e só permite adicionar novas tarefas. Tudo isso foi feito em apenas um arquivo que contém um pouco de HTML e um pouco de PHP.

Podemos continuar com esta estrutura, mas, com o tempo, o arquivo vai crescer e ganhar mais funcionalidades. Isso deixará nossa pequena aplicação mais complicada de ler e entender, uma vez que tudo estará em um arquivo muito grande e com mais de uma linguagem (PHP, HTML, CSS). Por isso, aqui entra uma decisão importante: separar nossa aplicação em dois arquivos. Um deles fará o processamento de entrada de dados e manipulação da sessão, e o outro exibirá o formulário de cadastro de tarefas e a lista das tarefas cadastradas, ou seja, será o nosso **template**.

Não adicionaremos código novo, vamos apenas separar o código atual em dois arquivos — um arquivo será o `tarefas.php` com este conteúdo:

```
<?php  
  
session_start();  
  
if (array_key_exists('nome', $_GET)) {  
    $_SESSION['lista_tarefas'][] = $_GET['nome'];
```

```
}

$lista_tarefas = [];

if (array_key_exists('lista_tarefas', $_SESSION)) {
    $lista_tarefas = $_SESSION['lista_tarefas'];
}

include "template.php";
```

Perceba que apenas juntamos os trechos de PHP que antes estavam separados em apenas um arquivo que contém somente código PHP. Outro detalhe importante neste arquivo é o uso da instrução `include`, que serve para incluir o conteúdo de outro arquivo no fluxo atual.

O legal do `include` é que ele adiciona o outro arquivo, e todas as variáveis e funções do arquivo atual continuam valendo no arquivo incluído. Por isso podemos, neste caso, incluir um arquivo com apenas o template do formulário de cadastro de tarefas e a lista de tarefas, e ainda podemos continuar usando a variável `$lista_tarefas`, que foi definida no arquivo `tarefas.php` e que contém um array com as tarefas cadastradas.

O arquivo com o template deve se chamar `template.php`, pois este é o arquivo que estamos incluindo usando o `include` no arquivo `tarefas.php`. Veja como fica o arquivo `template.php`:

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Gerenciador de Tarefas</title>
        <link rel="stylesheet" href="tarefas.css"
              type="text/css" />
    </head>
    <body>
        <h1>Gerenciador de Tarefas</h1>
        <form>
```

```
<fieldset>
    <legend>Nova tarefa</legend>
    <label>
        Tarefa:
        <input type="text" name="nome" />
    </label>
    <input type="submit" value="Cadastrar" />
</fieldset>
</form>
<table>
    <tr>
        <th>Tarefas</th>
    </tr>
    <?php foreach ($lista_tarefas as $tarefa) : ?>
        <tr>
            <td><?php echo $tarefa; ?></td>
        </tr>
    <?php endforeach; ?>
</table>
</body>
</html>
```

Agora temos apenas o `foreach` que é um código PHP, o restante é apenas HTML. E mesmo esse `foreach` foi feito usando a sintaxe recomendada para templates, o que ajuda ainda mais a deixar os arquivos bem diferentes um do outro e cada um com sua funcionalidade.

Assim fica mais fácil para adicionar e alterar as funcionalidades, pois os arquivos estão menores e, o que é mais importante, estão com suas responsabilidades separadas.

SEMPRE DEVO SEPARAR MEUS PROJETOS EM VÁRIOS ARQUIVOS?

Esta é uma pergunta difícil e a resposta é um pouco vaga: depende, mas em geral, sim.

Tudo depende do tamanho que sua aplicação terá e do quanto você julgar que vale a pena investir um tempo separando as responsabilidades entre os arquivos. No geral, vale bastante a pena fazer esta separação.

Veja que, no nosso caso, o programa ainda está bem pequeno e, mesmo assim, a separação de arquivos já melhorou bastante o entendimento das partes.

No geral, não estamos perdendo tempo quando paramos para organizar melhor um projeto. Esse tempo investido agora pode se tornar uma grande economia de tempo nas futuras manutenções do código.

Imagine que, mais para a frente, você decida investir mais na programação PHP e deixar a parte do HTML e CSS para alguém que decidiu investir mais nessas linguagens? Ter arquivos com responsabilidades separadas vai ajudar bastante a manter cada profissional fazendo o seu melhor e sem interferir no trabalho dos demais, mesmo que sem querer.

6.2 ADICIONANDO MAIS INFORMAÇÕES ÀS TAREFAS

Para que nosso sistema de controle de tarefas fique mais prático, vamos adicionar algumas informações para melhor descrever e organizar as tarefas.

Por enquanto, já temos um título para a tarefa. Agora vamos adicionar uma descrição, um prazo para conclusão, um nível de prioridade e uma confirmação de que a tarefa já está concluída. Para isso, vamos primeiro adicionar os novos campos no formulário HTML:

```
...
<label>
    Descrição (Opcional):
    <textarea name="descricao"></textarea>
</label>
<label>
    Prazo (Opcional):
    <input type="text" name="prazo" />
</label>
<fieldset>
    <legend>Prioridade:</legend>
    <label>
        <input type="radio" name="prioridade" value="baixa"
               checked />Baixa

        <input type="radio" name="prioridade" value="media" />
        Média

        <input type="radio" name="prioridade" value="alta"/>
        Alta
    </label>
</fieldset>
<label>
    Tarefa concluída:
    <input type="checkbox" name="concluida" value="sim" />
</label>
<input type="submit" value="Cadastrar" />
...

```

Se você usar o CSS dos exemplos do livro, seu formulário deverá ficar parecido com este:

Gerenciador de Tarefas

Nova tarefa

Tarefa:

Descrição (Opcional):

Prazo (Opcional):

Prioridade:

Baixa Média Alta

Tarefa concluída:

Figura 6.1: Formulário completo para o cadastro de tarefas

O legal é que só precisamos alterar o arquivo `template.php`, enquanto o `tarefas.php` continuou o mesmo. Se você tentar cadastrar uma tarefa, vai perceber que o sistema continua funcionando e apenas ignora os dados adicionais enviados.

Agora vamos alterar também o arquivo `tarefas.php` para pegar as novas informações. Vamos usar o campo com o nome da tarefa como a base para saber se devemos cadastrar uma nova tarefa, já que o seu nome é a única informação essencial para nós.

Para isso, vamos alterar aquele `if` logo após o `session_start()` e também seu conteúdo. É claro que o bloco do `if` vai ficar maior, pois agora são vários campos que precisamos pegar do formulário enviado:

```
<?php  
session_start();
```

```

if (array_key_exists('nome', $_GET) && $_GET['nome'] != '') {
    $tarefa = [];
    $tarefa['nome'] = $_GET['nome'];

    if (array_key_exists('descricao', $_GET)) {
        $tarefa['descricao'] = $_GET['descricao'];
    } else {
        $tarefa['descricao'] = '';
    }

    if (array_key_exists('prazo', $_GET)) {
        $tarefa['prazo'] = $_GET['prazo'];
    } else {
        $tarefa['prazo'] = '';
    }

    $tarefa['prioridade'] = $_GET['prioridade'];

    if (array_key_exists('concluida', $_GET)) {
        $tarefa['concluida'] = $_GET['concluida'];
    } else {
        $tarefa['concluida'] = '';
    }

    $_SESSION['lista_tarefas'][] = $tarefa;
}
// ...

```

Repare que usamos sempre a função `array_key_exists()` para saber se os índices do array `$_GET` existem. Isso é necessário pois o navegador não envia os campos em branco, nem os campos desmarcados.

Agora é só alterar a tabela para exibir todos os dados:

```


| Tarefa | Descrição | Prazo | Prioridade |
|--------|-----------|-------|------------|
|--------|-----------|-------|------------|


```

```

<th>Concluída</th>
</tr>
<?php foreach ($lista_tarefas as $tarefa) : ?>
<tr>
    <td><?php echo $tarefa['nome']; ?></td>
    <td><?php echo $tarefa['descricao']; ?></td>
    <td><?php echo $tarefa['prazo']; ?></td>
    <td><?php echo $tarefa['prioridade']; ?></td>
    <td><?php echo $tarefa['concluida']; ?></td>
</tr>
<?php endforeach; ?>
</table>

```

Acesse o formulário e cadastre algumas tarefas. Veja como ficou a minha página com duas tarefas cadastradas:

Gerenciador de Tarefas

Nova tarefa

Tarefa:

Descrição (Opcional):

Prazo (Opcional):

Prioridade:
 Baixa Média Alta

Tarefa concluída:

Cadastrar

Tarefa	Descrição	Prazo	Prioridade	Concluída
Aprender PHP	Preciso aprender mais sobre PHP		baixa	
Comprar leite	passar na padaria da esquina e comprar leite e queijo	01/10/2015	alta	

Figura 6.2: Formulário e página com novos campos para o cadastro de tarefas

6.3 RESUMO

Neste momento, temos um sistema de gerenciamento de

tarefas já bem interessante. Durante o desenvolvimento até aqui, já foram usadas funcionalidades do PHP para manter a sessão e para incluir arquivos com trechos de código.

Graças a esta separação em arquivos, o nosso projeto ficou mais legível. Separamos as responsabilidades de cada arquivo, um para tratar os dados, e outro para exibir o formulário e a lista de tarefas.

Nesta questão de separação de arquivos, poderíamos ir além, separando um arquivo para o formulário e outro para a lista de tarefas. Mas isso fica para você testar e decidir se vai ficar mais organizado, ou se vai separar demais as responsabilidades.

Por enquanto, nossa aplicação permite apenas a adição das tarefas e funciona apenas enquanto o navegador está aberto. Quando trocamos de navegador, ou quando reiniciamos o navegador atual, a sessão se perde e, com isso, perdemos as atividades cadastradas.

Isso quer dizer que as sessões são ruins? Não! Isso diz apenas que elas não servem para o nosso problema atual, que é guardar a lista de tarefas e acessar de outro navegador sem perder os dados. Mas em cenários onde é necessário, por exemplo, manter um login de usuário, uma sessão é o ideal.

Nos próximos capítulos, vamos passar a guardar os dados em um banco de dados, pois isso resolverá o problema de não termos as tarefas acessíveis em mais de um navegador.

6.4 DESAFIOS

Agora, mais alguns desafios para treinar e evoluir:

1. Continue a lista de contatos que está nos desafios do capítulo anterior. Além do nome, do telefone e do e-mail de cada contato, adicione um campo para descrição, um para a data de nascimento e um checkbox para dizer se o contato é favorito ou não.
2. Separe as responsabilidades da lista de contatos, mantendo um arquivo para cada, assim como fizemos na lista de afazeres.

CAPÍTULO 7

ACESSANDO E USANDO UM BANCO DE DADOS

Nos últimos capítulos, construímos uma lista de tarefas que funciona muito bem quando usamos apenas um navegador e mantemos a sessão sempre aberta.

Para a nossa aplicação, este não é o cenário ideal, pois a ideia é poder acessar a lista de tarefas de qualquer navegador e não queremos ter uma nova lista sempre que iniciarmos uma nova sessão do navegador. Por isso é necessário guardar nossos dados em um local mais adequado, de onde vamos conseguir recuperá-los sempre que necessário, mesmo estando em outro navegador, fechando o navegador e abrindo-o novamente, ou mesmo usando outro computador.

É para resolver este tipo de problema que entra o banco de dados e, no nosso caso, o famoso MySQL.

Como nossa aplicação já tem uma estrutura bacana e já funciona bem usando as sessões, o que precisamos fazer é trocar o uso das sessões pelo uso do banco de dados. Assim, vamos começar por criar o banco de dados, definindo quais dados queremos guardar, e faremos com que o PHP se conecte ao banco

para poder inserir e recuperar esses dados.

A manipulação do banco de dados é feita com a utilização de outra linguagem, a **SQL**, que nos permite executar comandos para inserir, pesquisar e remover dados, fazer relatórios etc. Neste livro, veremos apenas um básico sobre a SQL, focando mais nos comandos necessários para o desenvolvimento dos projetos.

Se você ainda não tem familiaridade com SQL, recomendo que também invista um tempo estudando esta linguagem. Assim, você poderá usar SQL para trabalhar com diversos bancos de dados além do MySQL, integrando com várias outras linguagens de programação além do PHP.

7.1 O BANCO DE DADOS MYSQL

MySQL é um software livre para banco de dados. Isso significa que você pode usá-lo em seus projetos e ainda pode contribuir com o desenvolvimento do próprio MySQL, se assim desejar ou necessitar. O MySQL é bastante usado em aplicações web por sua versatilidade e por ser suportado em diversas plataformas e diferentes linguagens.

Todas propriedades do MySQL se aplicam também ao MariaDB, pois este banco tem como ideia principal ser uma alternativa transparente ao MySQL. Ou seja, código feito para o MySQL vai funcionar no MariaDB, sem necessidade de adaptações, pelo menos durante a escrita deste livro, mas pode ser que no futuro os projetos se distanciem um pouco.

Por padrão, o MySQL não tem uma interface gráfica para administração e uso, mas é possível encontrar diversas ferramentas

que fazem este tipo de trabalho. Uma das mais conhecidas é o MySQL Workbench, que tem versões para Windows, Linux e Mac OS X. Não usaremos o MySQL Workbench, mas você pode pesquisar mais sobre este software e usá-lo para administrar seus bancos MySQL.

Instalando o MySQL

Como já fizemos a instalação do XAMPP, o MySQL veio junto. Então, só precisamos usar o Painel no XAMPP para iniciar o seu serviço. Fique sempre de olho para ver se não é necessário liberar o serviço do MySQL no firewall do Windows.

MySQL NO LINUX E NO MAC

Para quem usa Linux, mais especificamente os derivados do Debian e do Ubuntu, basta instalar o MySQL com o seguinte comando:

```
sudo apt-get install mysql-server
```

Usuários de Mac OS X que instalaram o XAMPP poderão iniciar o MySQL usando o painel do XAMPP.

7.2 PHPMYADMIN, ADMINISTRANDO O BANCO DE DADOS

A instalação do XAMPP contém uma ferramenta muito interessante para a gestão de bancos MySQL, o **PHPMyAdmin**. Ela é escrita em PHP, e é usada para gerenciar o MySQL, com

opções para criar novos bancos, usuários, tabelas, e também para inserir, pesquisar e remover registros etc.

INSTALAÇÃO DO PHPMyADMIN

Se você optou pelo XAMPP, o PHPMyAdmin já vem instalado e pronto para usar. Caso a sua instalação não seja com o XAMPP, ou outros pacotes neste estilo, você também poderá baixar o PHPMyAdmin em seu site oficial: <http://www.phpmyadmin.net/>.

Caso você use Ubuntu, Debian ou outras distros Linux, procure pelo pacote do PHPMyAdmin. No Debian/Ubuntu, basta usar o apt-get para fazer a instalação:

```
apt-get install phpmyadmin
```

Se você optou pelo XAMPP, será necessário iniciar o serviço do MySQL para poder acessar o PHPMyAdmin. Para isso, acesse o painel do XAMPP e clique na opção Start no serviço MySQL:

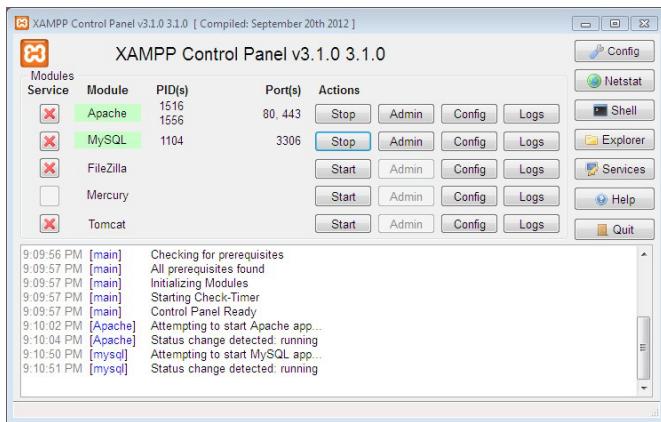


Figura 7.1: Painel do XAMPP, clique em Start no serviço MySQL

Lembre-se de que pode ser necessário autorizar o serviço no firewall do Windows. Se este for o caso, ele abrirá uma janela pedindo autorização.

Para acessar o PHPMyAdmin, utilize o endereço <http://localhost/phpmyadmin>. Você poderá ver uma página parecida com esta:



Figura 7.2: Página de login do PHPMyAdmin

Nesta página, você pode escolher o idioma para usar o PHPMyAdmin. Acesse seu PHPMyAdmin usando o usuário **root** e a senha **root**, ou a senha que você cadastrou na instalação do MySQL.

ATENÇÃO!

O XAMPP pode entrar direto na página de administração do PHPMyAdmin, sem pedir uma senha! Isso é normal em algumas versões dele quando instalado a partir do XAMPP.

Após acessar, você verá uma página como esta:

The screenshot shows the main interface of the PHPMyAdmin application. On the left, there's a sidebar with a tree view of databases: cdcol, information_schema, mysql, performance_schema, phpmyadmin, test, and webauth. The main content area has several sections: 'General Settings' (Server connection collation set to utf8_general_ci), 'Appearance Settings' (Language set to Portuguese - Brazilian portuguese, Theme set to pmahomme, Font size set to 82%), and 'Database server' (details about the MySQL server including host, version, and user). There's also a 'Web server' section listing Apache and MySQL details, and a 'phpMyAdmin' footer with links to documentation, a wiki, and support.

Figura 7.3: Página inicial do PHPMyAdmin

7.3 CRIANDO O BANCO DE DADOS

Vamos agora para a criação de um novo banco de dados. Se você escolheu o idioma português na página de login, selecione a opção **Bancos de dados**. Na próxima página, digite o nome do banco de dados, que será **tarefas**, na caixa de entrada **Criar banco de dados**. Na opção **Colação**, ou **Collation**, selecione a opção **utf8_general_ci**.

Banco de Dados

A screenshot of a form titled 'Criar banco de dados'. It has two input fields: 'Nome' (Name) containing 'tarefas' and 'Colação' (Collation) containing 'utf8_general_ci'. A 'Criar' (Create) button is visible to the right.

Figura 7.4: Formulário para criar um novo banco de dados no MySQL

Agora, basta clicar no botão **Criar** e o novo banco deve

aparecer no menu à esquerda:



Figura 7.5: Veja o banco 'tarefas' na lista dos bancos

Clicando no nome, seremos informados de que o banco ainda não possui tabelas.

7.4 CRIANDO A TABELA

Nosso projeto é um gerenciador de tarefas, então vamos precisar apenas de uma tabela com os campos necessários para guardar os dados que já temos nos arrays de tarefas dentro da `$_SESSION`. Nossa tabela deverá ficar assim:

tarefas	
*id	integer
*nome	varchar(20)
°descricao	text
°prazo	date
*prioridade	integer(1)
*concluida	boolean

Figura 7.6: Modelagem da tabela tarefas

Repare que a tabela é muito parecida com os arrays que armazenam as tarefas. As diferenças são pequenas, como o campo `id` e os campos `nome`, que foi limitado para até 20 caracteres, e `prioridade` que é um número inteiro com apenas um dígito.

O campo `id` será uma identificação única das nossas tarefas e será um número crescente. Dessa forma, teremos as tarefas `id` 1, 2, 3 e assim por diante, sem nunca repetir o número. Isso é importante para identificarmos as tarefas e não misturarmos quando precisarmos fazer referência a uma delas.

Imagine que usássemos o campo `nome` para isso. Fatalmente teríamos duas tarefas com o mesmo nome e isso atrapalharia na hora de saber qual é qual.

O campo `nome` agora tem um limite de até 20 caracteres. Isso é algo comum em bancos de dados. Já que não precisamos de nomes muito grandes para as tarefas, 20 caracteres devem ser suficientes. Mas você pode usar um valor maior, se assim desejar.

O campo `prioridade` é um número com apenas um dígito. Vamos usar desta forma, pois fica mais simples guardar no banco as prioridades 1, 2 e 3, em vez de baixa, média e alta. Isso também é

bastante comum em bancos de dados, pois reduz o espaço utilizado e fica fácil de controlar na aplicação.

Existem duas maneiras de criar a tabela usando o PHPMyAdmin. A primeira é executando diretamente o código SQL para criação de tabelas, e a segunda é usando uma interface fornecida pelo PHPMyAdmin.

Vou mostrar a primeira opção, pois exige menos passos e, acredite, é mais simples depois que se aprende um pouco de SQL. Isso porque interfaces podem mudar, mas o código para criar uma tabela tem mais chances de permanecer o mesmo no decorrer do tempo.

Usando o PHPMyAdmin e estando com o banco tarefas selecionado, clique na aba SQL . Você verá uma página com um campo para digitar comandos SQL. A página é parecida com esta:

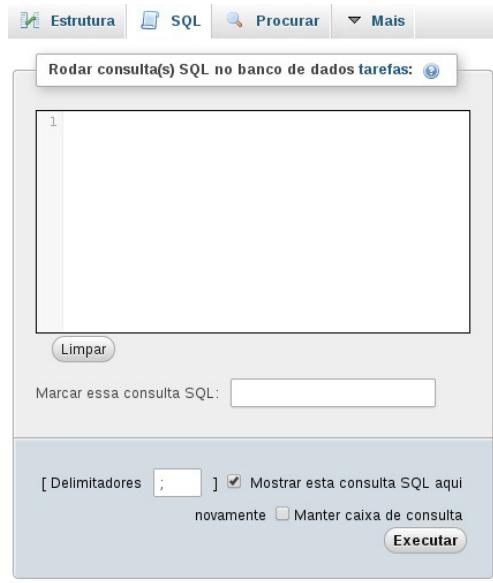


Figura 7.7: Campo para digitar comandos SQL

O código para a criação da tabela é este:

```
CREATE TABLE tarefas (
    id          INTEGER AUTO_INCREMENT PRIMARY KEY,
    nome        VARCHAR(20) NOT NULL,
    descricao   TEXT,
    prazo       DATE,
    prioridade  INTEGER(1),
    concluida   BOOLEAN
);
```

O comando `CREATE TABLE` cria uma tabela e usamos os parênteses para passar a lista de campos a serem criados nela. Para cada campo, temos uma estrutura do tipo `nome tipo opções`.

Veja por exemplo o primeiro campo, chamado de `id`, do tipo `integer`, ou seja um número inteiro. Ele possui as opções `AUTO_INCREMENT`, para fazer com que ele seja incrementado

automaticamente de 1 em 1 a cada nova tarefa inserida, e `PRIMARY KEY` que indica que o campo serve para identificar uma tarefa como única (vamos precisar disso na hora de editar e excluir tarefas).

Outros tipos de campos que podemos ver neste código SQL são:

- `VARCHAR`, que é um número variável de caracteres, podendo guardar até 20 caracteres;
- `TEXT`, que é para textos longos;
- `DATE` para datas;
- `INTEGER`, mas desta vez indicando que queremos apenas um dígito, ou seja, podemos guardar neste campo apenas os números de 0 a 9;
- `BOOLEAN` para valores verdadeiro ou falso, que o MySQL guarda como 0 (zero) para falso e 1 (um) para verdadeiro.

Digite o código SQL no campo em branco no PHPMyAdmin e clique no botão `executar`. Com isso, a tabela será criada, e poderemos então começar a adicionar e manipular os dados.

7.5 CADASTRANDO E LENDO OS DADOS DE UMA TABELA

Com a tabela pronta, podemos treinar um pouco mais de SQL antes de fazer a integração com o PHP. Vamos usar SQL para adicionar dados na tabela.

O comando usado para inseri-los é o `INSERT`. Nele devemos informar quais são os campos que queremos preencher e seus

valores.

Sempre que quisermos executar códigos SQL usando o PHPMyAdmin, basta acessarmos o banco e usar a aba SQL . Agora, veja um exemplo para inserir os dados de uma tarefa na tabela tarefas :

```
INSERT INTO tarefas  
  (nome, descricao, prioridade)  
VALUES  
  ('Estudar PHP', 'Continuar meus estudos de PHP e MySQL', 1)
```

Após digitar o código, use o botão Executar para inserir os dados no banco de dados.

Repare na estrutura do comando: nós estamos dizendo para o MySQL quais os campos que queremos preencher, que não são todos os campos da tabela. Neste caso, os campos que não preenchermos ficarão vazios.

Tenha atenção, pois é obrigatório manter a mesma sequência dos campos informados e seus valores. Sem isso, o MySQL vai cadastrar os valores nos campos incorretos. Veja que estamos colocando a mesma sequência de nome , descricao e prioridade na lista de campos e também na lista de valores, após a palavra VALUES .

Agora, vamos selecionar os dados da tabela. O comando para selecionar os dados é o SELECT . Digite o comando a seguir na caixa de texto na aba SQL :

```
SELECT * FROM tarefas
```

Com isso, o PHPMyAdmin vai para uma nova página com o resultado da pesquisa, parecida com a próxima figura:

	<input type="text"/> T	<input type="button"/> id	<input type="text"/> nome	<input type="text"/> descricao	<input type="text"/> prazo	<input type="text"/> prioridade	<input type="text"/> concluida
<input type="checkbox"/>	<input type="button"/> Editar	<input type="button"/> Copiar	<input type="button"/> Remover	1 Estudar PHP Continuar meus estudos de PHP e MySQL	NULL	1	NULL

Figura 7.8: Resultado do SELECT

No comando `SELECT`, nós informamos quais campos queremos. No nosso caso, usei o **asterisco**, que significa **todos os campos**, mas também podemos indicar os campos que queremos. Faça o `SELECT` a seguir para buscar apenas dois campos da tabela:

```
SELECT nome, prioridade FROM tarefas
```

Agora o resultado conterá apenas os campos `nome` e `prioridade`. Adicione mais algumas tarefas na tabela, usando o comando `INSERT` que vimos há pouco. No total, cadastrei quatro tarefas. O meu `SELECT`, com os campos `nome`, `descricao` e `prioridade`, trouxe este resultado:

	<input type="text"/> T	<input type="button"/> nome	<input type="text"/> descricao	<input type="text"/> prioridade
<input type="checkbox"/>	<input type="button"/> Editar	<input type="button"/> Copiar	<input type="button"/> Remover	Estudar PHP Continuar meus estudos de PHP e MySQL
<input type="checkbox"/>	<input type="button"/> Editar	<input type="button"/> Copiar	<input type="button"/> Remover	Estudar HTML HTML é importante!
<input type="checkbox"/>	<input type="button"/> Editar	<input type="button"/> Copiar	<input type="button"/> Remover	Comprar leite Desnecessário
<input type="checkbox"/>	<input type="button"/> Editar	<input type="button"/> Copiar	<input type="button"/> Remover	Arrumar as fotos das Só quando der tempo

Figura 7.9: Resultado do SELECT com mais linhas

DEVO USAR CAIXA ALTA PARA OS COMANDOS SQL?

Uma curiosidade é que SQL não te obriga a escrever os comandos da linguagem, como SELECT , INSERT etc., usando caixa alta. Porém, é interessante manter este padrão, pois simplifica a leitura.

Veja que, nos nossos códigos SQL até aqui, existem comandos como o FROM e o VALUES , que eu não necessariamente disse para que servem, mas como eles estão em caixa alta, você conseguiu ler os comandos e deduzir o que é da linguagem e o que são nossos dados. Então, mantenha seu SQL com as palavras da linguagem em caixa alta, porque facilitará para você e para outras pessoas lerem seu código e entender o que ele faz.

7.6 FILTRANDO OS RESULTADOS DO SELECT

Usando o SELECT , podemos filtrar os resultados da busca. Por exemplo, se quisermos exibir as tarefas com a prioridade 1, faremos o SQL assim:

```
SELECT nome, descricao, prioridade FROM tarefas  
WHERE prioridade = 1
```

O resultado deve ser algo similar a isso:

	nome	descricao	prioridade
<input type="checkbox"/>	Editar Copiar Remover Estudar PHP	Continuar meus estudos de PHP e MySQL	1
<input type="checkbox"/>	Editar Copiar Remover Comprar leite	Desnatado	1

Figura 7.10: Resultado do SELECT filtrando com a prioridade 1

Também podemos filtrar por um texto que esteja dentro de outro texto. Para isso, usamos a instrução `LIKE` do SQL:

```
SELECT nome, descricao, prioridade FROM tarefas  
WHERE nome LIKE '%php%'
```

O `LIKE` aceita o uso daqueles sinais de porcentagem. Eles servem para dizer algo como: **procure por qualquer coisa, seguido de php, seguido de qualquer coisa.**

Então, se a nossa pesquisa fosse assim:

```
SELECT nome, descricao, prioridade FROM tarefas  
WHERE nome LIKE 'php'
```

Ela não teria resultados. Consegue entender o motivo? Se pensar um pouco, fica simples. Neste caso, ele pesquisa resultados nos quais o campo `nome` seja igual a `php`, sem textos antes ou depois. Também podemos usar a porcentagem apenas antes ou apenas depois de um texto.

7.7 RESUMO

Neste capítulo, usamos a linguagem SQL para manipular dados em um banco de dados MySQL ou MariaDB. Tanto o MySQL quanto o MariaDB podem ser instalados à parte, ou junto com o pacote do XAMPP. Usamos também o PHPMyAdmin como ferramenta para acessar o banco por meio do navegador.

Para manipular o banco de dados, usamos a instrução `CREATE TABLE` para criar uma tabela, `INSERT INTO` para cadastrar dados em uma tabela, `SELECT` para buscar os dados da tabela, e também filtramos os resultados do `SELECT` usando `WHERE` e `LIKE`.

No próximo capítulo, vamos fazer a conexão da nossa aplicação PHP com o banco de dados.

7.8 DESAFIOS

Hora de alguns desafios, dessa vez voltados para bancos de dados e para a linguagem **SQL**:

1. Faça mais testes com o `SELECT` , o `WHERE` e o `LIKE` usando o banco `tarefas` .
2. Crie um banco chamado `contatos` para guardar os dados dos contatos do desafio iniciado nos capítulos anteriores. Qual tabela será necessária para este banco? Quais campos serão necessários para cadastrar os dados que já estão sendo capturados pelo formulário? Não se esqueça de deixar um campo `id` para guardar a chave que identificará um contato como único, assim como foi feito para as tarefas. E se estiver com dificuldades, procure ajuda no fórum da Casa do Código.
3. Crie um banco chamado `estacionamento` para cadastrar os veículos estacionados. Neste banco, crie uma tabela chamada `veiculos` com os campos `id` , `placa` , `marca` , `modelo` , `hora_entrada` e `hora_saida` . Decida os tipos de dados que devem ser usados para cada campo. Cadastre alguns veículos e tente fazer pesquisas, como buscar todos os veículos de uma marca, ou os veículos que saíram do estacionamento antes das 14h.

CAPÍTULO 8

INTEGRANDO PHP COM MYSQL

No capítulo anterior, criamos o banco de dados **tarefas** e uma tabela, também nomeada de **tarefas**, para guardar os registros de nossas tarefas. Além disso, inserimos algumas tarefas no banco e usamos o comando `SELECT` da linguagem SQL para selecionar as tarefas no banco, e filtrar pelo nome e pela prioridade usando o comando `WHERE`.

Neste capítulo, vamos finalmente conectar o PHP ao MySQL! Como já temos a aplicação funcional, mas usando sessões, e o banco já está criado, nossa tarefa será menos árdua, pois precisaremos alterar apenas algumas partes da aplicação.

Aqui, faremos uma mudança na parte mais interna da aplicação e já colheremos alguns frutos da organização do projeto em arquivos diferentes, pois as mudanças necessárias no fluxo da aplicação serão pequenas. Claro que vamos precisar de código adicional para fazer a comunicação entre o PHP e o MySQL/MariaDB, mas este código não entrará diretamente nos arquivos atuais.

8.1 PHP E MYSQL

PHP e MySQL já são velhos amigos. É bem comum encontrar aplicações que fazem uso destas tecnologias em conjunto. Desde pequenos sites pessoais, até grandes sistemas de gestão, lojas virtuais e redes sociais, a web está repleta de casos de sucesso desta parceria.

O que torna esta parceria tão forte e estável é uma combinação de fatores, como o fato de o MySQL/MariaDB ser um banco fácil de aprender, leve e rápido, e o PHP ser fácil para iniciantes e flexível para avançados. Além disso, ambos possuem licenças permissivas para uso pessoal, comercial e em projetos de software livre.

PHP torna simples o uso do MySQL por meio de uma rica biblioteca de funções que conectam, executam código SQL e trazem os resultados para a aplicação. O MariaDB é visto pelo PHP como se fosse o MySQL, sem a necessidade de usar funções específicas.

O que faremos agora é aproveitar toda essa integração entre essas tecnologias para conectar nossa aplicação PHP com o nosso banco de dados.

8.2 CONECTANDO O PHP AO MYSQL

Primeiro, vamos fazer a parte mais simples, que é conectar ao banco e buscar os dados que estão cadastrados nele. O primeiro passo para realizar a conexão com o MySQL é criar um usuário no MySQL para que o PHP possa se autenticar.

Este é um passo importante e ajuda a manter o banco de dados mais seguro. É como aquele usuário e senha que você precisa

digitar quando liga o computador, ou quando vai acessar seu e-mail ou ainda o seu perfil em uma rede social.

Para criar o usuário no MySQL, acesse o PHPMyAdmin e selecione o banco `tarefas`. Nele, use a opção `Privilégios` no menu superior. Na página que abrir, clique na opção `Adicionar usuário` — uma janela abrirá pedindo os dados do novo usuário. Eu preenchi os campos conforme a figura a seguir:

Informação de login

Nome do usuário: Usar campo texto: sistema

Servidor: Qualquer servidor

Senha: Usar campo texto:

Re-digite:

Gerar senha:

Banco de Dados para usuário

None
 Criar Banco de Dados com o mesmo nome e conceder todos os privilégios
 Conceder todos os privilégios no nome coringa (`nome_do_usuário_%`)
 Conceder todos os privilégios no banco de dados "tarefas"

Privilégios globais (Marcar todos / Desmarcar todos)

Figura 8.1: Criação de usuário no MySQL

No campo novo usuário, informei **sistematarefa**, na senha digitei **sistema** e confirmei, depois cliquei no botão `Adicionar usuário`. Após a adição do usuário, teremos um usuário chamado **sistematarefa** com a senha **sistema**. Estes são os valores que informaremos ao PHP.

Agora vamos criar um novo arquivo para fazer a conexão com o banco. Ele será gravado na mesma pasta dos arquivos da lista de tarefas, ou seja a pasta `tarefas`, e seu nome será `banco.php`:

```
<?php  
  
$bdServidor = '127.0.0.1';  
$bdUsuario = 'sistematarefa';  
$bdSenha = 'sistema';  
$bdBanco = 'tarefas';  
  
$conexao = mysqli_connect($bdServidor, $bdUsuario, $bdSenha,  
                         $bdBanco);  
  
if (mysqli_connect_errno($conexao)) {  
    echo "Problemas para conectar no banco. Erro: ";  
    echo mysqli_connect_error();  
    die();  
}
```

No arquivo, estamos criando quatro variáveis que guardam o endereço do servidor MySQL (que é o nosso próprio computador, neste caso), o nome de usuário, a senha de acesso e o nome do banco que queremos acessar. É legal manter estes dados em variáveis, pois se for necessário alterar, alteramos apenas a criação da variável, sem ter de procurar onde ela está sendo usada.

Repare que, no código do arquivo `banco.php`, existem apenas três funções que não usamos até agora. A função `mysqli_connect()` recebe os dados de conexão com o banco e abre a conexão. Esta conexão é guardada na variável `$conexao`, e vamos precisar dela sempre que formos interagir com o banco.

Outra nova função é a `mysqli_connect_errno()`, que pega uma conexão e verifica se houve erros de conexão. Neste caso, a função está dentro de um `if`, para verificar se houve erros ou não. A outra função é a `mysqli_connect_error()` que vai

retornar um texto explicando o que aconteceu de errado ao conectar ao banco de dados.

Ah, tem mais uma função nova ali, a `die()`, que faz o que o nome diz: ela encerra o programa ali mesmo, sem ler o código que existe mais para a frente. Usamos esta função aqui, pois não faz sentido continuar a aplicação sem uma conexão ao banco de dados.

SEMPRE OUVI FALAR DA FUNÇÃO MYSQL_CONNECT()

Talvez você já tenha visto algum texto sobre PHP e MySQL em que a função de conexão era a `mysql_connect()`. Esta função existe, assim como várias outras funções que começam com `mysql_`, mas elas foram removidas do PHP a partir da versão 7. Por isso, devemos usar a nova e melhorada versão da biblioteca MySQL do PHP, a MySQLi.

No geral, basta adicionar a letra `i` nos nomes de função MySQL e um parâmetro aqui ou ali, que tudo funciona bem. Entretanto, existem algumas diferenças entre as bibliotecas, por isso não necessariamente podemos pegar um código mais antigo e atualizar apenas adicionando o `i`.

Então, a dica é: **sempre use a versão nova da biblioteca MySQL para PHP. Sempre.** Se você está usando a versão 7 do PHP, o que é bastante provável, a única opção é a MySQLi.

TAMBÉM OUVI FALAR DO PDO. NÃO SERIA MELHOR USAR O PDO?

Se você já conhece um pouco de PHP e de como ele se conecta a diversos bancos de dados, já deve ter ouvido falar, ou mesmo usado, o **PDO**. O PDO é uma biblioteca para conexão a bancos de dados no PHP que nos permite usar diversos bancos diferentes, não apenas o MySQL e o MariaDB. Ele também fornece algumas funcionalidades que a MySQLi não tem.

A MySQLi, por sua vez, oferece duas versões da biblioteca, uma procedural (ou seja, que faz uso de funções) e uma orientada a objetos. Neste ponto do livro, acredito que a versão procedural é mais indicada por ser mais fácil de usar para quem não tem ainda conhecimento em programação orientada a objetos.

Já o PDO oferece apenas uma versão de sua biblioteca orientada a objetos, e isso ainda pode ser complicado neste ponto do livro, pois ainda não passamos pelo tópico de Orientação a Objetos.

Mesmo assim, não há como negar que o PDO é realmente uma evolução muito bacana no mundo do PHP e é algo que todo desenvolvedor PHP deve conhecer. Por isso, veremos como converter a nossa aplicação das tarefas para usar o PDO nos capítulos finais deste livro, após os capítulos que tratam de Orientação a Objetos. Ou seja, você vai terminar o livro conhecendo duas bibliotecas de conexão a bancos de dados em PHP. ;)

Vamos fazer uma pequena experiência com a conexão ao banco. Acesse o arquivo banco.php diretamente pelo navegador, no endereço: <http://localhost/tarefas/banco.php>. Você deverá ver apenas uma página em branco, o que é bom, pois significa que a conexão funcionou.

Agora, experimente mudar a variável \$bdSenha para uma senha incorreta e acesse o arquivo novamente. Você verá a frase: **Problemas para conectar no banco. Erro:**, e mais uma mensagem de erro do PHP. Isso nos mostra que nosso código de conexão está funcional e também exibe erros caso não consiga conectar.

O ERRO DE CONEXÃO APARECE SEMPRE, MESMO COM OS DADOS CORRETOS

Você poderá passar pelo problema do erro de conexão de vez em quando. Caso isso aconteça, verifique se o servidor do MySQL está ativo no painel de controle do XAMPP, ou em outra ferramenta que você use para gerenciar o MySQL.

Uma outra forma de verificar se o banco está funcionando normalmente é acessando o PHPMyAdmin. Se ele também não conseguir conectar, o banco poderá estar mesmo desligado.

Sempre leia as mensagens de erro da linguagem, elas sempre indicam o exato problema para a conexão não funcionar. Se a mensagem estiver em inglês, você pode usar um tradutor para ajudar.

Dica: conseguir ler e entender as mensagens em inglês pode ajudar muito na carreira de desenvolvimento de software.

8.3 BUSCANDO DADOS NO BANCO

A próxima alteração que faremos em nosso programa é fazer com que ele busque as tarefas cadastradas no banco de dados, e não mais na sessão. Para isso, abra o arquivo `tarefas.php` e ache o trecho que verifica se a sessão com a lista de tarefas existe, para então colocar a lista de tarefas na variável `$lista_tarefas`, ou criar uma lista vazia, caso a sessão não exista. O trecho é este:

```
<?php  
// ...  
  
$lista_tarefas = [];  
  
if (array_key_exists('lista_tarefas', $_SESSION)) {  
    $lista_tarefas = $_SESSION['lista_tarefas'];  
}  
  
// ...
```

Este trecho será apagado e, no lugar dele, ficará apenas uma nova linha que chama uma função que retornará as tarefas cadastradas no banco:

```
<?php  
// ...  
  
$lista_tarefas = buscar_tarefas($conexao);
```

É claro que a função `buscar_tarefas()` ainda não existe, por isso vamos criá-la dentro do arquivo `banco.php`. Para isso, abriremos novamente o arquivo `banco.php` e, no final dele, colocaremos a nova função. Mas antes de escrevê-la, precisamos planejar um pouco o seu funcionamento.

A função `buscar_tarefas()` deverá usar uma conexão com o banco para executar comandos SQL que busquem a lista de tarefas. Após buscar as tarefas, ela deverá criar um array com os dados das tarefas e, então, deverá retornar esses dados.

Certo, planejando assim, fica mais simples alcançar nosso objetivo. Como já temos uma variável com a conexão com o MySQL, que foi criada no arquivo `banco.php`, poderemos usá-la. Vamos ao código da função:

```
<?php  
// ...
```

```
function buscar_tarefas($conexao)
{
    $sqlBusca = 'SELECT * FROM tarefas';
    $resultado = mysqli_query($conexao, $sqlBusca);

    $tarefas = [];

    while ($tarefa = mysqli_fetch_assoc($resultado)) {
        $tarefas[] = $tarefa;
    }

    return $tarefas;
}
```

Nesta função, foi criada uma variável com o comando SQL que vai buscar os dados no banco, no caso, o comando `SELECT * FROM tarefas`. Logo depois, a função `mysqli_query()` usa a variável com a conexão e a variável com o comando para ir ao banco, executar o comando e trazer o resultado. Este resultado fica armazenado na variável `$resultado`.

Na sequência, é criada a variável `$tarefas`, que é apenas um array vazio. O próximo bloco é o `while`, que é executado até que a função `mysqli_fetch_assoc()` tenha passado por todas as linhas do resultado, sendo que cada linha é armazenada na variável `$tarefa`, no singular, para diferenciar do array `$tarefas`, no plural. E então temos, finalmente, o `return` que devolve os dados contidos no array `$tarefas` para quem chamou a função.

Agora podemos atualizar a página `tarefas.php` e ver o resultado. Acesse o endereço

`http://localhost/tarefas/tarefas.php` e veja o que acontece. Um erro! Ele diz que a função `buscar_tarefas()` não existe! Mas nós acabamos de escrevê-la! O que está acontecendo?

Analizando o problema, é fácil identificar que o arquivo `tarefas.php` tem um `include` apenas no final para o arquivo `template.php`. Como ele não sabe da existência do arquivo `banco.php`, ele gera o erro.

Vamos corrigir o problema adicionando um `require` para o arquivo `banco.php`. Este novo `require` será adicionado no começo do arquivo, logo após a função `session_start()`:

```
<?php  
session_start();  
  
require "banco.php";  
  
// ...
```

REQUIRE OU INCLUDE?

Usamos o `include` para adicionar o arquivo `template.php`, e o `require` para adicionar o arquivo `banco.php`. Mas, qual é a diferença entre eles?

A funcionalidade dos dois é a mesma: eles vão ler, executar e incluir um arquivo PHP no contexto atual. A diferença está no modo como eles tratam os erros.

Um erro no carregamento do arquivo usando o `include` vai gerar uma mensagem de alerta do PHP e vai continuar a execução. Já um erro no carregamento do arquivo usando o `require` vai fazer com que a execução do programa seja interrompida.

O arquivo `banco.php` é fundamental para a execução da nossa lista de tarefas, por isso vamos usar o `require`. Aliás, o `template.php` também é fundamental.

Devemos usar o `require` para ele também? Eu digo que sim. Então, altere todos os `include` que usamos até agora e use o `require` no lugar.

Agora sim, atualize a página e você verá que ela exibe os dados cadastrados no banco! Veja como ficou a minha:

Gerenciador de Tarefas

Nova tarefa

Tarefa:

Descrição (Opcional):

Prazo (Opcional):

Prioridade:

Baixa Média Alta

Tarefa concluída:

Tarefa	Descrição	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL	1		
Estudar HTML	HTML importante!	2		
Comprar leite	Desnatado	1		
Arrumar as fotos	Quando der tempo	3		

Figura 8.2: Listando as tarefas cadastradas no banco de dados

A importância de uma boa estrutura

Veja como conseguimos alterar a nossa lista de tarefas para pegar os dados do banco com apenas algumas alterações pontuais. Isso foi possível pois a nossa aplicação tem uma estrutura que favorece esse tipo de alteração, e até mesmo um crescimento mais saudável.

Veja a geração da lista de tarefas, por exemplo. Se estivéssemos usando a superglobal `$_SESSION` diretamente lá, a alteração seria mais complicada e nos custaria mais tempo, além de poder gerar mais erros. Mas, como estamos usando um array chamado

`$lista_tarefas` , bastou alterar a forma como este array é gerado. Repare que nem mesmo precisamos tocar no arquivo `template.php` .

Não se preocupe se você não conseguir criar estruturas assim logo no começo do seu aprendizado. Mas lembre-se de sempre estudar e melhorar seus códigos depois de "finalizar" um projeto, pois um software dificilmente vai ficar "pronto".

8.4 CADASTRANDO AS TAREFAS NO BANCO

Bem, agora que já estamos buscando as tarefas no banco, é chegada a hora de cadastrarmos mais tarefas usando o nosso formulário. Será que aqui também será possível fazer pequenas alterações para termos os dados sendo gravados no banco, no lugar das sessões?

Analisando o código no arquivo `tarefas.php` , conseguimos ver que a linha responsável por colocar a tarefa na sessão é esta:

```
<?php  
// ...  
  
$_SESSION['lista_tarefas'][] = $tarefa;  
  
// ...
```

Essa linha coloca no índice `lista_tarefas` da sessão o conteúdo da variável `$tarefa` , que é um array com os dados da tarefa. Este array é praticamente o que precisamos para enviar nossas tarefas para o banco de dados!

Assim como fizemos com o trecho que buscava as tarefas na sessão, vamos então trocar esta linha por uma chamada a uma

nova função que gravará os dados no banco. Troque a linha que acabamos de ver por esta:

```
<?php  
// ...  
  
gravar_tarefa($conexao, $tarefa);  
  
// ...
```

A função `gravar_tarefa()` receberá a conexão que já temos, na variável `$conexao`, e também a nossa tarefa, que está no array `$tarefa`. Mas a função `gravar_tarefa()` não existe, então precisamos adicioná-la. Faremos isso no arquivo `banco.php`, pois ele está com o restante do código responsável pela comunicação com o banco MySQL. No final do arquivo, vamos adicionar a função `gravar_tarefa()`:

```
<?php  
// ...  
  
function gravar_tarefa($conexao, $tarefa)  
{  
    $sqlGravar = "  
        INSERT INTO tarefas  
        (nome, descricao, prioridade)  
        VALUES  
        (  
            '{$tarefa['nome']}',  
            '{$tarefa['descricao']}',  
            {$tarefa['prioridade']}
```

Veja que este código faz o que fizemos lá no PHPMyAdmin, a diferença é que ele coloca as variáveis do PHP para preencher o

código SQL. Não se esqueça de colocar as variáveis PHP dentro das chaves quando estiver colocando o conteúdo delas dentro de outras strings.

Uma **string** é uma sequência de caracteres, ou seja, as nossas variáveis com textos são variáveis do tipo string. Não deixe de reparar que abrimos a string `$sqlGravar` na linha onde ela é declarada, e a fechamos na linha depois do código SQL, antes de chamar a função `mysqli_query()`.

Nesta função, também fizemos o uso da função `mysqli_query()`, pois ela é bem versátil, servindo para buscar e também para gravar dados. Na verdade, a função `mysqli_query()` serve para **executar** código SQL. Ela devolve dados quando fizermos, por exemplo, um `SELECT`, ou insere dados quando executados um `INSERT`.

Bem, a aplicação já está quase pronta para funcionar usando o banco. Mas ainda é necessária uma alteração! O nosso formulário tem o campo para definir a prioridade da tarefa e este está com os valores **baixa**, **media** e **alta**. Mas a tabela **tarefas** no banco de dados não está preparada para estes valores, pois decidimos usar números para definir as prioridades.

Sendo assim, altere o formulário para que os valores das prioridades fiquem como **1**, **2** e **3**, respectivamente. Lembrando de que o formulário está no arquivo `template.php`:

...

```
<legend>Prioridade:</legend>
<label>
    <input type="radio" name="prioridade" value="1"
           checked /> Baixa
    <input type="radio" name="prioridade" value="2" /> Média
```

```
<input type="radio" name="prioridade" value="3" /> Alta  
</label>
```

...

Muito bem, agora acesse a página em <http://localhost/tarefas/tarefas.php> e tente cadastrar uma tarefa. Eu cadastrei uma nova e a minha lista ficou assim:

Tarefa	Descrição	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		1	
Estudar HTML	HTML importante!		2	
Comprar leite	Desnecessário		1	
Arrumar as fotos das	Quando der tempo		3	
Estudar MySQL	Cadastrado pelo formulário		1	

Figura 8.3: Cadastrando uma nova tarefa através do formulário

Agora, precisamos também exibir corretamente as prioridades na lista de tarefas, pois 1, 2 e 3 devem ser usados apenas no banco, e a aplicação deve exibir sempre como **baixa, média e alta**.

A linha que precisamos alterar é a que exibe a prioridade no arquivo `template.php`:

...

```
<td><?php echo $tarefa['prioridade']; ?></td>
```

...

Adicionar alguns `ifs` podem resolver o nosso problema, mas vão deixar nosso template mais difícil de ser lido. Veja como fica se adicionarmos a logica necessária para exibir os nomes das prioridades diretamente no template:

```
<td><?php
```

```
if ($tarefa['prioridade'] == 1) { echo 'Baixa'; }
if ($tarefa['prioridade'] == 2) { echo 'Média'; }
if ($tarefa['prioridade'] == 3) { echo 'Alta'; }
?></td>
```

Não está assim tão ruim, mas veja como esta outra maneira fica bem mais legível e fácil de entender:

```
<td><?php echo traduz_prioridade($tarefa['prioridade']); ?></td>
```

Bem mais simples, não? Claro que, neste caso, é necessário criar a função `traduz_prioridade()` e colocar nela os `ifs`. Mas a grande questão aqui é que o arquivo `template.php` fica mais simples e enxuto, muito mais fácil de entender.

Pegue o arquivo `banco.php` como exemplo: nele existem as funções que criamos para lidar com o banco de dados. Graças a isso, o restante da nossa aplicação precisa apenas chamar essas funções e não se preocupar em escrever SQL para executar no banco.

Vamos criar um novo arquivo chamado `ajudantes.php`. Ele vai conter funções que nos ajudarão em pequenas tarefas, como este caso de traduzir o código da prioridade para o nome dela.

O conteúdo do arquivo `ajudantes.php` será apenas a função `traduz_prioridade()`, por enquanto, pois vamos colocar mais funções neste arquivo conforme a necessidade:

```
<?php

function traduz_prioridade($codigo)
{
    $prioridade = '';
    switch ($codigo) {
        case 1:
            $prioridade = 'Baixa';
        case 2:
            $prioridade = 'Média';
        case 3:
            $prioridade = 'Alta';
    }
    return $prioridade;
}
```

```

        break;
    case 2:
        $prioridade = 'Média';
        break;
    case 3:
        $prioridade = 'Alta';
        break;
    }

    return $prioridade;
}

```

Repare que, na função `traduz_prioridade()` , utilizamos uma nova instrução do PHP, a `switch/case` . Ela serve para os casos em que uma variável pode ter diversos valores, e cada valor significa um fluxo diferente para o programa.

Você deve estar pensando que o mesmo resultado poderia ser obtido usando alguns `ifs` , e você tem razão. Mas tente escrever esta função usando apenas `if` e `else` , você verá que ela ficará maior e mais complexa de se entender. No caso do `switch/case` , as opções ficam mais claras e de fácil localização.

Ah, o `switch/case` não pode ser usado em casos nos quais precisamos comparar se um valor é maior ou menor do que outro. Usando `if` , podemos colocar uma condição como `$variavel < 10` , algo impossível no `case` , que só aceita valores fixos. É como se no `if` só se pudesse usar `$variavel == 10` .

Com a função `traduz_prioridade()` pronta, precisamos dizer para a nossa aplicação que ela existe. Para isso, no arquivo `tarefas.php` , vamos adicionar mais uma instrução `require` , logo abaixo da inclusão do arquivo `banco.php` :

```
<?php
    session_start();
```

```
require "banco.php";
require "ajudantes.php";

// ...
```

Atualize a página e você deverá ver as prioridades com os nomes, e não mais com os códigos.

AJUDANTES?

É bem comum encontrar em aplicações PHP, ou de outras linguagens, arquivos, pacotes, módulos etc., com o nome de **helpers**, que significam exatamente **ajudantes**.

Em geral, estes arquivos contêm pequenos trechos de código para auxiliar em tarefas recorrentes, como no nosso caso de exibir o tipo de prioridade ou a formatação de datas, que veremos ainda neste capítulo.

Você também pode agrupar os ajudantes para determinadas tarefas em arquivos separados. Alguns exemplos seriam ajudantes para desenhar tabelas em um arquivo chamado `ajudantes_tabelas.php`, ou um para criar formulários em um arquivo `ajudantes_formularios.php`. Tudo vai depender das suas necessidades e organização do código.

8.5 CADASTRANDO O PRAZO DAS ATIVIDADES

Até o momento, estamos gravando no banco apenas o nome, a descrição e a prioridade das tarefas. Vamos agora incluir o prazo.

Este é um campo que vai dar um pouco mais de trabalho, pois no Brasil nós usamos a data no formato DIA/MÊS/ANO, porém, para o MySQL, o formato da data é ANO-MÊS-DIA. Por isso, vamos precisar de dois tradutores: um para pegar a data que digitarmos e traduzir para o formato do MySQL; outro para pegar a data do MySQL e traduzir para exibir na lista de tarefas.

Em programação, sempre existem diversas maneiras de se resolver um problema. Neste caso, não é diferente, pois podemos transformar as datas usando diversas técnicas diferentes. Vamos começar com uma técnica que pode não ser a melhor ou a mais elegante, mas vai resolver o nosso problema e ficará bem simples de entender. Depois vou mostrar uma técnica diferente e mais limpa, mas que exige mais conhecimentos sobre o funcionamento do PHP.

Vamos analisar melhor o problema. Nos dois formatos de data, temos os mesmos valores, mas em posições diferentes e usando separadores diferentes. Então, o que precisamos fazer é arrumar uma forma de separar os valores e reconstruir a string colocando cada valor em sua posição usando os separadores esperados, que podem ser barras ou traços.

Vamos começar pela tradução da data digitada no formulário, no formato DIA/MÊS/ANO para o formato do MySQL. A estratégia aqui será separar os valores para montar a string com o novo formato.

Para isso, vamos usar uma função bem interessante do PHP, a `explode()`, que divide uma string em partes, de acordo com algum separador informado. O retorno da função `explode()` é um **array** com as partes da string. Veja um exemplo:

```
<?php

$texto_original = "Frodo;Sam;Merry;Pippin";
$hobbits = explode(";", $texto_original);

foreach ($hobbits as $hobbit) {
    echo $hobbit . "<br />";
}
```

A variável `$texto_original` contém os nomes dos quatro `hobbits` separados por ponto e vírgula. A função `explode()` recebe dois parâmetros, sendo que o primeiro é o separador que desejamos e o segundo é o texto que desejamos separar. O resultado é um `array` no qual cada item é um dos pedaços após a separação.

No caso da tradução do prazo para as nossas tarefas, vamos criar uma função chamada `traduz_data_para_banco()` no arquivo `ajudantes.php`. Esta função vai separar os valores de dia, mês e ano, e retornará uma string com o formato correto para o MySQL. Veja como ela fica:

```
<?php

function traduz_data_para_banco($data)
{
    if ($data == "") {
        return "";
    }

    $dados = explode("/", $data);

    $data_banco = "{$dados[2]}-{$dados[1]}-{$dados[0]}";

    return $data_banco;
}

// ...
```

Quando separamos a string da data, temos um array em que o índice zero é o dia, o índice 1 é o mês, e o índice 2 é o ano. Basta então construir uma nova string com os dados nas posições corretas e separando com o traço, assim temos o formato esperado pelo MySQL/MariaDB.

Outro detalhe é que, antes de tentar traduzir a data, verificamos se ela está vazia, já que é um campo opcional. Caso esteja vazia, retornamos uma string vazia, assim o MySQL gravará uma data vazia também.

Agora precisamos alterar o trecho no qual montamos o array com os dados enviados para usar a nova função, lá no arquivo `tarefas.php` :

```
<?php  
// ...  
  
if (array_key_exists('prazo', $_GET)) {  
    $tarefa['prazo'] =  
        traduz_data_para_banco($_GET['prazo']);  
} else {  
    $tarefa['prazo'] = '';  
}  
  
// ...
```

Certo, já temos a função e também já estamos traduzindo o prazo que for digitado no formulário. Só falta alterar a função `gravar_tarefa()` no arquivo `banco.php` para também gravar o prazo. Para isso, é necessário mudar o código SQL.

Uma nota importante é que a data precisa ser formatada como uma string para o MySQL, ou seja, ela precisa estar entre aspas. Veja como deverá ficar o código:

```
<?php
```

```

// ...

function gravar_tarefa($conexao, $tarefa)
{
    $sqlGravar = "
        INSERT INTO tarefas
        (nome, descricao, prioridade, prazo)
        VALUES
        (
            '{$tarefa['nome']}',
            '{$tarefa['descricao']}',
            {$tarefa['prioridade']},
            '{$tarefa['prazo']}'
        )
    ";
    mysqli_query($conexao, $sqlGravar);
}

// ...

```

Pronto, todas as pontas ligadas. Experimente cadastrar uma tarefa com um prazo e veja como fica. A minha lista ficou assim:

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	
Comprar leite	Desnataido		Média	
Prova de PHP	Estudar até o prazo	2013-08-05	Baixa	

Figura 8.4: O prazo da atividade está no formato do MySQL

Quase perfeito, não? Agora precisamos traduzir a data de volta para o nosso formato brasileiro para exibir na lista de tarefas. Seguindo uma lógica bem parecida, vamos criar uma função `traduz_data_para_exibir()`, que também estará no arquivo `ajudantes.php`:

```

<?php
// ...

function traduz_data_para_exibir($data)

```

```

{
    if ($data == "" OR $data == "0000-00-00") {
        return "";
    }

    $dados = explode("-", $data);

    $data_exibir = "{$dados[2]}/{dados[1]}/{dados[0]}";

    return $data_exibir;
}

// ...

```

Repare nas diferenças entre a função que traduz para o banco e a que traduz para exibição. Estamos separando a string usando separadores diferentes e depois montamos novamente também com separadores diferentes. Uma outra diferença na traduz_data_para_exibir() é que o if logo no começo faz duas verificações, pois a data no banco pode estar em branco, ou pode estar como uma string "0000-00-00".

Para fazer esta verificação, utilizamos a instrução OR , colocando uma condição de cada lado. Assim, o PHP verificará se uma das condições, ou as duas, são verdadeiras. Altere a exibição do prazo para usar a nova função:

```

<td>
    <?php echo traduz_data_para_exibir($tarefa['prazo']); ?>
</td>

```

Atualize a sua lista de tarefas e veja o resultado, agora com a data correta:

Tarefa	Descrição	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	
Comprar leite	Desnatado		Média	
Prova de PHP	Estudar até o prazo	05/08/2013	Baixa	

Figura 8.5: O prazo da atividade com o formato correto usado no Brasil

8.6 OUTRA MANEIRA DE FORMATAR AS DATAS

A formatação das datas que estamos fazendo é bastante simples e direta, e tudo o que estamos fazendo é manipular uma string. Não há nada de errado com esta forma.

Entretanto, existem outras maneiras de se fazer o mesmo e que podem se mostrar mais interessantes em diferentes cenários como, por exemplo, em uma aplicação que exibe datas em formatos diferentes para usuários em diferentes países.

Uma destas formas é usando uma classe do PHP chamada `DateTime`, que é utilizada para manipulação de dados relativos à data e hora. Veja como fica a função `traduz_data_para_exibir()` para usando a classe `DateTime`:

```
<?php
// ...

function traduz_data_para_exibir($data)
{
    if ($data == "" OR $data == "0000-00-00") {
        return "";
    }

    $objeto_data = DateTime::createFromFormat('Y-m-d', $data);

    return $objeto_data->format('d/m/Y');
}

// ...
```

Veja que agora não estamos mais separando a string usando o traço como separador, e não estamos mais criando uma string usando as posições de um array. O que o exemplo anterior faz é criar um tipo de dado especial do PHP chamado **objeto**, usando o

método `createFromFormat()` da classe `DateTime`.

Isso é confuso? É muita informação? Não se preocupe, veremos mais sobre classes e objetos mais para a frente, inclusive a criação de nossas próprias classes. Por enquanto, vamos focar em como essa função, ou método, trabalha.

Se traduzirmos o nome do método `createFromFormat()` para português, teremos algo como "criar a partir de um formato". E é exatamente isso que é feito, pois o método recebe um formato **Y-m-d**, que significa "ano com quatro dígitos, mês e dia, separados por traços" e a string com a data no formato indicado, que vem do MySQL.

Depois disso, usamos essa sintaxe esquisita `$variavel->metodo()` para executar a função `format()` que faz parte do objeto `$objeto_data`. Essa função recebe apenas o formato a ser usado, que é **d/m/Y** (que significa dia, mes e ano), pois a data a ser formatada já está guardada dentro do objeto.

A função ficou mais simples de ser lida e pode ser uma boa opção para sistemas com mais de um formato de data, porque a função `format()` poderia receber uma variável com o formato preferido de data do usuário. Veja como ficaria um exemplo com formatação diferente para diferentes regiões:

```
<?php  
// ...  
  
function traduz_data_para_exibir_por_regiao($data, $regiao)  
{  
    if ($data == "" OR $data == "0000-00-00") {  
        return "";  
    }  
}
```

```

$objeto_data = DateTime::createFromFormat('Y-m-d', $data);

$resultado = '';

if ($regiao == 'EUA') {
    $resultado = $objeto_data->format('m/d/Y');
} else {
    $resultado = $objeto_data->format('d/m/Y');
}

return $resultado;
}

// ...

```

Repare que, quando a região for EUA, a data será formatada usando o padrão mês/dia/ano e, para o restante do mundo, o formato será dia/mês/ano.

8.7 MARCANDO UMA TAREFA COMO CONCLUÍDA

Dos campos que temos no formulário, o único que ainda não estamos gravando no banco é o campo que indica se a tarefa está concluída. No checkbox do formulário, colocamos o valor como `sim`, mas no banco precisamos do número zero para tarefas não concluídas, e do número 1 para tarefas concluídas. Por isso, vamos mudar o checkbox para o valor `1`, no arquivo `template.php`:

```

<label>
    Tarefa concluída:
    <input type="checkbox" name="concluida" value="1" />
</label>

```

Agora, vamos alterar a parte onde este dado é adicionado no array `$tarefa`, no arquivo `tarefas.php`:

```
<?php
```

```
// ...

if (array_key_exists('concluida', $_GET)) {
    $tarefa['concluida'] = 1;
} else {
    $tarefa['concluida'] = 0;
}

// ...
```

Repare que, após verificar se o dado está definido na superglobal `$_GET`, o valor está sendo definido como 1, ou como 0 quando não estiver definido. Essa verificação é assim pois, quando um checkbox não é marcado, ele não é enviado pelo navegador junto com o formulário. Por isso, basta verificar se ele está ou não definido usando a função `array_key_exists()`.

Agora, é necessário alterar a função `gravar_tarefa()` para gerar o código SQL incluindo o campo `concluida`. Vale lembrar de que a função `gravar_tarefa()` está no arquivo `banco.php`:

```
<?php
// ...

function gravar_tarefa($conexao, $tarefa)
{
    $sqlGravar = "
        INSERT INTO tarefas
        (nome, descricao, prioridade, prazo, concluida)
        VALUES
        (
            '{$tarefa['nome']}',
            '{$tarefa['descricao']}',
            {$tarefa['prioridade']},
            '{$tarefa['prazo']}',
            {$tarefa['concluida']}
        )
    ";
}

// ...
```

Repare que o campo foi adicionado na lista de campos e o valor na lista de valores. Como o campo é um número, não precisamos usar aspas desta vez.

Agora, cadastre uma atividade e veja o resultado:

Tarefa	Descrição	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	
Comprar leite	Desnatado		Média	
Prova de PHP	Estudar até o prazo	05/08/2013	Baixa	
Comprar uma mochila	Comprar uma mochila nova		Alta	1

Figura 8.6: Mostrando a tarefa como concluída, mas mostra '1' no lugar de 'Sim'

Bem, ainda não está bom para o usuário, pois agora ele vai exibir o número 1 para tarefas concluídas, e o número 0 para atividades não concluídas. Vamos adicionar mais uma função para traduzir conteúdos do banco para apresentação.

No arquivo `ajudantes.php`, criaremos a função `traduz_concluida()`, que retorna Sim ou Não de acordo com o campo de tarefa concluída no banco:

```
<?php
// ...

function traduz_concluida($concluida)
{
    if ($concluida == 1) {
        return 'Sim';
    }

    return 'Não';
}
```

Precisamos alterar o arquivo `template.php` para usar a nova função para exibir o status de conclusão das tarefas:

```
...  
> <td><?php echo traduz_concluida($tarefa['concluida']); ?></td>
```

Atualize a página, ou cadastre uma nova tarefa, e você verá a lista com as palavras **Sim** ou **Não** na coluna **Concluída**:

Tarefa	Descrição	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	Não
Comprar leite	Desnatado		Média	Não
Prova de PHP	Estudar até o prazo	05/08/2013	Baixa	Não
Comprar uma mochila	Comprar uma mochila nova		Alta	Sim

Figura 8.7: Lista exibindo o status de conclusão como Sim e Não

Neste momento, temos uma lista de tarefas funcional, mas ainda não podemos editar as tarefas e teremos problema se digitarmos datas incorretas, nomes muito grandes ou se atualizarmos a página assim que adicionarmos uma tarefa. Aliás, tente fazer essas ações e tome nota dos resultados.

No próximo capítulo, vamos lidar com alguns destes problemas.

8.8 RESUMO

Neste capítulo, alteramos a nossa aplicação para trabalhar com o banco de dados no lugar das sessões. Experimente, neste momento, fechar o seu navegador e abri-lo novamente, ou então acesse a lista de tarefas em um navegador diferente. Perceba que as atividades estão lá! Ou seja, não dependemos mais do uso das sessões para que nossa aplicação guarde os dados.

Este capítulo também reforçou o conceito da separação de responsabilidades em diferentes arquivos, e também mostrou o conceito dos arquivos com funções ajudantes, que também são conhecidos no mundo da programação como **helpers**. Também fizemos a tradução de datas no formato usado no Brasil para o formato esperado pelo MySQL e/ou pelo MariaDB (e vice-versa).

Tivemos também um primeiro contato com programação orientada a objetos, usando uma classe para fazer a tradução das datas. Veremos mais sobre Orientação a Objetos nos próximos capítulos.

8.9 DESAFIOS

Hora de treinar fazendo mais alguns desafios:

1. Atualize a função `traduz_data_para_banco()` para também trabalhar com a classe `DateTime`, e fazer o retorno usando a função `format()`.
2. Continue a lista de contatos dos desafios anteriores, mas agora passando a guardar os dados em um banco de dados. Utilize todos os conceitos vistos até aqui, como a separação em arquivos, o uso de ajudantes e o uso das funções `mysqli_` do PHP para lidar com o banco de dados.
3. Crie a aplicação para trabalhar com o banco de dados **estacionamento**, criado nos desafios do capítulo anterior. Mais uma vez, utilize todos os conceitos que vimos até aqui.

CAPÍTULO 9

EDIÇÃO E REMOÇÃO DE REGISTROS

Nossa lista de tarefas funciona apenas para inclusão de tarefas, e ainda não valida as informações fornecidas pelo usuário. Neste capítulo, vamos adicionar as funcionalidades para editar e remover tarefas. No próximo, faremos a validação da entrada de dados.

9.1 EDIÇÃO DE TAREFAS

Para editar nossas tarefas, é necessário primeiro identificar a tarefa, para então buscar seus dados no banco e, finalmente, exibi-los em um formulário para serem editados pelo usuário. Precisamos, então, de uma forma de identificar uma tarefa como única.

Poderíamos usar o nome, mas diferentes tarefas podem ter o mesmo nome, como por exemplo, "comprar pão", que pode se repetir em dias diferentes. Para indentificar as tarefas como únicas, vamos usar o campo `id` da tabela `tarefas` no banco de dados.

Este campo já foi feito para esta situação, em que precisamos identificar um registro como único. Como este campo é do tipo autonumeração, cada tarefa terá seu próprio número, sem se

repetir.

Vamos alterar a lista de tarefas para incluir uma nova coluna **Opções**. Ela vai conter links para edição e remoção de tarefas. A primeira opção que adicionaremos é a de edição das tarefas. Para isso, edite o arquivo `template.php`, adicionando a linha que cria a coluna **Opções** e a linha que cria o link para edição:

```
...  
  
<tr>  
    <th>Tarefa</th>  
    <th>Descrição</th>  
    <th>Prazo</th>  
    <th>Prioridade</th>  
    <th>Concluída</th>  
    <th>Opções</th> <!-- A nova coluna Opções -->  
</tr>  
<?php foreach ($lista_tarefas as $tarefa) : ?>  
    <tr>  
        <td>  
            <?php echo $tarefa['nome']; ?>  
        </td>  
        <td>  
            <?php echo $tarefa['descricao']; ?>  
        </td>  
        <td>  
            <?php echo  
                traduz_data_para_exibir($tarefa['prazo']);?>  
        </td>  
        <td>  
            <?php echo  
                traduz_prioridade($tarefa['prioridade']);?>  
        </td>  
        <td>  
            <?php echo  
                traduz_concluida($tarefa['concluida']);?>  
        </td>  
        <td>  
            <!-- O campo com os links para  
                editar e remover -->  
            <a href="editar.php?id=<?php echo $tarefa['id']; ?>">
```

```
        Editar  
        </a>  
    </td>  
</tr>  
<?php endforeach; ?>  
  
...
```

Adicionei comentários no HTML anterior para ficar mais simples identificar quais são as novas adições, mas eles não são necessários no seu código. Veja a construção do link para a edição das tarefas. Ele nos levará para um arquivo PHP chamado `editar.php` e está passando um parâmetro que é o `id`.

Este `id` é preenchido pelo PHP pegando o `id` cadastrado no banco de dados e, quando clicamos no link, o parâmetro é enviado para o servidor usando a URL. Com isso, o PHP consegue acessar seu valor usando a super global `$_GET`.

Agora temos uma decisão importante a ser tomada, pois temos dois caminhos que podem ser trilhados: criar uma nova página com o formulário para a edição de tarefas; ou usar o formulário existente adicionando um pouco de lógica para que ele possa ser usado tanto para adicionar quanto para editar tarefas.

O caminho da criação da nova página com um novo formulário para a edição das tarefas é mais simples. Isso porque basta copiar o formulário para outro arquivo e fazer pequenas mudanças para que ele possa ser usado para editar uma tarefa.

O problema é que isso criaria dois formulários diferentes para os mesmos dados. No caso de, por exemplo, adicionarmos um novo campo às tarefas, seria necessário alterar em dois lugares em vez de apenas um.

Já o caminho do reaproveitamento do formulário que já existe será um pouco menos simples no começo, pois será necessário adicionar um pouco mais de lógica para que o formulário tenha duas utilidades. Entretanto, facilitará bastante as futuras alterações em nossa aplicação, pois não será necessário alterar dois formulários quando um novo campo for adicionado, ou se um bug for corrigido.

Ou seja, nem sempre o caminho mais simples vai levar para o melhor destino. Isso é bastante comum em desenvolvimento de software, em que o planejamento pode nos levar a um código melhor, mais fácil de alterar e com menos bugs.

Vamos seguir o caminho do reaproveitamento e usar apenas um formulário para fazer a adição e a edição das tarefas. Caso você queira experimentar como seria a outra opção, faça uma cópia do projeto em uma outra pasta (`tarefas2formularios`, por exemplo) e tente aplicar a lógica para incluir os arquivos diferentes para adição ou edição. De repente, isso pode ser uma experiência bacana para se entender melhor os prejuízos de se manter dois formulários.

Para fazer o reaproveitamento, precisamos alterar o nosso formulário para que ele saiba lidar com as duas situações. Mas antes, vamos fazer mais uma separação em nossa aplicação.

Atualmente, o arquivo `template.php` é responsável por exibir o formulário para cadastrar uma atividade e também a tabela com a lista de atividades. Isso funciona bem para a situação atual, já que não temos ainda a edição de tarefas. Porém, não funcionará bem para a edição, pois a ideia é que a edição não exiba a lista de tarefas, apenas o formulário.

Por isso, vamos melhorar um pouco o projeto atual separando o arquivo `template.php` em dois: um para o formulário, que vai se chamar `formulario.php` (sem acento); outro para a lista de tarefas, que vamos chamar de `tabela.php`.

Vale lembrar de que, depois desta alteração, o arquivo `template.php` ainda existirá, mas com uma nova estrutura. Este é o arquivo `formulario.php`:

```
<form>
    <fieldset>
        <legend>Nova tarefa</legend>
        <label>
            Tarefa:
            <input type="text" name="nome" />
        </label>
        <label>
            Descrição (Opcional):
            <textarea name="descricao"></textarea>
        </label>
        <label>
            Prazo (Opcional):
            <input type="text" name="prazo" />
        </label>
        <fieldset>
            <legend>Prioridade:</legend>
            <input type="radio" name="prioridade"
                value="1" checked />
                Baixa
            <input type="radio" name="prioridade" value="2" />
                Média
            <input type="radio" name="prioridade" value="3" />
                Alta
        </fieldset>
        <label>
            Tarefa concluída:
            <input type="checkbox" name="concluida" value="1" />
        </label>
        <input type="submit" value="Cadastrar" />
    </fieldset>
</form>
```

Repare que este arquivo é apenas o nosso formulário já existente, sem novas alterações. Porém, agora ele está separado em seu próprio arquivo. E o arquivo `tabela.php` vai ficar assim:

```
<table>
<tr>
    <th>Tarefa</th>
    <th>Descrição</th>
    <th>Prazo</th>
    <th>Prioridade</th>
    <th>Concluída</th>
    <th>Opções</th>
</tr>
<?php foreach ($lista_tarefas as $tarefa) : ?>
<tr>
    <td>
        <?php echo $tarefa['nome']; ?>
    </td>
    <td>
        <?php echo $tarefa['descricao']; ?>
    </td>
    <td>
        <?php echo
            traduz_data_para_exibir($tarefa['prazo']);?>
    </td>
    <td>
        <?php echo
            traduz_prioridade($tarefa['prioridade']);?>
    </td>
    <td>
        <?php echo
            traduz_concluida($tarefa['concluida']);?>
    </td>
    <td>
        <a href="editar.php?id=<?php echo $tarefa['id']; ?>">
            Editar
        </a>
    </td>
</tr>
<?php endforeach; ?>
</table>
```

Mais uma vez, apenas removemos a tabela do arquivo `template.php` e a colocamos em um arquivo apenas para ela, sem alterações em seu conteúdo.

O novo arquivo `template.php` ficará bem menor, já que não terá mais a lógica do formulário e da tabela. Mas agora ele terá uma nova lógica que verificará se a tabela deve ou não ser exibida.

Para isso, vamos usar uma variável de controle chamada `$exibir_tabela`, que será usada para decidir se vamos ou não exibir a tabela com a lista de tarefas. Esta variável deverá ser iniciada com o valor `false` logo no começo do arquivo `tarefas.php`, e então mudaremos o seu valor para `true` quando não estivermos editando uma tarefa.

Mas, primeiro, veja como deve ficar o arquivo `template.php`:

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Gerenciador de Tarefas</title>
        <link rel="stylesheet" href="tarefas.css"
              type="text/css" />
    </head>
    <body>
        <h1>Gerenciador de Tarefas</h1>

        <?php require 'formulario.php'; ?>

        <?php if ($exibir_tabela) : ?>
            <?php require 'tabela.php'; ?>
        <?php endif; ?>
    </body>
</html>
```

Gostou desta sintaxe do `if`? Em vez de abrir chaves para os blocos, coloquei os dois pontos para abrir o bloco do `if` e fechei o bloco com a instrução `endif`. Esta sintaxe é bem legal para

templates, pois deixa mais claro o que está sendo fechado.

Lembre-se de que podemos fazer isso com o `while`, usando `endwhile`, `for` e `endfor`, `foreach` e `endforeach`. Use esta sintaxe para templates — fica melhor para entender o código depois. O seu futuro eu agradecerá, e também o pessoal da sua equipe. E sim, eu já falei isso antes, mas é sempre bom ressaltar as boas práticas. :)

Agora precisamos definir a variável de controle dentro do arquivo `tarefas.php`:

```
<?php
// ...

require "banco.php";
require "ajudantes.php";

$exibir_tabela = false;

// ...
```

Se você atualizar a página neste momento, verá apenas o seu título e o formulário, pois acabamos de definir como `false` a exibição da tabela.

A nossa ideia é não exibir a lista de tarefas apenas quando estivermos editando uma tarefa. Já que a nossa lógica é exibir a lista quase sempre e não exibir em apenas um caso, então fica mais simples se deixarmos a variável `$exibir_tabela` como `true` em vez de `false`:

```
<?php
// ...

require "banco.php";
require "ajudantes.php";
```

```
$exibir_tabela = true;
```

```
// ...
```

Não tenha medo de mudar a lógica dos seus programas de vez em quando, se for para deixar mais simples e fácil de entender. Tente evitar ao máximo a síndrome do Gollum e ficar chamando seu código de **meu precioso.** =)

Agora vamos cuidar daquele novo arquivo, para onde o link de edição das tarefas nos leva, o `editar.php`. Este arquivo será muito parecido com o `tarefas.php`, mas com diferenças essenciais, como a variável `$exibir_tabela` com o valor `false` e a não necessidade de usar a função `buscar_tarefas()`, já que não exibiremos as tarefas.

Claro que ainda será necessário ir ao banco de dados para pegar os dados da tarefa que queremos editar, por isso vamos usar uma função nova, a `buscar_tarefa()`. Veja que o nome é bem parecido com a outra função, com a diferença de que a nova função busca apenas uma tarefa. É por isso que seu nome está no singular e o seu resultado será atribuído à variável `$tarefa`, também no singular.

Uma outra diferença importante no arquivo `editar.php` é que ele deverá pegar também o código de identificação das nossas tarefas, que é o campo `id`, enviado via URL e acessível no PHP usando a variável `$_GET`. Veja como deverá ficar o nosso arquivo `editar.php`:

```
<?php  
session_start();
```

```

require "banco.php";
require "ajudantes.php";

$exibir_tabela = false;

if (array_key_exists('nome', $_GET) && $_GET['nome'] != '') {
    $tarefa = [];

    $tarefa['id'] = $_GET['id'];

    $tarefa['nome'] = $_GET['nome'];

    if (array_key_exists('descricao', $_GET)) {
        $tarefa['descricao'] = $_GET['descricao'];
    } else {
        $tarefa['descricao'] = '';
    }

    if (array_key_exists('prazo', $_GET)) {
        $tarefa['prazo'] =
            traduz_data_para_banco($_GET['prazo']);
    } else {
        $tarefa['prazo'] = '';
    }

    $tarefa['prioridade'] = $_GET['prioridade'];

    if (array_key_exists('concluida', $_GET)) {
        $tarefa['concluida'] = 1;
    } else {
        $tarefa['concluida'] = 0;
    }
}

editar_tarefa($conexao, $tarefa);
}

$tarefa = buscar_tarefa($conexao, $_GET['id']);

require "template.php";

```

Veja que ele é praticamente uma cópia do arquivo tarefas.php , mas com algumas alterações fundamentais.

Precisamos criar a nova função `buscar_tarefa()`. Para manter nosso código mais organizado, esta nova função deverá ficar junto com as demais que lidam com banco de dados no arquivo `banco.php`:

```
<?php  
// ...  
  
function buscar_tarefa($conexao, $id) {  
    $sqlBusca = 'SELECT * FROM tarefas WHERE id = ' . $id;  
    $resultado = mysqli_query($conexao, $sqlBusca);  
    return mysqli_fetch_assoc($resultado);  
}  
  
// ...
```

Repare que sempre usaremos o campo `id` da tabela `tarefas` para fazer referência a uma tarefa como única. Isso é um padrão para os bancos de dados e, em geral, para diversas linguagens.

Agora vamos alterar o arquivo `formulario.php` para que ele já venha com os dados da tarefa preenchidos, de modo que o usuário possa fazer a edição.

Vamos primeiro adicionar o campo `id` no formulário, assim ele será enviado junto com o restante dos campos quando o usuário pedir para salvar a edição das tarefas. Este campo não precisa ser exibido para o usuário, pois não existe a necessidade de edição dele, então vamos usar um campo do tipo `hidden`:

```
<form>  
    <input type="hidden" name="id"  
          value="php echo $tarefa['id']; ?&gt;" /&gt;<br/    <fieldset>  
  
    ...
```

No restante dos campos, vamos preencher os valores usando o

atributo `value` ou o conteúdo do elemento quando for texto, que neste caso são os campos `tarefa`, `descricao` e `prazo`. Vou colocar apenas os campos no código a seguir, mas lembre-se de que eles estão separados e existe mais código entre eles:

```
<input type="text" name="nome"  
      value="php echo $tarefa['nome']; ?&gt;" /&gt;<br/  
<textarea name="descricao">  
    <?php echo $tarefa['descricao']; ?>  
</textarea>  
  
<input type="text" name="prazo"  
      value="php echo traduz_data_para_exibir($tarefa['prazo']);<br/            ?>"  
      />
```

Lembra da função `traduz_data_para_exibir()`? Olha só como ela está sendo útil mais uma vez, agora para exibir a data no formato correto para edição.

O preenchimento dos campos `prioridade` e `concluida` é um pouco diferente, já que precisamos do atributo `checked` para marcar o campo:

```
...  
    <input type="radio" name="prioridade" value="1"  
          <?php echo ($tarefa['prioridade'] == 1)  
              ? 'checked'  
              : '';  
          ?> /> Baixa  
  
    <input type="radio" name="prioridade" value="2"  
          <?php echo ($tarefa['prioridade'] == 2)  
              ? 'checked'  
              : '';  
          ?> /> Média  
  
    <input type="radio" name="prioridade" value="3"  
          <?php echo ($tarefa['prioridade'] == 3)
```

```
? 'checked'  
: '';  
?> /> Alta  
  
<!-- Agora o campo de concluída -->  
  
<input type="checkbox" name="concluida" value="1"  
    <?php echo ($tarefa['concluida'] == 1)  
        ? 'checked'  
        : '';  
    ?> />
```

9.2 OPERADOR TERNÁRIO

Não se assuste com o trecho anterior! Nele estamos usando o ternário do PHP, que é tipo um `if`, mas com uma sintaxe reduzida.

O ternário funciona assim: você coloca uma condicional, e logo depois uma opção para a condicional verdadeira. Em seguida, uma opção para a condicional falsa.

No começo parece estranho, mas logo se percebe que esta forma de usar condicionais pode ser muito útil em alguns casos. Veja um exemplo com um `if` normal, igual ao que já vimos até agora:

```
<?php  
  
$idade = $_GET['idade'];  
  
if ($idade >= 18) {  
    echo "Bem vindo!";  
} else {  
    echo "Entrada proibida!";  
}
```

Agora, veja o mesmo exemplo, usando ternário:

```
<?php  
  
$idade = $_GET['idade'];  
  
echo ($idade > 18) ? "Bem vindo!" : "Entrada proibida!";
```

Bem mais simples, certo? Agora toda a condição cabe em apenas uma linha e fica bem fácil de ler e sem um monte de código para apenas um **echo**.

O ternário funciona sempre assim: coloca-se uma condição, uma interrogação, o resultado da condição verdadeira, dois pontos e o resultado da condição falsa.

9.3 NOVIDADE NO PHP 7: O OPERADOR "??"

E por falar em ternário, uma novidade bacana no PHP 7 é o **Null coalescing operator**, que funciona como se fosse um ternário, mas é usado para pegar valores de variáveis, quando definidos, ou valores padrão.

Veja um exemplo que tudo fica mais simples de entender:

```
<?php  
  
// Em vez de:  
$nome = (array_key_exists('nome', $_GET))  
    ? $_GET['nome']  
    : "Anônimo";  
  
// Podemos fazer apenas:  
$nome = $_GET['nome'] ?? "Anônimo";
```

Bastante simples, né? Usamos o operador **??** para pegar o valor do índice `nome`, se ele existir, ou a string **Anônimo**. Bem menor e mais simples do que usando o ternário.

Mas lembre-se de que esta sintaxe está disponível apenas do PHP 7 em diante.

9.4 DE VOLTA À EDIÇÃO DE TAREFAS

Uma pequena mudança que podemos fazer é alterar também o texto do botão `submit` no final do formulário, que atualmente é `Cadastrar`. A ideia é exibir `Atualizar` quando estivermos editando uma tarefa. Neste caso, o ternário também será bastante útil:

...

```
<input type="submit" value="  
<?php echo ($tarefa['id'] > 0) ? 'Atualizar' : 'Cadastrar'; ?>  
/>
```

...

Muito bem, neste momento o formulário de edição já está exibindo os dados das tarefas. Mas ainda não escrevemos a função `editar_tarefa()`. Esta função receberá um `array` com os dados da tarefa, e então fará a atualização do registro no banco usando a instrução `UPDATE` da `SQL`. Veja um exemplo de como funciona a instrução `UPDATE`:

```
UPDATE tabela SET  
    campo1 = 'valor',  
    campo2 = 123  
WHERE id = 1;
```

A função `UPDATE` é simples. Nela informamos qual tabela queremos atualizar, informamos os novos valores usando `SET` e, o que é bastante importante, informamos qual registro (ou quais registros) deve ser atualizado usando o `WHERE`.

É importante lembrar de usar o `WHERE` sempre para evitar de atualizar todos os registros ao mesmo tempo, pois é isso que acontece quando não o usamos. É quase como chegar no banco e falar: *"Hey, banco, atualiza o nome da tarefa para Comprar Pão"*, e o banco responder algo como *"Ok, todas as suas 95 tarefas se chamam Comprar Pão agora"*.

Algumas configurações dos bancos de dados existem para impedir isso de acontecer, exatamente para evitar esse tipo de atualização por engano. Mais uma vez, vale lembrar de que nossas funções que lidam com o banco de dados são apenas geradoras e executoras de SQL. Por isso é importante se dedicar a aprender SQL tanto quanto ao aprendizado de PHP. ;-)

Veja como deverá ficar a função `editar_tarefa()`:

```
<?php
// ...

function editar_tarefa($conexao, $tarefa)
{
    $sqlEditar = "
        UPDATE tarefas SET
            nome = '{$tarefa['nome']}',
            descricao = '{$tarefa['descricao']}',
            prioridade = {$tarefa['prioridade']},
            prazo = '{$tarefa['prazo']}',
            concluida = {$tarefa['concluida']}
        WHERE id = {$tarefa['id']}
    ";
    mysqli_query($conexao, $sqlEditar);
}

// ...
```

Com esta função pronta, já podemos editar as tarefas. Mas se você acessar o arquivo `tarefas.php` neste momento, você verá

diversos erros na página, mais ou menos conforme a figura a seguir:

Gerenciador de Tarefas

The screenshot shows a web form titled "Nova tarefa" (New Task). The "Tarefa:" field contains the notice:
Notice: Undefined variable: tarefa in /home/junior/projetos/ph. The "Descrição (Opcional):" field contains another notice:
Notice: Undefined variable: tarefa in /home/junior/projetos/phpmysql /exemplos/afazeresbd2/formulario.php on line 11
. The "Prazo (Opcional):" field contains a third notice:
Notice: Undefined variable: tarefa in /home/junior/projetos/ph. The "Prioridade:" dropdown menu has two options: "Baixa" (Low) and "Alta" (High), with "Baixa" selected. The "Notice" option is also present in the dropdown.

Figura 9.1: Diversos erros no formulário

Nossa! Quantos erros! Mas se analisarmos melhor veremos que, na verdade, todos eles são referentes à mesma variável `$tarefa`. Este erro está acontecendo porque mudamos o formulário para preencher os dados de uma tarefa sendo editada, pois tomamos a decisão de usar apenas um arquivo para o formulário. Agora teremos de adaptar o formulário para o cadastro de novas tarefas, que antes já funcionava corretamente.

Neste caso, a correção do problema é simples: basta fornecermos uma tarefa em branco para o formulário de cadastro de tarefas. Assim, ele sempre terá uma variável `$tarefa` para exibir, evitando os erros de variável não definida.

No arquivo `tarefas.php`, adicionaremos esta nova `$tarefa` em branco, que será um array. Vamos adicionar essa variável logo antes do `require` do `template.php`:

```

<?php
// ...

$tarefa = [
    'id'          => 0,
    'nome'        => '',
    'descricao'   => '',
    'prazo'        => '',
    'prioridade'  => 1,
    'concluida'   => ''
];

require "template.php";
// ...

```

Como estamos fornecendo uma tarefa em branco, o formulário não gera mais erros. Então, vamos editar uma tarefa! Acesse o endereço <http://localhost/tarefas/tarefas.php>, e clique no link `Editar` em uma das tarefas. Você deverá ver apenas o formulário preenchido com os dados da tarefa, sem a lista de tarefas embaixo:

Gerenciador de Tarefas

Nova tarefa

Tarefa:
Estudar HTML

Descrição (Opcional):
HTML importante!

Prazo (Opcional):

Prioridade:
 Baixa Média Alta

Tarefa concluída:

Figura 9.2: Nosso formulário para edição de tarefas

Agora podemos editar nossas tarefas, uma vez que já temos o formulário. O arquivo `editar.php` já trata os dados e também já

estamos enviando a tarefa para a função `editar_tarefa()` que usa o comando `UPDATE` do MySQL para atualizar os dados.

Mas algo ainda não está legal. Perceba que, após salvar uma tarefa, continuamos na página de edição. Esta é uma boa hora para apresentar uma funcionalidade interessante da navegação via HTTP: os **redirecionamentos**.

A ideia aqui é quebrar o fluxo da aplicação logo após a execução da função `editar_tarefa()`, para que o usuário seja encaminhado novamente para a página inicial — ou seja, a lista de tarefas, lá no arquivo `tarefas.php`. O PHP em si não possui uma ferramenta de redirecionamento, pois isso é feito pelo HTTP, por meio de informações para o navegador.

Aqui vou mostrar uma forma simples de fazer este redirecionamento, mas lembre-se de mais tarde parar para estudar um pouco mais sobre o protocolo HTTP, ok? Como existe muita informação na internet, basta pesquisa por "como funciona o protocolo http" que você vai encontrar bastante coisa.

Para fazer o redirecionamento, precisamos falar para o navegador que ele deve ir para um local diferente do atual. Isso é feito por cabeçalhos HTTP. Neste caso, o cabeçalho que precisamos é o `Location`.

O PHP usa a função `header()` para mandar esses cabeçalhos para o navegador. Veja como fica o uso da função `header()` para redirecionarmos nossos usuários de volta para `tarefas.php` após a atualização de uma tarefa com a função `editar_tarefa()`:

```
<?php  
// ...
```

```
editar_tarefa($conexao, $tarefa);
header('Location: tarefas.php');
die();

// ...
```

Logo após a função `header()`, foi adicionada a função `die()`, que encerra o processo do PHP imediatamente. A chamada da função `die()` vai evitar que o restante do arquivo `editar.php` seja executado de forma desnecessária, já que o que precisamos após a edição de uma tarefa é apenas o redirecionamento para a lista de tarefas.

Com isso, temos a lista de tarefas já funcional para cadastro e também para edição de tarefas! Mas agora precisamos de uns ajustes importantes, como a possibilidade de remover tarefas e também a validação do formulário, para evitarmos, por exemplo, o cadastro de tarefas em branco ou com datas inválidas.

9.5 REMOÇÃO DE TAREFAS

A remoção das tarefas será bem mais simples que a edição, pois dessa vez não precisamos pegar dados no banco, preencher formulário etc. Agora tudo o que precisamos é do número da tarefa para remover, ou seja, seu `id` no banco de dados.

Vamos começar adicionando o link `Remover` na nossa lista de tarefas. Este link ficará junto do link `Editar`, na coluna `Opções` da tabela, então vamos editar o arquivo `tabela.php` para fazer a adição do novo link:

...

```
<td>
<a href="editar.php?id=<?php echo $tarefa['id']; ?>">
```

```
        Editar  
    </a>  
    <a href="remover.php?id=<?php echo $tarefa['id']; ?>">  
        Remover  
    </a>  
</td>  
  
...
```

Perceba que o link de remoção é muito parecido com o link de edição que já conhecemos. Afinal, para remover uma tarefa, também precisaremos do seu **id** no banco de dados, mas o endereço aponta para o arquivo `remover.php`, que ainda não existe.

O nosso arquivo `remover.php` também deverá utilizar o arquivo `banco.php` para acessar o banco, e também deverá redirecionar o usuário após a remoção da tarefa, assim como fizemos na edição. Este arquivo será bem menor do que os demais, porque precisa fazer menos trabalho do que a lista e a edição de tarefas. Veja como ele deve ficar:

```
<?php  
  
require "banco.php";  
  
remover_tarefa($conexao, $_GET['id']);  
  
header('Location: tarefas.php');
```

E pronto, este é o arquivo `remover.php` em todo o seu esplendor. Sim, apenas isso! É claro que ainda precisamos criar a função `remover_tarefa()` lá no arquivo `banco.php`, mas a lógica da remoção é basicamente esta.

No arquivo `banco.php`, vamos adicionar a função `remover_tarefa()`. Esta vai usar o comando `DELETE` da

linguagem **SQL**, que tem a sintaxe bem parecida com o `SELECT`. Veja o exemplo:

```
DELETE FROM tarefas WHERE id = 123;
```

Este exemplo removeria o registro da tabela `tarefas` com o **id** `123`. Mas também podemos fazer algo assim com SQL:

```
DELETE FROM tarefas WHERE concluida = 1;
```

Este comando removeria da tabela `tarefas` todos os registros onde o campo `concluida` fosse igual a `1`, ou seja, nossas tarefas concluídas.

Assim como o `UPDATE`, o `DELETE` deve sempre conter um `WHERE` para evitar remover dados de toda a tabela. Aqui também existem configurações no MySQL e outros bancos para prevenir o uso do `DELETE` sem especificar quais registros usando o `WHERE`.

Mas vamos voltar à função `remover_tarefa()` no arquivo `banco.php`:

```
<?php  
// ...  
  
function remover_tarefa($conexao, $id)  
{  
    $sqlRemover = "DELETE FROM tarefas WHERE id = {$id}";  
  
    mysqli_query($conexao, $sqlRemover);  
}
```

Com a função `remover_tarefa()` pronta, já podemos usar o novo link `Remover` na lista de tarefas para removermos tarefas. Faça um teste, deu certo? Se sim, vamos em frente! Caso tenha algum problema, revise seu código até aqui e leia as mensagens de erro que o PHP informa, assim fica mais fácil de resolver os

problemas.

9.6 EVITANDO O PROBLEMA COM A ATUALIZAÇÃO DE PÁGINA

Ainda temos um pequeno problema com a nossa lista de tarefas! Não acredita? Então faça o seguinte: adicione uma tarefa, adicionou? Agora atualize a página usando o F5 , ou o botão de atualizar do seu navegador. Viu o que acontece? Ele repete o último cadastro feito! Veja como ficou a minha lista:

Teste	Um teste	Baixa	Não	Editar Remover
Teste	Um teste	Baixa	Não	Editar Remover
Teste	Um teste	Baixa	Não	Editar Remover
Teste	Um teste	Baixa	Não	Editar Remover

Figura 9.3: Tarefas repetidas após atualizar a página depois de uma gravação

Esse problema acontece pois, quando atualizamos a página, o navegador manda os dados novamente para o PHP. Como o PHP recebeu dados, ele acaba disparando toda a rotina de gravação de novas tarefas.

Para evitarmos este problema, vamos usar o truque que aprendemos com as páginas de edição e de remoção, e vamos redirecionar o usuário de volta para a lista de tarefas após a gravação de uma tarefa. Para isso, usaremos a função `header()` logo após a função `gravar_tarefa()` no arquivo `tarefas.php` :

```
<?php  
// ...  
  
gravar_tarefa($conexao, $tarefa);
```

```
header('Location: tarefas.php');
die();

// ...
```

A função `die()` aqui tem o mesmo propósito de quando usada no arquivo `editar.php`: encerrar a execução do PHP logo após enviar o cabeçalho de redirecionamento para o navegador.

Faça um teste agora e, após a gravação, você verá o formulário e a lista de tarefas normalmente, e ainda poderá atualizar a página sem o problema de duplicação da última tarefa cadastrada.

9.7 RESUMO

Até este momento, já vimos bastante coisa. Saímos do zero e já temos um ambiente PHP e MySQL (ou MariaDB) com a nossa primeira aplicação: um gerenciador de tarefas!

Neste capítulo, fizemos a edição das tarefas e também sua remoção. Além disso, corrigimos alguns comportamentos usando o `Redirect` do HTTP, através da função `header()` do PHP.

Neste capítulo, também foram usados os comandos `UPDATE` e `DELETE` da linguagem `SQL`. No próximo, faremos a validação dos dados enviados pelo usuário, assim evitamos problemas como datas inválidas, tarefas vazias etc.

9.8 DESAFIOS

Existem diversas formas de melhorar o que já fizemos até aqui. Copie o projeto para uma nova pasta, algo como `tarefas_experimentos` e tente desenvolver funcionalidades

como:

1. Um botão de cancelar quando estiver editando uma tarefa. Este botão, ou link, envia o usuário de volta para a lista de tarefas.
2. Um botão para duplicar uma tarefa, junto com os botões de editar e remover.
3. Que tal uma opção para apagar todas as tarefas concluídas?
4. Otimizar o código para ter menos repetição. Veja que os arquivos `tarefas.php` e `editar.php` são muito parecidos. Você consegue pensar em maneiras de reutilizar partes deles? Que tal um arquivo novo com as bases para eles? Isso será feito nos capítulos finais desse livro, mas já vale a pena fazer um experimento.

Além destes desafios para a lista de tarefas, continue os outros desafios dos capítulos anteriores:

1. Adicione uma opção para editar e outra para remover os contatos da lista de contatos.
2. Adicione uma opção para listar apenas os contatos favoritos.
3. Adicione opções para remover os dados dos veículos cadastrados no banco de dados do estacionamento.
4. Faça uma opção no desafio do estacionamento para mostrar os veículos que entraram no estacionamento em uma determinada data.

CAPÍTULO 10

VALIDAÇÃO DE FORMULÁRIOS

Um problema comum em praticamente qualquer software que tenha de lidar com dados fornecidos por um usuário é a validação. Sabe quando você preenche um formulário em um site (ou um software desktop, um app para smartphone etc.) e, logo após enviar os dados, o formulário é exibido novamente com seus dados e mais alguns alertas de campos que são obrigatórios ou que foram preenchidos incorretamente? Pois então, este é um cenário onde aconteceu algum tipo de validação de entrada de dados.

A validação de dados é importante para exigir que o usuário cadastre as informações obrigatórias e também para impedir a entrada de dados que podem comprometer o sistema, como a data 31 de fevereiro.

10.1 VALIDAÇÃO NA LISTA DE TAREFAS

Nossa lista de tarefas tem um campo obrigatório, que é o nome da tarefa e um campo que pode gerar erros se preenchido incorretamente, que é o campo de prazo, onde usamos uma data.

Atualmente, estamos usando o campo de nome da tarefa para

decidir se vamos gravar a tarefa ou não. Isso é um pouco ruim, pois se alguém preencher os dados da tarefa, menos o nome, e enviar, perderá o que foi digitado e a tarefa não será gravada.

Podemos começar validando essa entrada, para saber se o formulário foi enviado independentemente do campo de nome da tarefa ser preenchido ou não. Mas, antes de fazer esta validação, vamos pegar um pedaço deste capítulo para falar da entrada de dados.

10.2 ENTRADA DE DADOS USANDO POST

Até este momento, estamos usando a superglobal `$_GET` do PHP para pegar os dados enviados pelo usuário no formulário, e também nas páginas de edição e de remoção para pegar o `id` da tarefa que queremos editar ou remover.

Você já deve ter usado diversos formulários em sites e outras aplicações online e percebido que os dados digitados em um formulário não ficam expostos no endereço, assim como nosso formulário de tarefas está fazendo.

Existem basicamente duas formas de enviar dados usando um formulário em um navegador: uma delas é utilizando o `GET`, e a outra é usando o `POST`. `GET` e `POST` são diferentes métodos do HTTP.

Usando o `POST`, os dados do formulário não ficam mais no endereço da página de destino, o que pode tornar a aplicação um pouco mais segura. Imagine um formulário que envia uma senha e ela fica exposta na URL de destino. Nada bom, né? Pois alguém sentado ao lado do usuário poderia ler a senha facilmente.

ENTÃO O POST É MAIS SEGURO?

Não exatamente, pois os dados ainda são enviados de forma aberta pelo navegador. Eles apenas não serão exibidos no endereço. Mas se você usar as ferramentas para desenvolvedores dos navegadores e olhar a aba rede ou network, você poderá ver os dados sendo enviados. E se alguém mal intencionado estiver analisando a sua conexão, poderá facilmente ler essas informações.

Por isso, para garantir mais segurança no envio de dados, o ideal é utilizar o HTTPS. Mas este assunto foge do escopo deste livro, então fica o conselho para que você também estude sobre HTTPS e sobre como usar o HTTPS em servidores web.

Uma dica bacana é estudar sobre o Let's Encrypt (<https://letsencrypt.org/>), uma autoridade certificadora que oferece certificados grátis para todos que desejam usar HTTPS.

Certo, mas como podemos trocar o uso do GET nos formulários da nossa lista de tarefas para o POST ? O primeiro passo é alterar o formulário para que use o método POST . Para isso, é necessário adicionar o atributo method com o valor POST no elemento form . Estas alterações devem ser feitas no arquivo formulario.php :

```
<form method="POST">  
  <input type="hidden" name="id"
```

```
        value=<?php echo $tarefa['id']; ?>"  
/>
```

...

Com isso, o formulário passa a enviar os dados usando o `POST`. O padrão dos navegadores é enviar usando `GET`, por isso esse método era usado quando não tínhamos informado explicitamente qual usar.

O HTTP possui outros métodos usados para diferentes tarefas, mas os navegadores em geral suportam apenas `GET` e `POST`. Então, vamos usar apenas estas opções.

Agora também é necessário alterar toda a parte que pega os dados do formulário para cadastrar no banco, porque estamos usando a superglobal `$_GET`. Esta alteração será bem simples, já que o PHP cuida da entrada via `POST` também.

Então, tudo o que temos de fazer é trocar `$_GET` por `$_POST` nos campos relacionados ao formulário. Essas alterações devem ser feitas nos arquivo `tarefas.php` e `editar.php`. Veja como fica a alteração no arquivo `tarefas.php`:

```
<?php  
// ...  
  
if (array_key_exists('nome', $_POST) && $_POST['nome'] != '') {  
    $tarefa = array();  
  
    $tarefa['nome'] = $_POST['nome'];  
  
    if (array_key_exists('descricao', $_POST)) {  
        $tarefa['descricao'] = $_POST['descricao'];  
    } else {  
        $tarefa['descricao'] = '';  
    }
```

```

if (array_key_exists('prazo', $_POST)) {
    $tarefa['prazo'] =
        traduz_data_para_banco($_POST['prazo']);
} else {
    $tarefa['prazo'] = '';
}

$tarefa['prioridade'] = $_POST['prioridade'];

if (array_key_exists('concluida', $_POST)) {
    $tarefa['concluida'] = 1;
} else {
    $tarefa['concluida'] = 0;
}

gravar_tarefa($conexao, $tarefa);
header('Location: tarefas.php');
die();
}

// ...

```

As alterações no arquivo `editar.php` também são simples:

```

<?php
// ...

if (array_key_exists('nome', $_POST) && $_POST['nome'] != '') {
    $tarefa = array();

    $tarefa['id'] = $_POST['id'];

    $tarefa['nome'] = $_POST['nome'];

    if (array_key_exists('descricao', $_POST)) {
        $tarefa['descricao'] = $_POST['descricao'];
    } else {
        $tarefa['descricao'] = '';
    }

    if (array_key_exists('prazo', $_POST)) {
        $tarefa['prazo'] =
            traduz_data_para_banco($_POST['prazo']);
    } else {

```

```
        $tarefa['prazo'] = '';
    }

$tarefa['prioridade'] = $_POST['prioridade'];

if (array_key_exists('concluida', $_POST)) {
    $tarefa['concluida'] = 1;
} else {
    $tarefa['concluida'] = 0;
}

editar_tarefa($conexao, $tarefa);
header('Location: tarefas.php');
die();
}

// ...
```

O uso do POST também é importante para formulários que enviam arquivos, mas veremos isso nos capítulos a seguir.

10.3 VALIDANDO O NOME DA TAREFA

Vamos à validação do nosso primeiro campo, que será o campo com o nome da tarefa a ser cadastrada. Atualmente, já existe uma espécie de validação que verifica se o nome da tarefa foi digitado ou não, e então decide se vai cadastrar a tarefa no banco.

Esse tipo de decisão vai sempre depender de o campo com o nome da tarefa ter sido digitado pelo usuário. Isso pode trazer problemas, pois o usuário pode ter digitado o restante dos campos, mas ter se esquecido do nome da tarefa. Neste caso, ele perderá tudo o que digitou no restante dos campos.

Vamos tratar primeiro da validação na página de cadastro, ou seja, no arquivo `tarefas.php`. Vamos começar alterando a linha que verifica se tem de entrar ou não no processo de gravação de

novas tarefas. Isso porque ela verifica apenas se o índice `nome` existe em `$_POST`, e não necessariamente se existem mais coisas no `$_POST`. Esta é a linha:

```
<?php  
// ...  
  
if (array_key_exists('nome', $_POST) && $_POST['nome'] != '') {  
  
// ...
```

Essa linha será substituída por esta:

```
<?php  
// ...  
  
if (tem_post()) {  
  
// ...
```

A função `tem_post()` ainda não existe, então vamos adicioná-la no arquivo `ajudantes.php`:

```
<?php  
// ...  
  
function tem_post()  
{  
    if (count($_POST) > 0) {  
        return true;  
    }  
  
    return false;  
}
```

Veja que a função `tem_post()` faz uma contagem do número de índices existentes em `$_POST`, e retorna `true` quando este número for maior que zero, uma situação que vai acontecer apenas se alguma informação for enviada via `POST`.

Certo, agora o processo de gravação inicia se tiver dados no POST , independente de o campo nome ter sido enviado. Isso é bom, pois vamos validar este campo e repreencher os dados no formulário, caso a validação não passe.

Vamos criar uma nova variável de controle chamada \$tem_erros e também um array para guardar os erros em cada campo chamado \$erros_validacao . Ambos ficarão logo acima do if que verifica se tem POST :

```
<?php  
// ...  
  
$exibir_tabela = true;  
  
$tem_erros = false;  
$erros_validacao = [];  
  
if (tem_post()) {  
  
// ...
```

No caso, \$tem_erros já começa como false , e será alterada para true conforme as validações acontecerem. O array \$erros_validacao também receberá seus valores conforme as validações acontecerem.

Para validar o campo com o nome da tarefa, precisamos apenas verificar se ele não está vazio. Para isso, vamos alterar a linha que pega o nome da tarefa no arquivo tarefas.php :

```
<?php  
// ...  
  
$tarefa['nome'] = $_POST['nome'];  
  
// ...
```

Esta linha está logo após o `if` que acabamos de alterar para usar a função `tem_post()`. No lugar dela, faremos uma verificação parecida com a verificação que já existe para os demais campos, com a diferença de que agora precisamos preencher o array com os erros e alterar a variável `$tem_erros` para `true`, caso o nome da tarefa esteja vazio:

```
<?php
// ...

if (array_key_exists('nome', $_POST)
    && strlen($_POST['nome']) > 0) {
    $tarefa['nome'] = $_POST['nome'];
} else {
    $tem_erros = true;
    $erros_validacao['nome'] = 'O nome da tarefa é obrigatório!';
}

// ...
```

A maior novidade neste trecho é o uso da função `strlen()`, que conta o tamanho de uma string. O que estamos fazendo é verificar se o nome está no `POST` e se a contagem de caracteres é maior do que zero.

Repare que o `else` está criando um índice `nome` no array `$erros_validacao` com uma frase indicando o erro que aconteceu. Além disso, também aproveitamos o mesmo `else` para alterar a variável `$tem_erros` para `true`.

Agora que temos uma variável que diz se algum erro aconteceu, podemos usá-la para decidir se vamos ou não fazer a gravação da tarefa. Como já temos uma função que faz a gravação, podemos colocar a chamada desta função dentro de um `if`:

```
<?php
// ...
```

```
if (! $tem_erros) {  
    gravar_tarefa($conexao, $tarefa);  
    header('Location: tarefas.php');  
    die();  
}  
  
// ...
```

Veja que usamos a exclamação para negar, inverter, o valor de `$tem_erros`, então esta condição pode ser lida como "se não tem erros". Caso algum erro exista, a função `gravar_tarefa()` não será chamada, e o arquivo `tarefas.php` continuará a ser executado, já que o usuário não será redirecionado para a lista de tarefas. Se você tentar incluir uma tarefa vazia neste momento, você simplesmente voltará ao formulário, mas sem avisos de erro.

10.4 ADICIONANDO O AVISO DE ERRO

Para adicionar o aviso de erro, vamos alterar o arquivo `formulario.php`. Mais especificamente, vamos alterar a parte que cria o campo para o nome da tarefa:

```
...  
  
<label>  
    Tarefa:  
    <input type="text" name="nome"  
           value=<?php echo $tarefa['nome']; ?> />  
</label>  
  
...
```

As variáveis `$tem_erros` e `$erros_validacao` serão usadas para decidir se o erro será exibido e para pegar o valor erro. Altere o trecho dentro da tag `label` para ficar assim:

```
...
```

```
<label>
    Tarefa:
    <?php
if ($tem_erros && array_key_exists('nome', $erros_validacao)) : ?
>
    <span class="erro">
        <?php echo $erros_validacao['nome']; ?>
    </span>
<?php endif; ?>
<input type="text" name="nome"
       value="<?php echo $tarefa['nome']; ?>" />
</label>
```

...

Estamos usando duas condições dentro do `if` : uma para verificar se `$tem_erros` é verdadeira, e outra para verificar se o índice `nome` existe dentro do array `$erros_validacao` . Esta verificação do índice é necessária, pois logo mais faremos a validação dos outros campos. Então pode ser que, por exemplo, `$tem_erros` seja `true` , mas que o erro não seja no campo `nome` .

O erro foi exibido dentro de um elemento `span` com a classe `erro` , assim podemos usar CSS para estilizar a frase de erro. No meu caso, eu deixei a frase em vermelho. Se você usar o CSS dos exemplos, terá o mesmo resultado.

Faça uma nova tentativa de cadastro sem digitar o nome da tarefa. Você deverá ver o formulário novamente, com o erro sendo exibido:

Gerenciador de Tarefas

The screenshot shows a form titled "Nova tarefa". It contains several input fields and a radio button group. The first field, "Tarefa", has a red error message: "Tarefa: O nome da tarefa é obrigatório!". Below it is a "Descrição (Opcional)" field with a text area. A "Prazo (Opcional)" field follows, containing a date input. Under "Prioridade:", there is a radio button group with three options: "Baixa" (selected), "Média", and "Alta". At the bottom left is a checkbox for "Tarefa concluída:" and a "Cadastrar" button on the right.

Figura 10.1: Exibição do erro no nome da tarefa

Mas ainda temos um problema. Se experimentarmos cadastrar uma tarefa apenas com uma descrição, voltaremos para a página do formulário, mas sem os dados da descrição.

Este é um ponto importante da validação, pois quando validamos um formulário e informamos para o usuário que existe algum campo faltando, não podemos apagar o restante das informações já fornecidas. Quem nunca passou por um formulário que apagou os dados por conta de apenas um campo esquecido? É bem chato, não?

Para preencher novamente os campos que já foram enviados, faremos uso do array `$tarefa`, que é gerado antes de incluirmos o arquivo `template.php`, lá no final do arquivo `tarefas.php`:

```
<?php  
// ...  
  
$tarefa = [  
    'id'        => 0,  
    'nome'      => '',  
    'descricao' => '' ,
```

```
'prazo'      => '',
'prioridade' => 1,
'concluida'   => ''
];
// ...
```

Este array é gerado com dados em branco. Então, precisamos verificar se existem dados enviados pelo `POST`, para que sejam usados no lugar dos valores em branco. Lembra do ternário? Aqui ele será bem útil:

```
<?php
// ...

$tarefa = [
    'id'          => 0,
    'nome'        => (array_key_exists('nome', $_POST)) ?
        $_POST['nome'] : '',
    'descricao'   => (array_key_exists('descricao', $_POST)) ?
        $_POST['descricao'] : '',
    'prazo'        => (array_key_exists('prazo', $_POST)) ?
        traduz_data_para_banco($_POST['prazo']) : '',
    'prioridade'  => (array_key_exists('prioridade', $_POST)) ?
        $_POST['prioridade'] : 1,
    'concluida'   => (array_key_exists('concluida', $_POST)) ?
        $_POST['concluida'] : ''
];
// ...
```

Deixei o ternário em duas linhas em cada um dos campos para facilitar a leitura aqui no livro, mas ele poderia ficar em apenas uma linha no código.

Perceba que, em cada um dos campos, é verificado se existe um valor no `POST`, e então este valor é usado. Caso não exista, usamos

um valor padrão em branco.

O campo que foge um pouco dos demais é o **prazo**. Nele estamos usando a função `traduz_data_para_banco()`. Mas, espera um pouco, na verdade não estamos enviando esta data para o banco! Então, por que usar esta função?

Vamos analisar a criação do campo do prazo, lá no arquivo `formulario.php`:

```
<label>
    Prazo (Opcional):
    <input type="text" name="prazo"
        value="<?php
            echo traduz_data_exibir($tarefa['prazo']);
        ?>">
</label>
```

Veja que na criação do campo no formulário, estamos usando sempre a função `traduz_data_exibir()`, que espera um texto vazio, ou uma data no formato *ANO-MÊS-DIA*, mas a data que entramos no formulário está no formato *DIA/MÊS/ANO*. Para evitar erros na formatação da data, vamos montar o array com o formato esperado pelo nosso formulário.

Agora, preencha os campos do formulário, menos o nome da atividade, e tente fazer a gravação. Você verá novamente o formulário com o erro de validação do nome, assim como da outra vez, mas agora os demais campos já estarão preenchidos com os dados informados:

Gerenciador de Tarefas

Nova tarefa

Tarefa: O nome da tarefa é obrigatório!

Descrição (Opcional):
Descrição da minha tarefa

Prazo (Opcional):
22/09/2013

Prioridade:
 Baixa Média Alta

Tarefa concluída:

Cadastrar

Figura 10.2: Exibição de erro no nome, mas com o formulário preenchido

Se preenchermos também o nome da tarefa, ela será gravada normalmente. O próximo passo é validar o prazo, pois os demais campos são opcionais e não precisam de validação de dados corretos.

10.5 VALIDANDO A DATA DIGITADA

O prazo é um campo opcional, mas existe o problema de o usuário digitar datas inválidas, como um 31 de fevereiro, ou mesmo textos que não estejam no formato DIA/MÊS/ANO que estamos esperando.

Vamos focar na validação do formato primeiro, depois vamos para a validação de datas que existem ou não. Faremos isso para agir em um problema de cada vez, o que vai facilitar a implementação.

Vamos definir então qual será o formato de data aceita em nossa aplicação. O dia poderá ter um ou dois dígitos e o mês

também, mas o ano deverá ter sempre quatro dígitos. Veja alguns exemplos de datas que serão consideradas válidas e outras inválidas também:

```
# Datas válidas
03/03/2016
3/07/2016
03/7/2016
3/7/2016
```

```
# Datas inválidas
003/7/2016
3/007/16
3/7/16
03/07/16
03/07
```

Certo. Tendo definido os formatos que serão aceitos pela aplicação, podemos fazer a validação. Agora, como faremos isso?

Já sabemos que podemos explodir uma string usando um separador e transformá-la em um array. Aí poderíamos contar os caracteres de cada item do array para saber se a data está no formato esperado. Isso com certeza vai dar um trabalhão e não será muito prático. Mesmo assim, veja mais ou menos como ficaria a validação usando esta técnica:

```
<?php

function validar_data($data)
{
    $partes = explode('/', $data);

    if (count($partes) != 3) {
        return false;
    }

    $dia = $partes[0];
    $mes = $partes[1];
    $ano = $partes[2];
```

```
if (strlen($dia) < 1 OR strlen($dia) > 2) {
    return false;
}

if (strlen($mes) < 1 OR strlen($mes) > 2) {
    return false;
}

if (strlen($ano) != 4) {
    return false;
}

return true;
}
```

Esta implementação é até fácil de ler, mas não é lá muito prática.

Imagine validar o formato de um CPF assim, ou um CEP, ou um CNPJ, ou um código de barras de um boleto bancário. Enfim, deve existir algo que verifica certos padrões em um texto e diz se ele está no padrão, certo?

10.6 EXPRESSÕES REGULARES

Este é um tópico muito interessante que merece um livro só para ele, mas vamos com calma. Mostrarei apenas o básico necessário para validarmos o prazo das nossas atividades, e deixarei a sugestão para que você se aprofunde no assunto, pois realmente vale a pena.

A expressões regulares nos permitem pesquisar textos dentro de outros textos, e verificar se determinados textos seguem um padrão predefinido. Usando expressões regulares, podemos fazer coisas como a validação do formato que queremos para a nossa

data do prazo de uma forma muito mais simples do que a forma manual, que vimos anteriormente.

Usar expressões regulares para validar o formato da nossa data é algo mais ou menos assim: *verifique se o texto começa com um ou dois dígitos, seguido de uma barra, seguido por mais um ou dois dígitos, seguido por mais uma barra e, por último, uma sequência de quatro dígitos*.

Este formato que deve ser encontrado é chamado de **padrão**, e um padrão que valida, por exemplo, apenas um número com um dígito é este:

[0-9]

Isso significa qualquer número de 0 até 9, ou seja, todos os dígitos do sistema decimal. Então, para validar dois números seguidos, poderíamos usar um padrão como o seguinte:

[0-9][0-9]

Basta repetir o padrão, assim teríamos uma validação para dois números seguidos. Mas imagine que precisássemos validar 10 números seguidos, escrever o padrão 10 vezes não é prático e, com certeza, dificultaria muito o entendimento do código. Por isso, existem também quantificadores para as expressões regulares. Veja o exemplo:

[0-9]{10}

Isso quer dizer 10 vezes um dígito. Mas, além de poder colocar uma quantidade fixa de repetições do padrão, podemos colocar intervalos:

[0-9]{3,5}

Isso significa um padrão de três a cinco dígitos. Com isso, podemos voltar ao padrão para validar o prazo das atividades, que ficaria assim:

```
[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}
```

Ler esta expressão regular é algo como: "*um ou dois dígitos, uma barra, um ou dois dígitos, uma barra, quatro dígitos*". Perceba que existem barras invertidas antes das barras, isso é necessário para "escapar" as barras. Ou seja, estamos dizendo para a expressão regular que ela não deve tratar a barra como um caractere especial.

Mas ainda podemos melhorar um pouco a expressão, pois temos de verificar se este é o único texto digitado no campo do prazo. Para isso, precisamos adicionar os caracteres que significam o **começo** e o **fim** do texto em um padrão, que são respectivamente o acento circunflexo e o cifrão:

```
^([0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4})$
```

Ufa, expressão regular já montada e pronta para usar. Aliás, uma dica legal aqui é usar o site **regexpal** para testar suas expressões regulares (<http://regexpal.com/>). Este foi o site que usei para construir e testar a nossa expressão regular:



Figura 10.3: Regexpal sendo usado para testar a validação da data

Certo, já sabemos um básico de expressões regulares, agora precisamos aplicar a validação dentro do PHP. Para isso, vamos usar a função `preg_match()`, que recebe um padrão e um texto, e verifica se o texto casa com o padrão. Criaremos uma nova função `validar_data()` no arquivo `ajudantes.php`:

```
<?php
// ...

function validar_data($data)
{
    $padrao = '/^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$/';
    $resultado = preg_match($padrao, $data);

    return ($resultado == 1);
}

// ...
```

Ei! Este padrão está diferente do que fizemos!

Calma, ele tem apenas uma barra no começo e outra no final, pois este é um padrão usado em algumas linguagens, inclusive na linguagem Perl, que é de onde o PHP herda algumas coisas.

Agora que já temos a função para validar o prazo, vamos adicioná-la no arquivo `tarefas.php`, já aproveitando para colocar o erro de validação:

```
<?php
// ...

if (array_key_exists('prazo', $_POST)
    && strlen($_POST['prazo']) > 0) {
    if (validar_data($_POST['prazo'])) {
        $tarefa['prazo'] =
            traduz_data_para_banco($_POST['prazo']);
    } else {
        $tem_erros = true;
        $erros_validacao['prazo'] =
```

```

        'O prazo não é uma data válida!';
    }
} else {
    $tarefa['prazo'] = '';
}
// ...

```

Veja que é necessário primeiro verificar se existe algum texto no campo prazo, para depois usar a função `validar_data()`, pois este é um campo opcional. Se usarmos apenas a função `validar_data()`, o usuário não vai conseguir adicionar tarefas sem um prazo.

Precisamos alterar também o arquivo `formulario.php` para exibir o erro de validação do prazo:

```

<!-- ... -->

<label>
    Prazo (Opcional):
    <?php
    if ($tem_erros
        && array_key_exists('prazo', $erros_validacao)) : ?>
        <span class="erro">
            <?php echo $erros_validacao['prazo']; ?>
        </span>
    <?php endif; ?>
    <input type="text" name="prazo" value=
        "<?php echo
            traduz_data_para_exibir($tarefa['prazo']); ?>">
    />
</label>

<!-- ... -->

```

Agora já podemos tentar cadastrar uma tarefa com o prazo incorreto, como por exemplo, "12/12/12":

Gerenciador de Tarefas

Nova tarefa

Tarefa: O nome da tarefa é obrigatório!

Descrição (Opcional):

Prazo (Opcional): O prazo não é uma data válida!

12/12/12

Prioridade:

Baixa Média Alta

Tarefa concluída:

Figura 10.4: Erro de validação do prazo

Mas se digitarmos algo como "amanhã" no prazo, ou uma data como "12/12", seremos agraciados com vários erros, pois as funções `traduz_data_para_exibir()` e `traduz_data_para_banco()`, usadas para montar o array `&tarefa` e para exibir o formulário, esperam datas com três partes separadas por traços ou barras.

Por isso, vamos adicionar uma pequena modificação nestas duas funções para que retornem a data da forma como a receberam, caso ela não tenha três partes separadas por barras. Vale lembrar de que as funções estão no arquivo `ajudantes.php`:

```
<?php  
// ...  
  
function traduz_data_para_banco($data)  
{  
    if ($data == "") {
```

```

        return "";
}

$partes = explode("/", $data);

if (count($partes) != 3) {
    return $data;
}

$objeto_data = DateTime::createFromFormat('d/m/Y', $data);

return $objeto_data->format('Y-m-d');
}

function traduz_data_para_exibir($data)
{
    if ($data == "" OR $data == "0000-00-00") {
        return "";
    }

    $partes = explode("-", $data);

    if (count($partes) != 3) {
        return $data;
    }

    $objeto_data = DateTime::createFromFormat('Y-m-d', $data);

    return $objeto_data->format('d/m/Y');
}

// ...

```

Veja que apenas adicionei um `if` para verificar se a quantidade de partes da data, após o uso da função `explode()`, é igual a três. Agora sim, podemos usar a lista de tarefas com a validação da data e sem os erros em caso de datas em formatos incorretos.

Entretanto, ainda temos um pequeno problema: a validação do prazo apenas verifica se o formato de entrada da data está correto,

sem considerar se a data existe ou não. Desse modo, podemos digitar datas como "99/99/9999" que não teremos um erro de validação, mesmo que esta data não exista.

Validar se uma data é válida é uma atividade bem chatinha. Sério, além de o fato de cada mês ter uma certa quantidade de dias, ainda temos os maravilhosos anos bissextos, nos quais fevereiro tem 29 dias.

Ainda bem que PHP possui a função `checkdate()` que verifica se uma data é válida. Então, vamos usá-la dentro da nossa função `validar_data()`:

```
<?php
// ...

function validar_data($data)
{
    $padrao = '/^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$/';
    $resultado = preg_match($padrao, $data);

    if ($resultado == 0) {
        return false;
    }

    $dados = explode('/', $data);

    $dia = $dados[0];
    $mes = $dados[1];
    $ano = $dados[2];

    $resultado = checkdate($mes, $dia, $ano);

    return $resultado;
}

// ...
```

A função `checkdate()` recebe três parâmetros, sendo o

primeiro o mês, o segundo o dia e o terceiro o ano. Esta função retorna `true` apenas quando os dados informados formarem uma data válida. Experimente cadastrar uma atividade para o dia 30 de fevereiro, ou para 31 de abril, ou para 32 de dezembro. Você sempre receberá o erro de validação dizendo que a data é inválida.

Neste momento, temos a validação funcional para a adição de novas tarefas!

10.7 VALIDANDO O FORMULÁRIO DE EDIÇÃO DE TAREFAS

Com o cadastro de tarefas já funcional com validação, podemos agora alterar a edição de tarefas para que a validação também funcione. No arquivo `editar.php`, adicione a variável de controle `$tem_erros` e `$erros_validacao`, logo após a variável `$exibir_tabela`:

```
<?php  
// ...  
  
$exibir_tabela = false;  
$tem_erros = false;  
$erros_validacao = [];  
  
// ...
```

Altere também o `if` que verifica se os dados foram enviados via `POST`, que atualmente usa o campo com o nome da tarefa, para usar a nossa função `tem_post()`:

```
<?php  
// ...  
  
$exibir_tabela = false;  
$tem_erros = false;
```

```
$erros_validacao = [];
```

```
if (tem_post()) {
```

```
// ...
```

O próximo passo é alterar o trecho que pega o nome da tarefa, para que faça a validação e preencha o array de erros, se necessário:

```
<?php
// ...

if (array_key_exists('nome', $_POST)
    && strlen($_POST['nome']) > 0) {
    $tarefa['nome'] = $_POST['nome'];
} else {
    $tem_erros = true;
    $erros_validacao['nome'] =
        'O nome da tarefa é obrigatório!';
}

// ...
```

Também devemos alterar a definição do campo prazo, validando com a função `validar_data()`:

```
<?php
// ...

if (array_key_exists('prazo', $_POST)
    && strlen($_POST['prazo']) > 0) {
    if (validar_data($_POST['prazo'])) {
        $tarefa['prazo'] =
            traduz_data_para_banco($_POST['prazo']);
    } else {
        $tem_erros = true;
        $erros_validacao['prazo'] =
            'O prazo não é uma data válida!';
    }
}

// ...
```

E no final do bloco do `if` , precisamos alterar a chamada da função `editar_tarefa()` para usar a variável `$tem_erros` :

```
<?php
// ...

if (! $tem_erros) {
    editar_tarefa($conexao, $tarefa);
    header('Location: tarefas.php');
    die();
}

// ...
```

Este trecho todo fica mesmo muito parecido com o arquivo `tarefas.php` , com apenas alguns pontos diferentes, mas importantes, como chamar a função `editar_tarefa()` no lugar de `gravar_tarefa()` .

Se você tentar editar uma tarefa neste momento, já verá a validação funcionando. Entretanto, o repreenchimento dos campos não estará funcionando corretamente. Os valores exibidos em caso de erro serão exatamente os mesmos que estão cadastrados no banco, ou seja, a tarefa atual.

Vamos acertar este problema verificando se existem dados no `POST` logo após a chamada da função `buscar_tarefa()` , que está bem no final do arquivo `editar.php` . Os dados que essa função retorna são armazenados na variável `$tarefa` , que é um array com os dados da tarefa.

Então, o que precisamos fazer é substituir os valores recuperados do banco pelos que estão no `POST` . Vai ficar até parecido com a parte da validação, com a diferença de que não será necessário validar:

```
<?php
// ...

$tarefa = buscar_tarefa($conexao, $_GET['id']);

$tarefa['nome'] = (array_key_exists('nome', $_POST)) ?
    $_POST['nome'] : $tarefa['nome'];

$tarefa['descricao'] = (array_key_exists('descricao', $_POST)) ?
    $_POST['descricao'] : $tarefa['descricao'];

$tarefa['prazo'] = (array_key_exists('prazo', $_POST)) ?
    $_POST['prazo'] : $tarefa['prazo'];

$tarefa['prioridade'] = (array_key_exists('prioridade', $_POST)) ?
    $_POST['prioridade'] : $tarefa['prioridade'];

$tarefa['concluida'] = (array_key_exists('concluida', $_POST)) ?
    $_POST['concluida'] : $tarefa['concluida'];

// ...
```

Olha o nosso amigo ternário dando as caras novamente. Veja que, nesta parte, o que fazemos é a verificação do campo no POST . Quando ele não é encontrado, usamos o valor padrão que já está no array \$tarefa .

Não precisamos mais de alterações para que a validação funcione também na edição das tarefas, pois o formulário já está preparado para exibir os erros quando necessário. Então, já temos a adição e a edição de atividades funcionais e com validação.

10.8 RESUMO

Ufa, este foi um capítulo e tanto, *hein?* Para fazer a validação de apenas dois campos, passamos por alguns tópicos bem interessantes, como as famosas expressões regulares e também a

validação de datas. Não foi necessário alterar profundamente nossa aplicação, somente alguns pontos chave para a lógica e também a exibição dos erros no formulário.

Para o repreenchimento dos campos em caso de erros de validação, usamos os dados do `POST`, aproveitando a requisição atual. Existem técnicas de validação que guardam os dados na sessão e redirecionam o usuário para uma página de erros.

Você pode experimentar outras técnicas e até mesmo bibliotecas PHP especializadas em formulários e validação, mas saber o funcionamento básico é importante. Por isso recomendo treinar mais validações desta forma que fizemos.

10.9 DESAFIOS

Pensou que ficaria sem desafios? Aqui estão mais alguns para praticar:

1. Faça um formulário que peça um CEP e valide o valor digitado usando o padrão XXXXX-XXX, ou seja, cinco dígitos, um traço e mais três dígitos.
2. Faça um formulário que peça um CPF e valide se o valor foi digitado no formato correto: XXX.XXX.XXX-XX (X são os números). Use expressões regulares aqui.
3. Adicione validação para a lista de contatos que já está sendo feita nos desafios há alguns capítulos. Tente validar a entrada do nome, que deve ser obrigatório, do telefone, usando o formato "(00)0000-0000", do e-mail e também da data de nascimento. Aqui também vale tentar validar números de telefones móveis, no formato "(00)00000-0000".

4. Adicione validação também para o projeto do estacionamento. As placas dos veículos devem seguir o padrão "AAA-0000", ou seja, três letras e quatro números.

CAPÍTULO 11

UPLOAD DE ARQUIVOS

Uma tarefa bem comum em aplicações web é o upload de arquivos. Vez ou outra, nós acabamos enviando fotos, documentos e outros tipos de arquivos para sites e outras aplicações web.

O processo de envio de arquivos é bem parecido com o envio de dados em formulários. A diferença é que escolhemos os arquivos em nossos computadores em vez de digitar ou selecionar opções em formulários.

Neste capítulo, veremos como enviar arquivos para que sejam tratados pelo PHP e armazenados para que o usuário possa, futuramente, baixar o arquivo novamente.

11.1 ANEXOS PARA A LISTA DE TAREFAS

Nossa lista de tarefas agora também terá anexos, pois não é difícil imaginar que o usuário queira cadastrar uma tarefa como **imprimir o relatório** e queira, é claro, anexar o relatório à tarefa. Esta é uma funcionalidade muito prática, e sua implementação vai nos ensinar mais alguns conceitos muito interessantes sobre como funciona o desenvolvimento para a web.

O armazenamento de arquivos pode ser feito de duas maneiras

diferentes. A primeira é o armazenamento dos arquivos no próprio banco de dados, assim como estamos fazendo com os dados das tarefas até aqui. A segunda é o armazenamento dos arquivos no sistema de arquivos do servidor, ou seja, em pastas/diretórios.

Usaremos uma técnica que mescla o uso do banco de dados com o sistema de arquivos, mantendo o arquivo no sistema de arquivos e apenas uma referência ao nome dele no banco de dados. Assim saberemos onde encontrar o arquivo novamente para que o usuário possa fazer o download.

Faremos isso para manter o banco de dados menor e não ter de converter os arquivos para um formato que o banco entenda, e depois converter de volta na hora de fazer o download dos arquivos ou o envio por e-mails no capítulo seguinte.

Caso você queira ter mais informações sobre o assunto, pesquise por "armazenamento de arquivo no banco de dados ou no sistema de arquivos". Você vai encontrar diversos pontos de vista e diversos experimentos sobre o assunto, mas vai perceber que no geral a opção de usar o sistema de arquivos é a mais utilizada.

Antes de colocar a mão na massa, precisamos definir como será esse sistema de anexos:

- Teremos uma página com os detalhes da tarefa, onde estará o formulário para adicionar os anexos.
- Uma tarefa poderá ter nenhum, um ou mais anexos.
- Permitiremos anexar apenas arquivos .zip ou .pdf .
- O usuário poderá remover o anexo, sem remover a tarefa.

Uau, temos algum trabalho pela frente, mas o resultado será bem interessante. Vamos começar alterando o banco de dados, depois vamos adicionar a página com os detalhes das tarefas e o formulário para adicionar anexos. Por último, faremos o tratamento do envio dos arquivos e listagem dos arquivos na página de detalhes da tarefa.

Esta ordem é importante, pois quando chegarmos no ponto no qual salvaremos os dados no banco de dados, já devemos ter o banco pronto para receber os dados.

11.2 MUDANÇAS NO BANCO DE DADOS

Nosso banco de dados atual possui apenas a tabela `tarefas`, então precisamos fazer alterações para que seja possível guardar os dados relacionados aos anexos. Se adicionarmos um novo campo chamado, por exemplo, `anexo` à tabela `tarefas`, poderemos adicionar apenas um anexo por tarefa. Mas como já definimos que uma tarefa poderá ter vários anexos, esta solução não nos atenderá.

Sendo assim, teremos de adicionar mais uma tabela em nosso banco para guardar exclusivamente os anexos. Esta nova tabela deverá guardar uma referência às tarefas para que se possa associá-las a seus anexos. Veja como fica o diagrama para a nova tabela `anexos` e sua associação com a tabela `tarefas`:

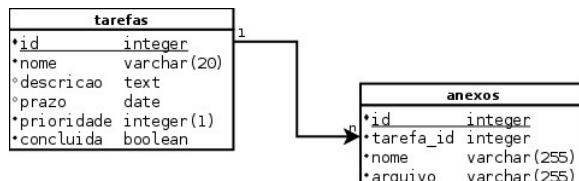


Figura 11.1: Diagrama do banco com a tabela anexos

A tabela `anexos` terá seu próprio `id` para identificação dos anexos como únicos. Terá também um campo `tarefa_id` que vai guardar uma referência ao campo `id` da tabela `tarefas`.

Falando um pouco mais sobre bancos de dados relacionais e suas teorias, os campos `id` em cada uma das tabelas são chamados de **chaves primárias**, pois reconhecem cada registro como único. Já o campo `tarefa_id` na tabela `anexos` guarda uma referência a um registro único em outra tabela, por isso ele é chamado de **chave estrangeira**.

Além disso, a nova tabela também terá dois campos adicionais: um para guardarmos o nome do arquivo sem sua extensão, e o outro a sua localização no sistema de arquivos.

Esse processo de separação de tabelas e uso de chaves primárias e chaves estrangeiras faz parte de uma disciplina chamada **normalização de bancos de dados**. Eu recomendo que você leia mais a respeito para evitar problemas comuns de duplicação de dados, ou até mesmo de perda de informações relacionais em bancos de dados.

Para criar a nova tabela, acesse o PHPMyAdmin e siga um processo parecido com o que usamos para criar a tabela das tarefas. Dessa vez, o código SQL necessário para criar a tabela `anexos` é o seguinte:

```
CREATE TABLE anexos (
    id      INTEGER AUTO_INCREMENT PRIMARY KEY,
    tarefa_id  INTEGER NOT NULL,
    nome     VARCHAR(255) NOT NULL,
    arquivo   VARCHAR(255) NOT NULL
);
```

Este código é bastante parecido com o que usamos

anteriormente para criar a tabela das tarefas, mudando apenas os nomes dos campos. Após executar este código, o PHPMyAdmin deverá exibir as duas tabelas no banco tarefas :

Figura 11.2: As tabelas no banco tarefas

11.3 PÁGINA COM OS DETALHES DAS TAREFAS

Com o banco de dados já pronto para receber os anexos das tarefas, vamos para a próxima etapa: a criação de uma página que exibirá os dados de uma tarefa e um formulário para cadastrar anexos.

A nossa nova página ficará no endereço <http://localhost/tarefas/tarefa.php?id=N>, onde N será o id da tarefa que queremos exibir. Este endereço estará em um link que vamos adicionar agora na lista de tarefas, usando o nome da tarefa. No arquivo `tabela.php`, altere a linha que exibe o nome, adicionando o link:

```
...
<td>
    <a href="tarefa.php?id=<?php echo $tarefa['id']; ?>">
        <?php echo $tarefa['nome']; ?>
    </a>
</td>
...

```

Este link aponta para o arquivo `tarefa.php`, que ainda não existe. Nossa próximo passo é a criação desse arquivo, que terá uma estrutura semelhante ao `tarefas.php`, mas deverá mostrar apenas uma tarefa por vez:

```
<?php

include "banco.php";
include "ajudantes.php";

$tem_erros = false;
$erros_validacao = [];

if (tem_post()) {
    // upload dos anexos
}

$tarefa = buscar_tarefa($conexao, $_GET['id']);

include "template_tarefa.php";
```

Perceba que já temos todas as funções que estão sendo usadas neste arquivo. Por enquanto, vamos deixar a parte do upload apenas com um comentário, e vamos criar o arquivo `template_tarefa.php`, que está sendo incluído na última linha:

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Gerenciador de Tarefas</title>
        <link rel="stylesheet" href="tarefas.css"
              type="text/css" />
    </head>
    <body>
        <div="bloco_principal">
            <h1>Tarefa: <?php echo $tarefa['nome']; ?></h1>
            <p>
                <a href="tarefas.php">
                    Voltar para a lista de tarefas
                </a>
            </p>
        </div>
    </body>
</html>
```

```
<p>
    <strong>Concluída:</strong>
    <?php echo
        traduz_concluida($tarefa['concluida']); ?>
</p>
<p>
    <strong>Descrição:</strong>
    <?php echo nl2br($tarefa['descricao']); ?>
</p>
<p>
    <strong>Prazo:</strong>
    <?php echo
        traduz_data_exibir($tarefa['prazo']); ?>
</p>
<p>
    <strong>Prioridade:</strong>
    <?php echo
        traduz_prioridade($tarefa['prioridade']); ?>
</p>

<h2>Anexos</h2>
<!-- lista de anexos -->

    <!-- formulário para um novo anexo -->
</div>
</body>
</html>
```

Este arquivo serve para exibir os dados da tarefa, sua lista de anexos e também o formulário para adição de novos anexos. Deixe os trechos com a lista de anexos e com o formulário de anexos em branco, pois vamos preencher estes trechos mais para a frente.

Acesse a lista de tarefas e clique no link no nome da tarefa para acessar a página com os seus detalhes. Você deverá ver uma página parecida com esta:

Tarefa: Estudar HTML

[Voltar para a lista de tarefas.](#)

Concluída: Não

Descrição: HTML é muito importante!

Prazo:

Prioridade: Média

Anexos

Novo anexo

Figura 11.3: Página com os detalhes da tarefa

11.4 O FORMULÁRIO PARA CADASTRAR ANEXOS

Ainda no arquivo `template_tarefa.php`, criaremos o formulário para o envio de anexos. Este será bem simples e terá apenas um campo para a seleção do arquivo, um campo `hidden` com o `id` da tarefa e o botão de envio:

```
<!-- ... -->

<!-- formulário para um novo anexo -->
<form action="" method="post" enctype="multipart/form-data">
    <fieldset>
        <legend>Novo anexo</legend>

        <input type="hidden"
            name="tarefa_id"
            value=<?php echo $tarefa['id']; ?> />

        <label>
            <?php if (
                $tem_erro
                &&
                array_key_exists('anexo', $erro_validacao)
            ) { ?>
```

```

):?>
    <span class="erro">
        <?php echo $erros_validacao['anexo']; ?>
    </span>
<?php endif; ?>

    <input type="file" name="anexo" />
</label>

    <input type="submit" value="Cadastrar" />
</fieldset>
</form>

<!-- ... -->

```

Repare nas diferenças do formulário de cadastro de anexos para o formulário de cadastro de tarefas. A tag `<form>` do formulário de cadastro de anexos tem a propriedade `enctype="multipart/form-data"`, que serve para indicar para o navegador que nosso formulário vai fazer o envio de arquivos.

Outra diferença é o `input` do anexo, que tem a propriedade `type="file"`. Ele serve para indicar que este campo deverá ser preenchido com um arquivo escolhido no computador do usuário.

Como já temos a experiência do formulário de cadastro de tarefas, também já podemos deixar o formulário de cadastro de anexos pronto para exibir erros de validação, mesmo sem ter feito a validação ainda.

Veja como fica este formulário na página com os detalhes da tarefa:

Novo anexo-

Selecionar arquivo... Nenhum arquivo selecionado.

Cadastrar

Figura 11.4: Form para envio de anexos

11.5 RECEBENDO ARQUIVOS PELO PHP

Após adicionar o formulário, vamos alterar o arquivo `tarefa.php` para que ele possa receber o arquivo. Faremos a validação e gravação do anexo naquele `if` que verifica se existem dados no post.

A estrutura que devemos construir verificará se o arquivo foi enviado e depois verificará se o arquivo tem a extensão `pdf` ou `zip`, afinal, estes foram os arquivos que decidimos aceitar. É só então que salvaremos o arquivo em um diretório e também faremos a gravação das informações do arquivo no banco de dados. Este trecho de código deverá ficar assim:

```
<?php
// ...

if (tem_post()) {
    // upload dos anexos
    $tarefa_id = $_POST['tarefa_id'];

    if (! array_key_exists('anexo', $_FILES)) {
        $tem_erros = true;
        $erros_validacao['anexo'] =
            'Você deve selecionar um arquivo para anexar';
    } else {
        if (tratar_anexo($_FILES['anexo'])) {
            $nome = $_FILES['anexo']['name'];
            $anexo = [
                'tarefa_id' => $tarefa_id,
                'nome' => substr($nome, 0, -4),
                'arquivo' => $nome,
            ];
        } else {
            $tem_erros = true;
            $erros_validacao['anexo'] =
                'Envie anexos nos formatos zip ou pdf';
        }
    }
}
```

```
    if (! $tem_erros) {
        gravar_anexo($conexao, $anexo);
    }
}

// ...
```

Perceba que esta estrutura é bem parecida com a gravação de tarefas, as diferenças estão no uso das duas novas funções: `tratar_anexo()` e `gravar_anexo()`. A função `tratar_anexo()` será responsável por verificar se o anexo está no formato correto, e também por copiá-lo para a pasta onde será armazenado. Já a função `gravar_anexo()` será responsável por cadastrar o anexo no banco de dados, no mesmo estilo da função `gravar_tarefa()`.

Uma outra novidade ali é a função `substr()`, que pega trechos de uma string. Neste caso, estamos usando esta função para pegar o nome do arquivo menos sua extensão, ou seja, sem os últimos quatro caracteres.

Para isso, passamos para a função o nome do arquivo, na variável `$nome`, o começo do trecho do texto que queremos extrair, que é a posição zero e, por último, até onde queremos cortar o texto, que é até faltar 4 caracteres para o final do nome. Assim, se o nome for algo como `relatorio.pdf`, o `.pdf` será removido e ficaremos apenas com o texto `relatorio`.

Na função `tratar_anexo()`, teremos de verificar se o arquivo é do tipo correto, ou seja, `pdf` ou `zip`. Quando o tipo for correto, vamos guardar o arquivo em uma pasta; e quando não for, exibiremos uma mensagem de erro no formulário.

GUARDAR ARQUIVOS EM PASTAS? E O BANCO DE DADOS?

Pode parecer estranho, mas não vamos usar o banco de dados para guardar os anexos dos usuários. O que vamos guardar no banco de dados é apenas o nome do arquivo. Assim, teremos uma referência para acessar cada arquivo quando necessário.

Já os arquivos serão armazenados em um diretório, uma pasta, no servidor, assim como guardamos arquivos em nossos computadores.

Esta é uma maneira simples e eficiente para lidar com o upload e armazenamento de arquivos. Ela mantém o banco de dados menor, apenas com os nomes dos arquivos e deixa os arquivos para o sistema de arquivos do sistema operacional. Faz sentido, certo?

Veja como deverá ficar a função `tratar_anexo()`, que será adicionada no arquivo `ajudantes.php`:

```
<?php
// ...

function tratar_anexo($anexo) {
    $padrao = '/^.+(\.pdf|\.\zip)$/';
    $resultado = preg_match($padrao, $anexo['name']);

    if ($resultado == 0) {
        return false;
    }

    move_uploaded_file(
        $anexo['tmp_name'],
        "anexos/{$anexo['name']}"
    )
}
```

```
 );
    return true;
}
// ...
```

A forma que estamos usando para verificar se o arquivo é do tipo correto é pela extensão. E aqui temos mais um uso importante para as expressões regulares.

Desta vez, estamos usando o ponto, que é uma espécie de curinga das expressões regulares e serve para casar com qualquer coisa. Logo após o ponto, temos o sinal de mais, que indica que o ponto (ou seja, qualquer coisa) deve casar pelo menos uma vez e pode casar diversas vezes.

Este trecho `.+` é o grande coringa das expressões regulares e serve para casar qualquer texto. Voltando ao nosso caso, queremos apenas os arquivos que terminem com `.pdf` ou `.zip`. Por isso, precisamos adicionar os trechos `\.pdf` e `\.zip`.

Perceba que nestes trechos queremos que a expressão case o ponto literal, não qualquer caractere, como fizemos antes. Para tanto, usamos a barra invertida antes dele, para fazer o **escape** do ponto. Isso significa algo como: *"ei, expressão regular, aqui eu quero um ponto mesmo, não qualquer coisa, como é o padrão quando se usa o ponto"*.

Ainda tem mais um detalhe importante nesta expressão regular: o uso do **ou**, pois precisamos que o arquivo seja `zip` **ou** `pdf`. Isso é feito usando os parênteses e o pipe (barra vertical), assim: `(isso|aquilo)`. Esta expressão casaria as palavras `isso` ou `aquilo`.

Não falei que as expressões regulares merecem um livro só para elas?

Muito bem, após casar a expressão regular, movemos o arquivo temporário para o seu destino final usando a função `move_uploaded_file()`.

ARQUIVO TEMPORÁRIO? O QUE TEM NO \$_FILES?

Quando enviamos um arquivo através de um formulário para o PHP, este cria a superglobal `$_FILES`. Sim, no mesmo estilo das superglobais `$_GET` e `$_POST`. Cada campo do tipo `file` é colocado em um array dentro de `$_FILES`, por isso conseguimos chamar a função `tratar_anexo()`, passando o índice `anexo`, que é o nome do campo do anexo no nosso formulário.

Cada índice do `$_FILES` é um array que contém os seguintes índices:

- `name` — nome do arquivo;
- `type` — o tipo mime do arquivo;
- `tmp_name` — o nome onde o arquivo foi salvo temporariamente;
- `error` — um número que representa um erro no upload, sendo que zero significa que está tudo certo;
- `size` — o tamanho do arquivo.

O arquivo enviado para o servidor é gravado em um nome temporário, por isso precisamos pegá-lo e copiá-lo para seu destino final. Veja que a função `tratar_anexo()` está movendo o arquivo temporário para uma pasta chamada `anexos`, que ainda não existe.

Para mover o arquivo para sua localização final, usamos a

função `move_uploaded_file()` . Ela faz o que seu nome diz, verificando se o arquivo de origem realmente foi enviado para o PHP através de um formulário na requisição atual, e depois move o arquivo de um lugar para outro.

A PASTA DE UPLOADS

Estamos enviando os arquivos para uma pasta chamada `anexos` usando a função `move_uploaded_file()` . Para que isso funcione, ela precisa existir. Então, crie uma pasta chamada `anexos` na mesma pasta onde está o arquivo `tarefas.php` (ou seja, a pasta `tarefas`).

Atenção: usuários de sistemas GNU/Linux e Mac OS X (ou outros baseados em Unix) deverão dar permissão de escrita para o Apache no diretório de upload. Um `chmod 777 anexos/` resolve, mas você também pode mudar o grupo deste diretório e dar permissão para o usuário do servidor web que, no Debian/Ubuntu, é o `www-data` , por padrão.

A função `tratar_anexo()` retorna `true` ou `false` . Caso seja `true` , criamos um array novo chamado `$anexo` e guardamos nele os dados do upload. Se for `false` , geramos um erro que será exibido no formulário.

11.6 GRAVANDO OS DADOS DO ANEXO NO BANCO DADOS

Estando tudo certo com o upload, vamos para a gravação dos

dados no banco usando a função `gravar_anexo()`, que ficará no arquivo `banco.php`:

```
<?php
// ...

function gravar_anexo($conexao, $anexo)
{
    $sqlGravar = "INSERT INTO anexos
        (tarefa_id, nome, arquivo)
    VALUES
    (
        {$anexo['tarefa_id']},
        '{$anexo['nome']}',
        '{$anexo['arquivo']}'
    )
    ";

    mysqli_query($conexao, $sqlGravar);
}

// ...
```

Sem muitos segredos, certo? A maior diferença entre a gravação dos dados de uma tarefa e de um anexo está no campo `tarefa_id`, que guarda uma referência à tarefa a qual um anexo pertence. Como já falei anteriormente neste capítulo, isso faz parte da normalização de bancos de dados e vale um estudo adicional sobre o assunto.

11.7 EXIBINDO OS ANEXOS

Agora precisamos buscar os anexos no banco para poder fazer uma lista na página com os detalhes da tarefa. Para isso, usaremos uma função nova chamada `buscar_anexos()`, que receberá, como sempre, a conexão e também o `id` da tarefa, neste caso. Adicione a chamada desta função no final do arquivo

tarefa.php , logo abaixo da linha que busca os dados da tarefa:

```
<?php  
// ...  
  
$tarefa = buscar_tarefa($conexao, $_GET['id']);  
$anexos = buscar_anexos($conexao, $_GET['id']);  
  
include "template_tarefa.php";  
  
// ...
```

Já a função buscar_anexos() deverá ser adicionada no arquivo banco.php :

```
<?php  
// ...  
  
function buscar_anexos($conexao, $tarefa_id)  
{  
    $sql = "SELECT * FROM anexos  
           WHERE tarefa_id = {$tarefa_id}";  
    $resultado = mysqli_query($conexao, $sql);  
  
    $anexos = [];  
  
    while ($anexo = mysqli_fetch_assoc($resultado)) {  
        $anexos[] = $anexo;  
    }  
  
    return $anexos;  
}  
  
// ...
```

Quase sem diferenças para a função buscar_tarefas() , né? Basicamente, muda só o código SQL que estamos usando. Veja que a consulta SQL filtra apenas as tarefas que contenham o campo tarefa_id com o valor da tarefa que queremos exibir.

Como o campo tarefa_id é uma chave em outra tabela (a

tabela `tarefas`), ela é chamada de chave estrangeira. Ops, já falei sobre isso, mas é bom reforçar. :)

Para exibir os anexos, precisamos adicionar uma tabela no arquivo `template_tarefa.php` , logo abaixo do H2 que diz Anexos e acima do formulário para cadastrar um anexo. Para criar a tabela, vamos usar a variável `$anexos` que contém o resultado da função `buscar_anexos()` :

```
<!-- ... -->

<h2>Anexos</h2>

<!-- lista de anexos -->

<?php if (count($anexos) > 0) : ?>
    <table>
        <tr>
            <th>Arquivo</th>
            <th>Opções</th>
        </tr>

        <?php foreach ($anexos as $anexo) : ?>
            <tr>
                <td><?php echo $anexo['nome']; ?></td>
                <td>
                    <a href=""anexos/<?php echo \$anexo\['arquivo'\]; ?>">Download
                </td>
            </tr>
        <?php endforeach; ?>

    </table>
<?php else : ?>
    <p>Não há anexos para esta tarefa.</p>
<?php endif; ?>

<!-- formulário para um novo anexo -->

<!-- ... -->
```

Veja que estamos verificando se a lista de anexos tem algum índice antes de tentar exibir a lista. Cada item da lista tem um elemento a que faz um link para o download do arquivo.

Aqui não tem segredos, é um link normal que aponta para o arquivo guardado na pasta anexos . Neste caso, não precisamos escrever mais código para lidar com o download do arquivo, pois o servidor web, no nosso caso o Apache, se encarrega disso.

Experimente fazer o upload do arquivo. Sua página de detalhes da tarefa deverá ficar parecida com esta:

Tarefa: Estudar HTML

[Voltar para a lista de tarefas.](#)

Concluída: Não

Descrição: HTML é muito importante!

Prazo:

Prioridade: Média

Anexos

Arquivo	Opções
textos.zip	Download

Novo anexo:

Nenhum arquivo selecionado.

Figura 11.5: Página com os detalhes da tarefa, a lista de anexos e o formulário para cadastrar novos anexos

11.8 REMOVENDO ANEXOS

A remoção dos anexos será bastante parecida com a remoção

das tarefas. Vamos usar um `id` enviado pela URL para apagar o registro no banco de dados.

Entretanto, aqui teremos ainda um passo adicional: remover o arquivo do anexo do sistema de arquivos. Pois, sem um registro no banco de dados, não faz sentido manter um arquivo no sistema de arquivos.

Vamos começar adicionando um link na tabela que lista os anexos para um novo arquivo chamado `remover_anexo.php`. Este é bastante semelhante ao link para remover as tarefas, e deve conter o `id` do anexo a ser removido.

Para adicionar o link para remoção do anexo, edite o arquivo `template_tarefa.php`, adicionando o link ao lado do link para download do anexo:

```
...
<td>
    <a href="anexos/<?php echo $anexo['arquivo']; ?>">
        Download
    </a>
    <a href="remover_anexo.php?id=<?php echo $anexo['id']; ?>">
        Remover
    </a>
</td>
...
...
```

O novo arquivo `remover_anexo.php` será bastante parecido com o arquivo `remover.php`:

```
<?php
include "banco.php";
$anexo = buscar_anexo($conexao, $_GET['id']);
```

```
remover_anexo($conexao, $anexo['id']);
unlink('anexos/' . $anexo['arquivo']);

header('Location: tarefa.php?id=' . $anexo['tarefa_id']);
```

Neste arquivo, estamos incluindo a conexão com o banco de dados e usando as funções `buscar_anexo()` e `remover_anexo()` (que ainda não existem), e também a função `unlink()` do PHP. Logo após o uso das funções que removem o anexo do banco e do sistema de arquivos, fazemos o redirecionamento do usuário de volta à página da tarefa.

Repare que, antes de remover os dados, fazemos uma busca no banco usando o `id` do anexo para ter em mãos o nome do arquivo a ser removido e também o `id` da tarefa na qual o anexo está ligado. Esses dados ficam guardados na variável `$anexo`.

A função `unlink()` recebe como parâmetro o caminho de um arquivo para ser removido do sistema de arquivos. Bastante simples, né?

A nova função `buscar_anexo()` deve ser adicionada no arquivo `banco.php` e é bastante simples, seguindo o padrão da função `buscar_tarefa()`:

```
<php

function buscar_anexo($conexao, $id)
{
    $sqlBusca = 'SELECT * FROM anexos WHERE id = ' . $id;
    $resultado = mysqli_query($conexao, $sqlBusca);

    return mysqli_fetch_assoc($resultado);
}
```

Já a nova função `remover_anexo()` deverá ser adicionada no arquivo `banco.php`. Ela também segue o padrão de uma função

já existente, a `remover_tarefa()` :

```
<?php

function remover_anexo($conexao, $id)
{
    $sqlRemover = "DELETE FROM anexos WHERE id = {$id}";

    mysqli_query($conexao, $sqlRemover);
}
```

Edições feitas e novo arquivo criado, tente remover um anexo clicando no link **Remover** na lista de anexos de uma tarefa. Após o teste, verifique se os dados do anexo foram apagados do banco de dados e se o arquivo foi apagado da pasta anexos. Se algo não saiu como o esperado, faça a revisão do código até aqui e se atente para as mensagens de erro do PHP.

11.9 RESUMO

Neste capítulo, tratamos do upload de arquivos usando PHP. Os uploads são colocados dentro da superglobal `_FILES`. Além disso, criamos uma nova tabela no banco, usamos mais um pouco de expressões regulares para validar o tipo de arquivo enviado. Também usamos a função `unlink()` do PHP para apagar arquivos do sistema de arquivos.

11.10 DESAFIOS

Depois de trabalhar duro na adição e remoção de anexos para as tarefas, vamos para mais alguns desafios!

Na lista de contatos (sim, aquela dos desafios anteriores):

- Adicione uma página para exibir os detalhes de cada contato.
- Adicione o upload de fotos para a lista de contatos. Lembre-se de validar se o arquivo é uma imagem, como jpg , png etc.
- Adicione a foto do contato na página de detalhes.
- Adicione uma opção para apagar a foto do contato.

No projeto do estacionamento:

- Adicione uma opção para upload da foto do carro no momento em que entrou e no momento em que saiu do estacionamento. Isso é uma garantia de que os clientes pediram, pois alguns disseram que seus veículos foram amassados dentro do estacionamento.
- Adicione uma opção para apagar as fotos.
- Pesquise na internet por outras formas de validar os tipos de arquivos e tente aplicá-las. Um exemplo seria usando o MIME Type do arquivo.

CAPÍTULO 12

LEMBRETES DE TAREFAS POR E-MAIL

É bem comum que diversos tipos de aplicações web enviem e-mails para os usuários com algum tipo de informação e também lembretes. Por isso, vamos adicionar uma opção em nossa lista de tarefas para enviar e-mails com lembretes das tarefas.

Este não será o tipo de lembrete que é enviado automaticamente no dia definido como prazo para a tarefa. O que faremos é a adição de um checkbox **Enviar lembrete** nos formulários de criação e edição de tarefas.

Quando esta opção estiver marcada, vamos enviar o e-mail para um endereço de e-mail que deixaremos cadastrado no código. E quando existente, também vamos anexar os arquivos `.pdf` e `.zip` das tarefas.

O PHP possui uma função para envio de e-mails chamada `mail()`. Esta função padrão oferece apenas alguns recursos simples e, no geral, é necessário recorrer a opções mais complexas para conseguir fazer autenticação em um servidor de e-mails ou anexar arquivos. Por isso, usaremos uma biblioteca adicional que facilitará bastante o nosso trabalho.

12.1 DEFININDO O E-MAIL DE AVISO

O e-mail de aviso será um HTML simples com os dados da tarefa, pois praticamente todos os clientes de e-mail conseguem exibir HTML. A ideia é mandar por e-mail uma página bem parecida com a página de detalhes e, quando necessário, adicionar também os anexos da tarefa ao e-mail. E é aqui que a biblioteca de envio de e-mails será especialmente bastante útil!

O corpo do nosso e-mail ficará em um novo arquivo, que vamos chamar de `template_email.php`. Este deverá ficar junto com os demais, dentro do diretório `tarefas`, e seu conteúdo será apenas este:

```
<h1>Tarefa: <?php echo $tarefa['nome']; ?></h1>

<p>
    <strong>Concluída:</strong>
    <?php echo traduz_concluida($tarefa['concluida']); ?>
</p>
<p>
    <strong>Descrição:</strong>
    <?php echo nl2br($tarefa['descricao']); ?>
</p>
<p>
    <strong>Prazo:</strong>
    <?php echo traduz_data_exibir($tarefa['prazo']); ?>
</p>
<p>
    <strong>Prioridade:</strong>
    <?php echo traduz_prioridade($tarefa['prioridade']); ?>
</p>

<?php if (count($anexos) > 0) : ?>
    <p><strong>Atenção!</strong> Esta tarefa contém anexos!</p>
<?php endif; ?>

<p>
    Tenha um bom dia!

```

</p>

Este trecho será usado como corpo do e-mail e seu conteúdo final será apenas HTML. O pouco de PHP usado no template é apenas para preencher os dados e indicar se a tarefa tem anexos ou não.

Repare que este template para o e-mail é bem parecido com a página de detalhes da tarefa, aquela que tem o formulário para adicionar anexos. A diferença é que aqui não precisamos exibir o link para voltar para a lista de tarefas e também não precisamos exibir os anexos, apenas uma informação de que existem anexos para a tarefa.

Agora vamos deixar esse arquivo guardado um pouco e vamos ajustar outros pontos da aplicação. Certo, eu sei que é chato escrever código e não ver o funcionamento, mas achei melhor já deixarmos este arquivo pronto agora, para não atrapalhar o nosso raciocínio mais à frente.

12.2 UNIFICANDO A CONFIGURAÇÃO DA APLICAÇÃO COM CONSTANTES

Até agora, nossa aplicação não tem muitos itens configuráveis. Na verdade, temos apenas os dados de conexão com o banco de dados. Mas, a partir de agora, teremos uma configuração para definir o endereço de e-mail para onde os e-mails de aviso serão enviados.

Tudo bem, ainda são poucas informações. Mas, acredite, é melhor começar a organização ainda pequeno, pois facilita o crescimento.

Muito bem, vamos criar então um novo arquivo para manter as configurações da nossa aplicação. Ele será o `config.php` e deverá ser colocado com os demais arquivos na pasta `tarefas`. Inicialmente, ele deverá conter os dados de conexão ao banco e também o e-mail de aviso das tarefas. Veja como deverá ficar seu conteúdo:

```
<?php

// Conexão ao banco de dados (MySQL)
define("BD_SERVIDOR", "127.0.0.1");
define("BD_USUARIO", "sistematarefa");
define("BD_SENHA", "sistema");
define("BD_BANCO", "tarefas");

// E-mail para notificação
define("EMAIL_NOTIFICACAO", "meuemail@meudominio.com.br");
```

Neste arquivo, estamos usando a função `define()` que serve para definir **constants** no PHP. A função `define()` recebe o nome da constante e o seu valor. Uma constante é um identificador para um valor que não será alterado durante a execução da aplicação.

Fica fácil entender as constantes quando já conhecemos bem as variáveis, que são identificadores para valores que podem mudar, por isso têm o nome de **variáveis**. Para usar o valor das constantes, utilizamos apenas seus nomes, sem o cifrão na frente, como as variáveis. Veja um exemplo:

```
<?php
$linguagem = "PHP";
define("BANCO", "MySQL");

echo "Eu uso {$linguagem} com o banco " . BANCO;
// Exibe: Eu uso PHP com o banco MySQL
```

Repare que estou deixando as constantes sempre em caixa alta, ou seja, todas as letras em maiúsculo. Isso não é obrigatório, mas é um padrão que o pessoal do PHP segue, pois desta maneira fica fácil de identificar onde as constantes estão em um código, já que não se usa o cifrão.

As constantes em PHP podem armazenar números e strings e, a partir da versão 7, também podem armazenar arrays. Apenas lembre-se de que depois de definido o valor de uma constante, ele não muda mais.

No nosso arquivo `banco.php`, definimos quatro variáveis para guardar os dados de acesso ao banco de dados. Mas esses valores não deverão mudar durante o uso da aplicação, por isso eles poderiam ficar em constantes. Agora, por exemplo, a variável `$tarefa` recebe novos dados e que podem ser alterados, por isso não pode ser uma constante.

No geral, as constantes são utilizadas para configurações, pois estes são valores difíceis de mudar, a não ser quando trocamos, por exemplo, o endereço ou o usuário do banco de dados. Mas isso não é algo que acontece o tempo todo.

Vamos então alterar o arquivo `banco.php` para usar nossas novas constantes para a comunicação com o banco. Apague a definição das variáveis com os dados de acesso ao banco, e mude a chamada da função `mysqli_connect()` para usar as novas constantes criadas no arquivo `config.php`:

```
<?php  
  
$conexao =  
mysqli_connect(BD_SERVIDOR, BD_USUARIO, BD_SENHA, BD_BANCO);
```

```
// ...
```

Veja como uma rápida olhada no código já nos revela que estamos usando constantes, graças ao uso da caixa alta. Se você tentar acessar sua aplicação agora, vai encontrar alguns erros dizendo que os identificadores usados não foram definidos, pois ainda não adicionamos o novo arquivo `config.php` na aplicação.

Nossa aplicação tem diversos pontos de entrada, que são os arquivos que estamos chamando diretamente pelo endereço no navegador. O arquivo `tarefas.php`, por exemplo, é um ponto de entrada, pois ele é chamado pelo navegador. Já o `template.php` é incluído por outros arquivos, então não é um ponto de entrada.

Precisamos alterar todos os pontos de entrada para incluir o arquivo de configuração. Então, adicione a linha a seguir no início dos arquivos `tarefas.php`, `tarefa.php`, `editar.php`, `remover.php` e `remover_anexo.php`:

```
<?php  
  
require "config.php"; // Esta é a nova linha  
require "banco.php";  
  
// ...
```

Lembre-se de que o arquivo `banco.php` depende das constantes definidas no arquivo `config.php`, então este deve ser incluído primeiro.

Antes de continuar, pense um pouco nesta última alteração. Nós tivemos de percorrer todos os pontos de entrada da aplicação para adicionar o nosso novo arquivo de configuração. Será que isso é bom? E se por um acaso nos esquecermos de alterar um ponto de entrada? A aplicação certamente vai quebrar em alguma página.

Você consegue pensar em alguma solução para esse problema? Nós veremos isso mais adiante no livro, mas por enquanto, apenas pense em possíveis formas de solucionar este problema.

Aliás, esse já foi um dos desafios dos capítulos anteriores. Se você já tentou fazer, então já deve entender melhor o problema que vários arquivos como pontos de entradas pode gerar.

12.3 ADICIONANDO A OPÇÃO DE AVISO POR E-MAIL

O usuário precisa informar que quer receber por e-mail um lembrete da tarefa. Por isso, precisamos alterar o formulário, adicionando um checkbox com a opção de envio do e-mail.

Vamos alterar o arquivo `formulario.php` para adicionar a nova opção logo abaixo da opção de tarefa concluída:

```
...
<label>
    Tarefa concluída:
    <input type="checkbox" name="concluida" value="1"
        <?php
            echo ($tarefa['concluida'] == 1) ? 'checked' : '';
        ?>
    />
</label>
<label>
    Lembrete por e-mail:
    <input type="checkbox" name="lembrete" value="1" />
</label>
...
...
```

Veja que é apenas um checkbox HTML com o nome `lembrete` e o valor `1`. Este campo será verificado durante a

gravação e durante a edição de uma tarefa. Caso o campo tenha sido checado, o e-mail com o aviso será enviado.

Vamos fazer a alteração primeiro na gravação de novas tarefas, que fica no arquivo `tarefas.php`. O envio do e-mail será feito usando a função `enviar_email()`, que ainda não existe. Esta receberá como parâmetros apenas os dados da tarefa e os dados dos anexos, sendo que nem sempre uma tarefa terá anexos. Logo, este parâmetro será opcional.

Opcional? Calma, veremos isso logo adiante neste capítulo.

A nossa primeira alteração será na gravação da tarefa. Então não precisaremos enviar os dados dos anexos, pois nossa aplicação só permite adicionar anexos na página de detalhes da tarefa.

No arquivo `tarefas.php`, logo após a gravação da tarefa usando a função `gravar_tarefa()`, adicione um `if` que verifica se a opção de lembrete foi checada, e então chama a função `enviar_email()`:

```
<?php
// ...

if (! $tem_erros) {
    gravar_tarefa($conexao, $tarefa);

    if (
        array_key_exists('lembrete', $_POST)
        && $_POST['lembrete'] == '1'
    ) {
        enviar_email($tarefa);
    }

    header('Location: tarefas.php');
    die();
}
```

```
// ...
```

Veja que a função `enviar_email()` está sendo chamada com apenas um parâmetro, que são os dados da tarefa.

Agora, altere também o arquivo `editar.php`. Essa alteração será bem parecida com a anterior, mas aqui precisamos mandar também os dados dos anexos. Então, fica assim:

```
<?php
// ...

if (! $tem_erro) {
    editar_tarefa($conexao, $tarefa);

    if (
        array_key_exists('lembrete', $_POST)
        && $_POST['lembrete'] == '1'
    ) {
        $anexos = buscar_anexos($conexao, $tarefa['id']);
        enviar_email($tarefa, $anexos);
    }

    header('Location: tarefas.php');
    die();
}

// ...
```

Como já temos a lógica para pegar os anexos dentro de uma função, basta chamá-la e pegar seu resultado para termos os anexos. Agora que já preparamos a aplicação para enviar os e-mails, vamos finalmente para a função `enviar_email()`!

12.4 A FUNÇÃO ENVIAR_EMAIL()

Antes de escrever a função para enviar e-mails, vamos pensar no que é necessário fazer para enviar um e-mail usando uma

aplicação como o Gmail ou outros webmails.

Vou colocar aqui o passo a passo do que é necessário para enviar um e-mail:

1. Acessar a aplicação de e-mails;
2. Fazer a autenticação com usuário e senha;
3. Usar a opção para escrever um e-mail;
4. Digitar o e-mail do destinatário;
5. Digitar o assunto do e-mail;
6. Digitar o corpo do e-mail;
7. Adicionar os anexos, quando necessário;
8. Usar a opção de enviar o e-mail.

Nossa! Uma tarefa tão simples como enviar um e-mail virou uma atividade dividida em oito passos! E eu nem adicionei "tomar um gole de café" no passo a passo. ;)

Bem, se este é o passo a passo para enviar um e-mail manualmente, o processo para enviar um email usando PHP, ou outras linguagens, deve ser algo bem parecido, pois programação é basicamente a automação de processos que antes eram manuais.

Sendo assim, criaremos a função para enviar os e-mails usando um monte de comentários, que vão nos guiar para a implementação do código. No arquivo `ajudantes.php`, adicione a função `enviar_email()`:

```
<?php  
// ...  
  
function enviar_email($tarefa, $anexos)  
{  
    // Acessar a aplicação de e-mails;  
    // Fazer a autenticação com usuário e senha;
```

```
// Usar a opção para escrever um e-mail;  
// Digitar o e-mail do destinatário;  
// Digitar o assunto do e-mail;  
// Escrever o corpo do e-mail;  
// Adicionar os anexos, quando necessário;  
// Usar a opção de enviar o e-mail.  
}
```

Agora já temos uma espécie de mapa que nos guiará. Mas, antes de seguir o mapa e fazer a função realmente mandar e-mails, repare nos parâmetros que ela pode receber. Veja que temos uma variável `$tarefa` e uma variável `$anexos`.

O problema que começa a nos perturbar é saber que, após a gravação da tarefa, estamos chamando essa função com apenas um parâmetro, enquanto que na edição estamos passando os dois. Isso certamente vai gerar um erro, pois precisamos fornecer os parâmetros obrigatórios de uma função.

12.5 AS FUNÇÕES E SEUS PARÂMETROS OPCIONAIS

Em geral, as funções recebem parâmetros e fazem algum tipo de trabalho repetitivo com eles. Mas, algumas vezes, podemos precisar de funções que podem receber um certo parâmetro, ou podem ter algum tipo de valor padrão para um ou mais parâmetros.

Este é exatamente o nosso caso na função `enviar_email()`, pois o que realmente será necessário sempre são os dados das tarefas, afinal, não podemos enviar um e-mail sem isso. Já os anexos serão opcionais, pois podemos muito bem ter tarefas sem anexos.

Para fazer com que um parâmetro seja opcional em uma função PHP, basta atribuir um valor logo após a declaração do parâmetro na função:

```
<?php  
  
function pular_linha($c = '<br />')  
{  
    echo $c;  
}
```

A função `pular_linha()` por padrão pula uma linha no HTML usando a tag `
`. Mas poderíamos trocar o comportamento padrão chamando a função e falando para pular linha em um arquivo de texto usando o caractere `\n` : `pular_linha("\n");` .

Como a lista de anexos é um array, podemos alterar a declaração da função `enviar_email()` para ter uma lista de anexos vazia por padrão, atribuindo um array vazio à variável `$anexos` :

```
<?php  
// ...  
  
function enviar_email($tarefa, $anexos = [])  
{  
    // ...  
}  
  
// ...
```

Agora vamos voltar à lista de passos para enviar o e-mail. Praticamente todos os passos da lista dependem de algum tipo de comunicação com o servidor de e-mails, exceto o item **escrever o corpo do e-mail**. Como já temos o arquivo `template_email.php` com o template, vamos começar por este item.

12.6 ESCREVENDO O CORPO DO E-MAIL USANDO UM ARQUIVO COM O TEMPLATE

Para colocar o texto do corpo do e-mail efetivamente dentro dele, precisamos usar alguma forma de ler o arquivo `template_email.php`, colocando o seu conteúdo já processado (com os valores das variáveis etc.) dentro de uma variável em vez de exibir o template, como estamos fazendo até agora.

Tente imaginar algo simples como um `include` sendo atribuído a uma variável, assim:

```
<?php  
// ...  
  
function enviar_email($tarefa, $anexos = [])  
{  
    $corpo = include 'template_email.php';  
}
```

O arquivo `template_email.php` precisa das variáveis `$tarefa` e `anexos`, que já estão sendo recebidas pela função. A ideia aqui é que a variável `$corpo` contenha o HTML resultante do processamento do arquivo `template_email.php`.

Infelizmente, esse código não vai funcionar, pois o PHP lerá o arquivo e o resultado do processamento do arquivo `template_email.php` será enviado para o navegador. Pois é, uma pena que não é assim tão simples ler e processar o conteúdo de um arquivo e colocar o resultado em uma variável.

Para resolver este problema, vamos separar essa lógica de ler o arquivo `template_email.php` em uma nova função, pois assim vai ficar mais simples ler e entender a função `enviar_email()`:

```
<?php
// ...

function enviar_email($tarefa, $anexos = [])
{
    $corpo = preparar_corpo_email($tarefa, $anexos);

    // ...
}

function preparar_corpo_email($tarefa, $anexos)
{
    // Aqui vamos pegar o conteúdo de template_email.php
}
```

Agora que separamos a lógica para preparar o corpo do e-mail, vamos pensar em como isso deve ser feito. O nosso maior problema é que, se incluirmos um arquivo com HTML usando o **include**, seu conteúdo será enviado para o navegador. Então, precisamos fazer um passo a passo dessa forma:

1. Falar para o PHP que não é para enviar o processamento para o navegador;
2. Incluir o arquivo `template_email.php`;
3. Guardar o conteúdo gerado pelo processamento do arquivo em uma variável;
4. Falar para o PHP que ele pode voltar a mandar conteúdos para o navegador.

Estamos cheios de listas descrevendo os procedimentos neste capítulo, não é? Mas é por uma boa razão, já que isso ajuda a pensar primeiro na lógica e depois na implementação usando PHP.

Uma dica que pode servir para diversas situações na carreira de um bom desenvolvedor é: "primeiro resolva o problema, depois escreva o código". Até que faz bastante sentido. ;)

Para escrever a função `preparar_corpo_email()`, vamos usar algumas funções do PHP que fazem exatamente o que foi descrito na lista anterior. Essas funções impedem o PHP de enviar conteúdos para o navegador, e nos permitem pegar esses conteúdos e guardar em variáveis. Veja como fica simples a implementação:

```
<?php
// ...

function preparar_corpo_email($tarefa, $anexos)
{
    // Aqui vamos pegar o conteúdo processado
    // do arquivo template_email.php

    // Falar para o PHP que não é para enviar
    // o resultado do processamento para o navegador:
    ob_start();

    // Incluir o arquivo template_email.php:
    include "template_email.php";

    // Guardar o conteúdo do arquivo em uma variável;
    $corpo = ob_get_contents();

    // Falar para o PHP que ele pode voltar
    // a mandar conteúdos para o navegador.
    ob_end_clean();

    return $corpo;
}
```

Eu dei os comentários de propósito, pois eles são a mesma lista de passos que fizemos para montar a lógica. Depois você pode apagá-los.

Na função `preparar_corpo_email()`, usamos a função `ob_start()`, que coloca uma espécie de semáforo com a luz vermelha acesa no fluxo que o PHP envia para o navegador. Isso

faz com que o PHP pare de enviar dados para o navegador.

Depois fazemos a inclusão do arquivo `template_email.php`, aquele que fizemos lá no começo deste capítulo, usando a instrução `include`. Isso vai fazer com que os dados fiquem parados no sinal vermelho.

Aí vem uma parte bem mágica em que usamos a função `ob_get_contents()`, que faz algo como copiar todo o conteúdo que estava esperando a luz verde do semáforo para continuar em seu caminho para o navegador e retorna esse valor para a variável `$corpo`.

Por último, usamos a função `ob_end_clean()` para finalizar o processo, que é tipo limpar o conteúdo que estava aguardando a luz verde do semáforo para finalmente seguir em frente.

Você pode se perguntar: "mas já não pegamos o conteúdo que estava aguardando para ser enviado para o navegador usando a função `ob_get_contents()` ?". A resposta é sim, pegamos, mas pegamos apenas uma cópia, por isso é necessário limpar o conteúdo antes de acender a luz verde.

Alias, essa área na qual os conteúdos ficam aguardando para serem enviados para o navegador se chama **buffer**. Uma descrição mais técnica dos passos que realizamos seria: ligar o buffer de saída; incluir o arquivo; pegar o conteúdo do buffer e colocar em uma variável; limpar o conteúdo do buffer; e desligar o buffer.

Complicado? Sim, um pouquinho, mas é só treinar para dominar essa técnica. Se não fosse por ela, teríamos de fazer algo como uma variável de texto com todo o HTML necessário e as variáveis. Isso seria mais trabalhoso e deixaria tudo misturado com

HTML e PHP.

12.7 INSTALANDO UMA BIBLIOTECA PARA ENVIAR E-MAILS

Nossa função `enviar_email()` já tem o corpo do e-mail, mas ainda tem vários passos que ela não faz e que serão feitos com a ajuda de uma biblioteca para e-mails.

Existem várias bibliotecas para envio de e-mails usando PHP, inclusive algumas opções embutidas no PHP. A ideia de usar uma biblioteca adicional neste livro é mostrar como isso pode ser feito, além de suprir alguns pontos que seriam bem trabalhosos de fazer com as opções padrão, como anexar arquivos.

A biblioteca escolhida é a PHPMailer. Para baixá-la, acesse <https://github.com/PHPMailer/PHPMailer>, e clique no botão `Download ZIP`, conforme a figura a seguir:

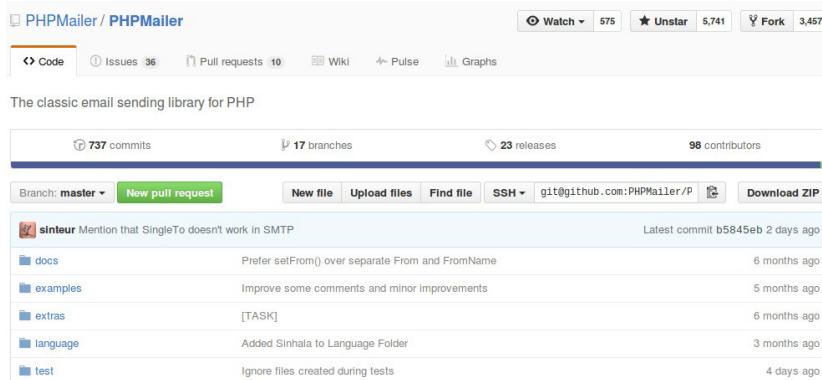


Figura 12.1: Baixando o PHPMailer no GitHub. Use o botão `Download ZIP`

Após baixar, crie um novo diretório, uma pasta chamada

bibliotecas , dentro da pasta tarefas , e descompacte o conteúdo do ZIP dentro dela. Após descompactar, você verá uma nova pasta chamada PHPMailer-master . Renomeie-a para apenas PHPMailer . Depois de renomear, você vai ficar com uma estrutura de pastas assim: tarefas > bibliotecas > PHPMailer .

Vale lembrar de que nosso projeto está dentro da pasta tarefas . Agora podemos voltar para a função enviar_email() .

12.8 FINALIZANDO A FUNÇÃO ENVIAR_EMAIL()

Para finalizar a função enviar_email() , precisamos usar a biblioteca PHPMailer. Veremos agora algumas linhas de código um pouco diferente do que já vimos até aqui, pois a biblioteca PHPMailer também faz uso da **Programação Orientada a Objetos** (POO), assim como a classe DateTime .

De forma simplificada, podemos pensar que um **objeto** é uma estrutura que pode ter diversas variáveis dentro dela, assim como os arrays podem ter diversos valores. Também pode ter funções dentro dela, que servem para manipular suas variáveis. Mas não se preocupe com isso agora, pois veremos mais sobre Orientação a Objetos logo mais.

Lembra da lista de passos necessários para se enviar um e-mail? Vamos agora preencher esta lista usando as opções da PHPMailer. Vou colocar o código passo a passo, depois juntamos tudo na função.

O primeiro passo é incluir a PHPMailer:

```
<?php  
// ...  
  
require "bibliotecas/PHPMailer/PHPMailerAutoload.php";
```

Os próximos passos são o acesso ao servidor de e-mails, a autenticação e a opção de escrever um novo e-mail:

```
<?php  
// ...  
  
// Acessar o servidor de e-mails;  
// Fazer a autenticação com usuário e senha;  
// Usar a opção para escrever um e-mail;  
  
$email = new PHPMailer(); // Esta é a criação do objeto  
  
$email->isSMTP();  
$email->Host = "smtp.gmail.com";  
$email->Port = 587;  
$email->SMTPSecure = 'tls';  
$email->SMTPAuth = true;  
$email->Username = "meuemail@gmail.com";  
$email->Password = "minhasenha";  
$email->setFrom("meuemail@gmail.com", "Avisador de Tarefas");
```

Neste último trecho, bastante coisa aconteceu! Mas vamos analisar passo a passo para ficar bem claro.

Primeiro, criamos o **objeto** \$email usando a classe PHPMailer . Não se preocupe com isso ainda, apenas entenda que, no objeto \$email , há variáveis e funções embutidas. Acessamos estas variáveis e funções usando a seta, que é o sinal de menos seguido por um sinal de maior: -> . Identificamos o que é variável e o que é função usando os parênteses nas funções.

Como quero enviar o e-mail usando o Gmail, precisei dizer

que o tipo de conexão é `SMTP` , o endereço do servidor é `smtp.gmail.com` , a porta para conexão é a `587` , e que ele tem de usar a criptografia `tls` .

Além disso, também falei que é necessário autenticar no SMTP usando o usuário (Username) e senha (Password) informados. Por último, coloquei o `From` do e-mail, para dizer quem é o seu remetente.

O próximo passo é adicionar o destinatário usando a nossa constante `EMAIL_NOTIFICACAO` :

```
<?php  
// ...  
  
// Digitar o e-mail do destinatário;  
$email->addAddress(EMAIL_NOTIFICACAO);
```

Agora tem o assunto do e-mail:

```
<?php  
// ...  
  
// Digitar o assunto do e-mail;  
$email->Subject = "Aviso de tarefa: {$tarefa['nome']}";
```

Tem também o corpo do e-mail, que vamos enviar usando HTML para deixar formatado:

```
<?php  
// ...  
  
// Escrever o corpo do e-mail;  
$corpo = preparar_corpo_email($tarefa, $anexos);  
$email->msgHTML($corpo);
```

Os anexos, quando existirem, são os próximos:

```
<?php  
// ...
```

```
// Adicionar os anexos quando necessário;
foreach ($anexos as $anexo) {
    $email->addAttachment("anexos/{$anexo['arquivo']}");
}
```

Veja que o `foreach` foi usado para os anexos. Caso o array `$anexos` esteja vazio, ele simplesmente não entra no laço.

Agora só falta a opção de enviar o e-mail! Veja como fica:

```
<?php
// ...

// Usar a opção de enviar o e-mail.
$email->send();
```

Ufa! Esse é todo o código necessário para enviar o e-mail! Se pensar bem, é tipo o passo a passo para enviar um e-mail, apenas codificamos em PHP.

Veja como fica a função `enviar_email()` completa e sem os comentários:

```
<?php

function enviar_email($tarefa, $anexos = [])
{
    include "bibliotecas/PHPMailer/PHPMailerAutoload.php";

    $corpo = preparar_corpo_email($tarefa, $anexos);

    $email = new PHPMailer();

    $email->isSMTP();
    $email->Host = "smtp.gmail.com";
    $email->Port = 587;
    $email->SMTPSecure = 'tls';
    $email->SMTPAuth = true;
    $email->Username = "meuemail@email.com";
    $email->Password = "minhasenha";
    $email->setFrom("meuemail@email.com", "Avisador de Tarefas");
```

```

$email->addAddress(EMAIL_NOTIFICACAO);
$email->Subject = "Aviso de tarefa: {$tarefa['nome']}";
$email->msgHTML($corpo);

foreach ($anexos as $anexo) {
    $email->addAttachment("anexos/{$anexo['arquivo']}");
}

$email->send();
}

```

A diferença é que eu coloquei o uso da função preparar_corpo_email() logo no começo.

Agora já podemos usar a opção de avisar por e-mail! Editei uma tarefa que tem anexos e marquei a opção para avisar:

Gerenciador de Tarefas

Nova tarefa

Tarefa:
Estudar HTML

Descrição (Opcional):
HTML é muito importante!

Prazo (Opcional):

Prioridade:
 Baixa Média Alta

Tarefa concluída:

Lembrete por e-mail:

Figura 12.2: Editando uma tarefa e marcando a opção de lembrete

E veja só como o e-mail chegou para mim:



Figura 12.3: E-mail de aviso no GMail, com anexo

O uso da PHPMailer facilita bastante o trabalho para envio de e-mails, pois reduz praticamente ao passo a passo seguido para enviar um usando um webmail ou um cliente de e-mail no desktop ou no mobile, sem maiores complicações.

NÃO USO O GMAIL

O pessoal do PHPMailer tem alguns exemplos de como fazer a parte da conexão usando outros servidores além do Gmail.

Para saber mais, acesse <https://github.com/PHPMailer/PHPMailer/tree/master/examples>. Está tudo em inglês nesta página, mas não é complicado de entender os exemplos, pois eles ficam bem no passo a passo que fizemos aqui.

DICAS IMPORTANTES!

Evite usar o seu e-mail pessoal para enviar e-mails em uma aplicação como a nossa, pois seu usuário e senha ficam expostos no código. Faça uma conta de e-mail para isso.

Ou se você estiver desenvolvendo algo na empresa onde trabalha, converse com a área responsável por administrar as redes e os servidores. Pergunte como deve ser a autenticação para mandar e-mails usando o servidor de e-mails da empresa (ou se realmente precisa autenticar etc.).

Mais uma dica se você for testar com a sua conta do Gmail: pode ser necessário autorizar a aplicação para mandar e-mails. Então, após fazer o teste, verifique se o Google enviou um e-mail para você para confirmar que você autoriza que a aplicação mande e-mails em seu nome.

12.9 GRAVANDO OS ERROS DO ENVIO DE E-MAILS

Uma dica importante para o final deste capítulo é fazer a gravação das mensagens em caso de erro ao enviar o e-mail. Podemos adicionar aqui o que é conhecido como "log" da aplicação, que é, pode-se dizer, uma espécie de diário do que um sistema faz.

Usar um log é sempre uma boa opção para entender como uma aplicação funciona, quando estamos entrando em um time, por

exemplo. Também ajuda a entender o motivo de certos problemas acontecerem.

Vamos tentar um exemplo simples: mude a senha que você usou para fazer o envio de e-mails para uma senha incorreta e tente fazer o envio do e-mail. Não vai funcionar, claro, mas não teremos uma mensagem para saber o que aconteceu. A solução seria feita através de algumas tentativas e erros até entender o que aconteceu de errado.

Para entender o que aconteceu de errado, a PHPMailer oferece uma mensagem de erro. Usaremos esta mensagem para fazer o tal do diário da aplicação.

Comece alterando o envio do e-mail para verificar se o retorno da função `send()` é falso, o que significa que algum erro aconteceu:

```
<?php

function enviar_email($tarefa, $anexos = [])
{
    include "bibliotecas/PHPMailer/PHPMailerAutoload.php";

    // Todo o código continua aqui...

    if (! $email->send()) {
        // salvar o erro em um arquivo de log
    }
}
```

Agora vamos criar uma nova função que salva textos em um arquivo. Ela deverá ser adicionada no arquivo `ajudantes.php`, será chamada de `gravar_log()`, e terá apenas um parâmetro: a mensagem a ser gravada.

```
<?php
```

```
function gravar_log($mensagem)
{
    $datahora = date("Y-m-d H:i:s");
    $mensagem = "{$datahora} {$mensagem}\n";

    file_put_contents("mensagens.log", $mensagem, FILE_APPEND);
}
```

Olha mais algumas novidades aí! A função `date()` formata uma data, de forma parecida com a função `format()` da classe `DateTime`. Estamos pegando a hora atual e formatando para "Ano-Mês-Dia Hora:Minuto:Segundo", pois assim saberemos exatamente quando o problema aconteceu.

A outra novidade é a função `file_put_contents()`, que salva conteúdos em um arquivo. No caso, estamos salvando o conteúdo da variável `$mensagem` no arquivo `mensagens.log`. Estamos também dizendo para a função `file_put_contents()` usar a opção `FILE_APPEND`, que significa que as mensagens devem ser sempre adicionadas no final do arquivo em vez de sobrescrever o conteúdo do arquivo com uma nova mensagem.

Ah, um outro detalhe é que a função `gravar_log()` também adiciona uma quebra de linha na mensagem a ser gravada. Assim, cada mensagem fica em uma linha.

Depois de criar a função `gravar_log()`, adicione uma chamada a ela, passando a mensagem de erro da PHPMailer em caso de problemas ao enviar o e-mail, que está na variável `ErrorInfo` do objeto `$email`:

```
<?php

function enviar_email($tarefa, $anexos = [])
{
```

```
include "bibliotecas/PHPMailer/PHPMailerAutoload.php";  
  
// Todo o código continua aqui...  
  
if (! $email->send()) {  
    gravar_log($email->ErrorInfo);  
}  
}
```

Agora tente mandar um e-mail novamente, por exemplo, editando uma tarefa. Veja se o arquivo `mensagens.log` foi criado e contém uma linha como esta: `2016-03-20 13:32:54 SMTP connect() failed..`.

Pronto, agora sabemos que algum problema aconteceu com a conexão, o que pode significar um usuário ou senha incorretos.

Esta forma de salvar os logs é bastante simples, e você pode adicionar chamadas à função `gravar_log()` em várias partes da aplicação. Você pode fazer isso em caso de erros na conexão ao banco de dados, ou mesmo no upload de arquivos, para saber se seus usuários estão tentando enviar arquivos não suportados pelo sistema (`pdf` e `zip`).

12.10 RESUMO

Este capítulo começou com uma ideia simples: mandar e-mails com lembretes das tarefas. Vimos como instalar e usar uma biblioteca com Orientação a Objetos para envio de e-mails, trabalhamos com constantes e com a função `define()` para centralizar a configuração da aplicação.

Além disso, usamos os controles de saída do PHP com as funções `ob_start()`, `ob_get_contents()` e `ob_end_clean()`

para conseguir fazer a inclusão de um template com o resultado em uma variável.

Vimos também como fazer a gravação dos logs da aplicação, que são uma espécie de diário da aplicação e que podem nos ajudar a entender os motivos de certos problemas acontecerem. Foi um capítulo bastante produtivo!

12.11 DESAFIOS

É claro que este capítulo também tem alguns desafios.

1. Ter quatro pontos de entrada dificultou um pouco a adição do arquivo de configuração. Faça uma cópia do projeto até aqui e pense em uma forma de manter apenas um ponto de entrada que consiga tratar todos os casos.

Vamos ver como fazer isso nos últimos capítulos do livro, mas já fica aqui uma experiência que você pode fazer. Se você já fez esta experiência nos capítulos anteriores, tente atualizar o código das experiências para enviar os emails.

2. O arquivo de configuração tem apenas o e-mail de destino. Adicione constantes para configurar o e-mail do remetente, o usuário e a senha para autenticar, e o nome que aparece no "from".
3. Na função `preparar_corpo_email()`, temos de pegar o conteúdo do buffer e depois limpar o buffer antes de desligá-lo. Tente usar a função `ob_get_clean()` para pegar o conteúdo e limpar o buffer ao mesmo tempo.
4. Na lista de contatos, adicione a opção de mandar os dados de

um contato para um endereço de e-mail.

5. Ainda na lista de contatos, adicione a foto do contato como anexo do e-mail.
6. Mais um para a lista de contatos: centralize as configurações usando constantes em um arquivo separado.
7. No sistema do estacionamento, adicione a opção de mandar os dados de uma estadia para um e-mail, mandando também as fotos do veículo.

Em todos os desafios, procure usar a técnica de ler o template do e-mail usando as funções `ob_start()` etc. Adicione também a gravação dos logs para ficar mais fácil entender o que pode gerar problemas.

CAPÍTULO 13

HOSPEDAGEM DE APLICAÇÕES PHP

Desenvolver aplicações web pode ser bastante divertido e útil, pois podemos desenvolver ferramentas que solucionam problemas que temos em nosso dia a dia. Como foi o caso do nosso sistema de tarefas desenvolvido durante os capítulos deste livro.

O problema é que, até este momento, ficamos presos aos nossos computadores, pois nossas aplicações funcionam apenas neles. Quando acessamos um site, um blog ou outra aplicação web, estamos acessando algo que foi desenvolvido provavelmente de uma maneira semelhante à nossa aplicação, mas que está hospedado em um servidor, que também é um computador e que está sempre conectado à internet.

Quando colocamos nossa aplicação em um servidor na internet, dizemos que a aplicação está **hospedada**. Quando nossa aplicação está em hospedada em um servidor na internet, ela pode ser acessada a partir de qualquer computador também conectado à internet, ou até mesmo de tablets, smartphones ou outros aparelhos conectados e que tenham um browser web.

13.1 SUA APLICAÇÃO PARA O MUNDO!

Bem, na verdade, ainda não é para o mundo, é apenas para você. Mas você poderá acessar estando em qualquer lugar, bastando ter uma conexão à internet.

Digo que é apenas para você, pois nossa lista de tarefas não tem separação por usuários, em que cada um pode ter sua própria lista de tarefas. Então serve para apenas um usuário de cada vez.

Para colocarmos nossa aplicação online, precisamos encontrar um servidor para hospedagem que suporte nossas necessidades, ou seja, a linguagem PHP e o banco de dados MySQL ou MariaDB. Depois teremos de configurar como será o funcionamento da aplicação no servidor. Então poderemos enviar nossos arquivos para lá, para finalmente executar a aplicação no servidor.

13.2 ESCOLHENDO UM SERVIDOR PARA HOSPEDAGEM

Para publicar uma aplicação na web, precisamos usar um servidor que tenha suporte às tecnologias que estamos usando. No nosso caso, é o PHP e o banco MySQL ou ainda o MariaDB.

Existem diversas opções para hospedar aplicações PHP e MySQL, desde empresas que oferecem o serviço gratuitamente (mas podem adicionar algum tipo de propaganda no site), até opções para quem quer gerenciar o servidor, instalar as ferramentas necessárias e manter tudo funcionando por conta própria.

No nosso caso, precisamos dos tipos mais básicos, pois estamos usando apenas para aprendizado e não teremos um grande volume de acesso. Além disso, a ideia aqui não é ensinar a configurar

servidores, precisamos apenas hospedar uma aplicação.

Claro que também é interessante conhecermos uma opção mais avançada. Por isso, demonstrarei dois serviços de hospedagem: um mais simples e grátis; e outro com mais recursos que oferece um período grátis para experimentar.

Os serviços de hospedagem demonstrados serão a hospedagem da Hostinger e o Jelastic da Locaweb:

- Hostinger — <http://www.hostinger.com.br/>
- Jelastic da Locaweb —
<http://www.locaweb.com.br/cloud/jelastic/>

13.3 HOSPEDAGEM COM A HOSTINGER

A hospedagem com a Hostinger segue um padrão de acesso via **FTP**, que é um protocolo para transferência de arquivos. Também faz uso de um painel de configuração bastante usado em diversos servidores, o **CPanel**.

O acesso via FTP é bastante usado, pois ele cria uma conexão que exibe os arquivos remotos e seus arquivos locais de uma maneira que fica simples para copiar de um para o outro. FTP significa *File Transfer Protocol*, ou Protocolo para Transmissão de Arquivos.

Existem diversos clientes de FTP que tornam o trabalho de envio de arquivos ainda mais simples, pois eles exibem o conteúdo do seu computador de um lado e o conteúdo do servidor do outro lado. Eles também permitem fazer a transmissão dos arquivos apenas arrastando de um lado para o outro.

O CPanel é uma ferramenta para administração de servidores via navegador. Esta ferramenta possui diversas opções de configuração e é bastante simples de utilizar, por isso é uma opção com um número bem grande de usuários.

Criação da conta na Hostinger

Para criar a conta para hospedagem de uma aplicação na Hostinger, acesse <http://www.hostinger.com.br/>. Você verá uma tela parecida com esta:



Figura 13.1: Site da Hostinger

Nesta página, clique no botão **Peça agora!**. Um formulário de cadastro será apresentado. Preencha os dados e clique no botão **Criar conta**:



Figura 13.2: Cadastro na Hostinger

A próxima tela, após o cadastro, pede para que você verifique o seu e-mail e clique no link de ativação:



Figura 13.3: Pedido de confirmação do cadastro por e-mail

Após clicar no link que chegou por e-mail e confirmar o cadastro, utilize a opção **Hospedagem > Nova conta** no menu superior. Você verá uma página para selecionar um plano de hospedagem. A opção **Gratuito** vai servir bem para testes e até para pequenos sites:



Figura 13.4: Escolha a opção Gratuito para os testes

Na próxima página, você deve criar uma nova conta. Escolhi um subdomínio gratuito. A senha escolhida aqui será usada na hora de enviar os arquivos da nossa aplicação para o servidor usando FTP. Explico um pouco mais sobre FTP logo adiante.

Aconselho a opção grátis para os testes:

Requisitar Nova Hospedagem "Gratuito" - Passo 2 de 3

1 Escolher Plano 2 Instalação de Hospedagem 3 Sumário de Pedidos

Digite domínio e senha

Escolher um Tipo de Domínio: Subdomínio

Subdomínio Gratuito tarefasphp .esy.es ▾

Escolha a Região do Servidor* Europa (Reino Unido) América do Norte (USA) Ásia (Singapura) América do Sul (Brasil) Russian Federation (RUS)

Senha*

Confirme Senha* Digite a senha novamente.

Figura 13.5: Escolha um subdomínio ou um domínio

Na próxima página, será necessário escrever com poucas palavras uma descrição da motivação por trás dos testes:

Confirme sua requisição

Plano:	Gratuito
Domínio Próprio:	tarefasphp2.esy.es
Preço:	R\$0.00
Descrição:*	<p>IMPORTANTE! Nós estamos recebendo mais de 20.000 novas ativações por dia e todas as contas são revisadas manualmente por nossa equipe. Isso ocorre devido a nossa política anti fraude. Por favor, preencha todas as informações corretamente.</p> <p>estou testando a hospedagem de sites usando php</p>
	<input checked="" type="checkbox"/> Não sou um robô  <small>reCAPTCHA Privacidade - Termos</small>
Voltar Comprar	

Figura 13.6: Descreva um motivo para o uso da hospedagem

Ele pode demorar um pouco para ativar a conta, mas dentro de alguns instantes você deve ver uma página como esta:

Contas de Hospedagem

Lista de Contas de Hospedagem					
10	Domínio Próprio	Plano	Expira Em	Estado	Ação
	tarefasphp2.esy.es	Gratuito	-	Ativo	<input checked="" type="checkbox"/> Upgrade

Figura 13.7: Conta criada na Hostinger

Após a ativação, clique no domínio para revelar as opções:

Contas de Hospedagem

The screenshot shows a web-based interface for managing domains. At the top, there is a navigation bar with 'Início > Hospedagem' and a search bar labeled 'Localizar..'. Below this is a section titled 'Lista de Contas de Hospedagem' with a dropdown menu set to '10'. A table lists one account: 'tarefasphp2.esy.es' (Domain Próprio), 'Gratuito' (Plan), 'Ativo' (Status), and an 'Upgrade' button. Below the table are four management icons: 'Gerenciar' (Manage) with a gear icon, 'Construtor de Website' (Website Builder) with a gear and website icon, 'Auto Instalador' (Automatic Installer) with a gear and database icon, and 'Contas de Email' (Email Accounts) with an envelope and gear icon. At the bottom right are navigation buttons for 'Anterior', 'Próximo', and page number '1'.

Figura 13.8: Opções de gestão do domínio

Finalmente teremos a conta criada e até o endereço já funcionando! Aqui, criei um subdomínio no endereço <http://tarefasphp2.esy.es/>. Ao acessá-lo, temos a página padrão da Hostinger:



Figura 13.9: Página padrão da Hostinger

Voltando na página das contas, clique no botão Gerenciar . Ele exibirá diversos blocos com áreas diferentes da administração. Navegue até o bloco intitulado Banco de Dados e clique na opção Bases de Dados MySQL :

■ Bancos De Dados

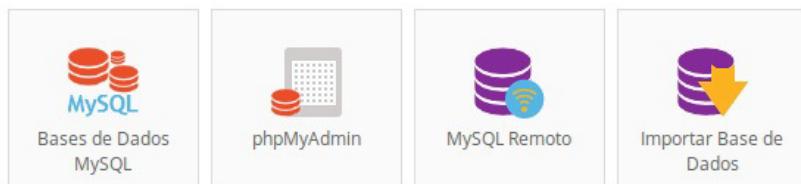


Figura 13.10: Use a opção Bases de Dados MySQL

Nesta página, será exibido um bloco chamado Criar Novo Banco de Dados e Usuário MySQL . Ele já coloca um prefixo para o nome do banco de dados e do usuário. Então, coloque o nome do banco e o usuário, depois a senha para acesso, e confirme usando o botão Criar :

A screenshot of a form titled 'Criar Novo Banco De Dados e Usuário MySQL'. It has four input fields: 'Nome do banco de dados MySQL' (with value 'u409809830_ taref'), 'Nome de Usuário MySQL' (with value 'u409809830_ taref'), 'Senha' (with value '.....'), and 'Senha novamente' (with value '.....'). Below the form is a blue button labeled 'Criar'.

Figura 13.11: Criando um novo banco e um usuário MySQL

A página será atualizada, e um bloco com os dados de acesso

será exibido:

The screenshot shows a table titled "Lista de Banco de Dados e Usuários MySQL Atuais". It has four columns: "Banco de Dados MySQL", "Usuário MySQL", "MySQL Host", and "Uso de Disco, MB". There is one row with the following data: "u409809830_taref", "u409809830_taref", "mysql.hostinger.com.br", and "0.02". A search bar labeled "Localizar..." is at the top right, and navigation buttons "← Anterior", "1", and "Próximo →" are at the bottom right.

Banco de Dados MySQL	Usuário MySQL	MySQL Host	Uso de Disco, MB
u409809830_taref	u409809830_taref	mysql.hostinger.com.br	0.02

Figura 13.12: Os dados de acesso ao MySQL

Guarde esses dados, pois a aplicação vai precisar deles. Agora devemos voltar ao painel Banco de Dados e usar a opção PHPMyAdmin. Será exibida a seguinte página:

The screenshot shows a table titled "Lista de Banco de Dados". It has three columns: "Banco de Dados MySQL", "Usuário MySQL", and "Ações". The "Ações" column contains a button labeled "Entrar phpMyAdmin". The data in the table is identical to Figure 13.12. Navigation buttons are at the bottom right.

Banco de Dados MySQL	Usuário MySQL	Ações
u409809830_taref	u409809830_taref	Entrar phpMyAdmin

Figura 13.13: Acessando o PHPMyAdmin

Clique em Entrar PHPMyAdmin e você verá o mesmo PHPMyAdmin que usamos em nosso ambiente local. Ele pode pedir o usuário e senha. Neste caso, coloque os dados que você acabou de criar, e então você terá acesso ao PHPMyAdmin. Agora, siga os passos para criação das tabelas dos capítulos *Acessando e usando um banco de dados* e *Upload de arquivos*.

Já temos o servidor pronto para receber a aplicação, então precisamos deixar a aplicação pronta para o servidor.

13.4 CONFIGURANDO A APLICAÇÃO PARA A HOSTINGER

Agora precisamos alterar a aplicação para que funcione no servidor. Na verdade, não é tanto trabalho, é apenas a configuração de acesso ao banco de dados. Altere o arquivo `config.php` e coloque os dados de acesso:

```
<?php  
  
define("BD_SERVIDOR", "mysql.hostinger.com.br");  
define("BD_USUARIO", "seu_usuario");  
define("BD_SENHA", "sua_senha");  
define("BD_BANCO", "seu_banco_tarefas");
```

Não esqueça de conferir os dados de acesso! Agora podemos, finalmente, enviar a aplicação para o servidor usando o FTP.

13.5 ENVIANDO A APLICAÇÃO PARA A HOSTINGER

Para enviar os arquivos usando FTP, você vai precisar de um software cliente FTP. FTP é um protocolo utilizado para transmissão de arquivos na internet. Esta é uma opção ainda bastante usada para se enviar sites e aplicações para hospedagens.

Existem diversas opções de clientes de FTP, mas recomendo o FileZilla, que funciona no Linux, no Windows e no Mac OS X. Ele pode ser baixado em <https://filezilla-project.org/>.

The screenshot shows the official website for FileZilla. On the left, there's a sidebar with links for Home, FileZilla (Features, Screenshots, Download, Documentation), FileZilla Server (Download), Community (Forum, Project page, Wiki), General (Contact, License, Privacy Policy), Development (Source code, Nightly builds, Translations, Version history, Changelog, Issue tracker). The main content area has a heading 'Overview' followed by text about the homepage, support through forums, wiki, and trackers, and documentation for compilation and nightly builds. It features two prominent download buttons: 'Download FileZilla Client' (All platforms) and 'Download FileZilla Server' (Windows only). Below these are sections for 'News' (with a link to '2013-08-07 - FileZilla Client 3.7.3 released') and 'Fixed vulnerabilities'.

Figura 13.14: Site do FileZilla

Use a opção **Download FileZilla Client** e, na página de download, escolha a opção para o seu sistema operacional. Siga os passos da instalação usando as opções padrão.

FILEZILLA NO LINUX

Procure no gerenciador de pacotes da sua distribuição pelo FileZilla. Usuários de Debian/Ubuntu podem instalar usando o apt: `sudo apt-get install filezilla`

Antes de abrir o FileZilla, precisamos dos dados de acesso FTP. Para consegui-los, vá ao painel da Hostinger e procure pelo bloco Arquivos e, nele, clique na opção Acesso FTP :

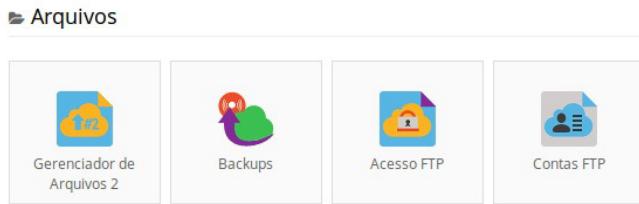


Figura 13.15: Use a opção Acesso FTP

Você verá a página com os dados de acesso FTP:

Acesso FTP	
Host FTP	ftp.tarefasphp2.esy.es
IP do FTP	31.220.16.94
Porta FTP	21
Nome de usuário FTP	u409809830
Senha FTP
Pasta para onde os arquivos serão enviados	public_html
Esqueceu sua senha de FTP?	Mudar senha de conta
Clientes FTP recomendados.	SmartFTP ou FileZilla

Figura 13.16: Dados de acesso via FTP

A senha é a usada na criação do domínio. Agora, abra o FileZilla e coloque os dados de acesso FTP exibidos pelo painel da Hostinger:

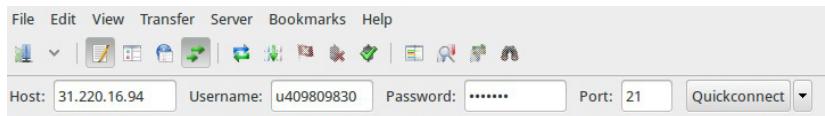


Figura 13.17: Preenchendo os dados de acesso FTP no FileZilla

Após preencher os dados, clique em **Conexão rápida**. O FileZilla tem dois painéis: um à esquerda com os seus arquivos locais (ou seja, no seu computador), e um à direita com os

arquivos no servidor.

No caso da Hostinger, será exibido um arquivo `default.php` e um arquivo `.htaccess`, que é um arquivo de configuração para o Apache e que não vou detalhar aqui, pois fica fora do escopo. Mas, é claro, aconselho que você pesquise um pouco sobre ele. Veja como ficou o meu FileZilla:

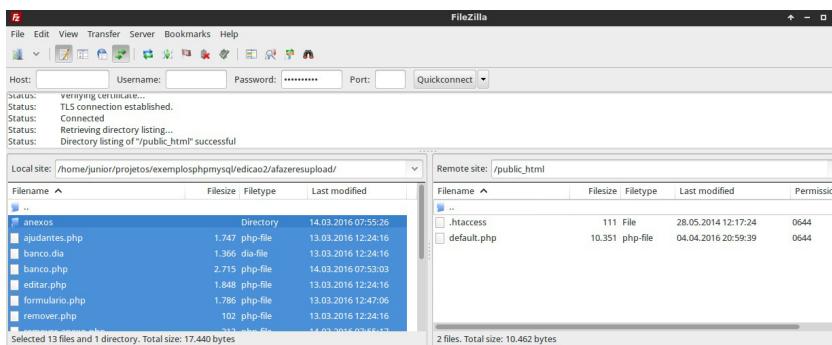


Figura 13.18: Acessando os arquivos usando FTP

Agora podemos navegar até a pasta onde estão nossos arquivos locais. Então, basta clicar e arrastá-los para o painel com os arquivos do servidor. O FileZilla vai iniciar a transmissão e, quando terminar, os arquivos serão exibidos no servidor também.

Após finalizar a transmissão dos arquivos, podemos acessar o sistema de tarefas já hospedado. O meu ficou hospedado em <http://tarefasphp2.esy.es/tarefas.php>.

Ah, mais uma dica importante! Você pode escolher a versão do PHP na opção Avançado > Versão do PHP. Então se você usou algum recurso presente apenas no PHP 7, faça a alteração conforme a figura a seguir:

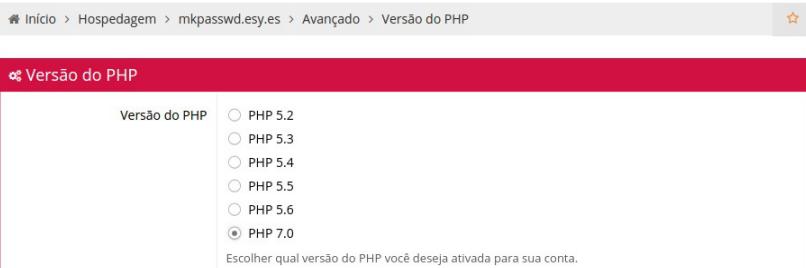


Figura 13.19: Escolhendo a versão do PHP

13.6 HOSPEDAGEM NO JELASTIC DA LOCAWEB

O Jelastic é um serviço bem interessante da Locaweb, pois nos permite configurar aplicações em caminhos diferentes, ter bancos de dados separados e pacotes com as diferentes versões da aplicação, hospedagem de outras tecnologias além do PHP e MySQL, configuração do PHP e do MySQL etc.

O Jelastic é um pouco mais complexo do que a Hostinger, mas é uma opção interessante para quem deseja ter mais controle sobre as opções do servidor.

Criação da conta no Jelastic da Locaweb

Para criar uma conta e hospedar uma aplicação no Jelastic, acesse <http://www.locaweb.com.br/cloud/jelastic/>. Você verá uma página como esta:



Plataforma on demand com escalabilidade automática

Seus projetos com mais controle de gastos no modelo pré-pago

Faça o deploy da sua aplicação em poucos minutos

• 14 DIAS
GRÁTIS

Digite seu e-mail

INICIE AGORA

• Como contratar

COMPRE AGORA
Fale com nossos especialistas

Figura 13.20: Site do Jelastic da Locaweb

Preencha o seu e-mail e clique em **inicie agora**. Após o processamento, você verá a mensagem de sucesso na criação da conta:

• 14 DIAS
GRÁTIS • Pronto! Verifique seu e-mail, acesse o link e faça seu deploy.

Figura 13.21: Sucesso na criação da conta

Verifique sua caixa de entrada, e você deverá ter recebido um e-mail com o endereço e os dados de acesso para configuração do ambiente de hospedagem:



Bem vindo ao Jelastic Cloud!

Olá,

Obrigado por seu interesse em conhecer o Jelastic Cloud Locaweb.

Sua conta *trial* já está disponível para você experimentar gratuitamente as vantagens desta plataforma escalável e automática. Acesse agora mesmo o [Painel de Controle](#) e crie seu ambiente!

Endereço: <http://app.jelasticlw.com.br/>

Login:

Senha:

Você pode acessá-lo diretamente através deste link: http://app.jelasticlw.com.br/SigninByAuthKey?key=_

Figura 13.22: E-mail de boas vindas do Jelastic

Clique no link do e-mail, ou acesse o endereço <https://app.jelasticlw.com.br/>, e informe o login e senha que recebeu no e-mail:

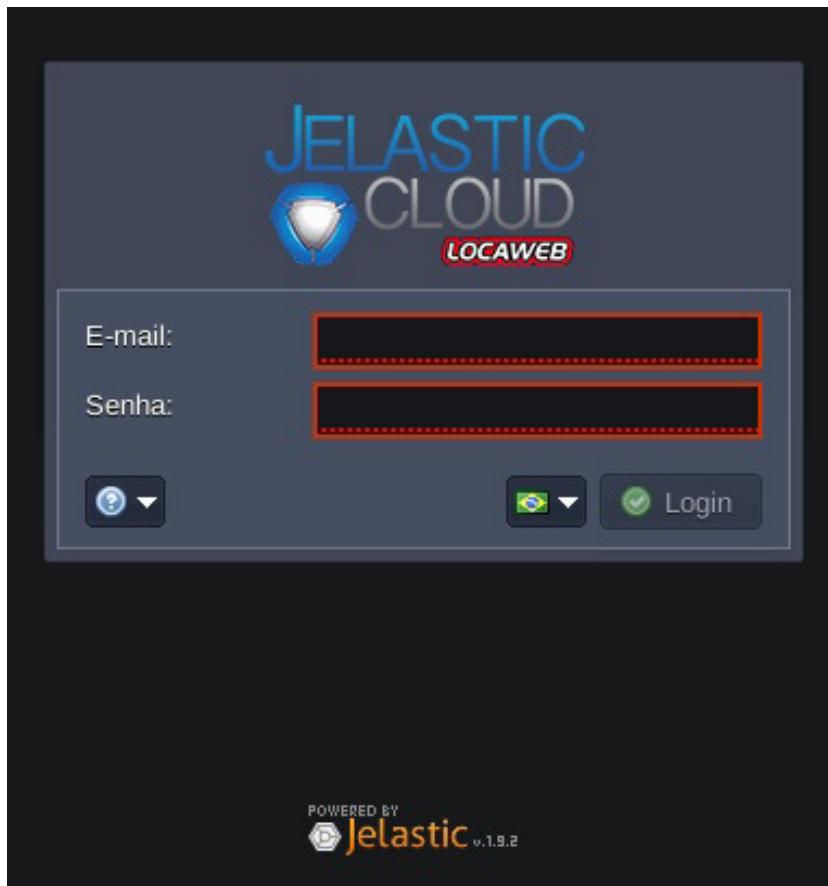


Figura 13.23: Tela de login do Jelastic

Nesta tela, você será recebido pelo assistente do Jelastic:



Figura 13.24: O assistente do Jelastic

Para a primeira utilização, recomendo seguir as instruções do assistente: basta clicar em **Vamos começar**, e então na opção **Criar ambiente** que fica em cima, à esquerda:

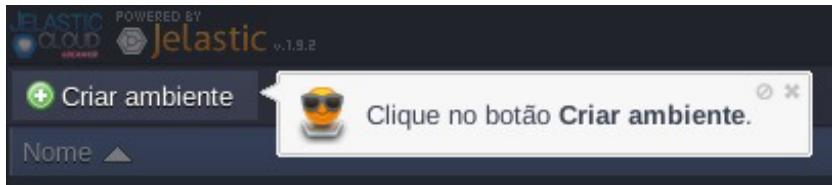


Figura 13.25: Opção para criar um novo ambiente

Nesta próxima tela, temos a opção de hospedar uma aplicação Java, PHP, Ruby, Node.js, Python ou ainda usando Docker (pesquise mais sobre estas tecnologias, pois tem muita coisa interessante em outras linguagens de programação):

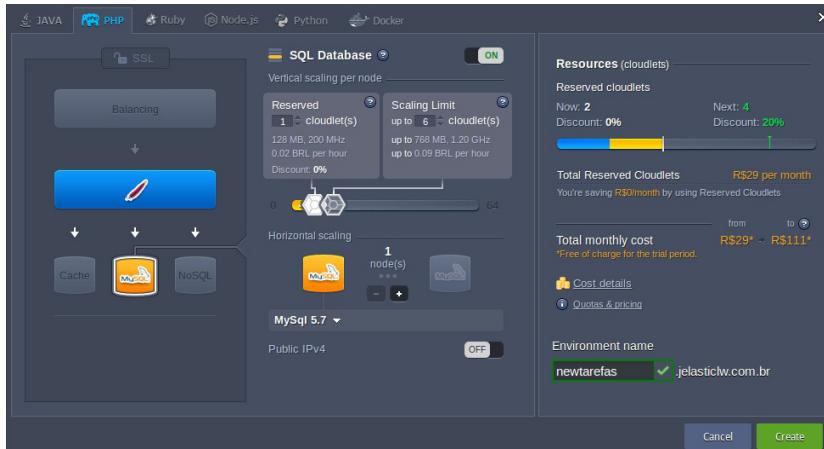


Figura 13.26: Seleção das tecnologias para rodar a aplicação

Selecione PHP na aba superior, Apache para o servidor HTTP e MySQL para o banco de dados. Aproveite escolher o endereço da aplicação e clique em Criar .

Agora, é só aguardar enquanto a aplicação é criada:

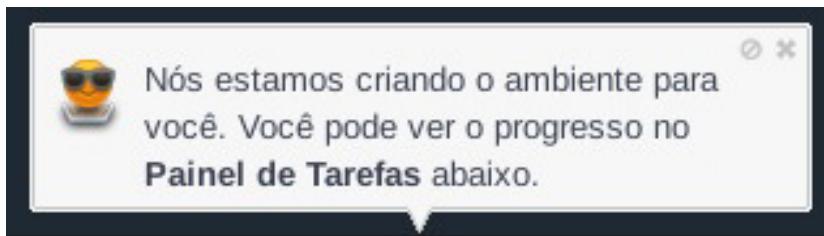


Figura 13.27: Aguarde a criação do ambiente

E assim que terminar, o assistente informa que o ambiente foi criado:

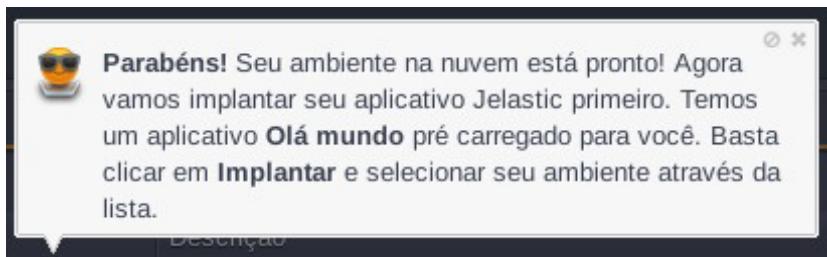


Figura 13.28: Sucesso na criação do ambiente para hospedagem

Verifique mais uma vez a sua caixa de entrada e você deverá ver um e-mail com os dados de acesso ao ambiente. Este meu e-mail veio em inglês, pois eu usei o ambiente do Jelastic em inglês. Mas se você deixar o seu ambiente em português, os e-mails serão enviados também em português.



Your Jelastic Environment has been successfully created

Hello,

Great news! Your [Jelastic Cloud](#) environment was just created - now you can start deploying your application(s) and enjoy the unique benefits of our scalable hosting platform.

Head over to the [dashboard](#) to deploy your application now!

Environment URL: <http://newtarefas.jelasticlw.com.br/>

Figura 13.29: Endereço de acesso à aplicação hospedada

Você também deverá ter recebido um e-mail com a indicação de que o ambiente do MySQL foi criado e seus dados de acesso:



MySQL node successfully added

Hello,

Just wanted to let you know that a new [Jelastic Cloud](#) MySQL server was added to your environment.

Figura 13.30: Dados de acesso ao MySQL

Agora, é necessário acessar o endereço do banco de dados. Lá tem um PHPMyAdmin. Informe seu usuário e sua senha, e use o PHPMyAdmin para criar um novo banco chamado `tarefas`.

Crie as tabelas criadas nos capítulos *Acessando e usando um banco de dados* e *Upload de arquivos* aqui do livro. Aqui não tem segredo, é só seguir os mesmos passos feitos no PHPMyAdmin do XAMPP, mas, desta vez, usando o PHPMyAdmin do Jelastic.

13.7 CONFIGURANDO A APLICAÇÃO PARA O JELASTIC

Após a criação das tabelas, edite o arquivo `config.php` da aplicação e coloque os dados de acesso ao MySQL do Jelastic:

```
<?php  
  
define("BD_SERVIDOR", "seu-mysql.jelasticlw.com.br");  
define("BD_USUARIO", "seu_usuario");  
define("BD_SENHA", "sua_senha");  
define("BD_BANCO", "tarefas");
```

Feito isso, crie um pacote zip da sua aplicação. Isso pode ser feito clicando com o botão direito na pasta tarefas e usando a opção de criar um arquivo zip . Este passo é importante, pois no Jelastic esta é a maneira padrão de enviar a aplicação.

13.8 ENVIANDO A APLICAÇÃO PARA O JELASTIC

Após fazer o pacote zip , volte ao painel do Jelastic e clique na opção **Upload** que está na aba **Gerenciador de Instalação**:

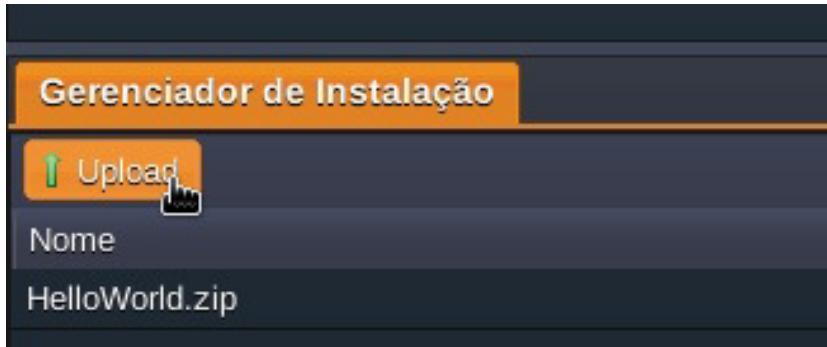


Figura 13.31: Opção de Upload da aplicação

Na próxima janela, selecione o pacote zip que você criou e dê uma descrição para esta versão do pacote:

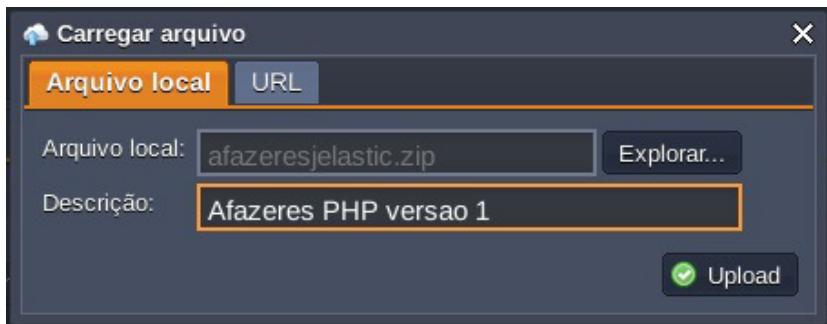


Figura 13.32: Selecione o zip e dê uma descrição para o pacote

Um bom nome ajuda a identificar o pacote quando você tiver mais de uma versão cadastrada. O próximo passo é usar a opção de enviar o pacote para produção usando o ícone que está na lista com os nomes dos pacotes:

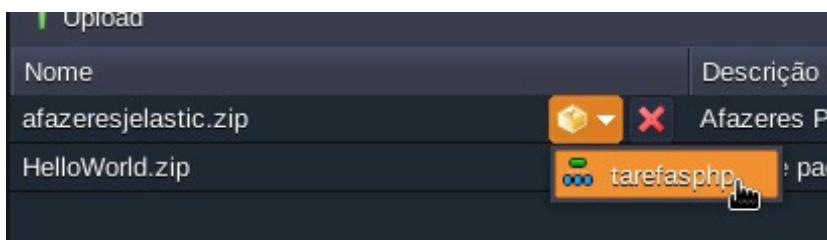


Figura 13.33: Selecione o ambiente para enviar a aplicação

Você será questionado sobre onde fazer a instalação do pacote. Deixe em branco para instalar na raiz do endereço (ou seja, em /), e clique em `instalar`:



Figura 13.34: Deixe em branco para instalar na raiz do endereço

Agora aguarde enquanto a aplicação é implantada.

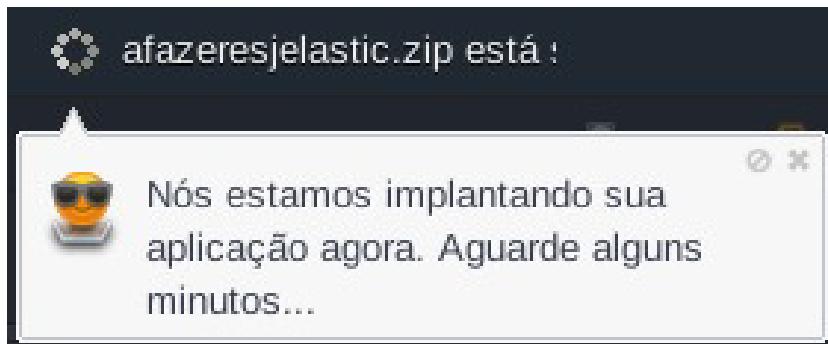


Figura 13.35: Aguarde a implantação

Após a conclusão, você pode clicar no ícone para ir para o endereço da aplicação:

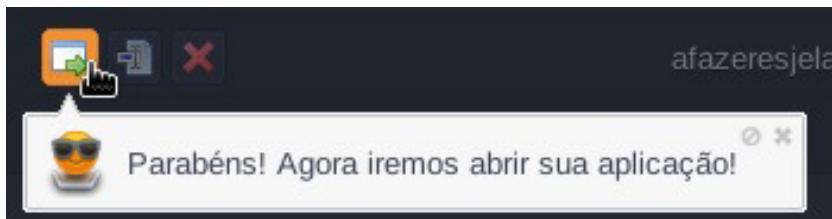


Figura 13.36: Aplicação instalada, basta acessar!

No meu caso a aplicação, ficou em <http://tarefasphp.jelasticlw.com.br/tarefas.php>. Ao acessar a sua, não se esqueça de colocar o nome do arquivo `tarefas.php` no endereço.

E é isso, sua aplicação estará instalada no Jelastic.

13.9 RESUMO

Neste capítulo foram apresentadas duas maneiras de hospedagem de aplicações PHP: uma usando o FTP e CPanel da Hostinger, que também são usados por diversos outros servidores; outra usando o Jelastic da Locaweb, que tem opções mais avançadas para configuração do servidor.

13.10 DESAFIOS

Achou que não teríamos desafios neste capítulo? Aqui estão eles!

1. Faça a hospedagem da aplicação dos contatos na Hostinger.
2. Faça a hospedagem da aplicação do estacionamento no Jelastic.

3. Procure mais uma opção de hospedagem PHP e MySQL, e faça a hospedagem do sistema de tarefas. Você pode procurar por hospedagens grátis em PHP ou hospedagens que ofereçam alguns dias para testes.

CAPÍTULO 14

PROGRAMANDO COM ORIENTAÇÃO A OBJETOS

Se você acompanhou o livro até aqui e fez todos os exemplos, você deve ter uma aplicação para a gestão de tarefas pessoais bastante funcional e já hospedada em um servidor na web. Agora vamos dar um importante passo à frente no mundo do PHP e da programação orientada a objetos. Não necessariamente adicionando mais funcionalidades à nossa aplicação, mas sim usando e alterando as suas bases para trabalhar com Programação Orientada a Objetos.

Este capítulo tem um grau de dificuldade um pouco maior para os iniciantes. Porém, com treino e paciência, essa etapa no aprendizado pode ser superada.

Você dificilmente vai querer programar sistemas sem Orientação a Objetos depois que entender bem como as classes e objetos podem ajudar a deixar um sistema ainda mais próximo do mundo que nos cerca.

Neste capítulo, também será muito importante que você tenha habilitado os erros do PHP e já esteja mais à vontade com a leitura dos erros informados e como encontrar as linhas problemáticas no

código. Digo isso, pois teremos grandes trechos de código pela frente, sem necessariamente executar o código a cada passo como temos feito no geral até aqui. Por isso, é importante conseguir interpretar os erros que o PHP nos mostra.

Lembra do uso da classe `DateTime` para fazer a formatação das datas? Lembra também do capítulo de envio de e-mails, no qual usamos a biblioteca `PHPMailer` através da criação de um **objeto** na variável `$email`? Em ambos os casos, usamos um pouco de programação orientada a objetos.

Você deve ter percebido que os objetos são compostos por diversas variáveis e funções, tudo embutido em apenas uma variável, quase como um array. A diferença é que um array não tem funções, apenas índices com valores.

Esta é uma forma bem interessante de se desenvolver aplicações, pois podemos pensar não apenas no fluxo da aplicação (ou seja, no recebimento da requisição pelo navegador e na devolução de uma resposta), mas também nas lógicas de cada parte separadamente.

A biblioteca `PHPMailer` é interessante, porque ela não tem as lógicas necessárias para o nosso gerenciamento de tarefas. Tudo o que ela faz é o envio de e-mails, preocupando-se apenas com isso e tentando fazer isso da melhor maneira possível.

Um dos grandes desafios da programação orientada a objetos é exatamente esta separação de responsabilidades entre as diferentes partes do código. Exatamente o que temos feito até aqui no livro.

Tentamos manter as funções separadas para acesso ao banco, exibição de templates, validação da entrada de dados etc. Mesmo

assim, algumas regras e lógicas estão espalhadas pela aplicação de controle de tarefas. E o que é mais importante, a representação do que é uma tarefa ou um anexo só pode ser encontrada no banco de dados, ou lendo e entendendo a construção dos arrays de tarefas.

Para reunir, por exemplo, a representação do que é uma **tarefa**, podemos usar uma espécie de **pacote** com os dados e métodos de uma tarefa. Este pacote recebe o nome de **classe**, e uma classe pode gerar os tais **objetos**. Olhando novamente para o uso da biblioteca PHPMailer, veja como foi criado o objeto `$email` :

```
<?php  
// ...  
  
$email = new PHPMailer();  
  
// ...
```

Neste caso, `PHPMailer` é a classe, e usamos o operador `new` para criar um novo objeto dela.

Se pensarmos em responsabilidades separadas, então podemos enxergar algumas partes da nossa aplicação que poderiam se transformar nesses "pacotes", que são as classes.

Temos algumas áreas bastante distintas da nossa aplicação que poderiam se tornar pacotes, ou melhor, classes com funcionalidades agrupadas e, de certa forma, isoladas do restante da aplicação, assim como é a classe `PHPMailer`. Mas neste capítulo, vou focar em apenas duas áreas centrais da aplicação:

- Representação no código do que é uma tarefa e um anexo;
- Manipulação do banco de dados.

Certo, já falei isso algumas vezes apenas aqui no começo do capítulo, mas vou tentar exemplificar de mais uma forma. Olhando para o código, e apenas para o código, você pode dizer, com precisão, o que é uma tarefa? Sim, no final do arquivo `tarefas`, antes de incluir o arquivo `template.php`, existe a criação de um array com todos os dados de uma tarefa. Mas, ainda assim, os anexos não aparecem neste array.

E o que é mais importante é que foi necessário ler o código do fluxo da aplicação para entender o que é uma tarefa. Isso pode até funcionar com a nossa aplicação do tamanho que está, mas acredite, isso não funciona se aplicação continuar crescendo.

O que precisamos aqui é da criação da representação do que é uma tarefa, e uma classe faz esse papel muito bem. Isso porque, como já sabemos, uma classe contém variáveis (que podemos chamar de propriedades) e funções (que podemos chamar de métodos).

Sendo assim, podemos pensar na criação de três classes para a nossa aplicação: uma chamada **Tarefa**, que vai conter os dados e a lógica de uma tarefa; uma chamada **Anexo**, com os dados e lógica de um anexo; e, por fim, uma chamada **RepositorioTarefas**, que fará a comunicação com o banco de dados quando o assunto for a manipulação de tarefas.

Aliás, **repositório** é uma nomenclatura bastante usada para se referir a uma fonte de dados que, no nosso caso, é um banco de dados MySQL. Entretanto, poderia ser qualquer fonte, como um arquivo texto, um site, uma outra aplicação etc.

Já as classes que usaremos para representar as tarefas e os

anexos podem ser chamadas de **entidades**, pois elas servem para representar uma entidade do mundo real, por assim dizer. Vamos então começar pela classe de tratamento de dados das tarefas.

14.1 A CLASSE TAREFA

Uma classe em programação orientada a objetos, como explicado anteriormente, é uma espécie de pacote que agrupa e isola determinadas funcionalidades. Para criar uma classe para manipular as nossas tarefas, precisamos planejar primeiro e definir o que é uma tarefa e quais ações ela pode sofrer e executar.

No nosso caso, esta não é uma tarefa complicada, pois já temos uma boa definição do que é uma tarefa. Já temos até mesmo uma tabela no banco de dados que descreve o que é uma tarefa. Sendo assim, vamos definir que uma tarefa é um conjunto de dados que contém os seguintes atributos:

- Um número de identificação único;
- Um nome;
- Uma descrição;
- Um prazo para a conclusão, que é uma data;
- Uma definição de estar ou não concluída;
- Nenhum, um ou mais anexos.

Estes são os atributos das nossas tarefas. Além dos atributos, uma classe, em geral, também tem funcionalidades. No caso das tarefas, podemos pensar no seguinte conjunto de funcionalidades por tarefa:

- Definir cada um dos atributos;
- Adicionar e listar os anexos.

Com base nesse pequeno estudo das tarefas, já podemos dizer que realmente este é um pacote que pode ser totalmente isolado do nosso sistema. Uma tarefa não precisa saber se os dados foram enviados via `POST` ou `GET`, ou ainda outras formas. Ela não precisa saber se seus dados serão exibidos em HTML ou em um relatório PDF.

Uma tarefa é apenas um conjunto de dados que pode ser manipulado por outras aplicações, não se importando sobre como essa aplicação funciona. Pensando nisso, podemos chegar a uma outra propriedade interessante de código orientado a objetos: **código orientado a objetos é reaproveitável**. E isso é realmente muito bom, pois podemos usar classes de uma aplicação em outra, e economizar tempo ao evitar reescrever funcionalidades semelhantes.

A classe `PHPMailer` não se importa com a origem do conteúdo de um e-mail, se é do banco de dados ou de um formulário preenchido pelo usuário. Ela é isolada do restante da aplicação, por isso pode ser reutilizada.

TODO CÓDIGO ORIENTADO A OBJETOS É REUTILIZÁVEL?

Esta é uma dúvida frequente para os iniciantes e a resposta é: **depende**. Ok, de novo esse papo de depende, mas é verdade.

Imagine que a classe `PHPMailer` funcionasse apenas com dados enviados via `POST`. Logo, ela não poderia ser usada em uma aplicação que usa `GET` para algum tipo de dado.

Da mesma forma, se nossa classe `Tarefa`, que ainda vamos construir, for acoplada ao formulário mandando um campo chamado `nome_tarefa`, ela não será muito reutilizável em outras aplicações. Ou, no mínimo, outras aplicações precisariam ter exatamente os mesmos nomes de campos para poder reutilizar a classe de tarefas.

Por isso, é sempre importante manter as classes isoladas do ambiente ao redor.

Vamos então à criação da classe `Tarefa`, que ficará no arquivo `Tarefa.php`, dentro da pasta `classes`:

```
<?php  
  
// arquivo classes/Tarefa.php  
  
class Tarefa  
{  
    private $id = 0;  
  
    public function setId($id)  
    {  
        $this->id = (int) $id;  
    }  
}
```

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

Este é apenas o começo da classe. Deixe-me explicar algumas coisas antes de continuarmos. Uma classe é um pacote com atributos e funcionalidades, simples assim.

No exemplo anterior, temos uma classe chamada `Tarefa` que tem um atributo **privado**, definido através da palavra `private`. Tem também duas funcionalidades, que chamamos de métodos, definidos usando uma sintaxe parecida com a das funções. A diferença aqui é a definição de que os métodos são públicos.

PRIVADO? PÚBLICO?

A ideia de criar classes é isolar determinadas funcionalidades do código, certo? Mas, às vezes, um(a) programador(a) pode sem querer, ou intencionalmente, acessar os atributos de uma classe e até mesmo quebrar seu funcionamento. Por isso, algumas linguagens de programação oferecem um recurso conhecido como visibilidade de atributos e métodos.

Isso é algo mais ou menos assim: a classe `Tarefa` possui uma propriedade chamada `$id` que deve ser um número inteiro. A melhor forma de garantir isso é protegendo essa propriedade para que apenas a própria classe possa acessá-la. Isso é feito por meio da palavra `private`, que torna a propriedade privada.

Já os métodos `setId()` e `getId()` devem ser públicos, pois é através deles que os usuários da classe podem definir e acessar o valor contido em `$id`.

É tipo aqueles guarda volumes em lojas e supermercados. Você não tem acesso direto ao que está lá dentro, e precisa pedir para o atendente guardar e depois buscar suas coisas.

Outra coisa nova é o uso da variável especial `$this`, que nada mais é do que uma referência ao objeto da classe. Complicado? Nem tanto.

Pense assim: o criador da classe não tem controle sobre qual nome será dado ao objeto, (ou objetos) criados pela classe. Logo, é

necessário ter um nome que faça referência ao objeto, não importando seu nome. Em PHP, essa referência é chamada de `$this`.

Veja que a classe possui atualmente dois métodos: um para definir o `id` da tarefa, chamado `setId()`; e um para pegar o `id` da tarefa, chamado `getId()`. Os nomes `get` e `set` vêm do inglês e significam, respectivamente, *pegar* e *definir*.

Nomear os métodos de acesso às propriedades como `get` e `set` é um padrão utilizado em diversas linguagens de programação. É bastante comum ouvir nomes como **getter** e **setter** para se referenciar a este tipo de método.

O método `setId()` recebe um parâmetro que tem o nome de `$id`. Este é convertido para um número inteiro, por meio de uma técnica chamada **casting**, usando a instrução `(int)`. E após ser convertido, o valor é atribuído à propriedade `id` do objeto da classe na linha `$this->id = (int) $id;`.

Atenção, repare que `$this->id` e `$id` são variáveis diferentes. A primeira é uma propriedade do objeto, e a segunda é uma variável recebida pelo parâmetro da função.

Já o método `getId()` apenas retorna o valor contido na propriedade `id` do objeto da classe, acessado usando a sintaxe `$this->id`. Mas uma tarefa não contém apenas o número de identificação, o `id`. Então, vamos adicionar o restante dos atributos e dos métodos da classe:

```
<?php  
// arquivo classes/Tarefa.php
```

```
class Tarefa
{
    private $id;

    private $nome;

    private $descricao;

    private $prazo;

    private $prioridade;

    private $concluida;

    private $anexos = [];

    public function setId($id)
    {
        $this->id = (int) $id;
    }

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

    public function setNome($nome)
    {
        $this->nome = $nome;
    }

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

    public function setDescricao($descricao)
    {
        $this->descricao = $descricao;
    }

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

```
public function setPrazo($prazo)
{
    $this->prazo = $prazo;
}

public function getPrazo()
{
    return $this->prazo;
}

public function setPrioridade($prioridade)
{
    $this->prioridade = $prioridade;
}

public function getPrioridade()
{
    return $this->prioridade;
}

public function setConcluida($concluida)
{
    $this->concluida = $concluida;
}

public function getConcluida()
{
    return $this->concluida;
}

public function setAnexos(array $anexos)
{
    $this->anexos = [];

    foreach ($anexos as $anexo) {
        $this->adicionarAnexo($anexo);
    }
}

public function adicionarAnexo(Anexo $anexo)
{
    array_push($this->anexos, $anexo);
}
```

```
public function getAnexos()
{
    return $this->anexos;
}
}
```

Ficou um pouco grande, né? Mas repare que todo o código é usado apenas para manipular os atributos da classe, usando os métodos get e set. Agora temos uma boa definição do que é uma tarefa através da classe `Tarefa`.

Perceba que uma classe documenta muito bem o que é uma tarefa, ao contrário do array com vários índices. A classe `Tarefa` está pronta e é bem isolada do que já temos até agora. Ela não depende de dados `POST` ou `GET`, não sabe o que é um banco de dados e não se importa com formulários. Ótimo!

Em algumas funções ou métodos da classe `Tarefa`, os parâmetros são precedidos de uma palavra. Você reparou?

O método `adicionarAnexo()` recebe um parâmetro chamado `$anexo` e este é do tipo `Anexo`. Ou seja, é um objeto da classe `Anexo`, que ainda não existe, mas que vamos criar em seguida. Perceba como fica mais simples de saber o que é um anexo, pois um anexo é um objeto da classe `Anexo`.

Já o método `setAnexos()` recebe um array, que poderia conter qualquer tipo de dados do PHP, como os números e as strings. Entretanto, basta ler o conteúdo do método para ver que ele chama o método `adicionarAnexo()`, que por sua vez, como já vimos, recebe um objeto da classe `Anexo`.

Quando declaramos o tipo de dados que uma função ou método de uma classe recebe, estamos usando uma funcionalidade

do PHP chamada **type hinting**. Isso significa algo como "sugestão do tipo de dados".

Esquisito? Sim, um pouco, no começo. Mas poder declarar o tipo de dados que um método ou uma função espera receber é um recurso muito bom para entender melhor o que uma classe representa.

Não acredita? Veja, por exemplo, os métodos `setDescricao()` e `setPrioridade()`. O que eles recebem? A descrição é um texto, ou seja, uma string. Certo, isso fica fácil de deduzir, mas a prioridade poderia ser uma string com o valor "urgente", ou um número de 1 a 3, ou ainda um booleano, apenas `true` ou `false`.

Veja como o código ficaria mais simples de entender se também declarassemos os tipos de dados esperados pelos demais métodos da classe `Tarefa`:

```
<?php

// arquivo classes/Tarefa.php

class Tarefa
{
    public function setId(int $id)

    public function setNome(string $nome)

    public function setDescricao(string $descricao)

    public function setPrazo(string $prazo)

    public function setPrioridade(int $prioridade)

    public function setConcluida(bool $concluida)

    public function setAnexos(array $anexos)
```

```
    public function adicionarAnexo(Anexo $anexo)
}
```

Agora a classe está bem mais descritiva, não é? Mas, infelizmente, temos um pequeno problema aqui. Até a versão 5.6 do PHP, a declaração dos tipos de dados esperados por uma função aceitava apenas arrays, nomes de classes/interfaces e callables , que podem ser strings com nomes de funções a serem executadas ou ainda closures (funções anônimas). Aliás, recomendo uma pesquisa sobre as closures em PHP, mas essa pesquisa pode ficar para depois de você finalizar a leitura do livro.
;)

Sobre a declaração dos tipos de dados que a função recebe, se você está usando a versão 5.6 do PHP ou anteriores, você não poderá declarar os tipo int , string e boolean , como foi feito no último exemplo. Isso porque esta funcionalidade foi adicionada apenas no PHP 7.

E por falar em novidades do PHP 7, olha só como ainda podemos deixar o código mais descriptivo sobre os tipos de dados:

```
<?php

// arquivo classes/Tarefa.php

class Tarefa
{
    public function getId(): int
    {
        return $this->id;
    }

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

```
public function getDescricao(): string  
// ...  
  
public function getPrazo(): string  
// ...  
  
public function getPrioridade(): int  
// ...  
  
public function getConcluida(): bool  
// ...  
  
public function getAnexos(): array  
// ...  
}
```

Essa também é uma novidade do PHP 7, a declaração do tipo de dados que uma função ou um método podem retornar, usando os dois pontos e o tipo esperado após a lista de argumentos e antes de abrir o corpo da função usando as chaves.

A declaração de tipos de retorno e os tipos esperados usando `int`, `string` e `bool` são itens a se considerar quando estiver desenvolvendo um projeto, pois a versão do PHP no servidor que vai hospedar a aplicação pode ser mais antiga do que a versão que você tem no seu computador. Por isso, quando começar um projeto, não esqueça de perguntar sobre como será o ambiente de produção, versões instaladas etc.

Eu vou manter os exemplos seguintes no livro sem a declaração dos tipos não aceitos até o PHP 5.6 e sem a declaração dos tipos de retorno. Mas se você está usando o PHP 7, siga em frente e adicione os tipos esperados e de retorno, pois o código fica bem melhor documentado, como você pode perceber.

14.2 O REPOSITÓRIO DE TAREFAS

O próximo passo é criar uma classe que sabe o que é um banco de dados (no nosso caso, o MySQL). Também sabe como gravar novas tarefas no banco, atualizar tarefas e como deletá-las. Este tipo de classe é conhecida como **Repositório**, e sua função é fazer o intermédio entre o sistema e o banco de dados ou outras fontes de informação.

Vejamos um possível esboço para esta classe:

```
<?php

// arquivo classes/RepositorioTarefas.php

class RepositorioTarefas
{
    public function salvar(Tarefa $tarefa)
    {
        // Salva uma tarefa no banco de dados
    }

    public function atualizar(Tarefa $tarefa)
    {
        // Atualiza uma tarefa no banco de dados
    }

    public function buscar($tarefa_id)
    {
        // Busca uma ou todas as tarefas no banco de dados
    }

    public function remover($tarefa_id)
    {
        // Remove uma tarefa do banco de dados
    }
}
```

Este esboço contém as quatro principais ações que o repositório precisa realizar, sendo elas: salvar, atualizar, buscar e

remover tarefas. Repare que os métodos `salvar()` e `atualizar()` recebem um objeto da classe `Tarefa` que criamos há pouco. Isso, mais uma vez, deixa bastante claro qual é a função desta classe.

Para que seja possível realizar essas atividades, o repositório precisa de acesso ao banco de dados, e isso será feito através da conexão que já estamos usando em nossas funções, que atualmente acessam o banco. A diferença é que não precisaremos mais ficar passando a conexão para cada função como estamos fazendo até aqui, pois faremos isso apenas uma vez quando criarmos um objeto da classe `RepositorioTarefas` usando o seu construtor.

Veja como fica o código e leia adiante para saber o que é um construtor:

```
<?php  
  
// arquivo classes/RepositorioTarefas.php  
  
class RepositorioTarefas  
{  
    private $conexao;  
  
    public function __construct($conexao)  
    {  
        $this->conexao = $conexao;  
    }  
  
    // ...
```

Criamos uma propriedade privada chamada `$conexao` que guardará a variável com a conexão ao banco que já estamos usando. Além disso, tem algo novo nessa classe, que é o método `__construct()`. Este é um método especial que será executado sempre que um novo objeto da classe for criado.

Não se assuste, pois vou explicar melhor na hora em que formos finalmente usar as nossas novas classes. Mas apenas tenha em mente que, sempre que criamos um novo objeto de uma classe usando o operador new , o PHP executa o método __construct() desta classe.

Agora tudo o que resta é preencher o restante das funcionalidades do repositório, e então adicionar as novas classes na aplicação. Repare que os métodos do repositório farão basicamente o que as funções no arquivo banco.php fazem. Logo, não há grandes novidades aqui, somente a nova sintaxe usando os objetos:

```
<?php

// arquivo classes/RepositorioTarefas.php

class RepositorioTarefas
{
    private $conexao;

    public function __construct($conexao)
    {
        $this->conexao = $conexao;
    }

    public function salvar(Tarefa $tarefa)
    {
        $nome = $tarefa->getNome();
        $descricao = $tarefa->getDescricao();
        $prioridade = $tarefa->getPrioridade();
        $prazo = $tarefa->getPrazo();
        $concluida = ($tarefa->getConcluida()) ? 1 : 0;

        if (is_object($prazo)) {
            $prazo = $prazo->format('Y-m-d');
        }

        $sqlGravar = "
            INSERT INTO tarefas
```

```

        (nome, descricao, prioridade, prazo, concluida)
VALUES
(
    '{$nome}',
    '{$descricao}',
    {$prioridade},
    '{$prazo}',
    {$concluida}
)
";
}

$this->conexao->query($sqlGravar);
}

public function atualizar(Tarefa $tarefa)
{
    $id = $tarefa->getId();
    $nome = $tarefa->getNome();
    $descricao = $tarefa->getDescricao();
    $prioridade = $tarefa->getPrioridade();
    $prazo = $tarefa->getPrazo();
    $concluida = ($tarefa->getConcluida()) ? 1 : 0;

    if (is_object($prazo)) {
        $prazo = $prazo->format('Y-m-d');
    }

    $sqlEditar = "
        UPDATE tarefas SET
            nome = '{$nome}',
            descricao = '{$descricao}',
            prioridade = {$prioridade},
            prazo = '{$prazo}',
            concluida = {$concluida}
        WHERE id = {$id}
    ";
}

$this->conexao->query($sqlEditar);
}

public function buscar($tarefa_id = 0)
{
    if ($tarefa_id > 0) {
        return $this->buscar_tarefa($tarefa_id);
    } else {

```

```

        return $this->buscar_tarefas();
    }
}

private function buscar_tarefas()
{
    $sqlBusca = 'SELECT * FROM tarefas';
    $resultado = $this->conexao->query($sqlBusca);

    $tarefas = [];

    while ($tarefa = $resultado->fetch_object('Tarefa')) {
        $tarefas[] = $tarefa;
    }

    return $tarefas;
}

private function buscar_tarefa($tarefa_id)
{
    $sqlBusca = "SELECT * FROM tarefas WHERE id = {$tarefa_id}";
    $resultado = $this->conexao->query($sqlBusca);

    $tarefa = $resultado->fetch_object('Tarefa');

    return $tarefa;
}

function remover($tarefa_id)
{
    $sqlRemover = "DELETE FROM tarefas WHERE id = {$tarefa_id}";
    $this->conexao->query($sqlRemover);
}
}

```

Você deve ter reparado uma pequena mudança em relação às funções do arquivo `banco.php`, que é o uso do objeto `$conexao`, não como parâmetro para as funções do MySQLi, mas sim como um objeto. Repare que, no lugar de

`mysqli_query($conexao, $sql)` , agora estamos usando apenas `$conexao->query($sql)` , que é a interface orientada a objetos da biblioteca MySQLi.

Você vai reparar nos exemplos que a interface orientada a objetos da MySQLi é muito parecida com a opção procedural. Basta remover o prefixo `mysqli_` do nome das funções, e usá-las como métodos do objeto `$conexao` .

Você deve ter reparado também que há dois métodos diferentes do que o esboço que fizemos. Os novos métodos são `buscar_tarefa()` e `buscar_tarefas()` , que buscam no banco apenas uma tarefa ou todas elas, respectivamente. Estes métodos são privados, o que significa que eles podem ser acessados apenas de dentro do contexto da classe.

A ideia para fazer esses novos métodos é deixar o uso da classe mais simples. Isso porque, quando estivermos usando o repositório, basta chamar o método `buscar()` sem parâmetros para buscarmos todas as tarefas, ou com um parâmetro (o `id` de uma tarefa) para buscar apenas uma tarefa.

Claro que toda esta lógica poderia simplesmente estar dentro do bloco do `if` . Porém, desta maneira fica bem claro o que cada método faz, além de simplificar bastante a leitura do método `buscar()` .

Algo que você deve ter reparado neste código é o uso do método `fetch_object()` do objeto `$resultado` . Este pega o resultado da busca no banco de dados e "transforma" em um objeto da classe que informamos, que no nosso caso é a classe `Tarefa` . Bem legal, não é?

Se não fosse por esse método, precisaríamos pegar o resultado em forma de array e atribuir os valores a cada um dos atributos da classe, usando os métodos que criamos, como `setId()` e `setDescricao()`.

Ainda não adicionamos os dados dos anexos aqui para deixar o contexto mais simples, mas faremos isso ainda neste capítulo. Vamos então conectar as novas classes à aplicação.

14.3 USANDO AS NOVAS CLASSES NA APLICAÇÃO

Ainda temos um pouco de trabalho pela frente antes de ter todos os dados da nossa aplicação passando pelo repositório de tarefas. Mas, neste momento, já é possível ligar algumas pontas e usar a classe `Tarefa` para exibir os dados das tarefas.

O primeiro passo é a inclusão dos novos arquivos usando o `require`. Vamos começar então pelo arquivo `tarefas.php`, responsável por exibir a lista de tarefas:

```
<?php  
  
require "config.php";  
require "banco.php";  
require "ajudantes.php";  
require "classes/Tarefa.php";  
require "classes/RepositorioTarefas.php";  
  
// ...
```

Agora vamos alterar o trecho antes da verificação dos dados no `post`. Vamos criar um objeto da classe `RepositorioTarefas` passando a nossa conexão com o MySQL, que está na variável `mysqli`. Também criaremos um objeto da classe `Tarefa` que

poderá receber os dados enviados via formulário, ou ser usado para exibir um formulário vazio:

```
<?php  
// ...  
  
$repositorio_tarefas = new RepositorioTarefas($mysqli);  
  
$exibir_tabela = true;  
  
$tem_erros = false;  
$erros_validacao = [];  
  
$tarefa = new Tarefa();  
$tarefa->setPrioridade(1);
```

Agora vamos pular todo o trecho dentro do `if`, e vamos para o final do arquivo, onde a lista de tarefas é recuperada do banco e um array com os dados da tarefa é criado para utilização no formulário. Esse trecho vai ficar bem mais simples:

```
<?php  
// Aqui termina o if tem_post()  
  
$tarefas = $repositorio_tarefas->buscar();  
  
include "template.php";
```

Nossa, o que aconteceu aqui? Removemos todo o trecho com a criação de uma tarefa em branco, ou com o preenchimento dos dados via `POST` para a exibição no formulário! Como assim?

Na verdade, é bem simples. Repare que, antes de verificar se tem dados via `POST`, nós criamos um objeto da classe `Tarefa`. E é este objeto que vamos usar em nosso formulário no arquivo `formulario.php`.

Por falar no arquivo `formulario.php`, vamos fazer as devidas alterações para que o formulário use o objeto da classe `Tarefa` e não mais um array para exibir os dados da tarefa:

```
<form method="POST">
    <input type="hidden" name="id"
        value="php echo $tarefa-&gt;getId(); ?&gt;" /&gt;
    ...
</pre
```

Este foi apenas um exemplo do que deve ser feito, que é a alteração de trechos como `$tarefa['id']` e `$tarefa['nome']` pelo uso dos métodos da classe `Tarefa`, como `$tarefa->getId()` e `$tarefa->getNome()`.

Como o arquivo é um pouco grande, mostrarei aqui apenas os trechos onde o array deve ser substituído pelo objeto `tarefa`:

```
<form method="POST">
    <input type="hidden" name="id"
        value="php echo $tarefa-&gt;getId(); ?&gt;" /&gt;

    Tarefa:
    &lt;input type="text" name="nome"
        value="<?php echo $tarefa-&gt;getNome(); ?&gt;" /&gt;

    Descrição (Opcional):
    &lt;textarea name="descricao"&gt;
        &lt;?php echo $tarefa-&gt;getDescricao(); ?&gt;
    &lt;/textarea&gt;

    Prazo (Opcional):
    &lt;input type="text" name="prazo"
        value="<?php echo traduz_data_exibir(
            $tarefa-&gt;getPrazo()
        ); ?&gt;" /&gt;

    &lt;legend&gt;Prioridade:&lt;/legend&gt;
    &lt;input type="radio" name="prioridade" value="1"
        &lt;?php echo ($tarefa-&gt;getPrioridade() == 1)
            ? 'checked' : '';</pre
```

```

    ?> /> Baixa
<input type="radio" name="prioridade" value="2"
    <?php echo ($tarefa->getPrioridade() == 2)
        ? 'checked' : '';
    ?> /> Média
<input type="radio" name="prioridade" value="3"
    <?php echo ($tarefa->getPrioridade() == 3)
        ? 'checked' : '';
    ?> /> Alta

Tarefa concluída:
<input type="checkbox" name="concluida" value="1"
    <?php echo ($tarefa->getConcluida())
        ? 'checked' : '';
    ?> />

<input type="submit"
    value="<?php echo ($tarefa->getId() > 0)
        ? 'Atualizar' : 'Cadastrar';
?>" class="botao" />

```

Nada muito especial, certo? Apenas a mudança do array para o objeto. O próximo passo é alterar o arquivo `tabela.php` para também usar o objeto da classe `Tarefa` no lugar do array. Veja como deve ficar o arquivo:

```

<table>
    <tr>
        <th>Tarefa</th>
        <th>Descrição</th>
        <th>Prazo</th>
        <th>Prioridade</th>
        <th>Concluída</th>
        <th>Opções</th>
    </tr>
    <?php foreach ($tarefas as $tarefa) : ?>
        <tr>
            <td>
                <a href="tarefa.php?id=<?php
                    echo $tarefa->getId(); ?>">
                    <?php echo $tarefa->getNome(); ?>
                </a>
            </td>

```

```

</td>
<td>
    <?php echo $tarefa->getDescricao(); ?>
</td>
<td>
    <?php echo
        traduz_data_para_exibir(
            $tarefa->getPrazo()
        );
    ?>
</td>
<td>
    <?php echo
        traduz_prioridade($tarefa->getPrioridade());
    ?>
</td>
<td>
    <?php echo
        traduz_concluida($tarefa->getConcluida()); ?>
</td>
<td>
    <a href="editar.php?id=<?php
        echo $tarefa->getId(); ?>">
        Editar
    </a>
    <a href="remover.php?id=<?php
        echo $tarefa->getId(); ?>">
        Remover
    </a>
</td>
</tr>
<?php endforeach; ?>
</table>

```

Mais uma vez, nada muito especial. Estamos apenas utilizando os métodos da classe `Tarefa` para recuperar os dados de uma tarefa e exibi-los na tabela. Neste momento, você deve ser capaz de acessar a aplicação e ver a lista de tarefas.

Mas, atenção, não tente adicionar uma nova, pois ainda não alteramos o trecho dentro do `if(tem_post())` no arquivo `tarefas.php` !

Se você experimentar adicionar uma tarefa, você verá erros do tipo usar um objeto como se fosse um array. Isso porque a variável `$tarefa`, usada como um array dentro do bloco do `if`, é agora um objeto da classe `Tarefa` e não mais um array.

Sendo assim, tente listar as tarefas e se certifique de que não há erros até aqui. Em seguida, altere o trecho dentro do `if(tem_post())` para preencher os dados de uma tarefa, usando o objeto da classe `Tarefa` e o repositório de tarefas para fazer a gravação dos dados no banco de dados:

```
<?php  
  
// ...  
  
if (tem_post()) {  
    if (array_key_exists('nome', $_POST)  
        && strlen($_POST['nome']) > 0) {  
        $tarefa->setNome($_POST['nome']);  
    } else {  
        $tem_erros = true;  
        $erros_validacao['nome'] =  
            'O nome da tarefa é obrigatório!';  
    }  
  
    if (array_key_exists('descricao', $_POST)) {  
        $tarefa->setDescricao($_POST['descricao']);  
    } else {  
        $tarefa->setDescricao('');  
    }  
  
    if (array_key_exists('prazo', $_POST)  
        && strlen($_POST['prazo']) > 0) {  
        if (validar_data($_POST['prazo'])) {  
            $tarefa->setPrazo(  
                traduz_data_br_para_objeto($_POST['prazo'])  
            );  
        } else {  
            $tem_erros = true;  
            $erros_validacao['prazo'] =  
                'O prazo não é uma data válida!';  
        }  
    }  
}
```

```

        }
    } else {
        $tarefa->setPrazo(' ');
    }

    $tarefa->setPrioridade($_POST['prioridade']);

    if (array_key_exists('concluida', $_POST)) {
        $tarefa->setConcluida(true);
    } else {
        $tarefa->setConcluida(false);
    }

    if (! $tem_erro) {
        $repositorio_tarefas->salvar($tarefa);

        if (isset($_POST['lembrete'])
            && $_POST['lembrete'] == '1') {
            enviar_email($tarefa);
        }

        header('Location: tarefas.php');
        die();
    }
}

// ...

```

Neste trecho, estamos preenchendo os dados do objeto `$tarefa` conforme a validação do formulário é feita. No final, caso não existam erros, usamos o método `salvar()` do repositório. Caso algum erro exista, o fluxo vai continuar normalmente e o arquivo `template.php` será incluído.

Neste caso, não precisamos criar um array com uma tarefa vazia ou com os dados do formulário, pois o objeto `$tarefa` já contém todos os dados.

Mas ainda existe uma função nova ali no meio, a `traduz_data_br_para_objeto()`, que é bem parecida com as

demais funções que traduzem a data para formatos diferentes. A diferença é que esta retorna um objeto da classe `DateTime` com a data. Essa nova função deve ser adicionada no arquivo `ajudantes.php` e deve ser assim:

```
<?php

// ...

function traduz_data_br_para_objeto($data)
{
    if ($data == "") {
        return "";
    }

    $dados = explode("/", $data);

    if (count($dados) != 3) {
        return $data;
    }

    return DateTime::createFromFormat('d/m/Y', $data);
}

// ...
```

Neste momento, você pode adicionar uma nova tarefa, mas ainda sem o uso da opção de lembrete por e-mail, pois não alteramos esta parte para usar o objeto da classe `Tarefa`. Então, mais uma vez, faça um teste para ver como as coisas estão funcionando até aqui, e verifique se o PHP não está informando erros.

14.4 USANDO AS CLASSES NO ENVIO DO E-MAIL

Após fazer seus testes, vamos fazer a alteração no envio do e-

mail para que este trecho do nosso código também use o objeto da classe Tarefa no lugar do array.

A função enviar_email() , que está no arquivo ajudantes.php , vai passar a receber apenas um parâmetro com o objeto da classe Tarefa em vez de um array para a tarefa e um array para os anexos. Faremos isso pois uma tarefa já contém anexos, então não é necessário passar os anexos em um segundo parâmetro.

Então, veja como deve ficar a função enviar_email() :

```
<?php

function enviar_email(Tarefa $tarefa)
{
    require "bibliotecas/PHPMailer/PHPMailerAutoload.php";

    $corpo = preparar_corpo_email($tarefa);

    $email = new PHPMailer();

    $email->isSMTP();
    $email->Host = "smtp.gmail.com";
    $email->Port = 587;
    $email->SMTPSecure = 'tls';
    $email->SMTPAuth = true;
    $email->Username = "seuemail@dominio.com";
    $email->Password = "senhasecreta";
    $email->setFrom(
        "seuemail@dominio.com",
        "Avisador de Tarefas"
    );
    $email->addAddress(EMAIL_NOTIFICACAO);
    $email->Subject = "Aviso de tarefa: {$tarefa->getNome()}";
    $email->msgHTML($corpo);

    foreach ($tarefa->getAnexos() as $anexo) {
        $email->addAttachment("anexos/{$anexo->getArquivo()}");
    }
}
```

```
$email->send();  
}
```

No corpo da função `enviar_email()`, a chamada da função `preparar_corpo_email()` também foi alterada para ter apenas um parâmetro. No restante, o que mudou foi apenas o assunto no e-mail (`Subject`) e a adição do anexo que vai usar o método `getArquivo()` da classe `Anexo`.

Ops, ainda não criamos a classe `Anexo` até o momento. Mas tudo bem, se uma tarefa não tiver anexos, aquele trecho do código não será executado e não teremos um erro.

Vamos deixar assim por enquanto. Logo mais, adicionaremos a classe `Anexo` e ligaremos as pontas soltas.

A função `preparar_corpo_email()` também deve ser alterada, mas apenas a sua declaração, para que receba apenas o objeto da classe `Tarefa`. Isso porque, na verdade, os dados são apenas utilizados no arquivo `template_email.php`:

```
<?php  
  
// ...  
  
function preparar_corpo_email(Tarefa $anexo)  
{  
    // ...  
}
```

E o arquivo `template_email.php` deve ficar assim:

```
<h1>Tarefa: <?php echo $tarefa->getNome(); ?></h1>  
  
<p>  
    <strong>Concluída:</strong>  
    <?php echo traduz_concluida($tarefa->getConcluida()); ?>  
</p>  
<p>
```

```
<strong>Descrição:</strong>
<?php echo nl2br($tarefa->getDescricao()); ?>
</p>
<p>
    <strong>Prazo:</strong>
    <?php echo traduz_data_para_exibir($tarefa->getPrazo()); ?>
</p>
<p>
    <strong>Prioridade:</strong>
    <?php echo traduz_prioridade($tarefa->getPrioridade()); ?>
</p>

<?php if (count($tarefa->getAnexos()) > 0) : ?>
    <p><strong>Atenção!</strong> Esta tarefa contém anexos!</p>
<?php endif; ?>
```

E esse foi mais um arquivo sem grandes surpresas, apenas a substituição dos índices do array pelos métodos da classe `Tarefa`.

Experimente cadastrar uma tarefa e fazer o envio do e-mail. Preste atenção nos possíveis problemas informados pelo PHP, e certifique-se de ter tudo funcional até aqui.

14.5 CRIANDO A CLASSE PARA OS ANEXOS

Com o que fizemos até aqui, uma boa parte da aplicação já está usando o novo formato com Orientação a Objetos. Porém, ainda falta a parte da edição da tarefa e, é claro, o tratamento dos anexos. Vamos primeiro à exibição e edição das tarefas.

Faça um teste. Tente exibir uma tarefa clicando no link no seu nome, que está na tabela com a lista de tarefas. O que acontece? Algum erro? Não, certo?

Isso acontece pois a exibição de tarefas é feita usando o arquivo `tarefa.php` e o template `template_tarefa.php`, e nenhum desses arquivos foi alterado para usar as classes. Sendo assim, eles

ainda estão funcionando corretamente com os arrays.

Bem, vamos em frente, pois precisamos alterar toda a aplicação para usar as classes. Assim, teremos um código uniforme e isso vai facilitar bastante as futuras possíveis alterações.

Esta parte da aplicação faz uso dos anexos, tanto para adicionar novos como para listar os anexos atuais. Sendo assim, esta é uma boa hora para criarmos a classe `Anexo`. Ela será bastante parecida com a classe `Tarefa`, pois vai representar um registro do banco de dados.

A nova classe vai ficar em um arquivo com o mesmo nome dentro do diretório `classes`. Este é o seu conteúdo:

```
<?php

// arquivo classes/Anexo.php

class Anexo
{
    private $id = 0;
    private $tarefa_id;
    private $nome;
    private $arquivo;

    public function setId($id)
    {
        $this->id = (int) $id;
    }

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

    public function setTarefaId($tarefa_id)
    {
        $this->tarefa_id = $tarefa_id;
    }
}
```

```
public function getTarefaId()
{
    return $this->tarefa_id;
}

public function setNome($nome)
{
    $this->nome = $nome;
}

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

public function setArquivo($arquivo)
{
    $this->arquivo = $arquivo;
}

public function getArquivo()
{
    return $this->arquivo;
}
}
```

Esta classe não tem novidades se comparada à classe `Tarefa`, certo? E mais uma vez, perceba que a classe `Anexo` não faz qualquer referência a um banco de dados, pois seu objetivo é apenas representar o que é um anexo.

E por falar em banco de dados, temos uma decisão importante agora: onde colocar a lógica para a comunicação com o banco de dados para os anexos?

Uma boa opção é a criação de mais uma classe do tipo "repositório", algo como **RepositorioAnexos**. Mas, para manter o contexto do livro menos complexo, optei por usar o repositório das tarefas para também lidar com os dados dos anexos. Afinal de

contas, os anexos são complementos dos dados das tarefas.

Sendo assim, precisamos adicionar na classe `RepositorioTarefas` os métodos para lidar com os anexos, além de alterar os métodos atuais para que preencham os anexos, quando necessário.

```
<?php

// arquivo classes/RepositorioTarefas.php

// ...

public function buscar_anexos($tarefa_id)
{
    $sqlBusca =
        "SELECT * FROM anexos WHERE tarefa_id = {$tarefa_id}";
    $resultado = $this->conexao->query($sqlBusca);

    $anexos = array();

    while ($anexo = $resultado->fetch_object('Anexo')) {
        $anexos[] = $anexo;
    }

    return $anexos;
}

public function buscar_anexo($anexo_id)
{
    $sqlBusca = "SELECT * FROM anexos WHERE id = {$anexo_id}";
    $resultado = $this->conexao->query($sqlBusca);

    return $resultado->fetch_object('Anexo');
}

public function salvar_anexo(Anexo $anexo)
{
    $sqlGravar = "INSERT INTO anexos
        (tarefa_id, nome, arquivo)
    VALUES
    (

```

```

        {$anexo->getTarefaId()},
        '{$anexo->getNome()}',
        '{$anexo->getArquivo()}'  

    )  

";  
  

$this->conexao->query($sqlGravar);  

}  
  

public function remover_anexo($id)  

{  

    $sqlRemover = "DELETE FROM anexos WHERE id = {$id}";  
  

    $this->db->query($conexao, $sqlRemover);  

}

```

Todos os novos métodos devem ser adicionados dentro da classe `RepositorioTarefas`. Após trabalhar um pouco com a classe `Tarefa`, os novos métodos não apresentam novidades, não é?

Ah, ainda existem algumas alterações pendentes no nosso repositório. Quando buscarmos uma tarefa ou mais, queremos que o array de anexos seja preenchido, pois uma lista de anexos faz parte da representação de uma tarefa. Para isso, altere os métodos `buscar_tarefa()` e `buscar_tarefas()` para chamar o método `buscar_anexos()`, para cada tarefa recuperada do banco de dados:

```

<?php  
  

// arquivo classes/RepositorioTarefas.php  
  

// ...  
  

private function buscar_tarefas()  

{
    $sqlBusca = 'SELECT * FROM tarefas';
    $resultado = $this->conexao->query($sqlBusca);

```

```

$tarefas = [];

while ($tarefa = $resultado->fetch_object('Tarefa')) {
    $tarefa->setAnexos(
        $this->buscar_anexos($tarefa->getId())
    );
    $tarefas[] = $tarefa;
}

return $tarefas;
}

private function buscar_tarefa($id)
{
    $sqlBusca = 'SELECT * FROM tarefas WHERE id = ' . $id;
    $resultado = $this->conexao->query($sqlBusca);

    $tarefa = $resultado->fetch_object('Tarefa');
    $tarefa->setAnexos(
        $this->buscar_anexos($tarefa->getId())
    );

    return $tarefa;
}

// ...

```

Perceba que usamos o método `setAnexos()` para enviar um array de anexos para cada objeto da classe `Tarefa`.

14.6 USANDO AS CLASSES NA EXIBIÇÃO E EDIÇÃO DAS TAREFAS

Com o repositório já tratando dos anexos, podemos alterar a área da aplicação que exibe uma tarefa e seus anexos.

Como o arquivo `tarefa.php` não é muito grande e boa parte dele deve ser alterada, achei mais simples colocar todo o arquivo aqui. Veja como ele deve ficar e faça as alterações no seu:

```

<?php

// arquivo tarefa.php

require "config.php";
require "banco.php";
require "ajudantes.php";
require "classe/Tarefa.php";
require "classe/Anexo.php";
require "classe/RepositorioTarefas.php";

$repositorio_tarefas = new RepositorioTarefas($mysqli);

$tem_erro = false;
$erros_validacao = array();

if (tem_post()) {
    $tarefa_id = $_POST['tarefa_id'];

    if (! array_key_exists('anexo', $_FILES)) {
        $tem_erro = true;
        $erros_validacao['anexo'] =
            'Você deve selecionar um arquivo para anexar';
    } else {
        $dados_anexo = $_FILES['anexo'];

        if (tratar_anexo($dados_anexo)) {
            $anexo = new Anexo();
            $anexo->setTarefaId($tarefa_id);
            $anexo->setNome($dados_anexo['name']);
            $anexo->setArquivo($dados_anexo['name']);
        } else {
            $tem_erro = true;
            $erros_validacao['anexo'] =
                'Envie apenas anexos nos formatos zip ou pdf';
        }
    }

    if (! $tem_erro) {
        $repositorio_tarefas->salvar_anexo($anexo);
    }
}

$tarefa = $repositorio_tarefas->buscar($_GET['id']);

```

```
include "template_tarefa.php";
```

As mudanças no início do arquivo são para incluir todas as classes. As mudanças no meio do arquivo são para usar a classe Anexo para criar um anexo e para fazer a gravação no banco de dados. A parte da validação continuou a mesma. Já no final a alteração feita foi para recuperar a tarefa do banco de dados usando o repositório.

As próximas mudanças devem ser feitas no template que exibe a tarefa, seus anexos e o formulário para adicionar anexos, no arquivo template_tarefa.php . O primeiro trecho exibe os dados das tarefas e deve ficar assim:

```
<h1>Tarefa: <?php echo $tarefa->getNome(); ?></h1>

<p><a href="tarefas.php">Voltar para alista de tarefas</a>.</p>

<p>
    <strong>Concluída:</strong>
    <?php echo traduz_concluida($tarefa->getConcluida()); ?>
</p>
<p>
    <strong>Descrição:</strong>
    <?php echo nl2br($tarefa->getDescricao()); ?>
</p>
<p>
    <strong>Prazo:</strong>
    <?php echo traduz_data_exibir($tarefa->getPrazo()); ?>
</p>
<p>
    <strong>Prioridade:</strong>
    <?php echo traduz_prioridade($tarefa->getPrioridade()); ?>
</p>
```

O segundo trecho mostra a lista de anexos ou apenas uma frase informando que a tarefa não tem anexos:

```
<h2>Anexos</h2>
<!-- lista de anexos -->
```

```

<?php if (count($tarefa->getAnexos()) > 0) : ?>
<table>
    <tr>
        <th>Arquivo</th>
        <th>Opções</th>
    </tr>
    <?php foreach ($tarefa->getAnexos() as $anexo) : ?>
    <tr>
        <td><?php echo $anexo->getNome(); ?></td>
        <td>
            <a href="anexos/<?php
                echo $anexo->getArquivo(); ?>">
                Download
            </a>
            <a href="remover_anexo.php?id=<?php
                echo $anexo->getId(); ?>">Remover</a>
        </td>
    </tr>
    <?php endforeach; ?>
</table>
<?php else : ?>
    <p>Não há anexos para esta tarefa.</p>
<?php endif; ?>

```

Veja que, neste trecho, devemos substituir o uso da variável `$anexos` pelo método `getAnexos()` do objeto `$tarefa`. Isso é possível, pois alteramos o repositório das tarefas para sempre adicionar os anexos de uma tarefa quando usarmos o método `buscar()`.

Já o último trecho do arquivo deve mudar apenas para usar o método `getId()` do objeto `$tarefa` no input `tarefa_id`:

```

...
<input type="hidden" name="tarefa_id"
       value="<?php echo $tarefa->getId(); ?>" />
...

```

Após estas alterações, tente exibir os dados de uma tarefa

usando o link no nome da tarefa na lista de tarefas. Tente também o upload de arquivos. Tudo deve funcionar normalmente.

A edição das tarefas utiliza boa parte do que já alteramos para a exibição do formulário de cadastro de tarefas. Sendo assim, precisamos alterar apenas o arquivo `editar.php`.

Este arquivo é bastante parecido com o trecho de validação dos dados para o cadastro de tarefas no arquivo `tarefas.php`. Porém, ele possui certas diferenças, como a necessidade de apagar os dados não obrigatórios, como a descrição e o prazo.

Como este arquivo também não é muito grande, coloquei-o na íntegra para facilitar a leitura:

```
<?php

require "config.php";
require "banco.php";
require "ajudantes.php";
require "classes/Tarefa.php";
require "classes/Anexo.php";
require "classes/RepositorioTarefas.php";

$repositorio_tarefas = new RepositorioTarefas($mysqli);
$tarefa = $repositorio_tarefas->buscar($_GET['id']);

$exibir_tabela = false;
$tem_erros = false;
$erros_validacao = [];

if (tem_post()) {

    if (isset($_POST['nome']) && strlen($_POST['nome']) > 0) {
        $tarefa->setNome($_POST['nome']);
    } else {
        $tem_erros = true;
        $erros_validacao['nome'] =
            'O nome da tarefa é obrigatório!';
    }
}
```

```

if (isset($_POST['descricao'])) {
    $tarefa->setDescricao($_POST['descricao']);
} else {
    $tarefa->setDescricao('');
}

if (isset($_POST['prazo']) && strlen($_POST['prazo']) > 0) {
    if (validar_data($_POST['prazo'])) {
        $tarefa->setPrazo(
            traduz_data_br_para_objeto($_POST['prazo'])
        );
    } else {
        $tem_erros = true;
        $erros_validacao['prazo'] =
            'O prazo não é uma data válida!';
    }
} else {
    $tarefa->setPrazo('');
}

$tarefa->setPrioridade($_POST['prioridade']);

if (isset($_POST['concluida'])) {
    $tarefa->setConcluida(true);
} else {
    $tarefa->setConcluida(false);
}

if (! $tem_erros) {
    $repositorio_tarefas->atualizar($tarefa);

    if (
        isset($_POST['lembrete'])
        &&
        $_POST['lembrete'] == '1'
    ) {
        enviar_email($tarefa);
    }

    header('Location: tarefas.php');
    die();
}
}

```

```
include "template.php";
```

O que mudou no começo do arquivo foi a adição de todas as classes que precisamos logo no começo do arquivo usando o `require`. Dentro do bloco da validação, estamos usando o objeto da classe `$Tarefa` no lugar do array, e estamos passando apenas este mesmo objeto para a nossa função `enviar_email()`.

Já no final do arquivo, todo o trecho de preenchimento do array de tarefas foi removido. Isso porque, caso a validação não tenha passado o objeto `$tarefa`, já terá todos os dados que o template precisa. Como o template e os demais arquivos já usam o objeto `$tarefa`, você já pode fazer a edição delas neste momento.

Finalmente, vamos ao último trecho que ainda precisa de alguma alteração: a remoção de tarefas e anexos. Estes trechos são bastante simples e usam apenas um arquivo cada um, sendo que estes são `remover.php` e `remover_anexo.php`. Veja como eles devem ficar:

```
<?php

// arquivo remover.php

require "config.php";
require "banco.php";
require "classe/RepositorioTarefas.php";

$repositorio_tarefas = new RepositorioTarefas($mysqli);
$repositorio_tarefas->remover($_GET['id']);

header('Location: tarefas.php');

<?php

// arquivo remover_anexo.php

require "config.php";
```

```
require "banco.php";
require "classe/Anexo.php";
require "classe/RepositorioTarefas.php";

$repositorio_tarefas = new RepositorioTarefas($mysqli);
$anexo = $repositorio_tarefas->buscar_anexo($_GET['id']);
$repositorio_tarefas->remover_anexo($anexo->getId());
unlink('anexos/' . $anexo->getArquivo());

header('Location: tarefa.php?id=' . $anexo->getTarefaId());
```

O arquivo `remover.php` é bastante direto e usa apenas o método `remover()` do repositório. Já o arquivo `remover_anexo.php` precisa primeiro recuperar os dados do anexo do banco de dados, pois esses dados são usados para fazer a remoção do arquivo e para fazer o redirecionamento para a página com os detalhes da tarefa.

Agora já é possível usar todas as opções da aplicação sem dependências com as funções que tratavam do acesso ao banco de dados. Alias, tente remover todas aquelas funções que buscam, salvam, atualizam e removem tarefas e anexos do banco de dados presentes no arquivo `banco.php`, e teste mais uma vez a aplicação. Tudo deve funcionar bem, pois transferimos as funções para os métodos da classe `RepositorioTarefas`.

14.7 RESUMO

Neste capítulo, foi apresentada uma pequena introdução à programação orientada a objetos. Não foi um capítulo simples, eu sei, mas finalmente estamos usando as novas classes em todo o sistema.

Não precisamos mais também dos arrays com as informações das tarefas e dos anexos. Com isso, conseguimos centralizar a

representação do que é uma tarefa e um anexo, e o nosso repositório é o ponto central para mudanças relacionadas ao banco de dados.

Orientação a Objetos é um tópico bastante extenso e vale a pena entender mais profundamente, pois grande parte dos projetos em PHP (e em outras linguagens) fazem uso dele para criar sistemas grandes e/ou pequenos, de forma mais organizada, facilitando a manutenção.

Se necessário, leia este capítulo novamente e tenha certeza de ter tudo funcionando e de que entendeu como as classes estão conectadas umas com as outras. Também recomendo bastante que você busque mais informações sobre Orientação a Objetos. O livro **Orientação a Objetos** da Casa do Código é uma boa recomendação, saiba mais em <https://www.casadocodigo.com.br/products/livro-oo-conceitos>.

No próximo capítulo, vamos trabalhar na segurança da nossa aplicação e a base será a aplicação já orientada a objetos. Aliás, o próximo capítulo nos mostrará como é mais fácil alterar a aplicação quando não deixamos, por exemplo, código que lida com o banco de dados espalhado por ela.

14.8 DESAFIOS

Agora mais alguns desafios, dessa vez para treinar Orientação a Objetos.

1. Crie uma classe `Contato` para representar os contatos no projeto dos contatos e transforme as funções de manipulação do banco de dados em métodos de uma classe chamada

`RepositorioContatos .`

2. No projeto do estacionamento, crie uma classe `Veiculo` que será a representação de um veículo estacionado. Crie também um repositório que será o responsável por manipular os dados dos veículos no banco de dados.
3. Tente separar a manipulação dos dados dos anexos em uma classe `RepositorioAnexos` . Mova os métodos que lidam com os anexos para esta classe e a utilize sempre que tratar de dados dos anexos.
4. Atualize a versão do nosso projeto das tarefas hospedada na Hostinger e no Jelastic da Locaweb para a nova versão orientada a objetos.

CAPÍTULO 15

PROTEGENDO A APLICAÇÃO CONTRA SQL INJECTION E XSS

Existem diversas formas de ataques a sites e aplicações web. Quando colocamos nosso código online, corremos o risco de receber algum tipo de ataque para quebrar a segurança de nossas aplicações e roubar informações. Ou mesmo para transformar nossas aplicações em robôs que enviam spam ou outros tipos de ameaças virtuais.

Um dos tipos mais comuns de ataque é o **SQL Injection**, uma técnica que pode ser até bastante simples de se executar em certos casos. Este tipo de ataque pode burlar o acesso ao banco de dados da aplicação e, em geral, consegue até mesmo apagar todos os dados.

Um outro ataque comum é o **Cross-site scripting**, ou **XSS**. Ele permite adicionar código que será executado na aplicação, mas apenas no lado do navegador.

Nossa aplicação das tarefas está vulnerável a estes tipos de ataque, pois um usuário poderia digitar algumas instruções SQL em alguns campos e estas poderiam ser executadas pelo banco,

sendo também possível adicionar código que será executado pelo navegador. Além disso, nossa aplicação não sabe como tratar certos caracteres, como as aspas e os apóstrofos.

Vamos agora tratar dos problemas da nossa aplicação começando pelo SQL Injection.

15.1 PROTEÇÃO CONTRA SQL INJECTION

Experimente cadastrar uma tarefa chamada `'Star Wars'`, e você verá que ela não será cadastrada. Isso acontece pois montamos o código SQL usando o apóstrofo para delimitar nossos campos de texto e, quando colocamos mais um apóstrofo no nome da tarefa, o código SQL é gerado incorretamente.

Veja um pequeno exemplo do que é um código SQL que vai funcionar corretamente:

```
INSERT INTO tarefas (nome)
VALUES ("Estudar SQL Injection");
```

Esta instrução SQL vai inserir um registro em uma tabela chamada `tarefas` com o valor `"Estudar SQL Injection"` no campo `nome`. Mas, imagine agora que você quer cadastrar essa tarefa usando aspas para dar destaque ao assunto a ser estudado, algo como: `Estudar "SQL Injection"`. Neste caso, o código ficaria assim:

```
INSERT INTO tarefas (nome)
VALUES ("Estudar "SQL Injection""");
```

Para nós, humanos, até parece correto. Mas, para um computador, o que acontece é que o valor do campo `nome` será apenas `"Estudar "`, depois disso, como as aspas "fecham", ele

trata o restante como código SQL e tenta executar o trecho "SQL Injection" até a abertura de novas aspas para formar uma nova string. Claro que, neste caso, o MySQL simplesmente geraria um erro por não entender uma parte da instrução e rejeitaria o comando.

Agora, imagine que essa informação vem de um formulário em uma página web, assim como a nossa aplicação de tarefas. E imagine que alguém vai preencher o campo de descrição não apenas usando aspas, mas também enviando código SQL válido para ser executado, algo como: `DELETE FROM tarefas WHERE id > 0`.

Isso certamente seria um grande problema, pois todas as nossas tarefas seriam apagadas! E essa técnica simples de usar, adicionando as aspas para finalizar campos de texto e então adicionar código SQL válido, é chamada de **SQL Injection**.

Certo, mas, como se proteger deste tipo de problema? Ou melhor, como cadastrar dados com aspas? Uma forma simples é usando a barra invertida para "escapar" as aspas. Desta forma, o MySQL vai entender que queremos usar aspas de verdade ali, e não as aspas de início e fim de uma string. Veja como ficaria o exemplo anterior:

```
INSERT INTO tarefas (nome)
VALUES ("Estudar \"SQL Injection\"");
```

Assim, o MySQL saberá que aquelas aspas ao redor de "SQL Injection" são aspas mesmo.

Existem formas simples de se proteger contra isso, como usando no PHP funções como `addslashes()` e

`addcslashes()` , que adicionam barras antes de aspas e apóstrofos. O problema é que SQL Injection não para simplesmente no uso de aspas, e outras técnicas podem ser usadas para quebrar uma aplicação. Por isso, a classe MySQLi contém métodos próprios para melhorar a segurança no caso de escapar strings.

15.2 PROTEGENDO-SE CONTRA SQL INJECTION USANDO MYSQLI

Para proteger nossa aplicação, precisamos escapar os caracteres que podem ser usados para compor código SQL, como é o caso de aspas e apóstrofos. Para escapar esses caracteres, devemos usar a função `mysqli_real_escape_string()` , ou o método `escape_string()` da classe `mysqli` . Eles são na verdade o mesmo método com nomes diferentes para a versão orientada a objetos e a versão procedural.

Veja como ficaria, por exemplo, o método `salvar()` da classe `RepositorioTarefas` :

```
<?php

class Tarefas
{
    // ...
    public function salvar(Tarefa $tarefa)
    {
        $nome = $this->conexao->escape_string(
            $tarefa->getNome()
        );
        $descricao = $this->conexao->escape_string(
            $tarefa->getDescricao()
        );
        $prioridade = $tarefa->getPrioridade();
        $prazo = $tarefa->getPrazo();
```

```

$concluida = ($tarefa->getConcluida()) ? 1 : 0;

if (is_object($prazo)) {
    $prazo = $prazo->format('Y-m-d');
}

$sqlGravar = "
    INSERT INTO tarefas
    (nome, descricao, prioridade, prazo, concluida)
    VALUES
    (
        '{$nome}',
        '{$descricao}',
        {$prioridade},
        '{$prazo}',
        {$concluida}
    )
";
$this->conexao->query($sqlGravar);
}

// ...

```

Perceba que os dados que contêm textos foram colocados em variáveis que são os retornos do método `escape_string()` do objeto `$this->conexao`, que é uma conexão com o MySQL usando a classe `mysqli`.

Escapar um texto é algo mais ou menos assim: o texto era isso `Assistir 'Star Wars'`, e virou isso `Assistir \'Star Wars\'`. Dessa forma, o MySQL trata o apóstrofo como um apóstrofo mesmo, e não como um delimitador de campos de texto.

Lembre-se de aplicar a mesma técnica, usando o método `escape_string()` do objeto `$this->conexao` nos outros métodos da classe `RepositorioTarefas` que enviam dados no formato de strings para o banco:

```

<?php

class RepositorioTarefas
{
    // ...

    public function salvar(Tarefa $tarefa)
    {
        $nome = $this->conexao->escape_string(
            $tarefa->getNome()
        );
        $descricao = $this->conexao->escape_string(
            $tarefa->getDescricao()
        );

        // ...
    }

    public function atualizar(Tarefa $tarefa)
    {
        $id = $tarefa->getId();

        $nome = $this->conexao->escape_string(
            $tarefa->getNome()
        );
        $descricao = $this->conexao->escape_string(
            $tarefa->getDescricao()
        );

        // ...
    }

    private function buscar_tarefa($id)
    {
        $id = $this->conexao->escape_string($id);
        $sqlBusca = 'SELECT * FROM tarefas WHERE id = ' . $id;

        // ...
    }

    public function salvar_anexo(Anexo $anexo)
    {
        $nome = $this->conexao->escape_string(
            $anexo->getNome()
        );

```

```

$arquivo = $this->conexao->escape_string(
    $anexo->getArquivo()
);

$sqlGravar = "INSERT INTO anexos
    (tarefa_id, nome, arquivo)
VALUES
(
    {$anexo->getTarefaId()},
    '{$nome}',
    '{$arquivo}'
)
";
}

$this->conexao->query($sqlGravar);
}

public function buscar_anexos($tarefa_id)
{
    $tarefa_id = $this->conexao->escape_string($tarefa_id);

    $sqlBusca =
        "SELECT * FROM anexos WHERE tarefa_id = {$tarefa_id}"
;

// ...
}

public function buscar_anexo($anexo_id)
{
    $anexo_id = $this->conexao->escape_string($anexo_id);

    $sqlBusca = "SELECT * FROM anexos WHERE id = {$anexo_id}"
;
    $resultado = $this->conexao->query($sqlBusca);

    return $resultado->fetch_object('Anexo');
}

public function remover($id)
{
    $id = $this->conexao->escape_string($id);
    $sqlRemover = "DELETE FROM tarefas WHERE id = {$id}";

    $this->conexao->query($sqlRemover);
}

```

```
}

public function remover_anexo($id)
{
    $id = $this->conexao->escape_string($id);
    $sqlRemover = "DELETE FROM anexos WHERE id = {$id}";

    $this->conexao->query($sqlRemover);
}
}
```

Como temos toda a comunicação com o banco acontecendo em apenas uma classe, ficou simples de fazer as alterações necessárias.

15.3 EXIBINDO E EDITANDO CAMPOS COM ASPAS

Agora que já estamos fazendo o escape de nossos campos de texto para o banco de dados, um novo problema aparece. Experimente adicionar uma tarefa chamada `Assistir "Star Wars"`, e você verá que o cadastro acontece normalmente.

Agora tente editar esta tarefa. Você perceberá que o campo `nome` aparece apenas com o texto `Assistir`. Isso está acontecendo pelo fato de o HTML gerado ser assim:

```
<input type="text" name="nome" value="Assistir "Star Wars"" />
```

Veja que as aspas antes da palavra `Star` estão finalizando o atributo `value` da tag `input`. Por isso, o restante do texto não aparece dentro do campo.

Para resolver este problema, precisamos transformar as aspas em caracteres especiais de aspas que o HTML entende. Isso não é a mesma coisa que o escape de strings, mas segue uma ideia

semelhante para não deixar que templates HTML quebrem.

A representação de aspas em HTML é o texto " . Em PHP, existe uma função que traduz todos os caracteres especiais do HTML em suas representações HTML. Esta função é a `htmlentities()`, e ainda neste capítulo tem uma explicação mais detalhada sobre ela e sobre as entidades HTML.

Por hora, veja como fica para exibirmos corretamente o nome das tarefas no formulário que está no arquivo `formulario.php`:

```
<input type="text" name="nome" value="<?php  
echo htmlentities($tarefa->getNome());  
?" />
```

Após esta alteração, o código HTML do campo será gerado assim:

```
<input type="text" name="nome"  
value="Assistir "Star Wars"" />
```

Veja que as aspas foram alteradas para " . O mesmo pode e deve ser feito com o campo descrição, pois o usuário também pode adicionar caracteres que quebram a aplicação nele:

Descrição (Opcional):
<textarea name="descricao"><?php
echo htmlentities(\$tarefa->getDescricao());
?></textarea>

Após esta alteração, os campos de nome e descrição da tarefa podem usar aspas e outros caracteres especiais do HTML.

15.4 PROTEÇÃO CONTRA XSS

Até este momento, já vimos PHP, HTML, SQL e até mesmo

um pouco de CSS neste livro. Mas ainda existe mais uma linguagem que é muito importante para o desenvolvimento para a web, o **JavaScript**.

O JavaScript começou como uma linguagem usada apenas no navegador para executar código na página já renderizada, após a resposta do servidor. Com ele, é possível fazer, por exemplo, validação de formulários enquanto o usuário digita as informações; exibir imagens em janelas dentro da página; adicionar animações em elementos HTML; e diversas outras interações com o usuário sem depender necessariamente de uma requisição ao servidor.

Hoje em dia, JavaScript também pode ser usado no lado servidor, fazendo algo parecido com o que estamos fazendo com o PHP até aqui. Ele é uma linguagem obrigatória para desenvolvimento web, então fica aqui a minha recomendação para que você estude JavaScript também.

Para adicionar JavaScript em uma página, basta usar a tag `<script>` do HTML. Veja um exemplo simples que exibe uma caixa de mensagem:

```
<script>
    alert("Olá, Javascript!");
</script>
```

Experimente colocar este código no final do arquivo `template.php`, antes da tag de fechamento do HTML. Perceba que, a partir deste ponto, você vai ver a mensagem toda vez que carregar a página.

Certo, agora remova o código JavaScript e vamos analisar o problema na segurança da nossa aplicação.

O campo descrição pode conter textos de qualquer tamanho, pois é um campo do tipo `TEXT` no MySQL. Seu valor é exibido na tabela com a lista de tarefas e também na página com os detalhes da tarefa. Agora, o que acontece se adicionarmos um pouco de JavaScript neste campo? Veja o que eu fiz:

Nova tarefa

Tarefa:

olá

Descrição (Opcional):

Mágica
<script>alert('Olá, Javascript!');</script>

Prazo (Opcional):

Figura 15.1: Adicionando JavaScript na descrição

Na descrição da tarefa, foi adicionado um pouco de JavaScript. E veja o que aconteceu depois que a tarefa foi salva:

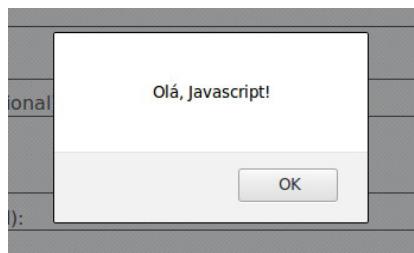


Figura 15.2: XSS em ação

Neste caso, usamos XSS apenas para exibir uma mensagem, mas um atacante poderia usar esta técnica para redirecionar o usuário para outros sites, roubar informações de sessão, manipular a página e exibir outras informações etc.

Para proteger a nossa aplicação contra XSS, existem duas opções simples. A primeira é tratando a exibição dos dados para que o JavaScript não seja executado, e a segunda é removendo qualquer tipo de código HTML e JavaScript antes de salvar as informações no banco de dados. Vamos ver como aplicar as duas formas de proteção separadamente e também em conjunto.

Vamos tratar primeiro da proteção na exibição dos dados. O grande problema aqui é que as tags HTML serão interpretadas pelo navegador, e o código será executado. Para prevenirmos esse tipo de ataque, precisamos novamente trocar os caracteres delimitadores das tags HTML por suas entidades em HTML, assim como fizemos no formulário de edição das tarefas.

ENTIDADES HTML

Entidades HTML são códigos para se exibir determinados caracteres em HTML. O caractere ¢, por exemplo, pode ser representado por ç.

Existem entidades HTML para representar os caracteres especiais do HTML como os sinais de menor e maior (< e >), que podem ser representados por < e > , respectivamente.

Estes caracteres são usados, por exemplo, em momentos nos quais queremos exibir coisas como 2 < 4 , e queremos evitar que o navegador entenda que estamos abrindo uma tag. Este exemplo ficaria assim: 2 < 4 .

E é claro que estes códigos também podem ser usados para fazer uma página explicando HTML. Tente colocar esta explicação em uma página HTML: Usamos a tag
 para pular uma linha .

O resultado na página vai ser a frase dividida em duas linhas, mas sem exibir a tag que queríamos exibir. Sendo assim, para exibir o exemplo corretamente em uma página HTML, o texto precisaria ser assim: Usamos a tag
 para pular uma linha .

O PHP tem uma função usada para converter os caracteres especiais do HTML em entidades HTML, a `htmlentities()` , já usada anteriormente. Altere o arquivo `template_tarefa.php` , na

linha onde a descrição da tarefa é exibida, para também usar esta função:

```
<p>
    <strong>Descrição:</strong>
    <?php echo nl2br(htmlentities($tarefa->getDescricao())); ?>
</p>
```

A função `nl2br()` já estava sendo usada antes, e a função `htmlentities()` foi adicionada para converter os caracteres especiais do HTML em entidades HTML, antes de adicionar as quebras de linha. Ao exibir os dados da tarefa, você deverá ver as tags HTML sendo exibidas em vez de executadas como JavaScript na descrição da tarefa:

[Voltar para a lista de tarefas.](#)

Concluída: Não

Descrição: Mágica

```
<script>alert('Olá, Javascript!');</script>
```

Figura 15.3: Exibindo o HTML sem executar o Javascript

Faça o mesmo para a exibição do campo nome no arquivo `template_tarefa.php` e também para os campos nome e descrição na tabela que exibe a lista de tarefas no arquivo `tabela.php`:

```
<td>
    <a href="tarefa.php?id=<?php echo $tarefa->getId(); ?>">
        <?php echo htmlentities($tarefa->getNome()); ?>
    </a>
</td>
<td>
    <?php echo htmlentities($tarefa->getDescricao()); ?>
</td>
```

Altere também no arquivo `template_tarefa.php` a exibição do nome dos anexos:

```
<td><?php echo htmlentities($anexo->getNome()); ?></td>
```

Agora temos as saídas dos campos com textos devidamente tratadas e protegidas, tanto para exibição quanto para edição dos dados.

A outra forma de proteger a aplicação contra XSS é tratar a entrada de dados e remover possíveis códigos HTML enviados pelos usuários. Isto é feito usando a função `strip_tags()`, que também remove tags PHP de um texto.

Para adicionar mais esta camada de proteção, altere o arquivo `RepositorioTarefas.php` adicionando a função `strip_tags()` na gravação e atualização de tarefas e na gravação dos anexos:

```
<?php

class RepositorioTarefas
{
    // ...

    public function salvar(Tarefa $tarefa)
    {
        $nome = strip_tags($this->conexao->escape_string(
            $tarefa->getNome()
        ));
        $descricao = strip_tags($this->conexao->escape_string(
            $tarefa->getDescricao()
        ));

        // ...
    }

    public function atualizar(Tarefa $tarefa)
    {
        $id = $tarefa->getId();

        $nome = strip_tags($this->conexao->escape_string(
            $tarefa->getNome()
        
```

```

    );
    $descricao = strip_tags($this->conexao->escape_string(
        $tarefa->getDescricao()
    ));

    // ...
}

public function salvar_anexo(Anexo $anexo)
{
    $nome = strip_tags($this->conexao->escape_string(
        $anexo->getNome()
    ));
    $arquivo = strip_tags($this->conexao->escape_string(
        $anexo->getArquivo()
    ));

    $sqlGravar = "INSERT INTO anexos
        (tarefa_id, nome, arquivo)
    VALUES
        (
            {$anexo->getTarefaId()},
            '{$nome}',
            '{$arquivo}'
        )
    ";

    $this->conexao->query($sqlGravar);
}

// ...
}

```

Após adicionar a função `strip_tags()`, tente adicionar ou editar uma tarefa usando tags HTML na descrição. Veja um exemplo usando a tarefa do exemplo anterior:

[Voltar para a lista de tarefas.](#)

Concluída: Não

Descrição: Mágica
`alert('Olá, Javascript!');`

Figura 15.4: Tags HTML removidas da descrição

Perceba que agora as tags não estão mais sendo exibidas, pois foram removidas na hora da gravação dos dados no banco de dados.

PRECISO MESMO TRATAR A ENTRADA E A SAÍDA DOS DADOS?

Com o uso da função `strip_tags()`, a função `htmlentities()` ficou meio que sem ter o que fazer, certo? Mais ou menos, pois a `htmlentities()` ainda vai tratar casos onde não temos código HTML, mas temos caracteres HTML, como o caso de exibir `2 < 4`, por exemplo. Inclusive, também tratará os casos em que temos aspas e/ou apóstrofos nos textos.

Além disso, se alguém encontrar uma falha de segurança na função `strip_tags()`, a nossa aplicação ainda terá uma camada de segurança contra XSS. Proteção nunca é demais. ;)

15.5 RESUMO

Este foi um capítulo extremamente importante! Nele, foram exibidos dois conceitos muito importantes do desenvolvimento web para ajudar na proteção da aplicação contra invasões: o tratamento contra **SQL Injection** e contra **Cross-site scripting** (ou **XSS**).

Proteger uma aplicação é um trabalho sem fim, pois, com o tempo, novas técnicas de invasão e injeção de dados são desenvolvidades. Assim, torna-se necessário manter a aplicação

sempre atualizada.

Uma dica importante é lembrar-se de sempre manter a segurança como um dos pontos principais durante o desenvolvimento e a manutenção de aplicações web. Acreditar que a sua aplicação é pequena e que nenhum malfeitor terá interesse em seus dados para tentar algum tipo de ataque é um pensamento bastante inocente e que deve ser evitado.

Por isso, fique de olho em notícias, blogs relacionados à segurança e nas atualizações do PHP, MySQL e demais softwares sendo utilizados em sua aplicação, para não deixar brechas na segurança de suas aplicações.

15.6 DESAFIOS

E aqui estão mais alguns desafios.

1. Faça o tratamento contra SQL Injection e XSS nos projetos dos contatos e do estacionamento.
2. Atualize a versão do nosso projeto das tarefas hospedada na Hostinger e no Jelastic da Locaweb para a nova versão com proteção contra XSS e SQL Injection.

CONHECENDO O PDO

A conexão e manipulação do banco de dados em nossa aplicação foi feita inicialmente usando a interface procedural do MySQLi, utilizando funções como `mysqli_query()` e `mysqli_fetch_assoc()`. Depois mudamos para a interface orientada a objetos, usando métodos como `$conexao->query()` e `$resultado->fetch_object()`.

Neste capítulo, vamos conhecer o **PDO**, ou **PHP Data Objects**. Ele é uma extensão do PHP utilizada para acessar diversos bancos de dados de uma forma mais consistente e unificada.

O PDO oferece um recurso muito bacana que o MySQLi não oferece (os parâmetros nomeados, que veremos logo mais). Ainda tem a vantagem de poder ser utilizado com vários bancos de dados diferentes, sem necessidade de alteração do código PHP.

Aliás, muitas pessoas vão vender a ideia do PDO focando exatamente na possibilidade de se poder trocar de banco de dados com menos esforço e com menos (ou nenhuma) reescrita de código PHP. E isso é verdade.

Mas a realidade que é não se altera o banco de dados de uma aplicação da noite para o dia apenas mudando o nome do banco de dados. Infelizmente, o código SQL também pode funcionar apenas

em MySQL e MariaDB, ou apenas em outro banco.

Sendo assim, a vantagem do PDO para quem desenvolve PHP é poder usar uma interface consistente para interagir com diferentes bancos de dados. Vamos começar então mudando a conexão com o banco que é realizada no arquivo `banco.php` para criar um objeto da classe `PDO`.

Mas antes veja a diferença entre MySQLi e PDO na hora de fazer a conexão com o banco:

```
<?php  
  
// conexão com MySQLi  
$mysqli = mysqli_connect(SERVIDOR, USUARIO, SENHA, BANCO);  
  
// conexão com PDO  
$pdo = new PDO(DSN, USUARIO, SENHA);
```

Neste exemplo, as constantes com os dados de conexão devem ser definidas previamente, assim como fizemos com as constantes que definimos no arquivo `config.php`. Repare que, para conectar ao banco de dados usando o PDO, é necessário ter uma string que indica o `DSN` (*Data Source Name*) ou *Nome da fonte de dados*. O DSN para conectar ao MySQL segue o seguinte padrão:
`mysql:dbname=nome_do_banco;host=servidor`

Um DSN do PDO é composto pelo tipo do banco que será utilizado. Ou seja, precisa conter o *driver* do PDO, que no nosso caso é `mysql`, o nome do banco (`dbname`) e o nome ou IP do servidor (`host`).

Antes de finalmente mudar a conexão no arquivo `banco.php`, vamos mudar o arquivo `config.php` adicionando o DSN para o PDO:

```
<?php

// Acesso ao banco de dados
define("BD_USUARIO", "sistematarefa");
define("BD_SENHA", "sistema");
define("BD_DSN", "mysql:dbname=tarefas;host=localhost");

// ...
```

Neste exemplo, foram removidas as constantes com o nome do servidor (`BD_SERVIDOR`) e com o nome do banco de dados (`BD_BANCO`), pois elas não serão mais usadas, já que essas informações devem ficar agora na configuração do DNS. Por isso foi adicionada a constante `BD_DSN` que contém, além do driver, o nome do banco e o host do servidor.

Agora vamos finalmente alterar o arquivo `banco.php` para realizar a conexão usando o PDO. Veja como deve ficar este arquivo:

```
<?php

try {
    $pdo = new PDO(BD_DSN, BD_USUARIO, BD_SENHA);
} catch (PDOException $e) {
    echo "Falha na conexão com o banco de dados: "
        . $e->getMessage();
    die();
}
```

Lembre-se de que, no capítulo anterior, movemos todas as funções para acesso ao banco para a classe `RepositorioTarefas`. Por isso, esse deve ser todo o conteúdo do arquivo `banco.php`.

O arquivo ficou um pouco diferente de antes, mas ele faz a mesma coisa que a versão usando a função `mysqli_connect()`. A diferença é que é função `mysqli_connect()` não gera uma exceção no momento da conexão com o banco. Logo, precisamos

usar a função `mysqli_connect_errno()` para descobrir se algum erro aconteceu no momento de conectar ao banco. Quando criamos um novo objeto da classe `PDO` e um erro como uma senha incorreta acontece, uma exceção será gerada.

16.1 EXCEPTIONS, TRY E CATCH

Exceções são interrupções no fluxo da aplicação que servem para parar o processamento e avisar que algo não funcionou da forma esperada. Esta é uma maneira mais elegante de se tratar erros do que usando instruções `if`, por exemplo. Isto também permite que os erros sejam capturados e tratados não apenas por quem desenvolve uma classe ou uma biblioteca, mas também por quem os utiliza.

No PHP, as exceções são objetos da classe `Exception` e suas derivadas. Um exemplo é a `PDOException`, usada para pegar a exceção na hora de criar o objeto com a conexão ao MySQL.

Vamos fazer um exemplo de como lançar uma exceção em PHP, assim fica mais fácil de entender o seu funcionamento. Crie um arquivo chamado `ex.php` no diretório `htdocs`, ou no diretório raiz do seu Apache, com o seguinte conteúdo:

```
<?php  
  
echo "<p>Iniciando o script</p>";  
  
throw new Exception("Algo deu errado");  
  
echo "<p>Finalizando o script</p>";
```

Reparou que usei o verbo "lançar" para falar das exceções? Quando trabalhamos com exceções, esta é a forma de dizer que

criamos uma exceção. Na verdade, dois verbos são muito usados para lidar com as exceções: um deles é o "lançar", o outro você vai descobrir nos próximos parágrafos.

Agora acesse o arquivo do exemplo digitando no navegador o endereço `http://localhost/ex.php`. Você deverá ver uma mensagem parecida com a seguinte: *Fatal error: Uncaught exception 'Exception' with message 'Algo deu errado' in ex.php:3*

O PHP está nos alertando de um erro que aconteceu por conta de não termos "capturado" a exceção. E olha aí o outro verbo que usamos para falar das exceções: "capturar".

Quando trabalhando com exceções, lembre-se de que elas devem ser lançadas e capturadas para tratamento. Quando uma exceção não é capturada dentro de nossas aplicações, ela acaba sendo capturada e tratada pelo PHP.

Aliás, foi exatamente isso que aconteceu. O PHP capturou a exceção que lancamos e exibiu suas informações na forma de um erro fatal.

Outro detalhe importante é que o PHP vai parar a execução do script quando uma exceção não capturada for lançada. Por isso não conseguimos ver a frase "Finalizando o script".

Seguindo o exemplo no arquivo `ex.php`, vamos adicionar a captura da exceção para que o PHP não exiba o erro e o script seja finalizado:

```
<?php  
echo "<p>Iniciando o script</p>";  
try {
```

```

echo "<p>Você vai ler este parágrafo</p>";
throw new Exception("Algo deu errado");
echo "<p>Mas não vai ler este parágrafo</p>";
} catch (Exception $e) {
    echo "Exceção capturada com a seguinte mensagem: "
        . $e->getMessage();
}
echo "<p>Finalizando o script</p>";

```

Chamamos esta construção de *Bloco Try/Catch*, pois tentamos executar algo dentro das chaves do `try`. Quando uma exceção é lançada, a capturamos no bloco `catch`, atribuindo seu valor a uma variável, no caso a `$e`.

Quando lançamos uma exceção, criamos um objeto da classe `Exception`. O construtor desta classe recebe como parâmetro uma string com um texto descrevendo o problema ocorrido. Esta string pode ser acessada usando o método `getMessage()` do objeto, indicado no bloco `catch`.

Boas mensagens de erro são de grande ajuda para quem usa a sua classe, por isso sempre que lançar exceções, tente escrever mensagens que descrevem bem o problema e de forma clara. Evite, por exemplo, mensagens do tipo *Um erro aconteceu: Código 853*, pois isso forçaria a pessoa usando o seu código a buscar em um manual ou mesmo na internet o que é o erro 853.

16.2 SUBSTITUÍNDΟ O MYSQLI PELO PDO

Depois da introdução aos blocos `try/catch`, fica mais fácil de entender o que está acontecendo no arquivo `banco.php`. Veja novamente o código:

```
<?php

try {
    $pdo = new PDO(BD_DSN, BD_USUARIO, BD_SENHA);
} catch (PDOException $e) {
    echo "Falha na conexão com o banco de dados: "
        . $e->getMessage();
    die();
}
```

Neste código, o método construtor da classe PDO vai tentar fazer a conexão com o banco de dados usando os dados fornecidos dentro do bloco try . Caso a conexão falhe, o bloco catch capturará a exceção e usará o método getMessage() do objeto \$e para nos informar qual é a mensagem de erro.

Agora vamos atualizar a nossa lista de tarefas, exibida quando acessamos o arquivo tarefas.php , para usar o **PDO**. Lendo este arquivo, vemos que a variável \$mysqli só é usada na hora de criar um objeto da classe RepositorioTarefas , logo no começo do arquivo.

Sendo assim, vamos alterar a criação do objeto da classe RepositorioTarefas para passar o novo objeto \$pdo para o método construtor:

```
<?php

// ...

repositorio_tarefas = new RepositorioTarefas($pdo);

// ...
```

Agora vamos alterar a classe RepositorioTarefas que está no arquivo classes/RepositorioTarefas.php . Vamos começar pelo método construtor, indicando que estamos usando um objeto

da classe PDO :

```
<?php

class RepositorioTarefas
{
    private $pdo;

    public function __construct(PDO $pdo)
    {
        $this->pdo = $pdo;
    }

    // ...
}
```

Perceba que o nome da variável privada foi alterado de `$conexao` para `$pdo`. Agora, o construtor tem a indicação de tipo (*type hinting*), dizendo que o método espera receber um objeto da classe PDO.

Se ignorarmos o bloco do `if`, que verifica se dados foram enviados via POST, vemos que o único método da classe `RepositorioTarefas` utilizado no arquivo `tarefas.php` é o `buscar()`. Por isso, vamos alterar apenas este método para já termos a lista de tarefas funcional usando o PDO.

Mas lembre-se de que o método `buscar()` usa o método privado `buscar_tarefas()`. Sendo assim, este é o método que precisamos alterar:

```
<?php

// arquivo classes/RepositorioTarefas.php

// ...

private function buscar_tarefas()
{
```

```

$sqlBusca = 'SELECT * FROM tarefas';

$resultado = $this->pdo->query(
    $sqlBusca,
    PDO::FETCH_CLASS,
    'Tarefa'
);

$tarefas = [];

foreach ($resultado as $tarefa) {
    $tarefa->setAnexos($this->buscar_anexos(
        $tarefa->getId()
    ));
    $tarefas[] = $tarefa;
}

return $tarefas;
}

// ...

```

A grande diferença no uso do PDO para o MySQLi é que já informamos no método `query()` qual é a classe que queremos usar para criar os objetos do resultado. Para isso, usamos a opção `PDO::FETCH_CLASS` e o nome da classe como argumentos do método `query()`.

Neste ponto, ainda não podemos executar nosso código sem receber alguns erros, pois o método `buscar_anexos()` também precisa ser atualizado para usar o PDO.

16.3 PREPARED STATEMENTS

A atualização do método `buscar_anexos()` será um pouco diferente, pois precisamos passar um valor adicional para a `query` com o id da tarefa da qual queremos os anexos:

```
<?php

// arquivo classes/RepositoryTarefas.php

// ...

public function buscar_anexos($tarefa_id)
{
    $sqlBusca = "SELECT * FROM anexos
        WHERE tarefa_id = :tarefa_id";
    $query = $this->pdo->prepare($sqlBusca);
    $query->execute([
        "tarefa_id" => $tarefa_id,
    ]);

    $anexos = array();

    while ($anexo = $query->fetchObject('Anexo')) {
        $anexos[] = $anexo;
    }

    return $anexos;
}

// ...
```

Agora temos mais diferenças com relação ao que fizemos com o MySQLi. Repare que, no código SQL da variável `$sqlBusca`, não colocamos o `id` da tarefa, mas sim a string `:tarefa_id`, assim mesmo, precedida pelos dois pontos.

Depois disso, usamos o método `prepare()` do PDO utilizando o código SQL que ainda está "incompleto", pois não tem o `id` da tarefa. O resultado do método `prepare()` é guardado na variável `$query`.

Finalmente, usamos o método `execute()` na variável `$query`, passando como parâmetro um array. Nele, o único valor tem a chave `tarefa_id`, que é a mesma string que colocamos

dentro do código SQL no lugar do número com o id da tarefa e precedida de dois pontos.

Nossa, o que foi que aconteceu? Simples, usamos um recurso conhecido como *Prepared Statements*.

As *Prepared Statements* (ou *Declarações Preparadas*) são uma forma de se preparar código SQL para ser executado. As vantagens das Prepared Statements são poder preparar o código SQL apenas uma vez e executar diversas vezes com parâmetros diferentes. Além disso, possui uma maior segurança, pois não é necessário fazer o "escape" dos parâmetros, já que o PDO (ou o banco de dados) se encarrega desta tarefa, prevenindo assim problemas de SQL Injection.

No nosso caso, temos o seguinte código SQL:

```
SELECT * FROM anexos WHERE tarefa_id = :tarefa_id
```

Este código vai ser preparado para a execução e ele sabe que, no lugar da string `:tarefa_id`, um outro valor será usado. Então, ele faz algo como deixar um espaço em branco que será preenchido no momento da execução.

Quando usamos o método `execute()`, temos de preencher este espaço em branco passando um array no qual os índices devem ser equivalentes aos nomes usados no código SQL, que neste caso é `tarefa_id`, mas sem os depois pontos.

O resultado será armazenado dentro da própria variável `$query`, que é um objeto da classe `PDOStatement`. Logo em seguida, usamos o método `fetchObject()` para colocar cada um dos resultados em um objeto da classe `Anexo`.

APENAS O PDO TEM PREPARED STATEMENTS?

Também é possível usar as Prepared Statements com o MySQLi. A diferença é que, no MySQLi, não é possível dar nomes aos parâmetros, por isso usamos interrogações para indicar os espaços a serem preenchidos durante a execução.

Um código SQL usando as Prepared Statements no MySQLi fica assim:

```
SELECT * FROM anexos WHERE tarefa_id = ?
```

Isso não é ruim quando se tem apenas um ou dois argumentos, mas pode ficar confuso com muitos parâmetros. Veja este exemplo usando três parâmetros:

```
SELECT * FROM tarefas  
WHERE prazo < ? AND prioridade = ? AND concluida = ?
```

Agora a versão com os parâmetros nomeados:

```
SELECT * FROM tarefas  
WHERE prazo < :prazo AND prioridade = :prioridade  
AND concluida = :concluida
```

Bem mais claro de entender, né?

Após alterar os métodos `buscar_tarefas()` e `buscar_anexos()`, atualize a página com a lista de tarefas (acessando o arquivo `tarefas.php`). Você deverá ver a lista de tarefas. Mas não tente cadastrar uma tarefa ou ver a página com os detalhes de uma tarefa, pois o restante da aplicação ainda espera um objeto MySQLi para fazer a comunicação com o banco.

16.4 USANDO O PDO NO RESTANTE DA APLICAÇÃO

Após alterar uma pequena parte da aplicação para usar o PDO, podemos alterar o restante de maneira bem simples. Vamos começar alterando todos os pontos de entrada da aplicação para criar o objeto da classe `RepositorioTarefas`, passando o objeto `$pdo` para o construtor:

```
<?php

// Altere os seguintes arquivos:
// - editar.php
// - remover.php
// - remover_anexo.php
// - tarefa.php

// Esta é a criação do objeto $repositorio_tarefas em todos
// os arquivos:

$repositorio_tarefas = new RepositorioTarefas($pdo);
```

Não foi difícil, pois temos apenas cinco arquivos que servem como pontos de entrada na aplicação. Porém, se fossem mais, esta seria uma tarefa mais complicada. Por isso, no próximo capítulo veremos como ter apenas um arquivo como ponto de entrada da aplicação.

O próximo passo é alterar todos os métodos da classe `RepositorioTarefas` para usarem o PDO. Como já fizemos esta mudança em dois métodos — uma usando apenas SQL sem parâmetros (`buscar_tarefas()`), e outra usando Prepared Statements (`buscar_anexos()`) —, vou apenas colocar o código da classe toda a seguir e você pode adaptar todos os métodos de acordo.

Este é o novo arquivo `classes/RepositorioTarefas.php` :

```
<?php

class RepositorioTarefas
{
    private $pdo;

    public function __construct(PDO $pdo)
    {
        $this->pdo = $pdo;
    }

    public function salvar(Tarefa $tarefa)
    {
        $prazo = $tarefa->getPrazo();

        if (is_object($prazo)) {
            $prazo = $prazo->format('Y-m-d');
        }

        // Definindo SQL com Prepared Statements
        $sqlGravar = "
            INSERT INTO tarefas
            (nome, descricao, prioridade, prazo, concluida)
            VALUES
            (:nome, :descricao, :prioridade, :prazo, :concluida)
        ";

        // Preparando a query
        $query = $this->pdo->prepare($sqlGravar);

        // Executando a query com os parâmetros nomeados
        $query->execute([
            'nome' => strip_tags($tarefa->getNome()),
            'descricao' => strip_tags($tarefa->getDescricao()),
            'prioridade' => $tarefa->getPrioridade(),
            'prazo' => $prazo,
            'concluida' => ($tarefa->getConcluida()) ? 1 : 0,
        ]);
    }

    public function atualizar(Tarefa $tarefa)
    {
```

```

$prazo = $tarefa->getPrazo();

if (is_object($prazo)) {
    $prazo = $prazo->format('Y-m-d');
}

// Lembre-se de que no update precisamos do WHERE
$sqlEditar = "
    UPDATE tarefas SET
        nome = :nome,
        descricao = :descricao,
        prioridade = :prioridade,
        prazo = :prazo,
        concluida = :concluida
    WHERE id = :id
";

$query = $this->pdo->prepare($sqlEditar);

// O parâmetro do WHERE também é incluído na execução
$query->execute([
    'nome' => strip_tags($tarefa->getNome()),
    'descricao' => strip_tags($tarefa->getDescricao()),
    'prioridade' => $tarefa->getPrioridade(),
    'prazo' => $prazo,
    'concluida' => ($tarefa->getConcluida()) ? 1 : 0,
    'id' => $tarefa->getId(),
]);
}

public function buscar($tarefa_id = 0)
{
    if ($tarefa_id > 0) {
        return $this->buscar_tarefa($tarefa_id);
    } else {
        return $this->buscar_tarefas();
    }
}

private function buscar_tarefas()
{
    // Aqui não precisamos de parâmetro adicionais
    // Pois buscamos todas as tarefas

    $sqlBusca = 'SELECT * FROM tarefas';
}

```

```

$resultado = $this->pdo->query(
    $sqlBusca,
    PDO::FETCH_CLASS,
    'Tarefa'
);

$tarefas = [];

foreach ($resultado as $tarefa) {
    $tarefa->setAnexos($this->buscar_anexos(
        $tarefa->getId()
    ));
    $tarefas[] = $tarefa;
}

return $tarefas;
}

private function buscar_tarefa($id)
{
    $sqlBusca = "SELECT * FROM tarefas WHERE id = :id";
    $query = $this->pdo->prepare($sqlBusca);
    $query->execute([
        'id' => $id,
    ]);

    $tarefa = $query->fetchObject('Tarefa');

    // Delegamos a busca dos anexos
    // para o método buscar_anexos()

    $tarefa->setAnexos($this->buscar_anexos(
        $tarefa->getId()
    ));

    return $tarefa;
}

public function salvar_anexo(Anexo $anexo)
{
    $sqlGravar = "INSERT INTO anexos
        (tarefa_id, nome, arquivo)
        VALUES
        (:tarefa_id, :nome, :arquivo)"

```

```

        ";
$query = $this->pdo->prepare($sqlGravar);
$query->execute([
    'tarefa_id' => $anexo->getTarefaId(),
    'nome' => strip_tags($anexo->getNome()),
    'arquivo' => strip_tags($anexo->getArquivo()),
]);
}

public function buscar_anexos($tarefa_id)
{
    $sqlBusca =
        "SELECT * FROM anexos WHERE tarefa_id = :tarefa_id";

    $query = $this->pdo->prepare($sqlBusca);
    $query->execute([
        'tarefa_id' => $tarefa_id,
    ]);

    $anexos = array();

    while ($anexo = $query->fetchObject('Anexo')) {
        $anexos[] = $anexo;
    }

    return $anexos;
}

public function buscar_anexo($anexo_id)
{
    $sqlBusca = "SELECT * FROM anexos WHERE id = :id";
    $query = $this->pdo->prepare($sqlBusca);
    $query->execute([
        'id' => $anexo_id,
    ]);

    return $query->fetchObject('Anexo');
}

public function remover($id)
{
    // Na remoção é muito importante usar o WHERE

    $sqlRemover = "DELETE FROM tarefas WHERE id = :id";
}

```

```

        $query = $this->pdo->prepare($sqlRemover);

        $query->execute([
            'id' => $id,
        ]);
    }

    public function remover_anexo($id)
    {
        $sqlRemover = "DELETE FROM anexos WHERE id = :id";
        $query = $this->pdo->prepare($sqlRemover);
        $query->execute([
            'id' => $id,
        ]);
    }
}

```

Perceba que todos os códigos SQL em que precisamos adicionar valores de variáveis foram feitos usando as Prepared Statements. Graças ao uso desta técnica, não precisamos mais tratar as entradas de dados usando, por exemplo, o método `escape_string()`, como fizemos na versão com o MySQLi.

Mas um tratamento que ainda é bom manter é a remoção de tags HTML usando a função `strip_tags()`. Isso porque as Prepared Statements não vão resolver o problema de exibição de tags HTML e possível vulnerabilidade a XSS.

Como exemplo, vamos analisar o método `salvar()` na versão sem e com Prepared Statements. Primeiro a versão sem:

```

<?php

// ...
public function salvar(Tarefa $tarefa)
{
    $nome = strip_tags($this->conexao->escape_string(
        $tarefa->getNome()
    ));
}

```

```

$descricao = strip_tags($this->conexao->escape_string(
    $tarefa->getDescricao())
));
$prioridade = $tarefa->getPrioridade();
$prazo = $tarefa->getPrazo();
$concluida = ($tarefa->getConcluida()) ? 1 : 0;

if (is_object($prazo)) {
    $prazo = $prazo->format('Y-m-d');
}

$sqlGravar = "
    INSERT INTO tarefas
    (nome, descricao, prioridade, prazo, concluida)
    VALUES
    (
        '{$nome}',
        '{$descricao}',
        {$prioridade},
        '{$prazo}',
        {$concluida}
    )
";
}

$this->bd->query($sqlGravar);
}

```

E agora a versão com Prepared Statements:

```

<?php

public function salvar(Tarefa $tarefa)
{
    $prazo = $tarefa->getPrazo();
    if (is_object($prazo)) {
        $prazo = $prazo->format('Y-m-d');
    }

    $sqlGravar = "
        INSERT INTO tarefas
        (nome, descricao, prioridade, prazo, concluida)
        VALUES
        (:nome, :descricao, :prioridade, :prazo, :concluida)
    ";

```

```
$query = $this->pdo->prepare($sqlGravar);

$query->execute([
    'nome' => strip_tags($tarefa->getNome()),
    'descricao' => strip_tags($tarefa->getDescricao()),
    'prioridade' => $tarefa->getPrioridade(),
    'prazo' => $prazo,
    'concluida' => ($tarefa->getConcluida()) ? 1 : 0,
]);
}
```

O código ficou menor e mais fácil de ler. Perceba que, na chamada do método `execute()`, passamos todos os dados descritos na SQL usando as marcações precedidas de dois pontos. Perceba, mais uma vez, que a função `strip_tags()` continua ali, pois a sua função é remover as tags HTML, coisa que o PDO não vai fazer, já que tags HTML não são prejudiciais para o banco de dados.

16.5 RESUMO

O PDO é uma ferramenta importante no mundo do PHP. Quando bem utilizado, pode permitir a troca do banco de dados da aplicação de forma mais simples do que mudar uma implementação usando a biblioteca MySQLi. Para isso, basta fazer a alteração do DSN da conexão.

Claro que o código SQL pode variar de um banco para o outro, e ajustes sempre serão necessários. Porém, aplicações no mundo real dificilmente são migradas de um banco para o outro apenas mudando o nome da conexão.

Este capítulo também nos mostrou o poder da centralização de funcionalidades, pois conseguimos mudar uma parte central da nossa aplicação alterando apenas uma classe. Isso mesmo, neste

capítulo, não alteramos a tabela, não alteramos os formulários e não alteramos os templates. Nem mesmo as classes `Tarefa` e `Anexo` foram alteradas! Muito legal, né?

16.6 DESAFIOS

Hora de praticar fazendo alguns desafios nos outros projetos.

1. Atualize as aplicações dos contatos e do estacionamento para usar o PDO.
2. Atualize a versão do nosso projeto das tarefas hospedada na Hostinger e no Jelastic da Locaweb para a nova versão com uso do PDO. Não se esqueça de atualizar o DSN para usar as credenciais de conexão de ambos os serviços.

CAPÍTULO 17

INTRODUÇÃO AO MVC

MVC (Model-View-Controller) é um padrão de arquitetura de projetos criado por Trygve Reenskaug no final dos anos 1970. Ele tem como objetivo separar a aplicação em três camadas com diferentes responsabilidades.

Neste capítulo, vamos aprender sobre MVC e veremos também uma breve descrição sobre o que são os famosos **Frameworks**. Além disso, vamos atualizar a nossa aplicação para ficar mais próxima dos moldes de um projeto MVC. Este é um exercício bastante interessante para ser feito antes de começar a utilizar um framework, pois nos faz compreender melhor os problemas que um framework se propõe a resolver.

Este capítulo parece ser mais complicado do que os demais por conta das alterações que faremos na estrutura da nossa aplicação. Mas não se engane, na verdade ele é mais simples do que o capítulo sobre Orientação a Objetos. No final das contas, vamos apenas mover os arquivos para diretórios diferentes e alterar algumas lógicas aqui e ali.

Mas antes de começar a alterar alguma coisa na aplicação, vou explicar o que são as diferentes camadas do MVC:

- **Model:** controla o acesso aos dados e contém as regras

de negócio da aplicação.

- **View:** a representação dos dados em uma interface com o usuário ou com outros sistemas.
- **Controller:** recebe as requisições do usuário e faz uso da Model e da View.

As camadas do MVC têm suas responsabilidades definidas e não devem fugir de seu escopo. Por exemplo, uma View, responsável por apresentar os dados, não deve ir ao banco de dados para buscar informações ou para atualizar um registro.

O padrão MVC, assim como outros padrões de arquitetura de software, não descreve exatamente como cada camada deve ser implementada. Por isso, existem diversas visões sobre como estas implementações devem ser feitas e diversas discussões nas comunidades de desenvolvimento de software sobre o assunto.

Durante o desenvolvimento da nossa aplicação de gestão de tarefas, buscamos deixar as camadas da nossa aplicação bem isoladas. Também, de certa forma, tivemos um bom resultado até aqui, pois no capítulo sobre atualizar a aplicação para usar o PDO nem mesmo precisamos tocar nos templates.

Mesmo assim, ainda temos algumas coisas para melhorar. Olha só como os arquivos estão organizados na pasta tarefas:

- `ajudantes.php`
- `bibliotecas/`
- `config.php`
- `formulario.php`
- `remover.php`
- `tarefa.php`

- tarefas.php
- template.php
- anexos/
- banco.php
- classes/
- editar.php
- remover_anexo.php
- tabela.php
- tarefas.css
- template_email.php
- template_tarefa.php

Um pouco confuso, certo? Quais arquivos são acessados pelos usuários? O que acontece, por exemplo, se alguém acessar o arquivo template_email.php pelo navegador? Eu tentei acessar usando `http://localhost/tarefas/template_email.php` e veja o resultado:

Tarefa:

```
Notice: Undefined variable: tarefa in template_email.php on line 1
```

```
Fatal error: Call to a member function getName() on null in template_email.php on line 1
```

Temos vários arquivos na raiz da nossa aplicação que não devem ser acessados pelos usuários, como os templates, os ajudantes e o arquivo de configuração, pois esses arquivos devem apenas ser incluídos pelos arquivos que são acessados pelos usuários.

Temos também os arquivos que podem ser acessados e esses

são os pontos de entrada da aplicação.

Ter muitos arquivos como pontos de entrada na aplicação é, no geral, algo ruim, pois quando precisamos mudar algo (como incluir um arquivo de configuração), precisamos mudar vários arquivos. Tivemos um bom exemplo disso no capítulo anterior, quando precisamos alterar vários arquivos para mudar a conexão para o PDO.

Um dos desafios no capítulo *Edição e remoção de registros* é sobre unificar as entradas da aplicação em apenas um arquivo. Se você tentou fazer o desafio agora, você vai poder comparar com uma possível solução neste capítulo.

Lembre-se de que não existe implementação MVC ideal, mas que o conceito geral está presente nas diferentes implementações que você pode encontrar.

UMA DICA IMPORTANTE

Já que este capítulo tem mudanças na estrutura do projeto, recomendo que você faça um backup/cópia do projeto de gestão de tarefas até aqui. Assim, você poderá refazer este capítulo de forma simples, apenas recuperando do backup.

17.1 O FRONT-CONTROLLER E OS CONTROLLERS

Chamamos de **Front-Controller** o arquivo que centraliza

todos os pontos de entrada de uma aplicação. Em PHP, é bastante comum que este arquivo tenha o nome `index.php`. E até mesmos os servidores web, como o Apache, vão procurar por este arquivo por padrão quando um arquivo não for informado.

Na nossa aplicação, temos diversos arquivos recebendo as requisições do usuário. Isso, além de deixar o projeto um pouco bagunçado e mais complicado de entender onde estão os pontos de entrada, ainda nos obriga a alterar diversos arquivos quando mudamos algo como o repositório de tarefas.

Então, para centralizar o recebimento das requisições e a configuração inicial da aplicação, vamos adicionar um front-controller. Isso será feito através da criação de um arquivo chamado `index.php` que deve ser colocado no diretório raiz da aplicação, a pasta `tarefas`, pois os servidores web usam este arquivo como padrão.

Mas antes de criar o arquivo, vamos planejar a sua lógica. Atualmente, a nossa aplicação tem diversos pontos de entrada e podemos dizer que cada um desses pontos "cuida da sua própria vida". Isso porque cada um faz o `require` das configurações, do banco, dos ajudantes e das classes que representam as tarefas e os anexos.

O front-controller, por outro lado, vai tomar conta da vida de todos os pontos de entrada, inserindo os arquivos necessários e iniciando o repositório. O front-controller também precisa entender as requisições e mandar cada uma para o código que vai efetivamente tratar da requisição.

Esse processo é chamado de **roteamento**. E para que ele seja

possível, precisamos adicionar uma variável que controlará as rotas.

Para facilitar o entendimento do front-controller, vamos fazer uma lista com o passo a passo do que é necessário, assim como fizemos na função `enviar_email()`. Crie um arquivo chamado `index.php` no diretório `tarefas`, e digite a lista com o passo a passo:

```
<?php

// Incluir as configurações, ajudantes e classes
// Criar um objeto da classe RepositorioTarefas
// Verificar qual arquivo (rota) deve ser usado para
//     tratar a requisição
// Incluir o arquivo que vai tratar a requisição
```

Certo, com essa pequena lista, podemos preencher o arquivo com o código:

```
<?php

// Incluir as configurações, ajudantes e classes

require "config.php";
require "banco.php";
require "ajudantes.php";
require "classe/Tarefa.php";
require "classe/Anexo.php";
require "classe/RepositorioTarefas.php";

// Criar um objeto da classe RepositorioTarefas

$repositorio_tarefas = new RepositorioTarefas($pdo);

// Verificar qual arquivo (rota) deve ser usado para
//     tratar a requisição

$rota = "tarefas"; // Rota padrão

if (array_key_exists("rota", $_GET)) {
```

```
$rota = (string) $_GET["rota"];
}

// Incluir o arquivo que vai tratar a requisição

if (is_file("${rota}.php")) {
    require "${rota}.php";
} else {
    echo "Rota não encontrada";
}
```

Esse arquivo não é muito diferente dos outros que recebem as requisições que fizemos até aqui. A diferença maior está na definição da variável \$rota e no uso da função `is_file()`, que verifica se um arquivo existe.

Agora altere os arquivos `editar.php`, `remover_anexo.php`, `remover.php`, `tarefa.php` e `tarefas.php` removendo as linhas iniciais, ou seja, as linhas com as instruções `require` e a criação do objeto da classe `RepositorioTarefas`. O começo do arquivo `tarefas.php`, por exemplo, deve ficar assim:

```
<?php

$exibir_tabela = true;

$tem_erros = false;
$erros_validacao = [];

$tarefa = new Tarefa();
$tarefa->setPrioridade(1);

// ...
```

Após estas alterações, acesse no navegador o endereço `http://localhost/tarefas/`, assim mesmo, sem informar o nome do arquivo. Você deverá ver a lista de tarefas e o formulário que apareciam acessando o arquivo `tarefas.php`.

Isso aconteceu pois o servidor HTTP (ou seja, o Apache) recebeu a requisição para um diretório e usou o arquivo padrão, `index.php`. Dentro desse arquivo, verificamos se o parâmetro `rota` existe e definimos a variável `$rota` com o valor `tarefas`. Logo em seguida, verificamos se o arquivo `tarefas.php` existe e fazemos a sua inclusão usando o `require`. Interessante, não é?

Com essa estrutura do front-controller funcionando, podemos começar a organização dos **controllers**, que são os trechos de código responsáveis por receber as requisições. Na raiz do projeto, crie uma pasta nova chamada `controllers`, e move os arquivos `editar.php`, `remover_anexo.php`, `remover.php`, `tarefa.php` e `tarefas.php` para dentro dela.

Agora altere o arquivo `index.php` para fazer o `require` dos controllers dentro da nova pasta:

```
<?php  
// ...  
  
// Incluir o arquivo que vai tratar a requisição  
  
if (is_file("controllers/{$rota}.php")) {  
    require "controllers/{$rota}.php";  
} else {  
    echo "Rota não encontrada";  
}
```

Agora é preciso observar um detalhe importante no arquivo `controllers/tarefas.php`. A inclusão do template no final do arquivo não vai mais funcionar, pois o arquivo `template.php` não está mais no mesmo diretório. Por isso, precisamos atualizar o último `require` para que ele saiba o caminho desse arquivo:

```
<?php
```

```
// arquivo: controllers/tarefas.php  
// ...  
  
require __DIR__ . "/../template.php";
```

Agora o require deve usar a "constante mágica" `__DIR__`, que contém o caminho do diretório do arquivo atual. Usamos então os dois pontos para dizer que queremos subir um nível na hierarquia de diretórios, e finalmente indicamos o arquivo `template.php`.

Neste momento, você já pode acessar no navegador o endereço `http://localhost/tarefas/` para ver a lista de tarefas. Se você tentar clicar nos links ou cadastrar uma nova tarefa, vai receber erros do servidor Apache dizendo que o arquivo `tarefas.php` ou `tarefa.php` não existe. Isso acontece pois todos os links ainda apontam para os nomes dos arquivos, mas vamos corrigir isso logo mais.

Agora, pare um momento e se certifique de ter entendido tudo até aqui. Em geral, essa parte de lidar com arquivos sendo incluídos em pastas/diretórios diferentes pode ser um pouquinho complicado no começo. Mas não se deixe assustar, você já chegou até esse ponto no livro e faltam apenas alguns passos para chegar ao final. ;)

17.2 ORGANIZANDO AS VIEWS

Antes de continuar com as correções para que os links funcionem corretamente, vamos usar a página inicial da nossa aplicação para fazer mais um ajuste em direção a uma estrutura MVC melhor organizada.

Na raiz da nossa aplicação, ainda temos todos os arquivos usados para gerar o HTML que será enviado para o navegador exibir. Esses arquivos são as nossas **views**, e vamos organizá-los da mesma forma que fizemos com os controllers.

Crie um novo diretório chamado `views` na raiz do projeto e mova os arquivos `formulario.php`, `tabela.php`, `template.php`, `template_email.php`, `template_tarefa.php` para lá. Após mover os arquivos, experimente acessar a aplicação.

Você verá um erro dizendo que o arquivo `template.php` não foi encontrado. Para corrigir o problema, mude o arquivo `controllers/tarefas.php` alterando o caminho usado no `require`:

```
<?php  
  
// arquivo: controllers/tarefas.php  
// ...  
  
require __DIR__ . "/../views/template.php";
```

Agora, atualize a página, e você deverá ver o formulário e a lista de tarefas normalmente.

17.3 ORGANIZANDO AS MODELS

Nesse momento, o projeto já está melhor organizado sem um monte de arquivos espalhados no diretório raiz. Mas ainda podemos melhorar a organização ou a facilidade de entendimento renomeando o diretório `classes` para `models`.

O conteúdo do diretório vai continuar exatamente o mesmo, mas o nome `models` vai expor ainda mais a estrutura MVC usada

na aplicação. Após renomear o diretório `classes` para `models`, altere o arquivo `index.php` para incluir os arquivos do novo diretório:

```
<?php  
  
// Incluir as configurações, ajudantes e classes  
  
require "config.php";  
require "banco.php";  
require "ajudantes.php";  
require "models/Tarefa.php";  
require "models/Anexo.php";  
require "models/RepositorioTarefas.php";  
  
// ...
```

Atualize a aplicação no navegador e você deverá ver tudo funcionando normalmente.

Até agora, já criamos um diretório `controllers` e movemos todos os arquivos que vão tratar as requisições dos usuários para lá. Criamos um diretório `views` e movemos todos os arquivos responsáveis pela apresentação do conteúdo para lá. Renomeamos o diretório `classes` para `models`, pois este nome faz mais sentido com a organização do projeto. E também criamos um arquivo `index.php`, nosso front-controller, e faz a conexão entre todos os outros.

17.4 LIGANDO AS PONTAS

Ainda podemos melhorar algumas coisas, como organizar os outros arquivos que ainda estão na raiz da aplicação. Porém, acredito que neste momento é melhor corrigir os detalhes no código para que tudo funcione na nova arquitetura MVC.

Vamos começar pela adição de tarefas. No arquivo controllers/tarefas.php , precisamos alterar a chamada da função header() para não mais direcionar o usuário para o arquivo tarefas.php , mas sim para index.php . Ele deverá utilizar o parâmetro de rota indicando a rota correta para o PHP tratar usando o GET no arquivo index.php :

```
<?php  
  
// arquivo controllers/tarefas.php  
  
// ...  
  
if (! $tem_erro) {  
    $repositorio_tarefas->salvar($tarefa);  
  
    if (isset($_POST['lembrete']) && $_POST['lembrete'] == '1') {  
        enviar_email($tarefa);  
    }  
  
    header('Location: index.php?rota=tarefas');  
    die();  
}  
  
// ...
```

Agora vamos alterar todos os links na lista de tarefas que fica no arquivo views/tabela.php para apontar para index.php com seus devidos parâmetros de rota e adicionais, como o id da tarefa, quando necessário:

```
...  
  
<?php foreach ($tarefas as $tarefa) : ?>  
    <tr>  
        <td>  
            <a href="">  
                <?php echo htmlentities\(\$tarefa->getNome\(\)\); ?>    </tr>
```

```

        </a>
    </td>
    <td>
        <?php echo htmlentities($tarefa->getDescricao()); ?>
    </td>
    <td>
        <?php echo traduz_data_exibir(
            $tarefa->getPrazo()
        ); ?>
    </td>
    <td>
        <?php echo traduz_prioridade(
            $tarefa->getPrioridade()
        ); ?>
    </td>
    <td>
        <?php echo traduz_concluida(
            $tarefa->getConcluida()
        ); ?>
    </td>
    <td>
        <a href="index.php?rota=editar&id=<?php
            echo $tarefa->getId();
        ?>">
            Editar
        </a>
        <a href="index.php?rota=remover&id=<?php
            echo $tarefa->getId();
        ?>">
            Remover
        </a>
    </td>
</tr>
<?php endforeach; ?>

...

```

Perceba que a alteração dos links segue um padrão de remover o `.php` do nome dos arquivos e adicionar o `index.php` seguido do parâmetro `rota`, como o `editar.php?id=X` que virou `index.php?rota=editar&id=X`. Lembre-se de que usamos o caractere `&` para separar os parâmetros na URL.

O primeiro link no arquivo `views/tabela.php` leva para a rota `tarefa`, que é o arquivo `controllers/tarefa.php`. Nele precisamos alterar apenas a última linha que faz o include do template:

```
<?php  
// ...  
include __DIR__ . "/../views/template_tarefa.php";
```

No arquivo `views/template_tarefa.php`, também precisamos alterar todos os links para apontar para `index.php`. Este arquivo é grande, mas temos apenas dois links que precisam de mudanças: o que volta para a lista de tarefas e o que é usado para remover os anexos.

```
...  


# Tarefa: <?php echo htmlentities($tarefa->getNome()); ?></h1> <a href="index.php?rota=tarefas"> Voltar para a lista de tarefas </a>. </p> ... <!-- lista de anexos --> <?php foreach ($tarefa->getAnexos() as $anexo) : ?> <tr> <td><?php echo htmlentities($anexo->getNome()); ?></td> <td> <a href="anexos/<?php echo $anexo->getArquivo(); ?>"> Download </a> <a href="index.php?rota=remover_anexo&id=<?php echo $anexo->getId(); ?>">Remover</a>


```

```
        </td>
    </tr>
<?php endforeach; ?>

...

```

Neste momento, você já pode clicar no nome da tarefa na lista de tarefas para ver os seus detalhes e a lista de anexos.

Agora, vamos para o controller que edita os dados das tarefas no arquivo `controllers/editar.php`. Este também é bem simples, pois precisamos alterar apenas a função `header()` e o `require` no final do arquivo:

```
<?php

// ...

if (! $tem_erro) {
    $repositorio_tarefas->atualizar($tarefa);

    if (isset($_POST['lembrete'])
        && $_POST['lembrete'] == '1') {
        enviar_email($tarefa);
    }

    header('Location: index.php?rota=tarefas');
    die();
}

require __DIR__ . "/../views/template.php";
```

O controller que remove uma tarefa, no arquivo `controllers/remover.php`, é bem pequeno e terá uma mudança apenas na função `header()`:

```
<?php

$repositorio_tarefas->remover($_GET['id']);
```

```
header('Location: index.php?rota=tarefas');
```

E o último controller a ser alterado é o controllers/remover_anexo.php , que precisa de uma mudança na função unlink() e na header() :

```
<?php  
  
$anexo = $repositorio_tarefas->buscar_anexo($_GET['id']);  
$repositorio_tarefas->remover_anexo($anexo->getId());  
unlink(__DIR__ . '/../../anexos/' . $anexo->getArquivo());  
  
header('Location: /index.php?rota=tarefa&id='  
    . $anexo->getTarefaId());
```

Após esta alteração, tudo já deve estar funcional. Experimente navegar pela aplicação, e adicionar e remover algumas tarefas. Caso encontre algum problema, verifique se as alterações foram feitas corretamente.

Ufa, quase lá! Certifique-se de ter compreendido as mudanças até aqui. Lembre-se de que estamos apenas movendo os arquivos para pastas/diretórios diferentes para melhor organizar o projeto.

17.5 ORGANIZANDO O RESTANTE DOS ARQUIVOS

Se você listar os arquivos no diretório raiz da aplicação, você vai perceber que ainda temos alguns arquivos espalhados que poderiam ser organizados em diretórios. O arquivo tarefas.css , por exemplo, pode ser colocado em um diretório novo chamado assets , que é o nome normalmente dado ao diretório que contém CSS, JavaScript e imagens.

Após criar o diretório assets e mover o arquivo

tarefas.css para lá, altere os arquivos views/template.php e views/template_tarefas.php para indicar a nova localização do arquivo de CSS:

```
<html>
  <head>
    <meta charset="utf-8" />
    <title>Gerenciador de Tarefas</title>
    <link rel="stylesheet"
          href="assets/tarefas.css" type="text/css" />
  </head>
  <body>
    ...

```

Também vamos criar um diretório chamado helpers (ajudantes), no qual colocaremos os arquivos ajudantes.php e banco.php . Após mover esses arquivos, precisamos alterar o começo do arquivo index.php para fazer os requires usando a nova localização:

```
<?php

require "config.php";
require "helpers/banco.php";
require "helpers/ajudantes.php";
require "models/Tarefa.php";
require "models/Anexo.php";
require "models/RepositorioTarefas.php";

//...
```

Você deve ter reparado que os diretórios criados neste capítulo estão sendo nomeados em inglês. Eu fiz isso de propósito para que o projeto fique com uma estrutura mais parecida com os diversos frameworks PHP. Por isso também vamos renomear o diretório bibliotecas para libs , apenas para manter esse padrão.

Precisamos também de algumas mudanças no arquivo

helpers/ajudantes.php , pois algumas funções neste arquivo apontam para outros arquivos e diretórios:

```
<?php

function enviar_email($tarefa)
{
    include __DIR__ . "/../libs/PHPMailer/PHPMailerAutoload.php";

    //
    //

    foreach ($tarefa->getAnexos() as $anexo) {
        $email->addAttachment(
            __DIR__ . "/../anexos/{$anexo->getArquivo()}"
        );
    }

    $email->send();
}

function preparar_corpo_email($tarefa)
{
    ob_start();
    include __DIR__ . "/../views/template_email.php";

    $corpo = ob_get_contents();

    ob_end_clean();

    return $corpo;
}

function tratar_anexo($anexo) {
    $padrao = '/^\.+\.(pdf|zip)$/';
    $resultado = preg_match($padrao, $anexo['name']);

    if (! $resultado) {
        return false;
    }

    move_uploaded_file(
        $anexo['tmp_name'],
        __DIR__ . "/../anexos/{$anexo['name']}"
    );
}
```

```
    return true;  
}  
  
// ...
```

Neste momento, tudo já deve estar funcional. Experimente navegar pela aplicação e usar todas funcionalidades. Se encontrar algum problema, leia atentamente a mensagem de erro que provavelmente será relacionada aos caminhos dos arquivos. Eu aconselho a ler este capítulo mais de uma vez para entender bem as mudanças feitas e seus propósitos.

17.6 FRAMEWORKS

Um framework não é uma aplicação pronta para usar, mas sim um conjunto de ferramentas e uma fundação para projetos de software. Um framework é uma ferramenta que ajuda no processo de desenvolvimento de aplicações, dando bases que podem e devem ser usadas e reutilizadas, e também definindo padrões sobre onde determinadas funcionalidades devem ser colocadas.

Graças aos padrões definidos pelos frameworks, a equipe desenvolvendo um projeto não precisa se preocupar com as bases genéricas de uma aplicação, como a conexão com o banco de dados e o tratamento contra ataques de SQL Injection e XSS.

Quando se utiliza um framework, a tarefa de introduzir uma nova pessoa na equipe também se torna mais fácil, pois ela vai saber onde procurar as funcionalidades dentro do código, caso já tenha familiaridade com o framework. Também poderá contar com o manual/documentação e, provavelmente, com comunidades online/offline voltadas a discutir e melhorar o framework.

Infelizmente, os frameworks fogem do escopo deste livro, mas aqui estão alguns nomes, links e uma breve descrição para você pesquisar:

- Symfony (<https://symfony.com/>) — Uma das opções mais robustas quando falamos sobre frameworks para PHP. O Symfony é também uma coleção de componentes reutilizáveis para PHP, oferecendo componentes para a Gestão de Assets, Configurações, Gestão de Dependências, Formulários HTML, Roteamento, Segurança, Templates, Validação e outros que podem ser usados independente de framework.
- Zend Framework (<http://framework.zend.com/>) — Muito utilizado em aplicações Enterprise e, junto com o Symfony, é um dos mais robustos do mercado. O Zend Framework também é um conjunto de componentes que podem ser usados de forma independente ou como um grande framework.
- Laravel (<https://laravel.com/>) — Foco em facilidade de uso e código expressivo, o Laravel é um dos frameworks mais utilizados atualmente e tem uma grande comunidade. Seus componentes também podem ser usados de forma independente, mas a sua força está no pacote tudo em um, que pode conter até mesmo uma máquina virtual de desenvolvimento usando o Laravel Homestead.
- CakePHP (<http://cakephp.org/>) — Foco em rapidez no desenvolvimento, o CakePHP é uma das opções que está a mais tempo do mercado e que evoluiu junto com o PHP.

- Yii (<http://www.yiiframework.com/>) — Também é bastante utilizado em aplicações Enterprise, pois tem foco em estabilidade. Contém ferramentas que geram código PHP e ajudam com as tarefas repetitivas quando criamos um novo projeto ou adicionamos novas funcionalidades.
- CodeIgniter (<https://codeigniter.com/>) — Foco em ser leve e rápido. Este framework é uma boa opção para se iniciar no mundo dos frameworks, pois sua estrutura é mais simples e direta.

Uma boa forma de aprender mais sobre os frameworks é fazendo alguns tutoriais e aplicações simples. Em geral, todo framework tem uma boa documentação e um tutorial para iniciantes que mostra como desenvolver uma aplicação. Tente aprender, pelo menos, uns dois ou três frameworks diferentes, pois eles podem se mostrar melhores para determinados tipos de projetos.

17.7 RESUMO

Neste capítulo, foi introduzido o conceito de MVC e organizamos a aplicação de gestão de tarefas para ter uma estrutura MVC mais clara. Essa organização foi feita criando diretórios para cada camada da aplicação, e então movendo os arquivos para os diretórios de acordo com suas funcionalidades.

Neste capítulo também aprendemos o conceito de front-controller, que é um arquivo que recebe todas as requisições, e carrega as bibliotecas e outros arquivos necessários para execução da aplicação. E por último, tivemos uma pequena introdução aos

frameworks e uma lista com alguns dos mais famosos e usados atualmente.

17.8 DESAFIOS

Muito bem, hora dos desafios!

1. Faça a adaptação da aplicação de gestão de contatos para a estrutura MVC. Lembre-se de criar os diretórios **models**, **views** e **controllers** e um front-controller.
2. Faça a adaptação da aplicação de gestão do estacionamento para a estrutura MVC seguindo as mesmas regras da aplicação de gestão de contatos.
3. Escolha dois ou três frameworks da lista anterior, ou outros que você já tenha ouvido falar, e tente ler a introdução da documentação e fazer a aplicação que em geral está no tutorial no site oficial. Caso o site oficial não ofereça um tutorial básico, tente pesquisar por "tutorial blog laravel", "clone twitter cakephp" ou ainda "galeria imagens codeigniter". Estas buscas vão trazer diversos resultados com tutoriais e dicas.
4. Tente também refazer a aplicação de gestão de tarefas em um ou mais frameworks para exercitar o seu uso.

CAPÍTULO 18

APENAS O COMEÇO

E aqui chegamos ao fim deste livro, mas não ao fim dos estudos. Se você deseja entrar no mercado de desenvolvimento de software ou deseja ter a programação como um hobbie, saiba que este livro é apenas um começo. Se você leu todo o conteúdo até aqui e praticou os exercícios, você já deve estar em boa forma para começar seus próprios projetos.

Lembre-se de que, para tocar bem um intrumento musical, são necessárias várias horas, dias, meses e até anos de treino e estudos. Claro que é possível conseguir tocar algumas notas sem muito esforço com apenas alguns minutos de prática, mas dificilmente você vai encantar as pessoas com suas músicas sem uma boa dose de foco e suor.

É claro que com programação não é diferente. Para conseguir desenvolver software com qualidade e chegar ao ponto de encantar as pessoas com seus códigos que parecem poesia, é necessário praticar, e muito, pois quanto mais se pratica, mais se aprende.

Às vezes, o problema maior para se praticar programação é saber por onde começar ou o que fazer. Então que tal tirar da gaveta aquela sua ideia de aplicação? Que tal começar a resolver os problemas que você enfrenta no dia a dia usando seus novos

conhecimentos para desenvolver pequenas aplicações usando PHP e MySQL? Este é um ótimo momento para começar a praticar mais.

Invente novos problemas para resolver, pois assim você se torna o primeiro usuário de suas aplicações. Crie um organizador de tarefas, um gestor para a sua coleção de livros, uma agenda digital, um clone do Twitter, uma loja virtual, ou automatize pequenas tarefas no seu dia a dia. O céu não é o limite.

Experimente outras linguagens de programação e outros bancos de dados também. Entenda as diferenças entre as linguagens e onde estão as vantagens e desvantagens de cada uma.

Python e Ruby são ótimas opções para se ter uma visão diferente do mundo PHP. JavaScript é algo que sempre vale a pena o investimento, pois é a única linguagem suportada pelos navegadores atualmente.

Ainda sobre as linguagens de programação, evite se tornar uma pessoa superprotetora da sua linguagem favorita. Lembre-se de que linguagens de programação são ferramentas e seria bem estranho ver alguém da área de construção civil dizendo que o martelo é a melhor ferramenta de todas e que devemos usá-lo para resolver todos os problemas (cortar, soldar, fixar, serrar, dobrar e outros tantos).

Um outro assunto importante: estude inglês. Praticamente tudo está em inglês nos sites, nas documentações e nas comunidades de linguagens de programação. Você não precisa necessariamente ser fluente em conversação, mas deve se sentir confortável lendo textos, programando e até mesmo assistindo

palestras técnicas.

O importante é continuar praticando, aprendendo e mantendo a mente aberta para as novas opções que surgem todos os dias no mundo da tecnologia da informação.

18.1 ONDE POSSO BUSCAR MAIS INFORMAÇÕES?

A internet é bem vasta e, às vezes, é complicado achar bons conteúdos. Por isso, deixo aqui algumas sugestões de onde buscar ajuda e procurar informações.

Para saber mais sobre os recursos do PHP, visite a documentação oficial em <http://php.net/docs.php>. Tem também a versão em português aqui http://www.php.net/manual/pt_BR/index.php. Às vezes, a versão em português pode estar um pouco defasada com relação à versão em inglês, por isso é bom comparar as duas.

Se você fala inglês, você tem uma grande vantagem, pois no geral o conteúdo é gerado primeiro em inglês. Aliás, se você fala inglês, recomendo que você ajude na tradução da documentação do PHP para português, assim você contribui de volta para linguagem e ainda ajuda o pessoal que ainda não teve oportunidade de aprender inglês.

Um site que tem muita informação importante sobre as melhores práticas para PHP é o <http://www.phptherightway.com/>. Existe também uma versão em português, em <http://br.phptherightway.com/>.

Participe do fórum de discussão do livro. Lá você pode tirar suas dúvidas, ajudar quem está passando por problemas diferentes e ainda conhecer pessoas interessadas no mesmo assunto, o que pode acelerar seu aprendizado. O fórum está neste endereço: <http://forum.casadocodigo.com.br/>.

Uma boa ideia é participar também de comunidades maiores como o GUJ (<http://www.guj.com.br>). Lá tem pessoas conversando sobre várias linguagens e tecnologias, e é possível aprender muito participando desde tipo de grupo.

Fique de olho no site da Casa do Código (<http://www.casadocodigo.com.br>) para saber dos últimos lançamentos.

Se você utiliza redes sociais, procure encontrar as pessoas influentes nas áreas em que você quer se especializar. Em geral, existem diversos palestrantes, escritores, membros de comunidades etc. compartilhando informações úteis, como dicas de bibliotecas para usar, boas práticas, blogs para ler e por aí vai.

Se você está no Twitter ,você pode me seguir. Sou o @InFog9 e, no geral, compartilho textos e vídeos sobre PHP, outras linguagens e TI em geral.

Tente participar também de eventos e grupos de estudos sobre os assuntos que você se interessa mais. Use os motores de busca na web para encontrar esses grupos na sua cidade. Caso não encontre, crie um grupo e comece a convidar pessoas para fazer parte. Em grupo, podemos aprender mais e nos manter mais motivados do que sozinhos. ;)

Vale lembrar de que você pode baixar todos os exemplos usados no livro. Eles estão no GitHub, em <https://github.com/InFog/phpmysql>.

Bons estudos!