**Aula 1 – CRUD-Produto em Laravel**

**Criando o Projeto da Aplicação**

**Objetivos:**

* Criar o projeto da Aplicação no Laravel;
* Conhecer alguns comandos do Artisan;
* Conhecer alguns arquivos de configuração;

**Sobre o Laravel:**

* Laravel é um framework para desenvolvimento de aplicações web;
* Utiliza a Linguagem PHP para desenvolvimento backend;
* Possui várias “facilidades” que aceleram o desenvolvimento.

**Requisitos para iniciar um projeto com Laravel:**

* Cada versão do Laravel necessita de uma versão mínima do PHP. Na época da escrita desta apostila a versão atual do Laravel era a 11.x, a qual exige no mínimo o PHP 8.2;
* PHP 8.2 ou maior (https://windows.php.net/download#php-8.2) é necessário para o Laravel 11.x a qual é a versão mais recente na época da escrita desta apostila. Baixar a versão VS16 x64 Thread Safe e instalar na raiz da unidade C:\php82.
* Importante: colocar o caminho C:\php82 na variável PATH do Windows;
* Composer (<https://getcomposer.org/>)

1. **Criando o Projeto Laravel da Aplicação**

Para criar um projeto Laravel para nossa aplicação, crie uma pasta para seus projetos. Abra um prompt de comandos e vá até esta pasta. Aqui vou criar a pasta Laravel na raiz da unidade C:

Tela de um aparelho eletrônico

Descrição gerada automaticamente com confiança baixa

Agora, pelo composer, crie o projeto Laravel para a aplicação:

composer create-project laravel/laravel produtoapp

Obs.: para utilizar uma versão específica:

composer create-project laravel/laravel produtoapp "11.\*"

Texto

Descrição gerada automaticamente

Note acima que houve um erro de falta da extensão openssl do PHP. Então, edite o arquivo php.ini do PHP e habilite a extensão:

Texto

Descrição gerada automaticamente

Salve o arquivo e tente criar o projeto novamente:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Note agora que não foi possível carregar outra extensão do PHP, a extensão zip. Faça o mesmo procedimento anterior:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Salve o arquivo e tente criar o projeto novamente:

Texto

Descrição gerada automaticamente

Note que aconteceram novos problemas associados à falta de extensões do PHP. Agora o problema foi com a extensão fileinfo. Para resolver, temos que habilitá-la no php.ini. Porém, ao invés de configurar o caminho para a pasta da extensão em cada diretiva, vamos configurar o caminho para a pasta de extensões na diretiva extension\_dir. Localize-a no php.ini e configure como a seguir:

Texto

Descrição gerada automaticamente

Agora localize a extensão fileinfo e habilite-a. Já vamos aproveitar e habilitar também as extensões gettext e mbstring, como a seguir:

Texto

Descrição gerada automaticamente com confiança média

Salve o arquivo, mas antes de testar, é necessário remover a pasta C:\Laravel\produtoapp criada pela tentativa anterior.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Agora execute novamente o comando do composer para criar o projeto. Veja na figura a seguir que o composer criou o projeto e instalou várias dependências.

Texto

Descrição gerada automaticamente

Porém, no final ainda aparece uma mensagem de warnning dizendo que não foi encontrado o driver para o sqlite. Novamente, é devido ao fato da extensão do sqlite desabilitada no PHP. Habilite-a no php.ini:

Interface gráfica do usuário, Texto, chat ou mensagem de texto

Descrição gerada automaticamente

Salve o php.ini. Novamente será necessário remover a pasta da aplicação e repetir o comando do composer:

Texto

Descrição gerada automaticamente

Note na figura anterior o final da criação do projeto Laravel para nossa Aplicação. Veja que finalmente tudo foi instalado sem erro nem warnnings.

Vá até a pasta e veja a estrutura de pastas e arquivos criados:

Interface gráfica do usuário, Tabela

Descrição gerada automaticamente

Embora ainda não tenhamos escrito nenhum código da nossa Aplicação, já é possível acessá-la através do servidor web embutido do PHP. Para isso, volte ao prompt de comandos, entre na pasta da aplicação e execute o comando:

php artisan serve

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Note que a aplicação está executando localmente (127.0.0.1) na porta 8000. Então abra um navegador e acesse o endereço. Será exibida uma página padrão do Laravel.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

1. **Configuração da Aplicação: arquivo .env e pasta /config**

Abra a pasta do projeto criado (produtoapp) em seu editor preferido. Aqui vou utilizar o VSCode.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Na raiz do projeto existem alguns arquivos de configuração importantes. Dentre eles, o .env, o qual contém configurações do ambiente de desenvolvimento da Aplicação.

Texto

Descrição gerada automaticamente

Estas configurações fazem parte do ambiente de desenvolvimento e, quando utilizado o GitHub para compartilhamento do código, esse arquivo não sobe para o GitHub, pois ele pode conter informações específicas do Ambiente de Desenvolvimento de cada colaborador do projeto. Esse arquivo está listado no arquivo .gitignore, o qual contém todos os arquivos que não vão para o GitHub. Assim, quando fazemos o deploy da Aplicação para o servidor de Produção via GitHub, essas configurações não vão existir no ambiente de Produção. Porém, essas configurações também podem ser feitas dentro dos arquivos da pasta config.

Tela de computador com texto preto sobre fundo azul

Descrição gerada automaticamente

Por exemplo, no arquivo config/app.php temos as configurações da Aplicação:

Texto

Descrição gerada automaticamente

Note que são as mesmas configurações do arquivo .env. Cada configuração tem uma documentação e é carregada através da função env(), Por exemplo:

'name' => env('APP\_NAME', 'Laravel'),

Nesse caso, a função primeiro verifica se existe algum valor no APP\_NAME, se existir ela pega esse valor. Caso não exista, pega o valor default 'Laravel'. Assim, quando a Aplicação for para produção podemos criar um arquivo .env específico com as configurações de Produção, ou colocá-las diretamente aqui para serem usadas no ambiente de Produção. Por exemplo, veja a configuração APP\_KEY, definida no .env da seguinte forma:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Note que o arquivo config/app.php NÃO tem uma configuração default:

Texto

Descrição gerada automaticamente

Sem essa configuração a Aplicação não funcionará. Assim, podemos copiar a chave do arquivo .env (ou gerar uma nova Chave via comando do artisan) e colocá-la aqui:

Texto

Descrição gerada automaticamente

1. **Comandos de Console do Artisan**

O Laravel possui vários comandos de administração da Aplicação que podem ser executados no Console do sistema. Mesmo console onde executamos o comando php artisan serve.

Voltar ao console e parar a execução do servidor usando <Ctrl> <C>

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Para ver os comandos do Artisan disponíveis, execute o comando php artisan list

Texto

Descrição gerada automaticamente

Por exemplo, o comando about exibe uma visão geral da configuração, dos drivers e do ambiente da Aplicação:

Texto

Descrição gerada automaticamente com confiança média

**O comando down**

Este comando do Artisan coloca a Aplicação em modo de Manutenção, não permitindo acesso. Interrompa o servidor e execute no console o comando:

php artisan down

Interface gráfica do usuário, Texto

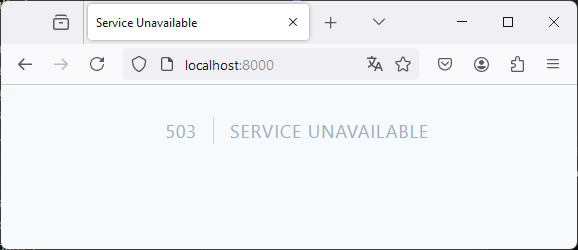
Descrição gerada automaticamente

Depois execute novamente o servidor: php artisan serve

Texto

Descrição gerada automaticamente com confiança média

Agora tente acessar a Aplicação:



Assim, é possível efetuar a manutenção da Aplicação sem ter acesso de usuários. O problema é que não será possível testar a Aplicação para verificar as alterações. Neste caso, o domando down do Artisan permite definir um token secreto para desvio do modo de manutenção. Para testar, primeiro execute o comando:

php artisan up

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Agora coloque a aplicação em manutenção definindo um token secreto de acesso:

php artisan down --secret="jldfjls9080394803--930409"

Texto

Descrição gerada automaticamente

Depois execute novamente o servidor: php artisan serve

Texto

Descrição gerada automaticamente com confiança média

Tente acessar a Aplicação na URL <http://localhost:8000>. Não será permitido.

Agora acesse a Aplicação utilizando o token na URL:

http://localhost:8000/jldfjls9080394803--930409

Note que agora foi permitido.

Após realizar a manutenção é preciso executar o comando php artisan up para voltar a Aplicação ao modo normal.

Uma descrição detalhada destes comandos e de outros pode ser obtida em <https://laravel.com/docs/11.x/configuration>

1. **Alterando o Nome da Aplicação**

No arquivo .env altere a configuração APP\_NAME para:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Este nome da Aplicação é utilizado pelo Laravel em notificações e outros elementos.

Também é bom colocar este nome no arquivo config>app.php:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

1. **Alterando o Timezone**

Outra configuração que devemos alterar dentro do arquivo .env é APP\_TIMEZONE a qual determina o timezone da Aplicação, influenciando nas funções de data e hora. Essa configuração pode ser alterada no php.ini, ou aqui nesta configuração do .env. Note que o valor atual é APP\_TIMEZONE=UTC . Este valor, dependendo o estado do Brasil, não mostrará a data e hora corretos.

Para verificarmos isso, vamos fazer um teste para nossa Aplicação mostrar a data e a hora.

Edite o arquivo routes>web.php. Acrescente a seguinte rota:

Route::get('/date', function () {

return date("d/m/Y h:i:sa");

});

O arquivo ficará assim:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Nota: A configuração anterior cria uma rota para nossa aplicação chamada /date, a qual retorna o resultado da função date("d/m/Y h:i:sa")do PHP. Rota é um recurso do Laravel e será explicado mais adiante.

Salve o arquivo e acesse a Aplicação pela URL: http://localhost:8000/date

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Verifique que a hora exibida será 3 horas a mais do que a hora do sistema do computador onde a Aplicação está executando.

Portanto, o Timezone está incorreto. Altere a configuração APP\_TIMEZONE do arquivo .env para:

APP\_TIMEZONE=America/Sao\_Paulo

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Salve o arquivo e acesse novamente a Aplicação pela URL: http://localhost:8000/date

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

1. **Acessando as configurações do PHP – phpinfo()**

Muitas vezes é necessário verificar as configurações do PHP através da função phpinfo(). Em uma Aplicação Laravel, podemos fazer isso facilmente através de uma rota. Acrescente a rota a seguir no arquivo routes>web.php:

Route::get('/phpinfo', function () {

return phpinfo();

});

Texto

Descrição gerada automaticamente

Salve o arquivo e acesse a Aplicação pela URL: http://localhost:8000/phpinfo

Texto, Email

Descrição gerada automaticamente com confiança média

1. **Estrutura de Diretórios da Aplicação no Laravel**

O Laravel estrutura o projeto da Aplicação em diferentes Diretórios ou Pastas, cada qual tem uma função.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

**A pasta app**

Esta pasta é onde fica toda a lógica da Aplicação. Onde colocamos a maior parte de nossas classes que implementam as regras de negócio da Aplicação. Como sabemos, o Laravel utiliza o modelo MVC. Assim, podemos ver dentro da pasta app, algumas das partes do MVC, como Http\Controllers, onde ficam os controladores da Aplicação, e Models, onde ficam os Modelos da Aplicação. Note que já existe, por default, o modelo User.php, o qual modela a tabela de Usuários de acesso ao sistema.

Texto

Descrição gerada automaticamente

Outras pastas poderão ser criadas dentro da pasta app, tais como: Broadcasting, Console, Events, Exceptions, Http, Jobs, Listeners, Mail, Models, Notifications, Policies, Providers e Rules. Estas pastas têm funções específicas e podem ser criadas com o comando make do Artisan.

**A pasta Bootstrap**

Contém o arquivo app.php que inicializa o framework. Esta pasta também abriga um diretório chamado cache que contém arquivos gerados pelo framework para otimização de desempenho, como os arquivos de cache de rota e serviços.

Texto

Descrição gerada automaticamente

**A pasta config**

Como já mostrado anteriormente, a pasta config contém todos os arquivos de configuração da Aplicação. Cada arquivo aqui possui configurações específicas para uma determinada funcionalidade da Aplicação. Por exemplo, app.php contém configurações gerais da aplicação, tal como nome, url de acesso, timezone etc. Já o arquivo database.php contém configurações de acesso ao banco de dados.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**A pasta database**

Contém as fábricas (factories) para criar registros para popular automaticamente suas tabelas. Além de conter também as migrations (migrações) de banco de dados que facilitam a criação das tabelas no banco de dados. Também configuramos nesta pasta os seeders (semeadores) da nossa Aplicação, os quais determinam como gerar registros para as tabelas.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**A pasta public**

Contém o arquivo index.php, que é o ponto de entrada para todas as solicitações que entram na Aplicação e configura o carregamento automático. Este diretório também abriga os arquivos públicos como imagens, JavaScript e CSS.

Texto

Descrição gerada automaticamente

**A pasta resources**

Contém as Views do modelo MVC e também CSS e JavaScript específicos da Aplicação. O Laravel trabalha com um sistema de templates para a interface com o usuário chamado Blade. Portanto, todos os arquivos de Views da Aplicação devem ter a extensão .blade.php.

Texto

Descrição gerada automaticamente

**A pasta routes**

Contém todas as definições de rota para sua aplicação. Por padrão, dois arquivos de rota são incluídos com o Laravel: web.php e console.php.

**A pasta storage**

Contém seus logs, modelos Blade compilados, sessões baseadas em arquivo, caches de arquivo e outros arquivos gerados pelo framework. Este diretório contém os diretórios app, framework e logs. O diretório pode ser usado para armazenar quaisquer arquivos gerados pela Aplicação. O diretório é usado para armazenar arquivos e caches gerados pelo framework e os arquivos de log da Aplicação.

O diretório storage/app/public pode ser usado para armazenar arquivos gerados pelo usuário, como avatares de perfil, que devem ser acessíveis publicamente. Você deve criar um link simbólico public/storage que aponte para esse diretório.

**A pasta tests**

Contém seus testes automatizados.

**A pasta vendor**

Contém as dependências do framework Laravel e da nossa Aplicação. Essa pasta é controlada pelo Composer.

1. **Ciclo de Vida de uma Requisição**

O ponto de entrada para todas as requisições para um aplicativo Laravel é o arquivo public/index.php.

Texto

Descrição gerada automaticamente

Todas as requisições devem ser direcionadas para esse arquivo pela configuração do seu servidor web (Apache/Nginx). O arquivo index.php não contém muito código. Em vez disso, é um ponto de partida para carregar o resto do framework. Ele carrega a definição do autoloader gerada pelo Composer e, em seguida, recupera uma instância do aplicativo Laravel em bootstrap/app.php. A primeira ação tomada pelo próprio Laravel é criar uma instância do contêiner de aplicativo/serviço .

Texto

Descrição gerada automaticamente