**Seção 5: Introdução as Rotas, Controllers e Views**

**Aula 23**

Para inicar a execução do Laravel, deve-se executar php -S localhost:\*porta disponível\* no console e deixa-lo executando para ser possível a sua utilização.

**Aula 24**

Mostrado o artisan, script responsavel por garantir execução de comandos uteis de maneira mais fácil, como por exemplo, basta executar o comando php artisan serve para rodar a aplicação http do laravel instalado com a porta padrão 8000, podendo ser alterada com o parâmetro --port=\*porta disponível\*.

**Aula 25**

Introução as rotas, que no ambiente laravel são segmentadas em 4 partes: API, channels, console e web. Cada uma possui sua própria função dentro do sistema a ser desenvolvido e serão utilizadas de forma mais prática futuramente.

**Aula 26**

Utilizada a rota Web para criar duas novas rotas, a de /sobre-nos e a /contato. É utlizado o método Route::\*método http\*($uri, $callback). O método http pode ser get, post, patch, put, delete, etc. Já a $uri é o local que deseja ser mapeado como no caso /sobre-nos e /contato. O $callback se trata do que será realizado quando a local for acessado, geralemente sendo uma função que retorna uma view.

**Aula 27**

Explicação breve sobre controladores. Se tratam da parte do código responsável pela lógica do negócio, após ser tatados pelas rotas que levarão até determinada página do site.

**Aula 28**

Adicionado os controllers da aplicação. Com o artisan, é possível criar um novo controller com o comando php artisan make:controller, que irá criar um novo arquivo no diretório /app/http/controllers/. Após a criação dos controllers, no primeiro momento foram criadas uma função por controller somente exibindo a mesma mensagem que já estava sendo trasmitida em forma de callback nas rotas. Agora nas rotas deve ser passada uma string, que será interpretada de forma a esperar o nome da controller e uma ação a ser realizada na chamada desse controller, isso no laraval 7 para trás. A nova utilização deve-se ser utilizado como [\*diretório\*::class, ‘\*funcão\*’].

**Aula 29**

Explicação sobre view. Se trata da visão produzida no lado do servidor responsável por exibir os elementos para o usuário. Também é dito que este modelo se trata do modelo tradicional de criação de sites, existindo um modelo mais moderno, tratando back e front-end com softwares diferentes, unidos por uma api.

**Aula 30**

Feita a criação das views do projeto. Estão presentes dentro da pasta \app\resources\views, em que é criada a pasta site que irá conter as 3 páginas criadas até o momento. Essas views irão conter o que será de fato exibido para o usuário.

**Seção 6: Avançando com Rotas (Routes)**

**Aula 31**

Mostrado como receber parâmetros em rotas. Deve-se criar uma nova rota que pode ser o mesmo endereço, porém com mais uma separação com / e entre {} é passado o parâmetro que com uma função de callcabk poderá ser utilizado como uma variável.

**Aula 32**

Exibido como fazer com que os parâmetros sejam opcionais e não obrigatórios. Para realizar isso, basta definir um safe null operator no final do parâmetro presente na rota e na função de callback, junto da declaração da variável deve ser definido um valor, que será tratado com o valor padrão, caso nenhum seja passado no parâmetro. O laravel possui a limitação de que o parâmetro opcional só pode estar em falta da direita para a esquerda, pois caso alguma parâmetro não seja passado, porém e sequente seja passado, a página não será encontrada.

**Aula 33**

Feita a aplicação de expressões regulares para tratar os parâmetros passados. Para fazer o tratamento dos parâmetros, ao final do método get do route, deve-se por a expressão ->where(), onde serão passados o nome do parâmetro a ser tratado e depois separado por vírgula as expressões regulares desejadas, ambos sendo contidos em strings.

**Aula 34**

Criada uma lista com os links das rotas criadas até o momento.

**Aula 35**

Criadas 4 novas rotas para login, clientes, produtos e fornecedores.

**Aula 36**

Realizado o agrupamento das rotas presentes na aula passada utilizando o método prefix do Route. Route::prefix(‘/app’)->group, esse método recebe uma função que deverá possuir as rotas que serão agrupadas por esse prefixo.

**Aula 37**

Atribuídos nomes as rotas definidas até o momento, que facilitam e muito em suas chamadas em links por exemplo, fazendo com que a chamada seja a mesma independente do diretório da aplicação, ou caso a rota em si seja alterada. Para definir um nome, basta ultilizar o método name->(‘\*nome do site\*’) no final da linha da rota que se deseja nomear.

**Aula 38**

Mostrado como realizar redirecionamento de rotas. Para isso, outras duas rotas foram criadas somente para teste. Existem duas maneiras de realizar o redirecionamento. Uma delas é por meio da função redirect()->route(‘\*rota destino\*’), que deve ser passado dentro da função de callback da rota original. Outra maneira de redirecionar uma rota é utilizando o método redirect de Route (Route::redirect(‘/rota2’, ‘/rota1’);).

**Aula 39**

Criada uma rota de fallback, que é acessada toda vez que a rota não é encontrada. Sua sintaxe é Route::fallback(function() {\*ação da rota fallback\*});

**Seção 7: Avançando com Controladores (Controllers e Visualizações (Views)**

**Aula 40**

Mostrado como encaminhar parâmetros da rrota para o controlador. Basta que no momento em que seja passado o controller responsável por aquela rota, deve-se receber os parâmetros na função do controlador. Exem: parâmetros /{p1}/{p2}, a função do controller deve possuir function controller($p1, $p2) {} (o nome não precisa ser o mesmo, o importante é a ordem dos elementos a serem recebidos, da esquerda para a direita.

**Aula 41**

Mostrado como passar as variáveis de um controller para uma view. Há 3 maneiras de fazer isso. Após criar o retorno da view e indicar qual view será atribuída, pode-se criar um array contendo uma string com o nome desejado para a vaariável na view recebendo(=>) a variável presente no controller, para adicionar outras variáveis basta criar outros índices no array, o nome dessa técnica é array associativo. Existe também a função nativa do php compact, em que basta apenas usar o método compact dentro do método view, passando como parâmetros do primeiro strings sem o $ com o nome da variável, fazendo com que uma variável de mesmo nome seja passada diretamente na view. A ultima maneira é utilizando o método laravel ->with(‘\*nome desejado\*’, $\*variavel\*), contendo somente uma variável por utilização. Para passar outras variáveis, basta repetir a funcão novamente após o termino da anterior.

**Aula 42**

Explicado o blade. Se trata de uma tecnologia de renderização de views, possibilitando uma escrita mais enxuta do php no front-end. Além disso, o blade ainda permite que a escrita habitual do php seja utilizada, apesar de não ser muito usual.

**Aula 43**

Explicado como incluir comentários com blade e blocos de php puros. Para comentar em blade basta utilizar {{-- comentário --}}. E para abrir blocos de php, basta utilizar @php e @endphp para recerrar o bloco. {{ \*variavel ou string\* }} é sinônimo de <?= \*variável ou string\* ?>.

**Aula 44**

Instalado extensão de highlight da sintaxe blade no VSCode.

**Aula 45**

Feito if/else no blade. Basata utilizar @if, @else, @elseif e @endif para realizar o if. Diferente do php nativo, não é necessário abrir ou fechar tags para a programação, facilitando o trabalho.

**Aula 46**

Utilizado sintaxe blade @unless. Se trata da inversão do if, comparando se a informação passada é falsa, mesmo procedimento do if(!).

**Aula 47**

Explicada utilização do @isset. Essa sintaxe poupa a necessiade de abrir um if somente para verificar a existencia de uma variável. Para fechar, basta usar @endisset.

**Aula 48**

Definição do @empty, que verifica caso o valor atribuído a variável não é vazio. Valores vazios: ‘’, 0, 0.0, ‘0’, null, false, array(), $var.

**Aula 49**

Explicação de operador condicional ternário, no php puro mesmo.

**Aula 50**

Explicação do operador condicional de valor default no blade. Trata-se da utilização de ?? após a impressão de uma variável no blade, seguido pela definição do valor default. O operador condicional verifica como um isset e não como empty, então somente de estar setada, a variável não cairá no default.

**Aula 51**

Mostrado switch case no blade. @switch para abrí-lo, @case para inciar uma condição, @break para encerrá-la e @endswitch para finalizar o switch.

**Aula 52**

Mostrado for no blade. @for para iniciá-lo e @endfor para finalizá-lo. É possível usar um índice de um array como $i diretamente, exem: @for($i = 0; $array[$i]; $i++).

**Aula 53**

Mostrado while no blade. @while para iniciar e @endwhile para finalizar. É necessário criar uma variável de contador para contabilizar a execução do while, possívelmente sendo necessário a abertura da tag @php para criar o contador.

**Aula 54**

Exibido foreach no blade. @foreach para inicicar e @endforeach para finalizar.

**Aula 55**

Mostrado o forelse, tag exclusiva da sintaxe blade. Funciona como um foreach, mas caso o array percorrido esteja vazio, com o forelse é possível definir um @empty para definir o que será realizado com esse array vazio. Para fechá-lo, basta usar @endforelse.

**Aula 56**

Para escapar a tag de impressão do blade({{ }}), basta por um @ na frente da tag, o que fará com que o elemento inteiro seja imprimido e não interpretado.

**Aula 57**

Ao utilizar um loop for ou foreach, existe uma variável que contém informações sobre a execução, para acessá-la deve-se usar {{ $loop->\*informação\*}}. Alguns exemplos são ->iteration, ->first, ->last, ->count.

**Aula 58**

Disponibilizados novos arquivos com elementos feitos das páginas do diretório site.

**Aula 59**

Explicados o que são assets, basicamente sendo elementos que ajudam a composição de um elemento html, sejam css, arquivos javascript, etc.

**Aula 60**

Baixados e incluidos imagens presentes nos arquivos da aula anterior. Mostrado a tag {{ asset(‘\*diretorio\*’) }} que realiza a inserção de assets. Sua vantagem é que o diretório que a função asset irá acessar pode ser alterado a qualquer momento, então caso seja necessário uma mudançã de diretórios, configurando em apenas um local irá mudar em todos os outros automáticamente.

**Aula 61**

Feita a inclusão do css contido na página em um arquivo externo, dentro da pasta public.

**Aula 62**

Criado um template da parte de fora da body dos sites e colocada em outra view. A view contendo a head do site pode ser acessada pelos outros por meio do @extends e as views com o conteúdo podem ser renderizadas na view head por meio do @section(‘\*nome do conteudo\*’) e depois finalizada com @endsection. Depois, no site com a head foi usado o @yield(‘\*nome do conteudo\*’), fazendo com que o conteúdo das outras páginas sejam gerados sem repetição e uma alteração no template da head irá alterar em todas as páginas.

**Aula 63**

Feito uma alteração no título por meio da section, que pode ser mandada sem o @endsection, passando somente uma informação. @section(‘\*variavel\*’, ‘\*conteudo\*’) irá passar a variável que pode ser vista com o @yield.

**Aula 64**

Feita a inclusão do menu superior das páginas no arquivo basico, esndo criada a parte \_partials, que deverá conter partes do html que podem ser reutilizadas, se tratando de elementos visuais e não somente da head. @include irá incluir tudo que estiver no determinado arquivo, no local em que for chamada a sintaxe.

**Aula 65**

Passada a ação e o método do formulário presente na página de contato. A ação foi passada usando {{ route(‘site.contato’) }}, pois o objetivo é enviar o formulário para seu próprio controller.

**Aula 66**

Modificado o formulário fazendo com que ele seja enviado por post. Para fazer isso, é necessário criar uma nova rota utilizando o método post. Todos parâmetros da rota podem ser iguais, porém usando o método get ou post, somente um ou outro serão recebidos, fazendo que nesse caso os dois sejam necessários. Também é necessário adicionar ao formulário uma linha contendo @csrf, que se trata de um token necessário para que o laravel aceite qualquer formulário enviado, por questões de segurança.

**Aula 67**

Explicada a importância do token csrf. O mesmo impede que requisições maliciosas sejam realizadas com informações armazenadas nos cookies de um site para o envio de formulários falsos, como o de envio de dinheiro em um internet banking, por exemplo. O token csrf é mandado pelo servidor para a página web autêntica do formulário e é enviada novamente com os dados preenchidos do formulário. Caso qualquer formulário que for enviado para o servidor não possuir o token, será imediatamente recusado.

**Aula 68**

Mostrado a sintaxe @component para retirar o formulário que estava se repetindo em duas páginas para que o mesmo seja utilizado de forma dinâmica. Foi criada uma nova pasta no diretório layout, nomeada components. Nela foi criada um arquivo que contém o formulário.

**Aula 69**

É possível passar parâmetros por meio do @component, que será recebido pela variável @slot diretamente no que estiver escrito entre os @components. Com o @component é possível passar uma ou mais variáveis que serão passadas e serão utilizáveis diretamente no componente. Exem: @component(‘site.teste’, ‘\*variavel\*’ => ‘\*valor\*’).

**Seção 8: Models, Migrations, Seeders, Factories, Banco de Dados, Tinker e Eloquent ORM**

**Aula 70**

Criado a Model SiteContato com o comando php artisan make:model SiteContato -m. O comando -m cria junto da model, uma migration, que será explicada mais a frente no curso.

**Aula 71**

Utilização e explicação da migration. Se trata de um arquivo que possui informmações escritas em php, que irão realizar a criação de elementos no banco de dados, facilitando assim por exemplo, o desenvolvimento em equipe, pois no código pode conter as informações necessárias para o criação de tabelas e elementos em um banco de dados. Acessar <https://laravel.com/docs/10.x/migrations> para a documentação com os comandos das migrations.

**Aula 73**

Mostrado a execução das migrations. Para realizar, primeiro é necessário conferir o arquivo database.php, que contem informações sobre vários tipos de bancos de dados diferentes. Neste arquivo, existe a utilização do método env em várias linhas. Esse método se refere as infromações do environment, que deve ser deifinido no arquivo .env, na raiz do projeto. Neste arquivo, é necessário conferir se o banco setado é o que deseja ser utilizado. Também é interessante criar um arquivo de mesmo nome do que está escrito no elemento database\_path(método esse que irá procurara no diretório database), no arquivo database.php, no caso do sqlite. Para que a leitura do database\_path aconteça, é necessário excluir a linha com DB\_DATABASE no arquivo .env.

**Aula 74**

Instalação do mysql server.

**Aula 78**

Mudada a migration para os paramêtros mysql, presente no arquivo .env.

**Aula 79**

Executada a migration.

**Aula 80**

No caso da criação de uma model sem o -m no final, fará com que uma migration não seja criada automáticamente, porém, para criar depois caso necessário, basta digitar o comando php artisan make:migration \*nome\_da\_migration\* (no caso create\_fornecedores\_table). Criada a model Fornecedor e sua migration com a coluna string nome.

**Aula 81**

Criada uma migration de alteração de tabela (alter\_fornecedores\_novas\_colunas), informando as novas colunas a serem informadas para a mesma tabela utilizada na migration anterior, fazendo com que caso novas colunas sejam adicionada sem que o conteúdo anterior das tabelas seja perdido, criando outro documento de migrate poderá resolver o problema. No caso, o objeto Schema chama a função table, que é responsável por selecionar uma tabela e não criá-la.

**Aula 82**

Explicação da diferença do método up e down das migrations. O método up é responsável por realizar ações no banco de dados, já o método down é responsável por desfazer as ações do método up. Para executar o método down de uma migration, é necessário usar o comando php artisan migrate:rollback. Esse comando executa a uma rollback(down) da mais recete para a mais antiga, então caso existam 6 batches de migrations, somente a sexta será revertida. É possível alterar qual batch será revertida adicionando --step=\*batch desejada\* ao comando migrate:rollback.

**Aula 83**

Descrição de como criar colunas que aceitam null e valores default para colunas nas migrations. Para null, basta adicionar o método ->nullable() ao final da criação de uma coluna. Para valores default, basta adicionar o método ->default(\*valor default\*), utilizando ‘’ para strings.

**Aula 84**

Criadas chaves estrangeiras por meio de migrations. Para realizar tal, foi criada uma nova migration da tabela produto\_detalhes, na qual irá receber o id da tabela produtos. É criada uma constraint $table->foreign(‘produto\_id’)->references(‘id’)->on(‘produtos’). Essa linha cria uma chave estrangeira produto\_id, ligada a coluna id da tabela produtos, criando assim a ligação. Também é utilizado $table->unique(‘produto\_id’), para que os registros das tabelas sejam únicos, já que o objetivo é uma relação um para um.

**Aula 85**

Criados relacionamentos um para vários com migrations. A diferença em usa implementação é ma criação é pela ausência do método unique. Também é interessante criar a coluna foreign como unsinegnedBigInteger, pois essa é a criação padrão de uma id, sendo útil definir da mesma forma para uma foreign key.

**Aula 86**

Criado um relacionamento de vários para vários. Esta ligação é feita com base na criação de uma tabela auxiliar, totalizando 3 tabelas em que as duas tabelas exteriores fazem uma ligação de um para vários, dessa forma totalizando uma ligação de vários para vários. É criada uma nova migration criando as duas tabelas, a tabela filial que irá ter uma relação de vários para vários com a tabela produtos, para isso é criada a tabela auxiliar produtos\_filiais. É então criada na última duas foreign keys, uma de cada tabela, estabalecendo assim o relacionamento desejado.

**Aula 87**

Mostrado o modificador after. Sua função é definir o local em que uma coluna deve ficar, após sua criação, quando é feita em uma tabela pré-existente. Sua sintaxe é ->after(‘\*coluna anterior desejada\*’).

**Aula 88**

Mostrado 4 comandos uteis com o migrate: status; reset; refresh e fresh. O comando status exibe todos as migrates e seus batches, além de exibir caso elas foram migradas ou não. O comando reset executa os comandos down de todas as migrates, mas nada além disso. Já o refresh executa todos os comandos down das migrates e em seguida executa os comandos up, refazendo o banco de dados. O fresh é bem parecido com o refresh, porém ao invéz de executar o comando down, ele dropa todas as tabelas e executa o comando up em seguida.

**Aula 89**

Introdução ao eloquent ORM, que ajudará a trazer o paradigma de orientação a objetos para os bancos de dados relacionais.

**Aula 90**

Introdução ao tinker. Auxilia no teste das classes criadas e models por exemplo, funcionando como um atalho, pois não será necessário criar uma interface web para visualizar o funcionamento das classes.

**Aula 91**

Mostrado como realizar inserções com o tinker. Após abrir o terminal do tinker (php artisan tinker), foi criado um novo objeto da classe \App\SiteContato() e foram passados os respectivos parâmetros para o objeto. Depois para inserir o objeto no banco de dados, basta usar o modificador ->save(), que o objeto será inserido normalmente no banco de dados.

**Aula 92**

Mostrado que o eloquent tem um padrão para a tradução do nome da model para o nome do banco de dados. Primeiro, as letras maiusculas terão um undeline adicionados a sua esquerda (SiteFornecedor -> Site\_Fornecedor). Depois, as letras maíusculas serão transformadas para minusculas (Site\_Fornecedor -> site\_fornecedor). Por último, algo que pode apresentar problemas. É colocado um s no final da última letra, para dar o sentido de plural (site\_fornecedor -> site\_fornecedors). Porém em alguns casos como o do exemplo, somente o s não completa o sentido da palavra, provavelmente diferenciando do banco que está criado. Para isso é possível definir o nome da tabela alvo do eloquent, usando protected $table = ‘\*nome da tabela\*’. A variável protegida $table é definida pelo padrão de tradução do eloquent automáticamente.

**Aula 93**

No tinker, também é possível inserir registros por métodos estáticos, sem depender da instanciação de um objeto. Para isto, deve-se usar \*model\*::create([\*array de parâmetros a serem inseridos\*]);. Dessa forma, um novo registro será inserido no banco referente a model colocada. Porém, esse comando inicialmente apresentará erro. Isso é por que na model é necessário que exista a definição de quais colunas podem ser preenchidas com o comando protected fillable = [\*colunas preenchiveis\*]. Após essa adição para a model, o método estático create irá funcionar corretamente.

**Aula 94**

Mostrado o método estático ::all();. Esse método recupera todos os elementos cadastrados em um banco de dados, facilitando a sua visualização. No tinker, é possível declarar use \*namespace completo da model\* e nas próximas utilizações da model, não é necessário o namespace completo, somente o nome da model.

**Aula 95**

Utilizado método estático ::find();. Por padrão este método pesquisa em um banco de dados pela id passada do registro. Caso o id passado não seja encontrado, simplesmente não haverá retorno. É possível passar um array de ids, que retornará uma coleção de registros, assim como utilizando o método all().

**Aula 96**

Introduzido o método estático ::where(). Com ele é possível fazer pesquisas simples e avançadas sem a necessidade de pesquisar no banco. Após declarar a model a ser pesquisada, são necessários 3 parâmetros para a utilização do where: ::where(‘\*nome da coluna\*’ , ‘\*operador de comparação(<, >, like)\*’ , ‘\*valor a ser comparado\*’). Quando o operador de comparação for =, é possível omitir o segundo parâmetro, contendo o parâmetro do nome da coluna e do valor a ser comparado. Será sempre retornado um conjunto de registros, mesmo resultando em um ou nenhum. Além disso, somente executando o método estático, será retornado um builder, sem o resultado. Para isso, é necessário a utilização do modificador ->get() ao final do comando.

**Aula 97**

Mostrados os métodos estáticos ::WhereIn() e ::WhereNotIn(). Ambos recebem dois parâmetros, o primeiro sendo o nome da coluna a ser buscada e o segundo, o(s) valor(es) a ser(em) buscado(s). Em caso de mais de um, pode ser utilizado um array, sendo passado os valores. No caso do whereIn, serão retornados os registros que atenderem os valores do segundo parâmetro, já o whereNotIn irpa retornar o oposto. Pode ser utilizado para datas, integers e strings.

**Aula 98**

Os métodos ::whereBetween() e ::whereNotBetween funcionam de maneira parecida com os da aula anterior, porém, além dos próprios valores a serem comparador por igualdade, também serão retornados os valores entre esses registros no caso do whereBetween. No caso do whereNotBetween o oposto será retornado.

**Aula 99**

Mostrado como utilizar mais de um where no tinker. Para realizar tal, basta que ao final do primeiro where, seja passado um próximo utilizando -> e em seguida o where desejado.

**Aula 100**

Quando é utilizado mais de um where em uma busa e espera-se que a operação entre os dois ou mais wheres seja de or e não and (operação padrão), pode-se inserir o or na frente da função de qualquer um dos wheres(orWhere, orWhereIn, orWhereBetween), fazendo com que a operação utilizada seja a de or.

**Aula 101**

Mostrado o método de pesquisa ::whereNull, que retorna os valores nulos, de acordo com a model e coluna selecionada. Logo, a função possui somente um parâmetro, que é o da coluna a ser pesquisada.

**Aula 102**

Exibido os comandos de pesquisa que envolvem datas. Eles são: whereDate(‘\*coluna\*’, ‘\*data\*’), whereDay(‘\*coluna\*’, ‘\*dia\*’), whereMonth(‘\*coluna\*’, ‘\*mês\*’), whereYear(‘\*coluna\*’, ‘\*ano\*’), whereTime(‘\*coluna\*’, ’\*operador de comparação\*’, ‘\*horário\*’). Lembrando que para que as pesquisas de datas funcionem, as colunas devem ser do tipo data e estarem preenchidas corretamente, pois o laravel não tem inteligência de interpretar horários diferentes do padrão.

**Aula 103**

Mostrado o comando whereColumn, que funciona de maneira muito parecida com o where, porém de o terceiro parâmetro ser um valor para a comparação, o whereColumn o compara com outra coluna do mesmo registro, podendo assim comparar se o horário criado da coluna é maior, igual ou menor que o horário de modificação, por exemplo.

**Aula 104**

Mostrado como fazer pesquisas mais específicas utilizando o eloquent. Existem casos em que a buscas podem possuir ‘camadas’: select \* from tabela where (... or ...) and (... and ...), retornando resultados diferentes caso não houvesse os parenteses. Para isso, é possível realizar consultas no eloquent utilizando o método where, passando uma função de callback que possui como parâmetro a prória query, sendo ela a utilizada para realizar as consultas de uma camada, por exemplo: $consulta = Model::where(function($query) { $query->whereIn(...)->orWhereDate(...);})->orWhere(function($query){ $query->whereBetween(...)->whereNull(...);});

**Aula 105**

Para organizar a ordem dos resultados no eloquent, deve ser usar o método ::orderBy(‘\*coluna\*’, ‘\*asc ou desc\*’). Por padrão o modo ascendente é selecionado e pode ser omitido caso for o desejado. É possível também stackar orderBy(), sendo que será ordenado de acordo com o primeiro e denstro dessa organização será pela segunda ordem. É possível utilizar filtros where também em conjunto ao orderBy.

**Aula 106**

Introdução as collections do laravel, sendo possível encontrar a documentação das mesmas no site do laravel.

**Aula 107**

Após realizar o get(), será retornado uma collection, que possui vários métodos, 3 deles são: ->first(), que irá retornar o primeiro elemento da collection; ->last(), que irá retornar o último e ->reverse que irá retornar a collection inteira porém invertida.

**Aula 108**

Mostrados os métodos ->toArray() e ->toJson() do eloquent. Fazem exatamente o que parece. Após utilizar esses dois métodos, não será possível usar os métodos de collections do laravel.

**Aula 109**

Mostrado o método ->pluck(‘\*coluna\*’, ‘\*coluna associativa\*’), que irá retornar somente a coluna inserida. É possível passar um segundo parâmetro que é o da coluna associativa, que assumirá como identificador para os elementos da coluna, exemplo: ->pluck(‘email’, ‘nome’) irá retornar uma collection com [‘santhiago’ => ‘santhiago@contato.com’].

**Aula 110**

Falado um pouco mais sobre a documentação laravel.

**Aula 111**

Mostrado o método save() no eloquent. É possível alterar registros no banco que já foram feitos. Para isso, basta recuperar o registro que se deseja modificar com o método ->find(), depois basta alterar os valores desejados, exemplo: $registro->nome = ‘novo nome’;. Após isso, basta utilizar o método save() que as alterações serão salvas no banco.

**Aula 112**

Usado o método ->fill(). Sua funcionalidade é modificar vários elementos de um registro de uma vez, por meio de um array, sua sintaxe é: $registro->fill([‘\*coluna1\*’ => ‘\*valor1\*’, ‘\*coluna2\*’ => ‘\*valor2\*’, ‘\*coluna3\*’ => ‘\*valor3\*’])->save(). Pronto, registro atualizado.

**Aula 113**

Utilizado o método ->update([‘\*coluna\*’ => ‘\*valor\*’]). Nele os valores passados serão substituídos de acordo com as colunas e os valores do array passsado. É possível também aplicar as alterações de acordo com um where que for utilizado, para várias mudanças ao mesmo tempo, por exemplo.

**Aula 114**

Mostrados os métodos delete() e destroy(). O método delete, serve para deletar um registro de acordo com as clausulas de where utilizadas para receber os registros. Já o método destroy(\*id1\*, \*id2\*, ...) é mais curto, porém serve somente para ids. Uma observação é que ao utilizar qualquer um dos dois métodos, caso seja uma variável que armazena os registros, será retornado um true ou false informando o sucesso ou não da operação, porém, caso esses métodos sejam utilizados em diretamente na model, será então retornado um int informando quantos elementos foram excluídos.

**Aula 115**

Mostrado o softDelete. Faz com que quando um registro seja deletado, na verdade ele seja desativado, tendo uma coluna no banco de dados informando a data da desativação, evitando a exclusão do registro. Para sua implementação, basta que a model possua outra herança, algo que normalmente não seria possível, mas o laravel contorna o problema com o use. Primeiro deve-se passar a classe do softDelete, que fica no diretório Illuminate\Database\Eloquent\SoftDelete, depois devese usar a sintaxe use SoftDelete na model desejada. Após isso deve-se criar outra coluna em uma migration de preferência, utilizando a sintaxe $table->softDeletes(); e $table->dropSoftDeletes() no método down. Pronto, quando algum registro for excluido na determinada tabela, somente será adicionado a data na coluna deleted\_at. Ainda é possível excluir um registro de verdade utilizando a função forceDelete(), com ela o registro será completamente removido da tabela.

**Aula 116**

Mostrado como selecionar e restaurar elementos excluídos com softDelete. Para visualizar os elementos excluídos, pode-se usar o método withTrashed(), que irá retornar tanto os registros excluidos quanto os não excluidos. Utilizando o onlyTrashed, será retornado somente os elementos excluidos. Para restaurá-los, é necessário utilizar um dos comando anteriores, pois somente eles capturam os elementos excluidos e armazenar a collection retornada em uma variável. Após isso, é necessário selecionar a index do registro desejado como se fosse um array, utilizando o método restore(), que irá colocar o valor nulo na coluna deleted\_at, restaurando o registro, exemplo: $collection[0]->restore();.

**Aula 117**

Mostrado os seeders. São classes responsaveis por popular tabelas a fim de testes. Para criar uma seeder, basta utilizar o comando artisan make:seeder \*nome do seeder\*. Após isso acessando a nova classe seeder criada, é necessário instanciar a model que se deseja popular com o método use. Então deve-se criar os registros para popular a tabela, são mostrado 3 modos diferentes. O primeiro é instanciando o objeto utilizando $variavel = new Model () e preenchendo o registro e utilizando o método save no final. O segundo foi feito utilizando o método estático create (Model::create([‘\*array com informações\*’]). O ultimo foi utilizando o método insert direto da classe DB (DB::table(‘\*tabela\*’)->insert([\*array com informações\*]). Também é necessário passar qual model será executada pelo artisan no arquivo DatabaseSeeder.php, informando o comando $this->call(Model::class). Após isso basta executar o comando php artisan db:seed e pronto, tabelas populadas.

**Aula 118**

Criado um novo seeder, porém dessa vez, um para a model SiteContato. Após a criação do seeder,

Para não executar novamente o criado na aula anterior, é possível passar diretamente o nome da seeder desejada, exemplo: php artisan db:seed --class=MinhaSeeder.

**Aula 119**

Criada uma factory, responsável por criar automaticamente quantos registros forem desejados. Para criar uma, basta utilizar php artisan make:factory \*nome da factory\* --model=\*model desejada\*, após cria-la é preciso passar os elementos a serem criados. As factorys do laravel utilizam a biblioteca faker, a documentação está em <https://fakerphp.github.io>. Após isso, deve-se definir o que será passado nas colunas por exemplo: ‘nome’ => $this->faker->name. Após isso deve-se passar chamar a factory criada na seeder da respectiva model com factory (\App\Models\Model::factory()->count(\*vezes desejadas\*->create()). Dessa maneira, na próxima vez que a seeder for executada, os registros serão criados automaticamente.