SUMÁRIO

1. CONCEITOS ARQUITETURAIS DO LARAVEL

O objetivo desse documento é descrever o geral de como o framework Laravel opera, parte a parte – o seu, digamos, "ciclo" de operação. Como referência será utilizada essa seção da documentação.

2. PARTE 1: REQUISIÇÕES

2.1. O ENTRY POINT

O ponto de entrada para todas as solicitações de um aplicativo Laravel é o public/index.php. Todas as solicitações são direcionadas a este arquivo, e ele não contém muito código. Em vez disso, é um ponto de partida para carregar o resto da estrutura.

O index.php carrega a definição do autoloader gerada pelo Composer, e então recupera uma instância do aplicativo Laravel de bootstrap/app.php.

2.2. ARQUIVOS KERNEL: KERNEL HTTP E KERNEL DE CONSOLE

Em seguida, a solicitação de entrada é enviada para o **kernel HTTP**, app/http/kernel, ou **kernel do console**, app/console/kernel, dependendo do tipo de solicitação que está entrando no aplicativo.

Esses dois kernels servem como o local central por onde passam todas as solicitações. Pense no kernel como uma grande caixa preta que representa todo o seu aplicativo. Ele recebe requisições, e retorna respostas.

Uma das ações de inicialização do kernel mais importantes é carregar os provedores de serviço para o aplicativo. Todos os provedores de serviço do aplicativo são configurados no arquivo de configuração config/app.php, na matriz "providers".

3. PARTE 2: CONTAINER DE SERVIÇO

Entendendo "serviço" como um código encapsulado, e que pode ser transmitido entre as partes de um sistema, e entre sistemas, o container de serviço Laravel é uma **ferramenta poderosa para gerenciar dependências de classes e executar injeção de dependências**.

Mas isso é uma explicação super rasa e genérica. Para aprofundar o tema, são necessários mais alguns conceitos.

3.1. INJEÇÃO DE DEPENDÊNCIA

A injeção de dependência é uma frase sofisticada que essencialmente significa o seguinte: as dependências da classe são "injetadas" na classe por meio do construtor ou, em alguns casos, dos métodos "setter", ao invés de serem definidas de forma “hard code”. Isso aumenta a flexibilidade da classe dentro do sistema.

Diz-se, assim, que uma injeção de dependência causa uma “inversão de controle”, efeito esse que tem uma sigla no mundo da programação: IoC.

3.2. INVERSION OF CONTROL: IOC

A injeção de dependência é uma abordagem específica de IoC, ou de “inversão de controle”. Isso, “IoC”, é na verdade um princípio de design de sistemas orientados a objetos para que se obtenha módulos fracamente acoplados a outros.

O sentido genérico do termo “inversão” no nome desse design de software, é porque o sujeito principal, que depende de alguma coisa, não controla mais a sua dependência; ele delega a tarefa para outro, externo, um terceiro, e apenas recebe o que precisa dele.

No caso da injeção de dependência, ao invés de uma classe A instanciar, de forma “hard code”, a sua dependência B, ela apenas recebe B. Entendendo a classe A como um módulo, e B como outro, dizemos que A e B estão fracamente acoplados.

3.3. CONTAINER IOC

Um contêiner IoC pode tornar o processo de injeção de dependência mais eficiente. É uma classe simples, capaz de salvar e fornecer dados quando solicitados, ou, em outras palavras, capazes de fornecer um gerenciamento automatizado de inversão de controle.

No caso do Laravel essa tecnologia é utilizada para gerenciar as instâncias das classes que são descritas como “serviços”, e das suas dependências.

3.4. AGORA SIM: ENTENDENDO O CONTAINER DE SERVIÇOS

O container de serviço, então, nada mais é do que um container IoC, e que é utilizado no contexto do Laravel para gerenciar e automatizar as instâncias dos serviços.

Sem grandes detalhes, o que ocorre é que o container de serviço é nutrido, por debaixo dos panos, com as especificações da classe a ser instanciada – coisa possível de ser feita com a classe reflection. Assim, se um serviço X depende de uma classe Y, o container de serviço irá primeiro instanciar a classe Y, e depois a X enviando para o seu construtor a instância de Y.

Isso é realizado no Laravel o tempo inteiro, não apenas para as classes já existentes no framework, na sua inicialização, mas também para as criadas ao longo do desenvolvimento do aplicativo. Por exemplo, os controladores criados para as rotas têm dependências, e elas são resolvidas com esse mecanismo de forma automatizada.

Mas, existem casos em que a vinculação manual ao container de serviço é necessária, e isso nos leva ao próximo subtópico.

3.4.1. QUANDO É NECESSÁRIO DEFINIR UMA CLASSE NO CONTAINER?

Afinal, quando isso será necessário? De acordo com a documentação do Laravel serão raras as ocasiões. São citados nela – na documentação – apenas dois exemplos, mas um deles é o principal: **quando a classe depende de uma interface, ou de qualquer outra coisa que não é obtido por uma instância.**

Esse é um caso nítido. Se digo que o container de serviço automatiza a instanciação das classes, e das suas dependências, estou admitindo, de forma implícita, que as dependências são classes. Mas, se a dependência não se trata de uma classe, mas de outra coisa, o container não conseguirá resolver a classe sozinho; neste caso, precisará de definições adicionais.

3.4.2. COMO ENSINAR AO CONTAINER A RESOLUÇÃO DA NOVA CLASSE?

Para registrar novas classes no container de serviço, e suas ligações, o que no contexto Laravel é chamado de "**binding**", é utilizado o **método "register"** do provider Providers/AppServiceProvider.

Em um provedor de serviços, você sempre tem acesso ao contêiner por meio do termo **$this->app**, seguido do método singleton, para realizar instâncias únicas, e depois do método bind, para realizar a ligação.

O método bind recebe o nome da classe a ser registrada, e uma função, cujo parâmetro $app é o próprio container. O retorno é a dependência da classe.

Veja o exemplo abaixo. A classe Foo depende do nome da aplicação, que é um valor configurado com chave e valor no sistema. Não é uma classe. Então é preciso registrar essa ligação. Isso é feito dessa forma:

```
$this->app->singleton(\App\Foo::class, function ($app) {
    return new \App\Foo($app->config['app.name']);
});
```

4. PARTE 3: PROVEDOR DE SERVIÇO

Os provedores de serviço são o local central de todo o bootstrap de aplicativos Laravel. Eles são responsáveis por inicializar todos os vários componentes da estrutura, como banco de dados, fila, validação e componentes de roteamento.

Essencialmente, todos os principais recursos oferecidos pelo Laravel são inicializados e configurados por um provedor de serviços que é assistido pelo container de serviços.

Se você abrir o arquivo config/app.php verá um array “providers”. Nessa estrutura estão configuradas todas as classes de provedor de serviço que serão carregadas para o aplicativo na sua inicialização, ou durante a sua execução, quando forem requisitadas.

5. PARTE 5: FACADES

Facade é um padrão de projeto, que pode ser encontrado em livros ou cursos sobre Design Patterns. Por conceito, basicamente uma Façade é um objeto que provê uma interface simplificada para um corpo de código maior.

No caso do Laravel, podemos fazer a analogia dos facades com métodos estáticos, embora não sejam, na verdade. Eles são responsáveis por preparar e retornar uma instância de alguma classe disponível no conteiner de serviços, resolvendo todas as dependências da classe e retornando uma instancia.

Como diz na documentação, os facades fornecem uma interface "estática" para classes que estão disponíveis no contêiner de serviço do aplicativo, fornecendo acesso a quase todos os recursos do Laravel com o benefício de uma sintaxe concisa e expressiva.

```
use Illuminate\Support\Facades\Cache;

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

    return Cache::get('key');

});
```

FONTES

https://laravel.com/docs/8.x/lifecycle

https://dev.to/fhsinchy/laravel-service-container-and-service-providers-explained-5a1

https://www.tutorialsteacher.com/ioc/inversion-of-control

https://stackoverflow.com/questions/49348681/what-is-a-usage-and-purpose-of-laravels-binding