5 Desenvolvimento para servidores 1

Durante o quarto semestre na disciplina de Desenvolvimento para Servidores 1, foi proposto aos alunos a criação de uma aplicação de gestão de livros de uma livraria, esse projeto foi idealizado e feito com a linguagem de programação PHP, e no meu caso utilizando o framework Laravel, o framework mais utilizado no desenvolvimento PHP para backend de aplicações.

Na criação do projeto foi utilizado todo o ecossistema do framework escolhido e por isso tive o auxílio de diversas ferramentas, como um sistema de rotas completo, o Eloquent ORM utilizado para poder utilizar o banco de dados sem precisar escrever queries que deixam o código menos legível e por fim um sistema de gerenciamento de filas, que é muito utilizado para executar funções em horários específicos e não depender da influência humana para o seu acionamento.

Por ser um framework o Laravel tem um formato de projeto já estipulado por si próprio, no caso desse framework ele é completamente desenvolvido utilizando o sistema MVC, esse sistema/design pattern, separa os componentes da aplicação em 3 partes a Model (M) que faz toda a parte de conexão com o banco de dados e na maioria das vezes também serve como entity que é a classe de construção das tabelas do banco, além das *Models*, temos os Controllers (C) que serve como uma espécie de garçom e alimenta e faz a conexão entre as views e os dados, ou seja, o controller serve para poder especificar para a *Model*, o que as views precisam em relação de dados, além de também tratá-los para poder ter uma disposição dos dados de uma forma mais amigável e sucinta para as view, e por fim temos as Views (V) que são nada mais nada menos que a camada de visualização da aplicação, ou seja, a porta de entrada da aplicação, onde o usuário tem acesso às funcionalidades do programa, site ou aplicativo.

No caso do projeto por ele ser uma API e não ter uma interação direta com o usuário, as views são substituídas por rotas, que são nada mais nada menos que as URI 's que a aplicação frontend irá consumir para poder gerar tudo que o usuário precisa.

Esse tipo de arquitetura favorece muito as aplicações pois impede que algum problema em um endpoint/ uri diferente do que o usuário está acessando prejudique ou impossibilite a experiência do usuário com a sua aplicação.

Figura 5.1: Classe de Model de livros

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Books extends Model
{
    use HasFactory;
    protected $fillable=['title','sinopsis','author','gender'];
    protected $guarded=['id','created_at','updated_at'];
}
```

Na figura 5.1, podemos ver a implementação da classe model de livros, onde na primeira linha temos somente a hiper tag <?php que nos faz entender que enquanto aquela tag estiver aberta o código que será escrito será em PHP, logo após na duas linhas seguintes, temos apenas importações padrão para o funcionamento do Laravel, no caso importamos a função HasFactory e a classe *Model* para que possamos instanciar a nossa classe e dizer para o framework que ela é um model de uma tabela do banco de dados. Na linha seguinte está a declaração da classe Books, que está estendendo da classe Model, o que significa que essa classe terá além dos seus atributos, os atributos de sua classe Mãe no caso *Model*. Logo abaixo, temos a indicação de que esse modelo utilizará factories, que nada mais é do que uma classe que serve como fábrica para inserir dados básicos no banco de dados, comumente usada para poder inserir dados de teste. Abaixo temos as declarações das propriedades *fillable* e guarded, que servem para poder mapear as colunas que podem ser preenchidas, e para declarar quais terão dados definidos pelo próprio banco de dados, respectivamente.

Uma outra parte muito importante para o funcionamento e configuração da aplicação no ambiente de desenvolvimento, são as migrações, que quando trabalhado com banco de dados servem para poder criar todas as tabelas necessárias automaticamente, com apenas um comando no terminal, na maior parte das vezes.

Figura 5.2:Classe de criação de migrations

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateBooksTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('books', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->string('sinopsis');
            $table->string('author');
            $table->string('gender');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('books');
    }
}
```

Na figura 5.2 temos a classe que cria uma migration, no caso está criando a migration que cria a tabela de books, nas primeiras linhas da figura temos conceitos já destacados anteriormente, mas a partir da linha que nos dá a criação da função temos conhecimentos diferentes, primeiro temos a chamada ao método estático da classe Schema o create, esse método serve para realmente criar a tabela no banco de dados, dentro desse método estamos passando dois argumentos, o primeiro a string books que é o nome da tabela, e o segundo argumento uma função que recebe um atributo $table do tipo Blueprint, esse atributo é basicamente objeto que carrega os valores das colunas da tabela, o que acontece nas linhas seguintes onde temos o objeto $table, que vai recebendo cada coluna passando o seu tipo com função do próprio eloquent e o seu nome como parâmetro para essa função.Após definida toda a função up que é a de criação de uma tabela, temos a criação da função down, que faz a exclusão da tabela.

Figura 5.3:Função do controller que faz a inserção de dados que vem do frontend

```php
/**
 * Show the form for creating a new resource.
 *
 * @param Request $request
 * @return \Illuminate\Http\Response
 */
public function create(Request $request)
{
    try {
        $book = new Book;
        $book->title = $request->title;
        $book->sinopsis = $request->sinopsis;
        $book->author = $request->author;
        $book->gender = $request->gender;
        $book->save();

        return [
            'status'=>200,
        ];
    } catch (\Throwable $th) {
        return [
            'status'=>400,
            'error'=>$th
        ];
    }
}
```

Na Figura 5.3 temos a função create, que basicamente serve para receber os dados que vem da chamada da api ao backend, esses dados vem através da variável $request, que pega automaticamente automaticamente do body da requisição. Na função create temos toda a construção do objeto que é instância da model da classe que você quer adicionar um novo dado, no caso a chamada ao método save faz essa inclusão esse método pega toda a instância de books e se encarrega de criar toda query insert para guardar os dados no banco de dados.