Programação Web

Informática (1º Ciclo)

Rui Ribeiro <rmribeiro@ismai.pt>

March 14, 2019

Instituto Universitário da Maia

Table of contents

Introdução

Laravel

Laravel é uma framework PHP para desenvolvimento de aplicações web. Segue o paradigma de programação orientada a objetos e implementa o padrão MVC *(Model View Controller)*.

MVC I

Model View Controller é um padrão de desenvolvimento que divide a aplicação em três camadas distintas:

Model
O modelo representa os dados da aplicação, regras de negócio, bem como, toda lógica associada ao seu comportamento. *Claro que devemos extrair funcionalidades e responsabilidades para não se obter modelos impossíveis de manter.*

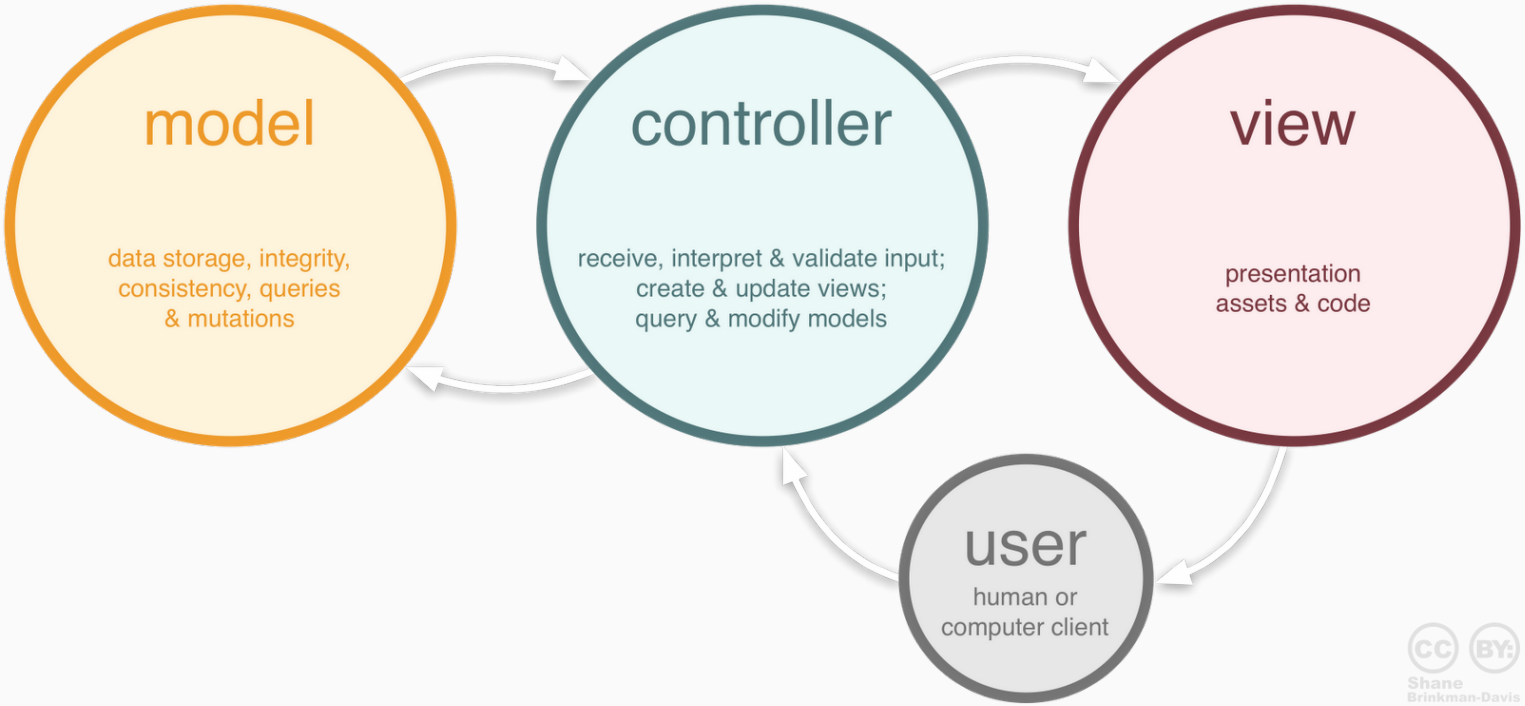
View
A visualização está associada à representação e apresentação dos dados. No caso das aplicações web HTML, CSS e JavaScript. *Assim como os modelos, a visualização podem ser dividida em componentes.*

MVC II

Controller
O controlador, ponto de entrada na aplicação, responde a eventos e articula as chamadas entre *models* e *views* encaminhando os pedidos que recebe.

Finalidade?
Promover separação de conceitos, reutilização de código, orientar o programador a uma estruturação clara do código. Estabelecer *guidelines* que permitam melhor comunicação dentro de uma equipa de desenvolvimento.

MVC III



Rotas

Na web as rotas definem *endpoints*, *URI (Universal Resource Identifier)* ou URL (Universal Resource Locator) através dos quais a uma aplicação deve responder.

```
1  Route::get('/', function () {
2      return view('welcome');
3  });
4
5  Route::get('foo', 'Photos\AdminController@method');
6
7  Route::resource('photo', 'PhotoController');
```

Templates

As templates fazem parte da camada de apresentação, são utilizadas para geração de páginas dinâmicas. Dentro das templates ainda temos *layouts* e templates parciais.

Em Laravel o sistema de templates é designado de Blade.

```
1  <h1>Laravel</h1>
2
3  Hello, {{ $name }}.
```

ORM (Object Relational Mapper)

Como ORM, Laravel fornece uma biblioteca designada de Eloquent que permite fazer o mapeamento entre os modelos e o repositório de dados.

No caso de uma base de dados relacional permite-nos o mapeamento entre tabelas e modelos (classes) com integração dos respetivos mapeamentos. Para além disso, abstrai o SQL (Structured Query Language) uma vez que é utilizado PHP para expressão das consultas.

```
1  Student::where('active', 1)->orderBy('name')->take(10)->get();
```

Laravel

Estrutura de Pastas I

App
Alberga toda a lógica da aplicação, rotas, controladores, modelos, etc.

Config
Ficheiros de configuração para todos os componentes da framework.

Database
Localização das migrações e classes referentes à inicialização da base de dados (*seeders*).

Routes
Local para definição das rotas utilizadas pela aplicação, embora não seja obrigatório, as rotas podem ser organizadas em ficheiros que representam diferentes tipos de pedido.

Estrutura de Pastas II

Public
Recursos estáticos da aplicações e local a partir de onde serão servidos aos clientes.

Resources
Templates, traduções, CSS e JavaScript para pré-processamento.

Storage
Local onde a framework despeja dados como informação de sessões, cache das páginas e logs.

Vendor
Bibliotecas das quais a framework depende.

Rotas

routes/{web.php, api.php, *}
Ponto de entrada das rotas conhecidas pela aplicação. Nestes ficheiros descrevem-se as rotas e a forma como devem ser tratadas ou encaminhadas para os controladores.

```
1  Route::get('/', function () {
2      return view('welcome');
3  });
4
5  Route::get('product/{id}', 'ProductController@show');
6  Route::get('welcome', 'WelcomeController@index');
7  Route::resource('team', 'TeamController');
8  Route::resource('team.result', 'ResultController');
9
```

Controladores

app/Http/Controllers

Seguindo as *guidelines* da framework os controladores devem ser colocados nesta pasta. Claro que podem ser organizados em sub-pastas ou mesmo organizados de outra forma.

```
1  <?php namespace App\Http\Controllers;
2
3  class SiteController extends Controller {
4      public function index() {
5          return view('index');
6      }
7  }
```

Os controladores podem ser criados manualmente ou utilizando o comando *artisan*.

```
1  php artisan make:controller SiteController
```

Resource Controllers I

Figure 1: Route::resource('team', 'TeamController')

Method	URI	Name	Action
POST	team	team.store	TeamController@store
GET	team	team.index	TeamController@index
GET	team/create	team.create	TeamController@create
DELETE	team/{id}	team.destroy	TeamController@destroy
PUT/PATCH	team/{id}	team.update	TeamController@update
GET	team/{id}	team.show	TeamController@show
GET	team/{id}/edit	team.edit	TeamController@edit

Resource Controllers II

```php
<?php namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TeamController extends Controller
{
    public function index() {}

    public function create() {}

    public function store(Request $request) {}

    public function show($id) {}

    public function edit($id) {}

    public function update(Request $request, $id) {}

    public function destroy($id) {}
}
```

resources/views
Esta é localização das templates segundo a configuração da framework. Neste diretório a estrutura e organização das templates compete ao programador, é comum organizar por controlador, onde o nome do controlador dá origem a uma pasta que contém as respetivas *views*.

Em Laravel o sistema de templates designa-se de Blade, os ficheiros devem manter a extensão **.blade.php**, caso contrário serão avaliados como PHP.

Templates

Blade tenta simplificar a criação de templates apresentando um dialeto mais amigável que PHP.

Todas as expressões Blade começam com **@**, por exemplo @{{ }}, @{!! !!}, @section, @if, @for.

Blade suporta as mesmas condições e ciclos que PHP, @if, @elseif, @else, @for, @ foreach, @while e adiciona funcionalidades como @section, @extends, @yield e @include.

```
1  <ul>
2  @foreach ($users as $user)
3      <li>Hello {{ $user->name }}</p></li>
4  @endforeach
5  </ul>
```

Layouts I

Os layouts permitem definir *master templates* a serem utilizadas pelas views, um pouco como quando utilizamos herança na programação orientada a objetos.

```
1  <!doctype html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>@yield('title')</title>
6      </head>
7      <body>
8          @yield('content')
9      </body>
10 </html>
```

Onde observamos **@yield** será injetado conteúdo das templates.

Layouts II

Para herdar do layout utiliza-se a diretiva **@extends**, para injetar o conteúdo pretendido utiliza-se a diretiva **@section**.

```
1  @extends('layouts.frontend')
2
3  @section('title', 'Listagem Utilizadores')
4
5  @section('content')
6      <ul>
7          @foreach ($users as $user)
8              <li>Hello {{ $user->name }}</p></li>
9          @endforeach
10     </ul>
11 @endsection
```

Views Parciais

Para situações onde se pretende reutilizar parte de uma view a diretiva **@include** ou **@each** são úteis.

@include
Inclusão de uma view com ou sem passagem de dados (segundo argumento). Dados são opcionais, uma vez que os herda da view principal: *@include('shared.user')*.

@each
Idêntica à diretiva @include, mas com a particularidade de iterar por todos os valores do array passado como segundo argumento: *@each('shared.user', $users, 'user')*.
Pode ainda levar um quarto argumento que indica a view a carregar caso o array esteja vazio.

Migrações I

database/migrations

As migrações permitem, de certa forma, visualizar a evolução do esquema da base de dados e as alterações feitas em determinado ponto. São compostas por código PHP que quando executado é transformado em SQL e corrido na base dados.

O corpo de uma migração é composto por um método **up**, **down**, bem como, as respetivas alterações definidas com **Schema::create** ou **Schema::table**.

Comandos

```
1  php artisan make:migration nome_da_migração
2  php artisan migrate
3  php artisan migrate:rollback
```

Migrações II

```
1  public function up() {
2    Schema::create('entries', function (Blueprint $table) {
3        $table->increments('id');
4        $table->string('title', 255);
5        $table->string('slug')->nullable();
6        $table->tinyInteger('status')->default(1);
7        $table->integer('user_id');
8        $table->timestamps();
9
10       $table->foreign('user_id')->references('id')->on('users');
11       $table->index('slug', 'idx_entries_slug');
12   });
13 }
14
15 public function down() {
16   Schema::drop('entries');
17 }
```

Modelos

app

Finalmente temos o **M** do padrão **MVC**. Em Laravel seguem o padrão **Active Record** servindo de ponte entre a nossa aplicação e a base de dados. Para além de armazenarem informação do sistema são também utilizado para exprimir queries.

Variáveis Importantes

$fillable, $guarded, $table, $primaryKey

Comandos

```
1  php artisan make:model nome_do_modelo
```

Modelos Queries

```php
Entry::find(1);
Entry::findOrFail(1);

$entries = Entry::all()

foreach ($entries as $entry) {
    echo $entry->name;
}

Entry::where('status', 1)
    ->orderBy('title', 'desc')
    ->take(5)
    ->get();

Entry::where('id', '>', 100)->firstOrFail();
```

Relações

Através dos seguintes métodos podemos definir relações entre as tabelas, note que estas chamadas devem ser feitas numa instância de um modelo. Por exemplo,

```
1  public function entries() {
2      $this->hasMany('App\Entry');
3  }
```

- hasOne
- hasMany
- belongsTo
- belongsToMany

Forms

TODO csrf_field method_field old

Validation

app/Http/Requests

```
1  public function rules()
2  {
3      return [
4          'title' => 'required|max:255',
5          'slug' => 'required|unique:entries',
6      ];
7  }
```

Comandos

php artisan make:request StoreEntryRequest

References I


Laravel framework.
`http://laravel.com/.`


Model view controller pattern (mvc).
`https://ist.berkeley.edu/as-ag/pub/pdf/mvc-seminar.pdf.`


M. Fowler.
Patterns of Enterprise Application Architecture.
Addison-Wesley Longman Publishing Co., Inc., 2002.