IPL
escola superior
de tecnologia e gestão
instituto politécnico
de leiria

Laravel - 1

Laravel Basic Concepts with a Demonstration

Marco Monteiro

Contributors

- Author(s):
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

Summary

1. Composer
2. Starting & Artisan
3. Architecture & Structure
4. Simple demo
5. Summary
6. References

1 – COMPOSER

Composer

- Composer is a package manager for PHP that install and handles dependencies of php packages
  - Installs packages/libraries in a directory inside the project ("vendor")
  - Finds out which versions of which packages/libraries need to be installed
  - Prepares a file for autoloading all of the classes in any of the libraries that it downloads
  - Project's dependencies stored in file **composer.json**

Composer

- Alternative composer syntax:

```
composer <…>
```

```
php composer.phar <…>
```

- Create a Laravel project with composer:

```
composer create-project --prefer-dist laravel/laravel prj
```

Composer – some useful commands


```
composer install
```

```
composer update
```

Read composer.json file and install/update all required packages (including all dependencies)

```
composer require nameof/package
```

Add a package to the composer.json file

```
composer remove nameof/package
```

Removes a package from the composer.json file

```
composer self-update
```

Updates the composer itself

```
composer dump-autoload
```

Update the autoloader (*it is known to solve some issues during development*)

2 – STARTING & ARTISAN

Clean Laravel Installation

- With composer:

```
composer create-project --prefer-dist laravel/laravel prjName
```

Specifying Laravel Version (e.g.: latest Laravel 9 version):

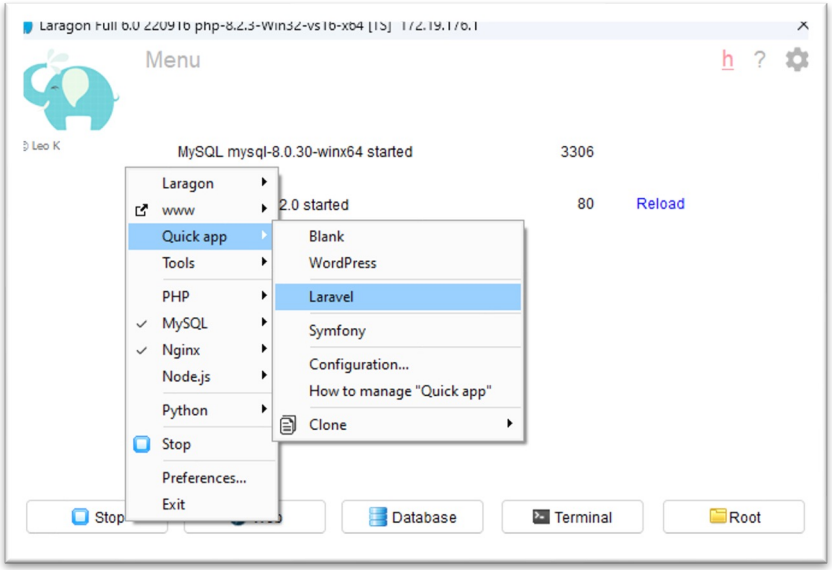
```
composer create-project laravel/laravel prjName  "9.*" --prefer-dist
```

- With docker (Laravel Sail):

```
curl -s "https://laravel.build/prjName" | bash
```

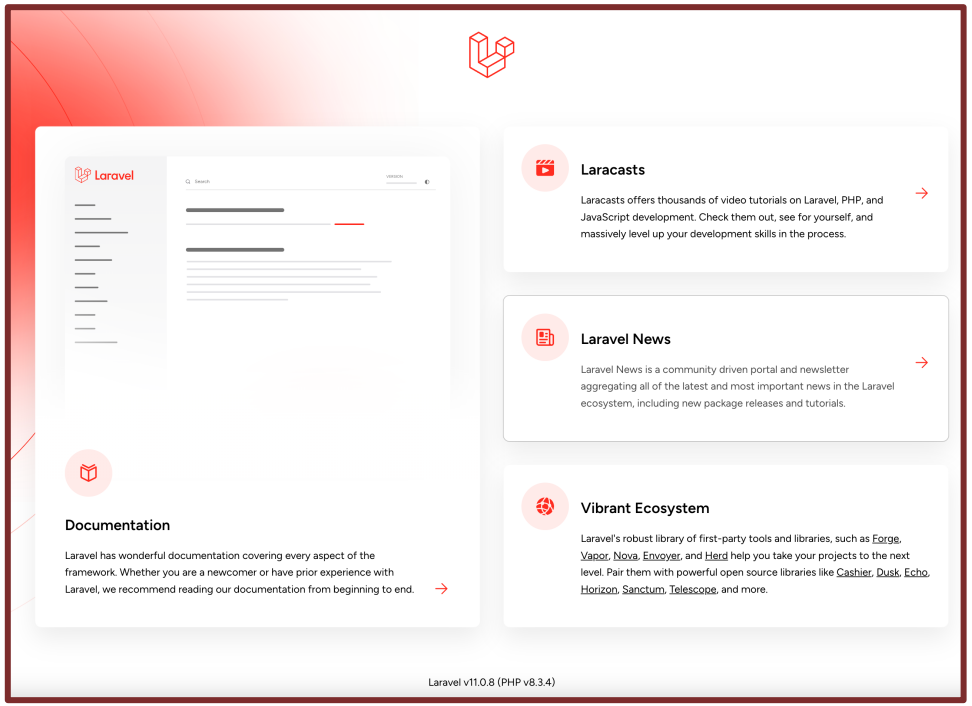
- With Laragon "Quick App":

Right-click on Laragon application and choose the option "Quick app" / "Laravel"



Clean Laravel Installation

- After a clean installation, Laravel application should display the home page (content depends on the Laravel version):



- Note: application must be deployed on a Web Server
- Alternatively, if your local machine has the correct version of PHP installed, it is possible to *"execute"* the Laravel application with this command:

```
php artisan serve
```

Use this alternative (php artisan serve) only on simple projects

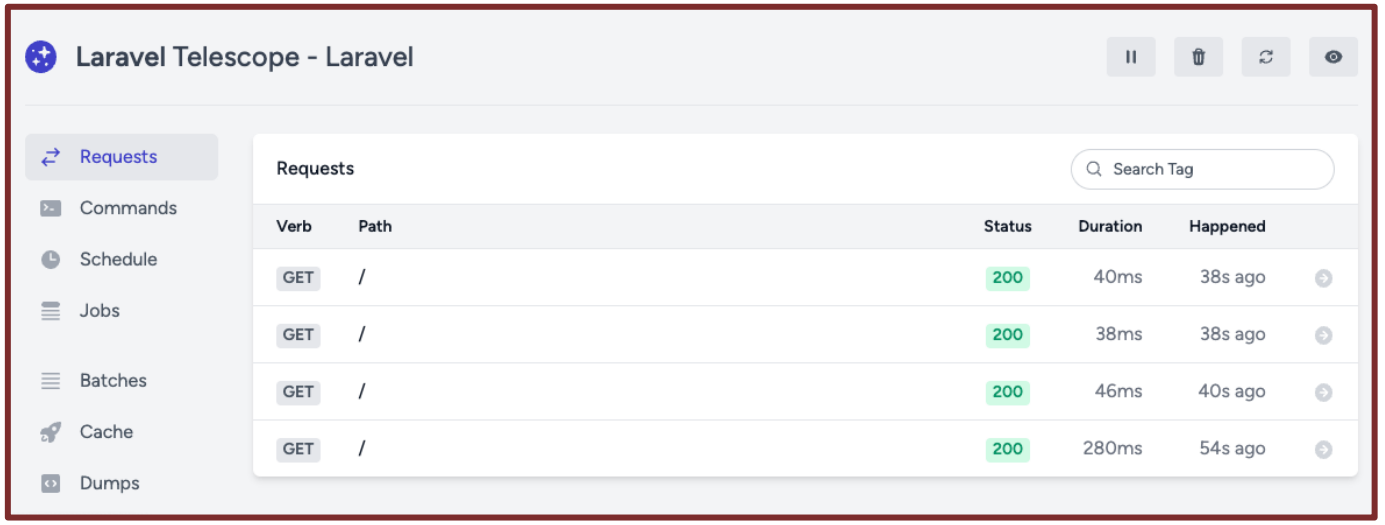
Laravel Telescope

- Optional: install "Laravel Telescope" on the project (*only locally*), with the following commands:

```
composer require laravel/telescope --dev
php artisan telescope:install
php artisan migrate
```

  - Analyze how the HTTP request is handled by the Laravel framework.

```
http://your-prj-domain/telescope
```



Artisan

- Artisan is the command-line interface (**CLI**) included with Laravel, with commands to assist application development.
- Example of some useful artisan commands:

```
php artisan list
```

List all available artisan commands

```
php artisan serve
```

Serve the application on the PHP development server

```
php artisan clear-compiled
```

Remove the compiled class files

```
php artisan view:clear
```

Clear all compiled view files

Artisan


```
php artisan make:<something>
```

Creates a new resource/file/class/etc... Examples:

```
php artisan make:controller UserController
```

```
php artisan make:model Product
```

```
php artisan route:list
```

List all registered routes of the application

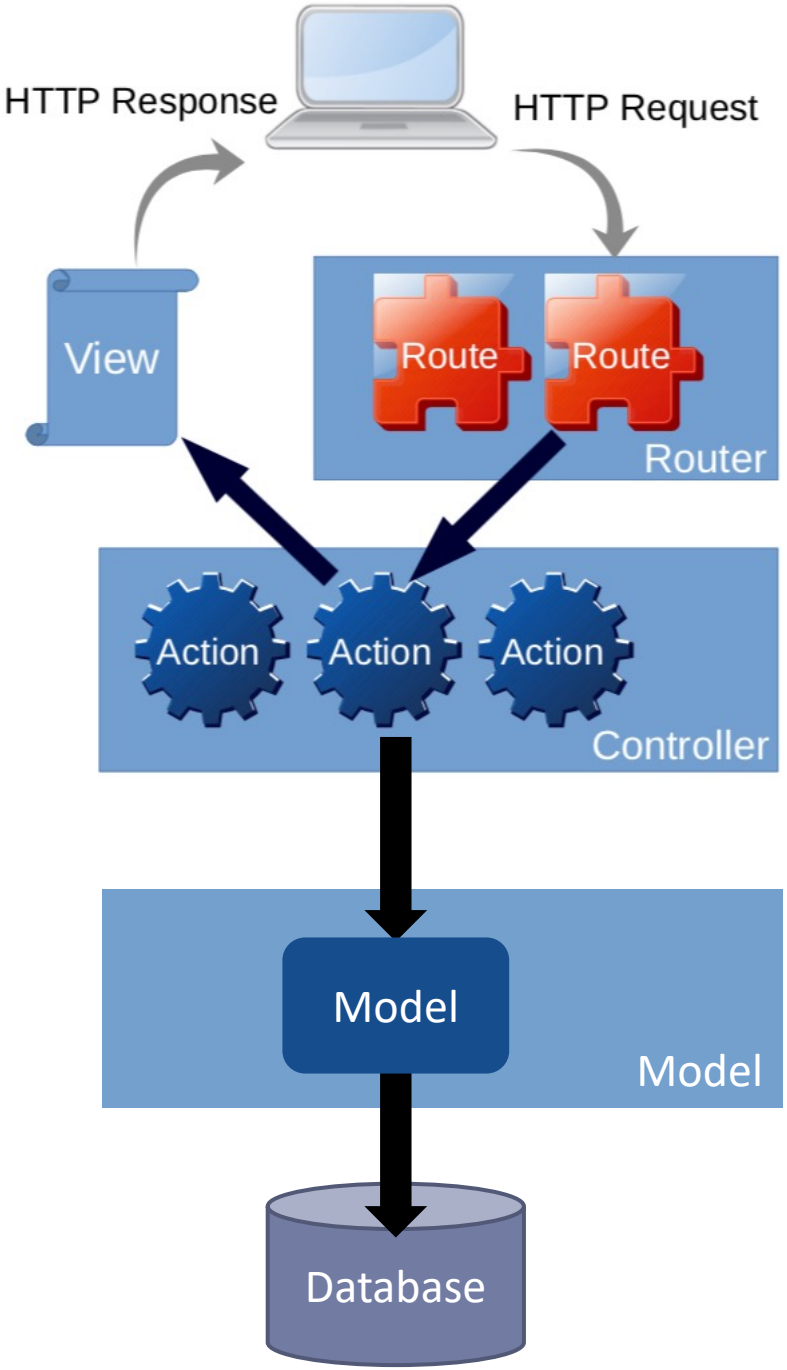
```
php artisan route:clear
```

Remove the route cache file

3 – ARCHITECTURE & STRUCTURE

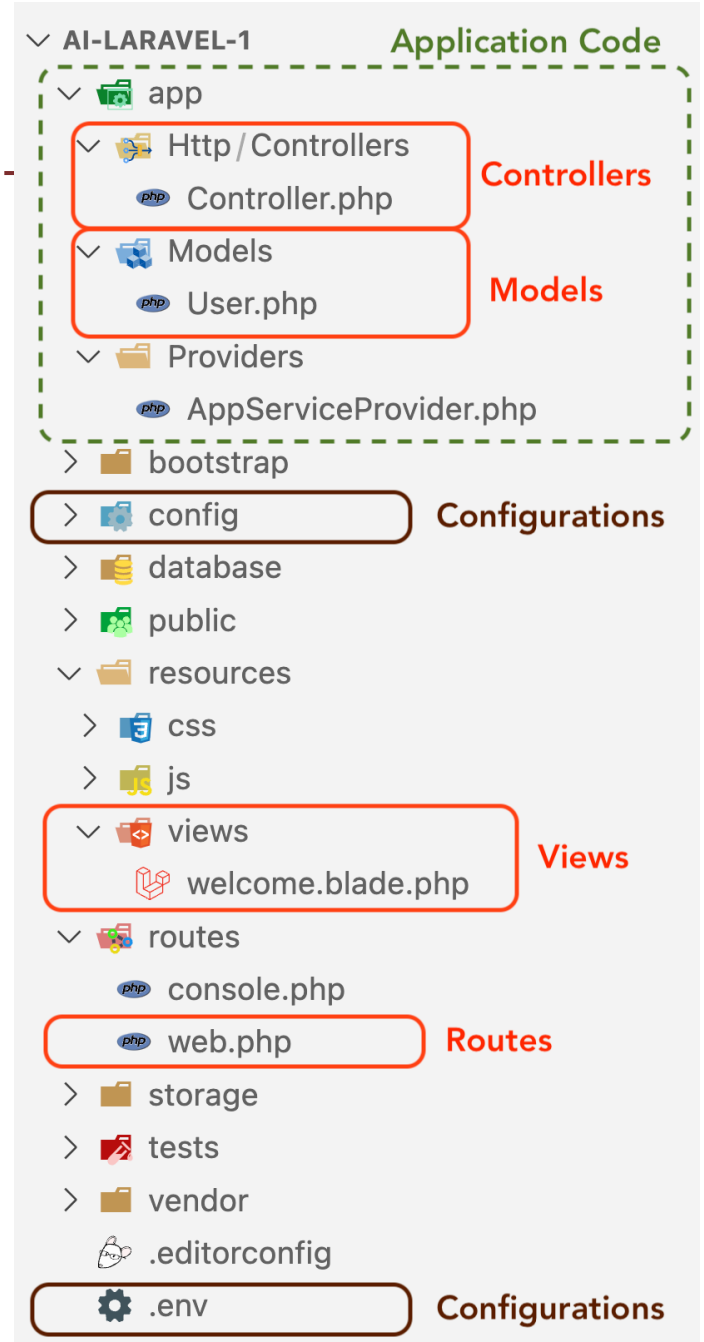
Architecture

- Laravel is an **MVC** (**M**odel-**V**iew-**C**ontroller) based Web framework.
- Includes a Router component (Routing)
  - Maps HTTP requests to controller actions
  - Defines which controller action will handle an incoming HTTP request



Basic Structure

- Controllers
  - `app/Http/Controllers`
- Models
  - `app/Models`
- Views
  - `resources/views`
- Routes
  - Route registration file for Web applications:
  - `routes/`**`web.php`**
- Configuration files
  - **`.env`** file
  - `config` folder
- Public folder
  - What is directly "*accessible*" by the client
  - `index.php` – bootstraps (startup code) Laravel (*"executed" on every HTTP request*)



4 – SIMPLE DEMO

Simple demo application

- Tutorial to build a very simple application to manage a list of disciplines
  - Discipline's list
  - Add discipline
  - Edit discipline
  - Delete discipline

Database

- Create a clean Laravel project
- Database configuration:
  - For this project, we'll create an empty database – use your preferred MySQL database administrator application
  - *Note: when creating the Laravel project with "Laragon quick apps", a database with the same name as the application is also created*
  - On the configuration file **.env** file (on the root of the laravel project) configure the database to use on the project. Examples:

```
DB_CONNECTION=mysql
DB_HOST=mysql
. . .
DB_DATABASE=demo1
DB_USERNAME=sail
DB_PASSWORD=password
```

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
. . .
DB_DATABASE=demo1
DB_USERNAME=root
DB_PASSWORD=
```

Database

- Remove the database folder and replace it with the provided database folder (includes custom migrations and custom database seeders)
- Database migration – create/update DB structure
  - Execute the following command:

```
php artisan migrate
```

  - If it does not work, execute the command: "php artisan migrate:fresh"
- Database seeder – add data (real or fake) data to DB
  - Execute the following command:

```
php artisan db:seed
```

  - If it does not work, execute the command : "composer dump-autoload" and then repeat the command "php artisan db:seed"

Models

- Use artisan to create the Discipline model

```
php artisan make:model Discipline
```

- It will create the file: "app/Models/Discipline.php", with the following code:

```
<?php

namespace App\Models;
. . .

class Discipline extends Model
{
    . . .
}
```

Models

- **Eloquent** is the ORM technology used by Laravel
- "Discipline" class extends

`\Illuminate\Database\Eloquent\Model`

  - This means that it already has methods to retrieve
    - e.g. Discipline::all() or Discipline::where(…)->get() -
    insert, update, delete rows from the table “disciplines”

  - By convention, "**Discipline**" model will be associated with "**disciplines**" table, and it will have the same attributes (properties) as "disciplines" table columns

Controllers

- Use artisan to create the DisciplineController

```
php artisan make:controller DisciplineController
```

- It will create the file:
  "app/Http/Controllers/DisciplineController.php"

Controllers

- Code for the DisciplineController, with the method "index":
  - Gets data from the model (*all "disciplines"*)
  - Creates a view and passes data to it (2 variables: $disciplines and $pageTitle)
  - Returns the view

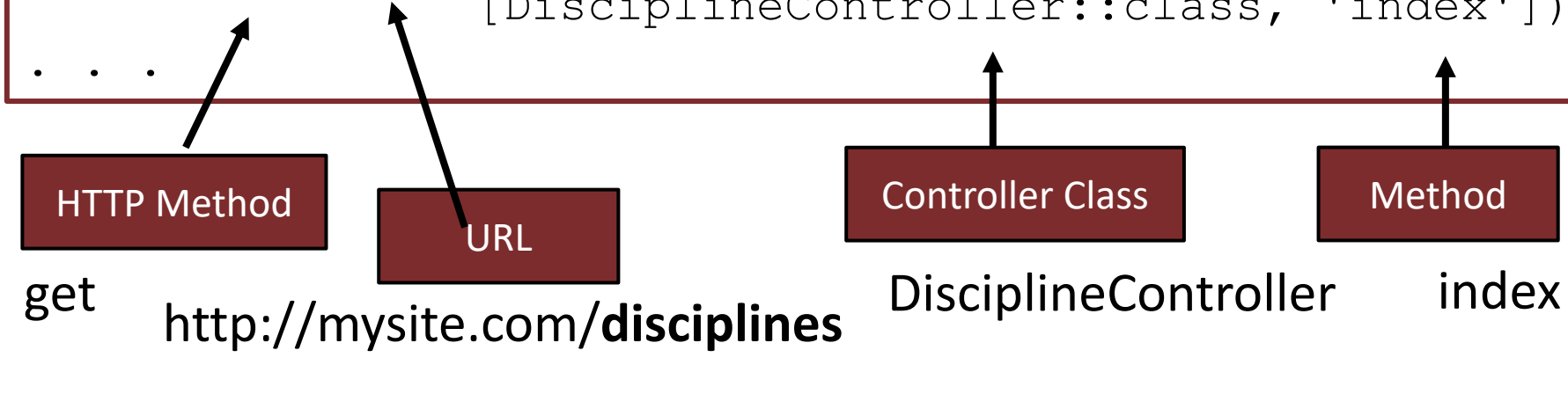
```
<?php
namespace App\Http\Controllers;
. . .
use App\Models\Discipline;
use Illuminate\View\View;

class DisciplineController extends Controller
{
    public function index(): View
    {
        $disciplines = Discipline::all();
        return view('disciplines.index')
            ->with('disciplines', $disciplines)
            ->with('pageTitle', 'Disciplines List');
    }
}
```

Routing

- Web Application routes registration is defined on "**routes/web.php**" file.
- Add a route to execute the "index" method of the "DisciplineController", when the server receives a HTTP GET request with the URL: "/disciplines"

```
. . .
use App\Http\Controllers\DisciplineController;
. . .
Route::get('disciplines',
                [DisciplineController::class, 'index']);
. . .
```



Views

- Views are not created with the "artisan" tool.
  We must create the view files manually
  - Create the file "template.blade.php" on the "**resources/views**" folder
  - Create the folder "resources/views/disciplines"
  - Create the file "index.blade.php" inside newly created folder

- Note: Views will use the **blade** engine for rendering
  - Has specialized instructions to simplify rendering definition
  - Implements a template mechanism
  - Blade view file names follow the format:

`view_name`**`.blade.php`**

Views

- index.blade.php (1/2)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{{ $pageTitle }}</title>
    <style>
        table, th, td {
            border: 1px solid black;
            border-collapse: collapse;
        }
    </style>
</head>
<body>
<h1>{{ $pageTitle }}</h1>
<p><a href="disciplines/create">Create a new discipline</a></p>
<table>
<thead>
    <tr>
        <th>Abbreviation</th>
        <th>Name</th>
        <th>Course</th>
        <th>Year</th>
```

Views

- index.blade.php (2/2)

```
        <th>Semester</th>
        <th></th>   <th></th>   <th></th>
    </tr>
</thead>
<tbody>
    @foreach ($disciplines as $discipline)
    <tr>
        <td>{{ $discipline->abbreviation }}</td>
        <td>{{ $discipline->name }}</td>
        <td>{{ $discipline->course }}</td>
        <td>{{ $discipline->year }}</td>
        <td>{{ $discipline->semester }}</td>
        <td> <a href="disciplines/{{ $discipline->id }}">View</a> </td>
        <td> <a href="disciplines/{{ $discipline->id }}/edit">Edit</a> </td>
        <td> <form method="POST" action="disciplines/{{ $discipline->id }}">
                @csrf
                @method('DELETE')
                <button type="submit" name="delete">Delete</button>
            </form> </td>
    </tr>
    @endforeach
</tbody>
</table> </body> </html>
```

Simple application

- Execute – http://mysite/disciplines

Disciplines List

Create a new discipline

Abbreviation	Name	Course	Year	Semester			
AL	Linear Algebra	EI	1	1	View	Edit	Delete
AM	Mathematical Analysis	EI	1	1	View	Edit	Delete
FA	Applied Physics	EI	1	1	View	Edit	Delete
Prog I	Programming I	EI	1	1	View	Edit	Delete
SC	Computer Systems	EI	1	1	View	Edit	Delete

Views - Templates

- **Template** (file = "template.blade.php")
  - Defines the main structure of the HTML content
  - Has the content that is common to all views

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{{ $pageTitle }}</title>
    <style>
        table, th, td {
            border: 1px solid black;
            border-collapse: collapse;
        }
    </style>
</head>
<body>
<h1>{{ $pageTitle }}</h1>
<div>
   @yield('content')
</div>
</body>
</html>
```

Views - Templates

- index.blade.php (using the template) 1/2

```
@extends('template')
@section('content')
 <p><a href="disciplines/create">Create a new
discipline</a></p>
    <table>
        <thead>
            <tr>
                <th>Abbreviation</th>
                <th>Name</th>
                <th>Course</th>
                <th>Year</th>
                <th>Semester</th>
                <th></th>   <th></th>   <th></th>
            </tr>
        </thead>
        <tbody>
```

Views - Templates

- index.blade.php (using the template) 2/2

```
    @foreach ($disciplines as $discipline)
    <tr>
        <td>{{ $discipline->abbreviation }}</td>
        <td>{{ $discipline->name }}</td>
        <td>{{ $discipline->course }}</td>
        <td>{{ $discipline->year }}</td>
        <td>{{ $discipline->semester }}</td>
        <td> <a href="disciplines/{{ $discipline->id }}">View</a> </td>
        <td> <a href="disciplines/{{ $discipline->id }}/edit">Edit</a> </td>
        <td> <form method="POST" action="disciplines/{{ $discipline->id }}">
                @csrf
                @method('DELETE')
                <button type="submit" name="delete">Delete</button>
            </form> </td>
    </tr>
    @endforeach
</tbody>
</table>
</body>
</html>
@endsection
```

Views – Blade Engine

- Some notes about the blade engine
  - Template view has 1 or more "*inject*" points, defined with `@yield('sectionName')`
  - Views use a template, by extending it with `@extends('templateName')`
  - View sections will be "*injected*" at one "*inject*" point in the template, with `@section('sectionName')`
  - `{{$a}}` - replaces `<?=$a?>` or `<?php echo $a?>`

    `{{$a}}` - sanitizes (uses htmlspecialchars) value of $a

    Note: `{!!$a!!}` - $a value is not sanitized.

    Try both with $var = "something"
  - Blade includes instructions like `@foreach`; `@if`; `@for`; `@while`; `@switch`; `@isset`; `@method` ; etc . . .

Simple application

- Execute – http://mysite/disciplines

Disciplines List

Create a new discipline

Abbreviation	Name	Course	Year	Semester			
AL	Linear Algebra	EI	1	1	View	Edit	Delete
AM	Mathematical Analysis	EI	1	1	View	Edit	Delete
FA	Applied Physics	EI	1	1	View	Edit	Delete
Prog I	Programming I	EI	1	1	View	Edit	Delete
SC	Computer Systems	EI	1	1	View	Edit	Delete

Add Operation – Model

"Discipline" model extends from Eloquent Model, which means that it already implements insert operations

However, to support bulk insert operations (which we will use), we have to change the Discipline model:

```
<?php
namespace App\Models;
. . .
class Discipline extends Model
{
    . . .
     public $timestamps = false;

     protected $fillable = [
       'course', 'year', 'semester', 'abbreviation', 'name',
       'name_pt', 'ECTS', 'hours', 'optional',
     ];
}
```

Add Operation – Model

- Some notes about the Discipline model
  - `$fillable` – attribute of the model that defines (using an array) which fields will be included on bulk database operations
    - E.g. Discipline::create($array_data) is a bulk operation, which we will use to create a new discipline (Eloquent ORM will translate this operation to an insert command)
  - `$timestamps = false`

    By default, the model expect that the associated table has 2 timestamps fields ("created_at" and "updated_at").

    If your table does not have these fields (which is the case), we must set the attribute "$timestamps" to false.

    If this attribute value is true (the default value), the insert or update operations of “Discipline” would fail, as they expect the presence of the timestamp fields

Add Operation - Routes

- 2 new routes:
  - get "disciplines/create" -> render the form to create
  - post "disciplines" -> when user submits the form to create a new "discipline"

```
. . .
Route::get('disciplines/create',
           [DisciplineController::class, 'create']);

Route::post('disciplines',
           [DisciplineController::class, 'store']);
```

Add Operation - Controller

- Add 2 new methods to the DisciplineController
  - **create** – shows the form to create a new discipline
  - **store** – creates a new discipline on the DB (through the model) – invoked when user submits the form

```
. . .
public function create(): View
{
    return view('disciplines.create')->withPageTitle('Add Discipline');
}
```

this passes a variable named $pageTitle to the view

- *Alternatives to pass variables to the view:*

```
return view('disciplines.create')->with('pageTitle', 'Add Discipline');
```

```
$pageTitle = 'Add Discipline';
return view('disciplines.create', compact('pageTitle'));
```

```
return view('disciplines.create', ['pageTitle' => 'Add Discipline']);
```

Add Operation - Controller

- **store** – invoked when user submits the form
  - Stores (inserts) the inputted data on the database
  - After a success insert operation, returns a **redirect** header to the client
    - Note: after receiving the redirect header on the HTTP response, the client (browser) will automatically send a new HTTP GET request to the URL specified on the redirect header

```
. . .
use Illuminate\Http\RedirectResponse;
. . .
public function store(Request $request): RedirectResponse
{
    Discipline::create($request->all());
    return redirect()->action([DisciplineController::class, 'index']);
}
```

Add Operation - Controller

Some notes about the store method

- `store(Request $request)` - $request is an instance of the current HTTP request. It is obtained via dependency injection.
  - When a method of the controller has a parameter of Request type, Laravel injects an instance of Request to it. Otherwise, $request is not available
- `$request->all()` - returns an array with all field values of the form (all GET and/or POST parameters on the HTTP Request)
- `Discipline::create($array_data)` - inserts a record on "disciplines" table. Data to be inserted is defined by the $array_data argument (array with all required fields)
- `return redirect()->action([DisciplineController::class, 'index'])` - returns an HTTP response with information for the client (browser) to redirect to another page – this is done by sending a "location" header
  - action() - returns the url associated to the route that has the specified action

Add Operation - View

- View “disciplines/create.blade.php”

```
@extends('template')
@section('content')
<form action="{{ action([App\Http\Controllers\DisciplineController::class,
'store']) }}" method="post">
    @csrf
   <div>
        <label for="inputCourse">Course</label>
        <input type="text" name="course" id="inputCourse">
    </div>
    … repeat for fields: abbreviation, name, year, semester, etc…
    <div>
        <label for="inputOptional">Optional</label>
        <input type="hidden"   name="optional" id="inputOptional" value="0">
        <input type="checkbox" name="optional" id="inputOptional" value="1">
    </div>
    <div>
        <button type="submit" name="ok">Save</button>
        <button type="reset" name="cancel">Cancel</button>
    </div>
</form>
@endsection
```

Add Operation - View

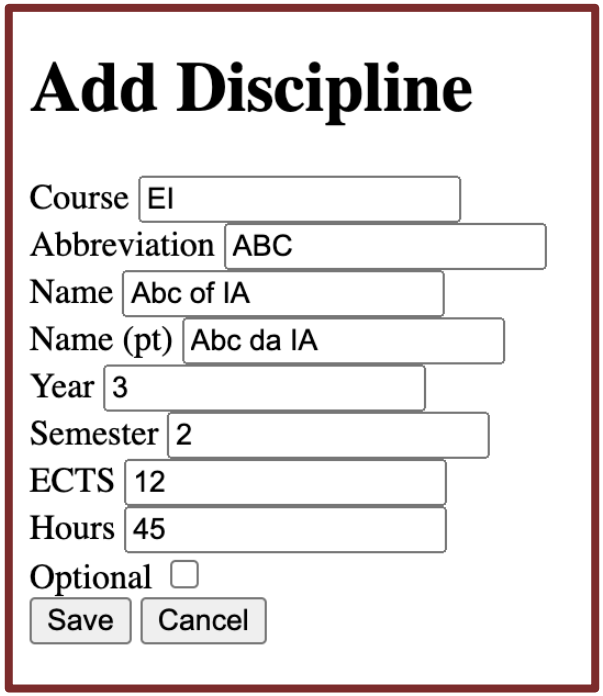
- Some notes about the previous view (with a form)
  - `@csrf` – adds a CSRF "token" to the form
    - To protect from **cross-site request forgery** (CSRF) attacks.
      Cross-site request forgeries are a type of malicious exploit whereby unauthorized commands are performed on behalf of an authenticated user.
  - `action([App\Http\Controllers\DisciplineController::class, 'store'])`
    Returns the URL address that is associated (through the routing mechanism) to the store method of the DisciplineController controller (the method that saves the data)
  - `Form fields names == Table columns names`
    Name of the form fields is the same as the name of the database columns. When saving data, no extra mapping is required

Add Operation - View

- Some notes about the previous view (with a form)
  - **`Trick for checkbox fields`**
    - The problem? When the user does not check the box (when the value is false), the checkbox field is not sent on the HTTP request.
    - The solution: **immediately before** the checkbox field, we add a **hidden** field with the **same name** as the checkbox field and with the **value** that represents **false** (in this case = "0")
    - How does it work?
      - If the user does not check the box (value false), the checkbox is not sent to the server, but the hidden field value is sent to the server (with the same name and with the value that represents false)
      - If the user checks the box, both the hidden field and the checkbox field are sent to the server, with the same name but conflicting values. When that happens, the server assumes the value of the last field passed on to the server - on this case, the server will assume the value that represents true, which is what we want

Add Operation - Test

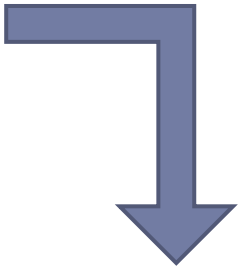
- Execute – http://mysite/disciplines/create
- Try to submit (save button) data



- When user submit data (click on save button) the server saves the new discipline on the DB.

- After correctly saving the record on the DB, server returns a redirect response (header "location").

- Client receives the **redirect** response and call the other page (makes a **new HTTP request**)



Proj	Project	MCD	2	2	View	Edit	Delete
Diss	Dissertation	MCD	2	2	View	Edit	Delete
Estg	Internship	MCD	2	2	View	Edit	Delete
ABC	Abc of IA	EI	3	2	View	Edit	Delete

Add Operation – Validation (Controller)


```
public function store(Request $request): RedirectResponse
{
    $validated = $request->validate([
        'course' =>       'required|exists:courses,abbreviation',
        'abbreviation' => 'required|string|max:20',
        'name' =>         'required',
        'name_pt' =>      'required',
        'year' =>         'required|integer|between:1,5',
        'semester' =>     'required|in:1,2',
        'ECTS' =>         'required|integer|between:1,60',
        'hours' =>        'required|integer|between:1,999',
        'optional' =>     'required|boolean',
    ], [  // Custom Error Messages
        'course.required' => '"Course" is required.',
        'course.exists' => '"Course" is invalid! ' .
            'It does not exist on the database.',
    ]);
 // If something is not valid, execution is interrupted.
 // Remaining code is only executed if validation passes
    Discipline::create($validated);
    return redirect()->action([DisciplineController::class, 'index']);
}
```

Add Operation – Validation (Controller)

- Some notes about validation
  - `$request->validate` – method that validates the values passed on the HTTP request (GET or POST parameters)
    - Validation based on **rules**.
    - Invalid values generate errors messages. To change the default error messages, add custom messages to the second argument of validate method.
  - How does it work?
    - If a validation error occurs, validator **automatically** redirects to the previous location (in this example, to the create form "disciplines/create"). In addition, all the validation errors messages will be flashed to the session – and available on the view with the variable **$errors**

      The remaining code of the controller is not executed
    - If no validation error occurs, the validate function returns an array with all validated fields (only these) and the controller continues the execution

Add Operation – Validation (View)

- When there is a validation error, the validation function redirects the client to the previous location (the page that submitted the data)
- Validation errors messages will be flashed to the session, and automatically passed on to the view on the variable **$errors**
- Add the code "@dd($errors)" to the view "disciplines/create.blade.view"

```
@extends('template')
@section('content')
<form . . .>
    . . .
</div>
</form>
@dd($errors)
@endsection
```

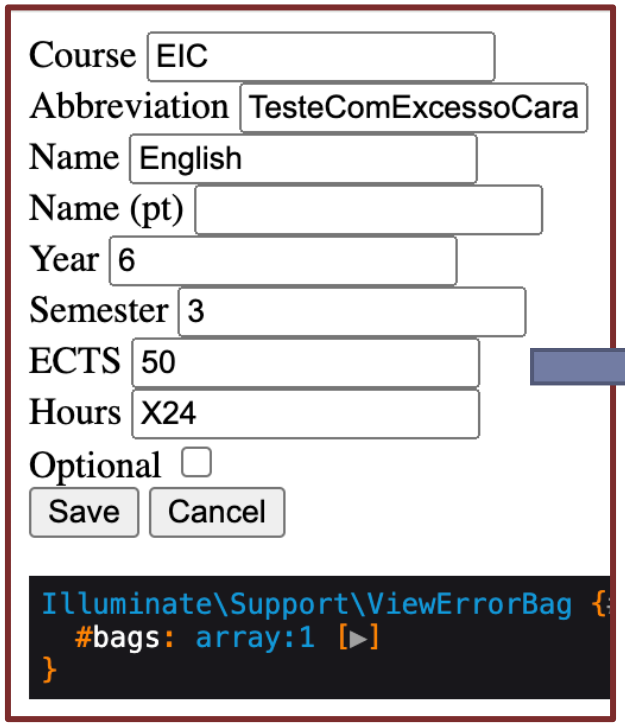
@dd($var) – prints the value of $var on the page and terminates the rendering of the view.
For debug purposes only

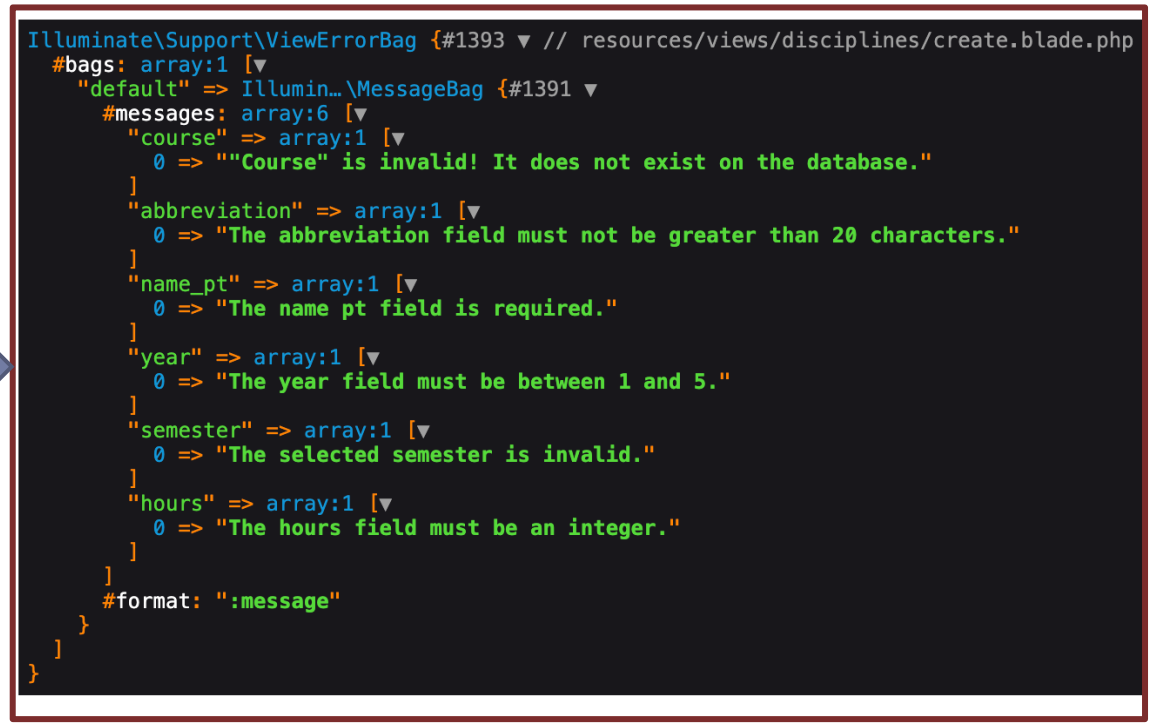
Alternative: @dump($var)
does not terminate the rendering

Add Operation – Validation (View)

- Try to add a discipline with invalid values:

@dd($errors)





Add Operation – Validation (View)

- Change view "disciplines/create.blade.php" to handle **validation errors** and fill "old" value
- **old** – HTML helper function that returns the value that the user has inputted previously (or a default value otherwise)
- **@error('field_name')** – Blade directive – shows a section if the field with the specified name is invalid
- **$message** – Variable injected by Blade inside the @error section, with the error message relative to that section

```
. . .
 <input type="text" name="course" id="inputCourse"
                    value="{{ old('course') }}">
 @error('course')
        <em>{{ $message}}</em>
 @enderror
 … repeat for fields: abbreviation, names, year, semester, etc…
…
<input type="checkbox" name="optional" id="inputOtcional" value="1"
       {{ old('optional') ? 'checked' : '' }}>
…
```

Add Operation – Validation

- Execute – http://mysite/disciplines/create
- Try to submit some invalid data

Course EIC *"Course" is invalid! It does not exist on the database.*
Abbreviation TesteComExcessoCara *The abbreviation field must not be greater than 20 characters.*
Name English
Name (pt) *The name pt field is required.*
Year 6 *The year field must be between 1 and 5.*
Semester 3 *The selected semester is invalid.*
ECTS 50
Hours X24 *The hours field must be an integer.*
Optional ☐
Save Cancel

Update Operation - Model

- "Discipline" model requires **no further modifications**
  - Extends from Eloquent, which means that it already supports all required operation to update the DB
  - It is already configured to support bulk operations and to support a table with no timestamps

```
<?php
namespace App\Models;
. . .
class Discipline extends Model
{
    . . .
     public $timestamps = false;

     protected $fillable = [
       'course', 'year', 'semester', 'abbreviation', 'name',
       'name_pt', 'ECTS', 'hours', 'optional',
     ];
}
```

Update Operation - Routes

- 2 new routes:
  - get "disciplines/X/edit" -> renders the form to edit the discipline with id = X
  - put "disciplines/X" -> when user submits the update form (relative to discipline where id = X)

```
Route::get('disciplines/{id}/edit', [DisciplineController::class, 'edit']);
Route::put('disciplines/{id}', [DisciplineController::class, 'update']);
```

  - **{id} – Route Parameter** (name = id) included on routes URL specification.
  - At the **{id}** placeholder, the URL accepts a value which is passed on to the controller's method:

```
public function update(Request $request, $id)
```

  - Mapping (between route parameters and method parameters) is done **by position** – not by name

Update Operation - Controller

- Add 2 new methods to the DisciplineController
- **edit** – show the form to edit a discipline
- **update** – update the discipline on the DB (through the model) – invoked when user submits the form

Arguments of the controller method will be automatically filled by Route Parameters of the URL

```
. . .
public function edit($id): View
{
  $discipline = Discipline::findOrFail($id);
  $pageTitle = 'Update Discipline';
  return view('disciplines.edit', compact('discipline', 'pageTitle'));
}
```

this passes 2 variables to the view, namely $discipline and $pageTitle. Variable names are the same as the names on the edit method

Update Operation - Controller


```
public function update(Request $request, $id): RedirectResponse {
    $validated = $request->validate([
        'course' =>       'required|exists:courses,abbreviation',
        'abbreviation' => 'required|string|max:20',
        'name' =>         'required',
        'name_pt' =>      'required',
        'year' =>         'required|integer|between:1,5',
        'semester' =>     'required|in:1,2',
        'ECTS' =>         'required|integer|between:1,60',
        'hours' =>        'required|integer|between:1,999',
        'optional' =>     'required|boolean',
    ], [  // Custom Error Messages
        'course.required' => '"Course" is required.',
        'course.exists' => '"Course" is invalid! ' .
        'It does not exist on the database.',
    ]);
  // If something is not valid, execution is interrupted.
  // Remaining code is only executed if validation passes
    $discipline = Discipline::findOrFail($id);
    $discipline->fill($validated);
    $discipline->save();
    return redirect()->action([DisciplineController::class, 'index']);
}
```

Update Operation - Controller

- Some notes about the update method
  - `update(Request $request, $id)`
    $request is an instance of the current HTTP request, $id will be filled by a route parameter from the URL.
  - `Discipline::findOrFail($id)`
    returns the Discipline model (instance of the model) with a specific id (primary key). If it does not exist, the controller is terminated, and a "404 Not Found" HTTP response is created
  - `$discipline->fill($disciplineInput);`
    updates the $discipline model property values with the data on the $disciplineInput array (which was previously validated)
  - `$discipline->save();`
    save the changes of the $discipline model on DB. Eloquent creates and sends an update SQL command.

Update Operation - View

- View "disciplines/edit.blade.php"

```
. . .
<form action="{{ action([App\Http\Controllers\DisciplineController::class,
                         'update'], $discipline->id) }}" method="post">
    @csrf
    @method("PUT")
    <div>
      <label for="inputCourse">Course</label>
        <input type="text" name="course" id="inputCourse"
               value="{{ old('course', $discipline->course) }}">
        @error('course')
            <em>{{ $message}}</em>
        @enderror
    </div>
   … repeat for fields: abbreviation, name, year, semester, etc…
    . . .
    <input type="checkbox" name="optional" id="inputOptional" value="1"
        {{ old('optional', $discipline->optional) ? 'checked' : '' }}>
. . .
```

Update Operation - View

- Some notes about the “disciplines/edit.blade.php” view
  - `action( [App\Http\Controllers\DisciplineController::class, 'update'], $discipline->id)`
    Returns the URL address that is associated (through the routing mechanism) to the update method of the DisciplineController controller and passes a parameter to it
  - `@method("PUT")`
    Adds the hidden field named "_method" with the value "PUT". This will allow the server to handle the HTTP Request as a PUT request (instead of a POST request – because forms only support GET or POST methods)
  - `value="{{old('course', $discipline->course)}}"`
    The value of the "course" field is initially filled with the value of $discipline->course (second argument is the default value). Afterwards, old helper function will return the value that the user has inputted

Update Operation - Test

- Execute – http://mysite/disciplines/1/edit
- Try to submit (save button) data
- Try some invalid data

Course EI
Abbreviation Prog I
Name Programming I - INIT
Name (pt) Programação I - INIT
Year 1
Semester 1
ECTS 7
Hours 75
Optional
Save Cancel

Course EIXPTO *"Course" is invalid! It does not exist on the database.*
Abbreviation Prog I
Name Programming I
Name (pt) Programação I
Year 6 *The year field must be between 1 and 5.*
Semester 1
ECTS 7
Hours 75
Optional
Save Cancel

Abbreviation	Name	Course	Year	Semester			
AL	Linear Algebra	EI	1	1	View	Edit	Delete
AM	Mathematical Analysis	EI	1	1	View	Edit	Delete
FA	Applied Physics	EI	1	1	View	Edit	Delete
Prog I	Programming I - INIT	EI	1	1	View	Edit	Delete

Delete Operation - Model

- "Discipline" model requires **no further modifications**
  - Extends from Eloquent, which means that it already supports all required operation to delete on the DB

```
<?php
namespace App\Models;
. . .
class Discipline extends Model
{
    . . .
     public $timestamps = false;

     protected $fillable = [
       'course', 'year', 'semester', 'abbreviation', 'name',
       'name_pt', 'ECTS', 'hours', 'optional',
     ];
}
```

Delete Operation - Route

- Add a new route:
  - delete "disciplines/X" -> when user submits the delete form (relative to discipline where id = X)

```
Route::delete('disciplines/{id}',
       [DisciplineController::class, 'destroy']);
```

Delete Operation - Controller

- Add the method destroy to the DisciplineController
- **destroy** – delete the discipline from the DB (through the model) – invoked when user submits the form

Arguments of the controller method will be automatically filled by Route Parameters of the URL

```
. . .
public function destroy($id) : RedirectResponse
{
    Discipline::destroy($id);
    return redirect()->action(
                [DisciplineController::class, 'index']);
}
```

- `Discipline::destroy($id)`
  Method of the model to delete from the DB, the "discipline" with "id" that equals $id

Delete Operation - View

- Delete operation **does not have a view file**
  - Unless if associated to a confirmation form
- However, for invoking the delete operation we must create a form associated with a route to delete (method delete).
  - This was done previously on the view "index", that show the list of disciplines. Section of code where the form is created (repeated for all disciplines on the table):

```
<form action="disciplines/{{ $discipline->id }}" method="POST">
    @csrf
    @method("DELETE")
    <input type="submit" value="Delete">
</form>
```

Using CSS, we can transform the submit button of the form so that it looks (visually) as a hyperlink

Delete Operation - Test

- Execute – http://mysite/disciplines
- Try to click on "delete" submit button (looks like a hyperlink)

Disciplines List

Create a new discipline

Abbreviation	Name	Course	Year	Semester			
AL	Linear Algebra	EI	1	1	View	Edit	Delete
AM	Mathematical Analysis	EI	1	1	View	Edit	Delete
FA	Applied Physics	EI	1	1	View	Edit	Delete

5 – SUMMARY

A summary of the concepts used by previous example, complement by some new associated concepts

Configuration

- System application configuration on **.env** file
  - This allows to have different configuration values based on the environment where the application is running.
  - By default, .env file will not commit with git – because different developers may have different environments
  - Also accessible through code, with **env** helper

    e.g. `$x = env('APP_DEBUG', false);`
- Configuration is really defined on classes of **config** folder
  - Most configuration settings use environment variables.
  - e.g. `'default' => env('DB_CONNECTION', 'mysql'),`
  - Also accessible through code, with **config** helper

Routes

- Routes registration on “Routes/Web.php” file
- Each route has a HTTP method, a URL pattern and a Controller/action. Other route definition alternatives are possible – latter on we will revisit the routing system

```
. . .
Route::get('disciplines', [DisciplineController::class, 'index']);
Route::get('disciplines/create', [DisciplineController::class, 'create']);
Route::post('disciplines', [DisciplineController::class, 'store']);
Route::get('disciplines/{id}/edit', [DisciplineController::class, 'edit']);
Route::put('disciplines/{id}', [DisciplineController::class, 'update']);
Route::delete('disciplines/{id}', [DisciplineController::class, 'destroy']);
```

- URL pattern may include parameters eg: `users/{id}/edit`
- To view all routes, execute the following artisan command: **`artisan route:list`**

Models

- Models use **Eloquent ORM** – extend from ..\Eloquent\Model
- By convention, each Model class is associated to a database table (User class => users table)
  - It is possible to change the associated table with the attribute $table. E.g. `protected $table = 'my_user';`
- Model automatically creates object attributes that represent the columns of the table
- Model includes several instance and static methods to retrieve, insert, update, delete record from the database. E.g.:

```
User::all()                 User::where( . . . )
User::create($user)         User::destroy($id)
User::find($id)             User::findOrFail($id)
$user->save()               $user->fill($user)
```

. . . MANY MORE ARE AVAILABLE . . .

Controllers

- Controllers are defined by extending ”Controller” class
- They use the model (static or instance method of the Model class), HTTP Request information (data from forms, etc); Validators; etc… to create **views** or **redirect** to another location
- If the URL pattern of the route includes a parameter, add a parameter to the controller’s method

```
public function edit($id)
```

- If the controller’s method need to access the HTTP Request, just add (at the beginning) a parameter of Request type

```
public function update(Request $request, $id)
```

Controller actions - patterns

- Controller Actions (methods) typically follow **2 patterns**:

1. They return a **view** (associated to a **get** request)
   - Usually, they will retrieve data from the database, pass that data to the view and return the view

```
public function index(): View
{
    $data = EntityModel::all();
    return view('entity.index')->with('data', $data);
}
```

2. They perform an operation and return a **redirect** (associated to **post**, **put**, **patch** or **delete** request)
   - Usually, they will insert, update, delete data on the database, or execute some other operation, and redirect the end-user to a page that shows data/content

```
public function store(Request $request): RedirectResponse
{
    Discipline::create($request->all());
    return redirect()->action([DisciplineController::class, 'index']);
}
```

Controllers / Request

- To retrieve all input data (data fields values) as an array

```
$input = $request->all();
```

- To retrieve the input data of one form field:

```
$name = $request->input('name');
$name = $request->input('name', 'default_value');
```

Or:

```
$name = $request->name;
```

- To retrieve all query string data (URL parameters) as an array

```
$input = $request->query();
```

- To retrieve one query string parameter:

```
$name = $request->query('name');
$name = $request->query('name', 'default_value');
```

- To check if a value is present on the request:

```
if ($request->has('name')) . . .
```

Controllers / Request Validation

- The request data can be validated, using validation **rules**, custom messages, etc..

```
$disc = $request->validate([
         'curso' => 'required|exists:cursos,abreviatura',
         'ano'   => 'required|integer|between:1,5',
        ], [     // Custom Messages:
            'ano.required' => 'Ano é obrigatório.',
        ]);
```

- If everything is OK, validated data* from the form is retrieved as an array

  * *only the validated fields*

- If there is any error, Laravel will automatically:
  - redirect to previous location (typically the form where data was inputted)
  - Flash the error messages and inputted values to the redirect location – in the view, error messages are available as **$errors** variable

Views

- Views use the **Blade** rendering engine
- Blade templates:
  - Template view "*inject*" points:
    `@yield('sectionName')`
  - Views use a template, by extending it with
    `@extends('templateName')`
  - View sections will be "*injected*" at one "*inject*" point in the template, with: `@section('sectionName')`

Views

`{{$a}}` - sanitized $a (*htmlspecialchars($a)*)

`{!!$a!!}` - NOT sanitized $a

- Blade includes instructions like:

`@foreach`; **`@if`**; **`@for`**; **`@while`**; **`@switch`**; **`@isset`**;

- Blade can use HTML helper like:

`old()` **`action()`** **`route()`** **`url()`** . . .

Views / Forms

- Laravel includes instructions to inject csrf and _method spoof hidden fields on the form:

```
@csrf    @method("PUT")    @method("DELETE")
```

- Use old() HTML helper to maintain form state. E.g.

```
value= {{ old('name' , $user->name) }}
```

- Show errors

```
@error('name'))
        {{ $message }}
$enderror
```

5 – REFERENCES

References

- Official (Laravel documentation)
  - https://laravel.com/docs
- Laracasts
  - https://laracasts.com/