DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

HTTPS://WWW.IPLEIRIA.PT/ESTG-DEI/

APLICAÇÕES PARA A INTERNET

Engenharia Informática

Marco Monteiro

5 - Laravel - 2

Objectives:

(1) Comprehend and use main concepts of Laravel Framework.

(2) Install and use Tailwind CSS

(3) Implement layouts (templates) with Laravel.

(4) View Components

(5) Forms and validations.

Note the following:

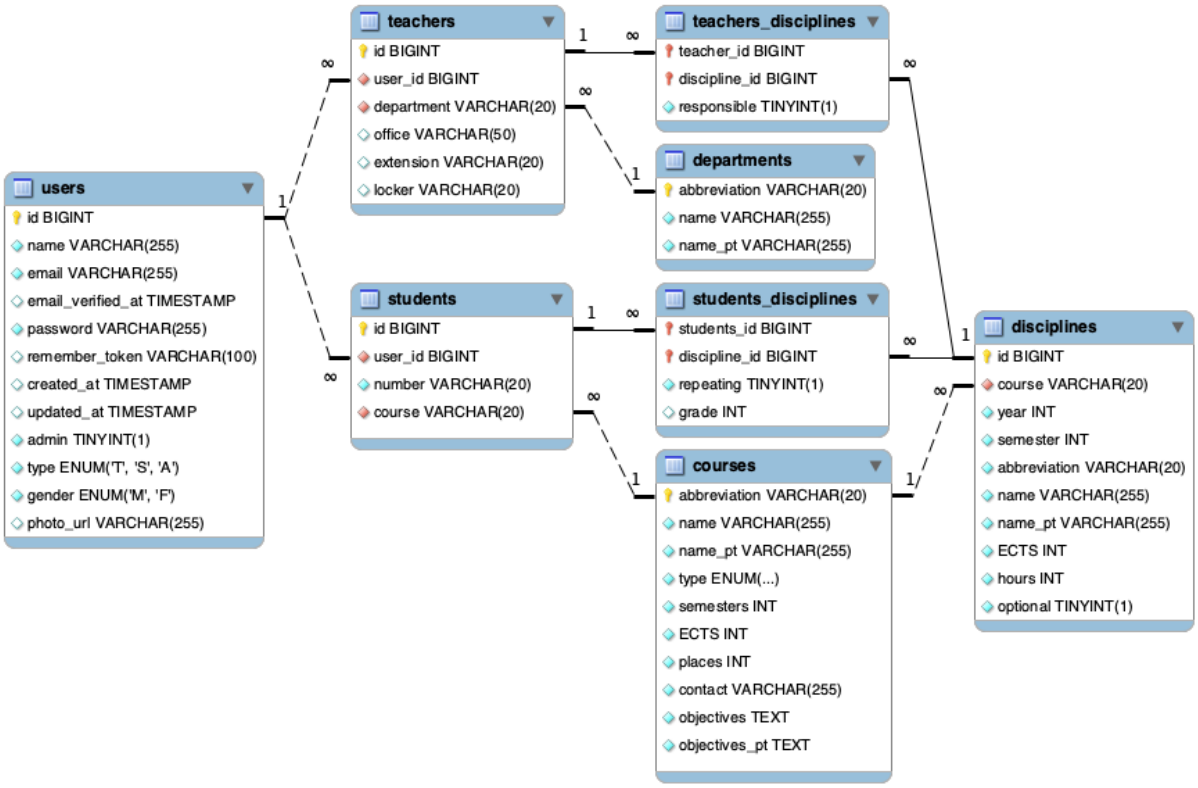
- Before starting the exercise, read the related content on Moodle;
- During the resolution of the exercises, consult the Laravel documentation (https://laravel.com) and other online resources.

Scenery

This worksheet will continue the development of the Web Application started on the first Laravel worksheet that used Laravel Framework and included all CRUD operations for the courses and disciplines. In the current worksheet we will apply a template (Tailwind CSS based template) to define the web application layout, create and use view components, use pagination for large sets of data, implement validation on create and update operations, as well as alert messages using flash data.

Database

This worksheet will use the same database as the one used on the last worksheet. The structure of the database is the following:



1. Preparation

To run the exercises of this worksheet we will use Laragon (https://laragon.org), or Laravel Sail (https://laravel.com/docs/sail). For the database, we'll preferably use a MySQL server, or if that's not possible, a SQLite database.

To create the project for the current worksheet we have 3 options:

1. Copy the provided project and configure it as a new project, using Laragon.
2. Copy the provided project and configure it as a new project, using Laravel Sail.
3. Merge the provided project into the project that was implemented on previous worksheet (fastest option). Works with Laragon or Laravel Sail.

Consult the tutorial "**tutorial.laravel.01-laravel-install-configuration**", available on Moodle, to check for details on installation and configuration of Laravel projects.

1.1. New Laravel Project – with Laragon

1. Copy the provided zip file (`start.ai-laravel-2.zip`) into the Laragon root folder and decompress it on that folder.
2. Previous command will create the folder `ai-laravel-2`, that will be the worksheet project folder, inside the Laragon root folder. The worksheet project folder should be available as `C:\<laragon_www_root>\ai-laravel-2`. For example, `C:\laragon\www\ai-laravel-2` or `D:\ainet\ai-laravel-2` (it depends on the Laragon root folder)
3. Use previous database (from the first Laravel worksheet) or create a new database. Configure .env file accordingly. Typical database configuration for Laragon (with the database name "Laravel")

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

4. Run Laragon and start all services (in ESTG computers, before starting the services, it might be necessary to stop `vmware` services).
5. Open Laragon terminal and execute the following command on the project folder (`ai-laravel-2`), to rebuild the "`vendor`" folder:

```
composer update
```

6. To define the database structure and fill (seed) the data on the database, execute:

```
php artisan migrate:fresh
```

```
php artisan db:seed
```

7. Use the "`http://ai-laravel-2.test`" URL to access the content.
8. Test CRUD operations for courses (`http://ai-laravel-2.test/courses`) and disciplines (`http://ai-laravel-2.test/disciplines`)

1.2. New Laravel Project – with Laravel Sail

9. Copy the provided zip file (`start.ai-laravel-2.zip`) into any folder and decompress it.
10. Previous command will create the folder `ai-laravel-2`, that will be the current worksheet project folder.
11. Execute the following command on the project folder (`ai-laravel-2`), to rebuild the "`vendor`" folder – this will also install the required package Laravel Sail

```
composer update
```

    - To execute previous command, it is necessary that the composer tool is installed on your local machine. Check https://getcomposer.org to install composer if necessary.
    - If for some reason it is not possible to install composer on your machine, copy the provided zip file (`start.ai-laravel-2.all-folders.zip`) that already includes the vendor folder.

12. Ensure that Docker Desktop (or other similar application) is running.
13. On the `ai-laravel-2` folder execute the following command:

```
./vendor/bin/sail up -d
```

    - If the sail alias is already configured, it is possible to execute the alternative command:

```
sail up -d
```

14. To define the database structure and fill (seed) the data on the database, execute:

```
sail php artisan migrate:fresh
```

```
sail php artisan db:seed
```

15. Use the "`http://localhost`" URL to access the content, and "`http://localhost:8080`" to access the `adminer` tool (for database administration)
16. Test CRUD operations for courses (`http://localhost/courses`) and disciplines (`http://localhost/disciplines`)

1.3. Merge Projects – with Laragon or Laravel Sail

17. Copy the provided zip file (`start.ai-laravel-2.zip`) into any folder and decompress it.
18. Previous command will create the folder `ai-laravel-2`, with the base project for the current worksheet. However, instead of using this new folder, we will continue to use the last worksheet project folder. With this approach we will reuse the folder "vendor" and "storage", as well as the database.
19. On the last worksheet project folder (that we want to continue using), **remove** the following folders:
    - `config`
    - `app`
    - `resources`
    - `routes`
20. Copy the 4 folders (`config`, `app`, `resources` and `routes`) from the provided folder (`ai-laravel-2`) to the last worksheet project folder (that we want to continue using).
21. If you are using Laragon, run Laragon and start all services (in ESTG computers, before starting the services, it might be necessary to stop `vmware` services).
    - Use the same URL as the last worksheet (probably "`http://ai-laravel-1.test`") to access the content.
    - Test CRUD operations for courses (`http://ai-laravel-1.test/courses`) and disciplines (`http://ai-laravel-1.test/disciplines`)
22. If you are using Laravel Sail, execute the following command on the root of the last worksheet project:

```
./vendor/bin/sail up -d
```

    - If the sail alias is already configured, it is possible to execute the alternative command:

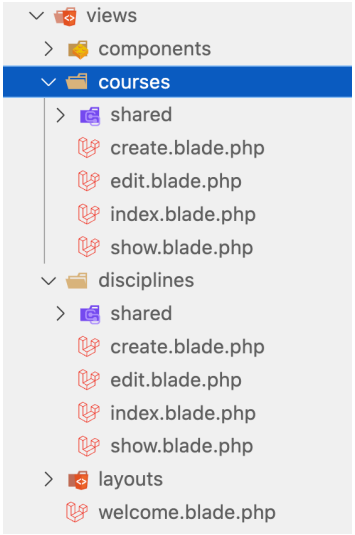
```
sail up -d
```

    - Use the same URL as the last worksheet (probably "`http://localhost`") to access the content.
    - Test CRUD operations for courses (`http://localhost/courses`) and disciplines (`http://localhost/disciplines`)

- Test "`adminer`" tool for database administration: (`http://localhost:8080`)

2. Layouts

Our application has several views related to the courses and disciplines, and all these views fully define the HTML document – all views include the `<html>`, `<body>` and all the page content.



However, this is not a correct approach for larger applications, where a set of different views share a common part/layout, so that the web application maintains visual consistency through all their pages/views. Let´s convert our application, so that all existing views use a common layout.

23. On the "`resources/views`" folder, add the file `layout.blade.php` (file name: `resources/views/layout.blade.php`) with the following content:

```
<!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>ESTG</title>
    <style>
        table,
        th,
        td {
            border: 1px solid black;
            border-collapse: collapse;
        }
```

```
        html {
            height: 100%;
        }

        body {
            height: 100%;
            display: flex;
        }

        body>nav {
            min-width: 150px;
            background-color: lightgray;
            margin-right: 20px;
        }

        body>nav ul {
            list-style-type: none;
            padding-left: 15px;
            margin-bottom: 10px;
        }

        body>nav li {
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
    <nav>
        <ul>
            <li>
                <a href="{{ route('courses.index') }}">Courses</a>
            </li>
            <li>
                <a href="{{ route('disciplines.index') }}">Disciplines</a>
            </li>
        </ul>
    </nav>
    <div class="main">
        <header>
            <h1>@yield('header-title')</h1>
        </header>
        <div class="content">
            @yield('main')
        </div>
    </div>
</body>
</html>
```

- Previous file (`layout.blade.php`) defines what will be common to all pages of the Web Application – it defines the **template**.
- `@yield(…)` blade directive, defines where the views that will use the layout, can inject their own code.
- This layout has 2 `@yield` directives, the first one (named "`header-title`") is used to define a custom title (the text of the <h1> heading). The second yield (named "`main`") is where the view will inject their specific code – the part of the page that is specific to each view.

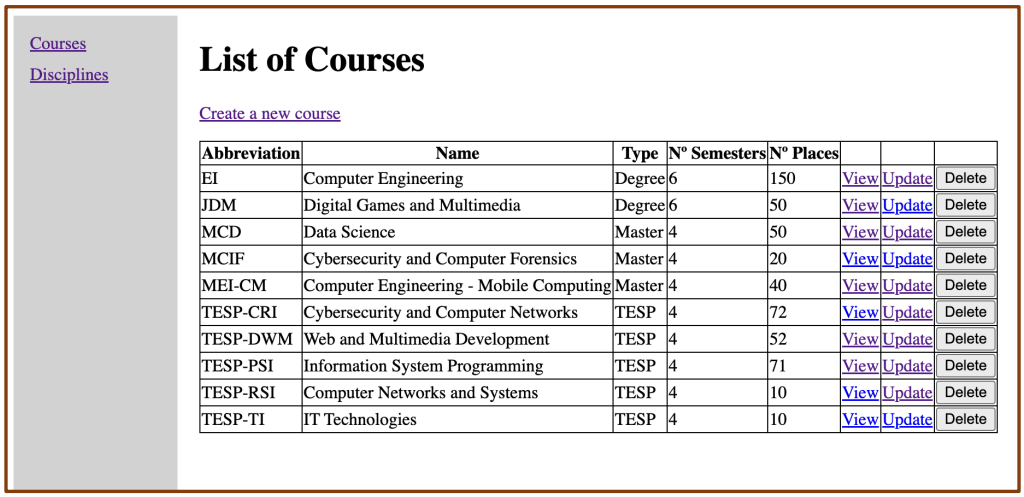
24. Next, we will adapt the view "`courses.index`" to use the layout previously defined. Change the code of the file "`resources/views/courses/index.blade.php`" to:

```
@extends('layout')

@section('header-title', 'List of courses')

@section('main')
    <p>
       <a href="{{ route('courses.create') }}">Create a new course</a>
    </p>
    <table>
        . . .
    </table>
@endsection
```

- `@extends(…)` directive defines that the current view will use the specified layout. In this example, the current view will use the layout with the name: layout - `@extends('layout')`.
- Note that a layout is also a view, so the naming convention is the same – "`layout`" is specified by the file "`resources/views/layout.blade.php`"
- `@section('name-of-section')` directive defines the content that is specific to the current view. The content of a specific section will be injected on the correspondent `@yield` on the layout.
- The content of a section can be defined by the second parameter of the @section directive:
    - `@section('header-title', 'list of Courses')` – `header-title` is the name of the section – corresponds to the `@yield('header-title')` on the layout. "List of Courses" is the content of the section.

- The content of a section can also be defined by the block of code between the `@section` directive and the `@endsection` directive. This is what happens on the case of the "`main`" section.

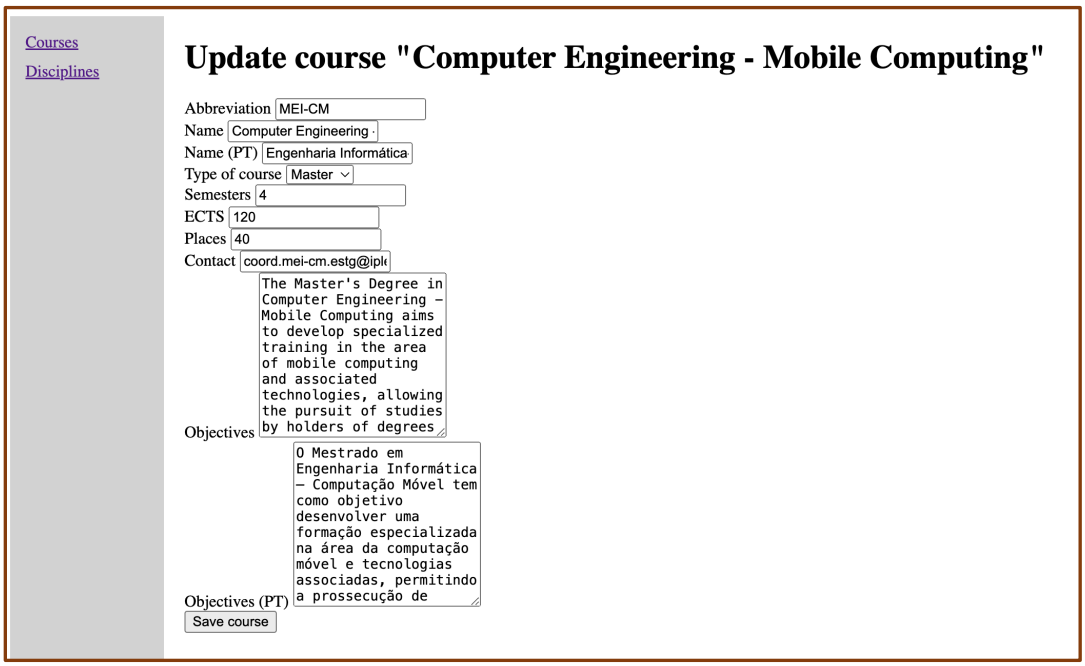
25. Test the application with the following URL: `http://yourdomain/courses`. The page should look like this:



26. Next, we will adapt the view "`courses.edit`" to use the layout previously defined. Change the code of the file "`resources/views/courses/edit.blade.php`" to:

```
@extends('layout')

@section('header-title', 'Update course "' . $course->name . '"')

@section('main')
    <form method="POST" action="{{ route('courses.update', ['course' => $course]) }}">
        @csrf
        @method('PUT')
        @include('courses.shared.fields')
        <div>
            <button type="submit" name="ok">Save course</button>
        </div>
    </form>
@endsection
```

27. Test the application with the following URL: `http://yourdomain/courses/MEI-CM/edit`. The page should look like this:



28. Redefine all remaining views (except subviews) relative to the courses and disciplines, so that all views use the same layout.
29. Thy the application and verify that it maintains a consistent layout and visual appearance – for instance, the menu is present on all views.
30. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.2.zip`").

3. TailwindCSS and Vite

Although the web application has a template (defined by the layout), its design is very rudimentary. To enrich the design of our web application, we will use a TailwindCSS based template. The provided project already includes most of the files of an example of a Tailwind CSS template - we will use that template through the exercises.

On this section we will execute the required steps to install, configure and use the TailwindCSS framework, as well as the **vite** tool. These are required steps to use a Tailwind CSS based template.

31. For our Laravel project, we will use the TailwindCSS framework (https://tailwindcss.com) with **Vite** tool (https://vitejs.dev) as the CSS and JS builder (compiles the resulting CSS and

JS files). To install TailwindCSS with Vite on Laravel, we will follow the instructions available on https://tailwindcss.com/docs/guides/laravel#vite

32. In the project root folder, execute the following command:

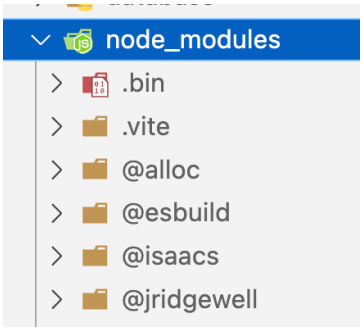
```
npm install -D tailwindcss postcss autoprefixer
```

- This command will install `TailwindCSS` framework, as well as the required tools `postcss` (https://postcss.org/) and `autoprefixer` (https://www.npmjs.com/package/autoprefixer), for transforming CSS with Javascript.

33. Open the file `package.json` (on the project root folder). This file includes the client (Node.js) module dependencies for our project.

```
{
    "private": true,
    "type": "module",
    "scripts": {
        "dev": "vite",
        "build": "vite build"
    },
    "devDependencies": {
        "autoprefixer": "^10.4.19",
        "axios": "^1.6.4",
        "laravel-vite-plugin": "^1.0",
        "postcss": "^8.4.38",
        "tailwindcss": "^3.4.1",
        "vite": "^5.0"
    }
}
```

- The client (Node.js) modules are installed on the folder "`node_modules`". Open this folder to check what modules are installed:



- `node_modules` folder includes more modules than the ones specified on the `package.json` file. This is because the modules specified on the `package.json` file are themselves dependent of other modules, creating a complex tree of dependencies.
- To reinstall client (Node.js) modules we just execute the following command:

```
npm install
```

- When the folder `node_modules` is not available we have to execute previous command. We can execute "`npm install`", even when the folder is already present, or alternatively, we can execute the command "`npm update`". Try to execute both and analyze the differences between them.

34. Next, execute the following command to create the configuration files `tailwind.config.js` and `postcss.config.js`:

```
npx tailwindcss init -p
```

35. Configure all the files that have content to compile – the files that, when changed, will rebuild the resulting CSS and JS files. Add the paths to the `tailwind.config.js` file on of the root folder:

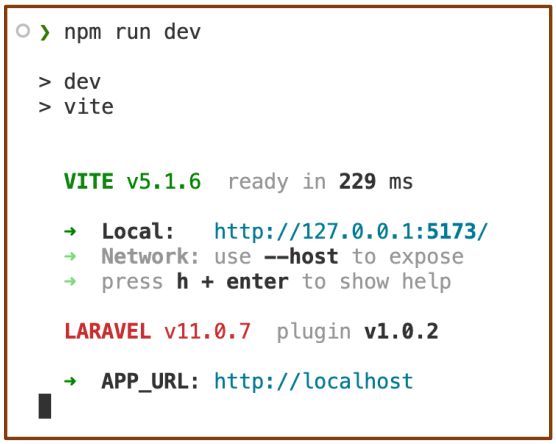
```
/** @type {import('tailwindcss').Config} */
export default {
    content: [
    "./resources/**/*.blade.php",
    "./resources/**/*.js",
    "./resources/**/*.vue",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

36. The file `resources/css/app.css` will be the initial CSS file – the file where we can add our own CSS rules. We will add Tailwind directives to this file (`resources/css/app.css`):

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

37. Start your build process by executing the following command:

```
npm run dev
```



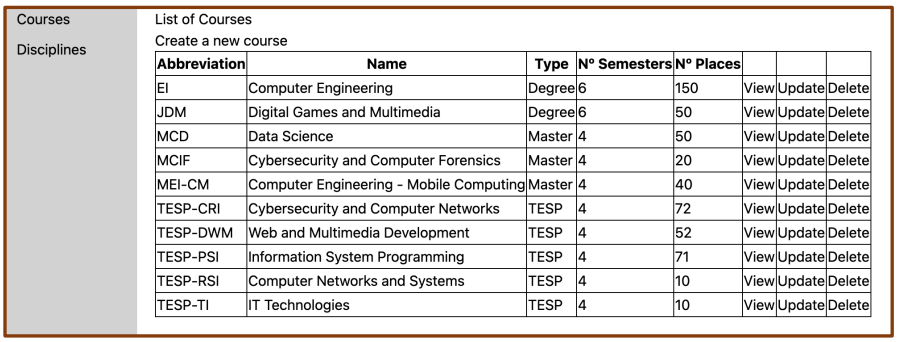
- This command will use **vite** tool to dynamically build all CSS and JS required by our pages.
- CSS files on the folder `resources/css` will compile to the CSS loaded by the pages
    - `app.css` file is the entry point css file
- JS files on the folder `resources/js` will compile to the JavaScript loaded by the pages
    - `app.js` file is the entry point JavaScript file
- Build process runs at the background and watches for changes on the CSS and JS files, as well as changes to the content specified on the `tailwind.config.js` configuration file.

38. To use the compiled CSS (with the TailwindCSS framework) in our pages, we will include the following code on the `<head>` section of the `resources/views/layout.blade.php` file (layout view):

```
<!DOCTYPE html>
    . . .
    </style>
    @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
<body>
```

- The `@vite` will add to the final HTML content, the necessary `<link>` element to load the resulting CSS and the `<script>` element to load the resulting JavaScript code.

39. Test the application with the following URL: `http://yourdomain/courses`. The page should look like this:



Courses
Disciplines

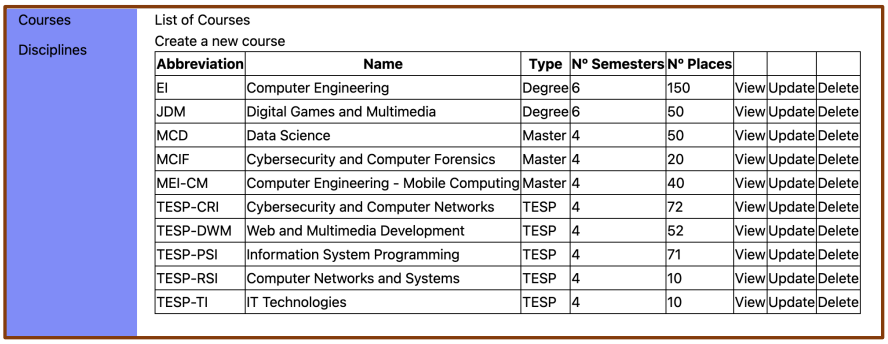
List of Courses
Create a new course

Abbreviation	Name	Type	Nº Semesters	Nº Places			
EI	Computer Engineering	Degree	6	150	View	Update	Delete
JDM	Digital Games and Multimedia	Degree	6	50	View	Update	Delete
MCD	Data Science	Master	4	50	View	Update	Delete
MCIF	Cybersecurity and Computer Forensics	Master	4	20	View	Update	Delete
MEI-CM	Computer Engineering - Mobile Computing	Master	4	40	View	Update	Delete
TESP-CRI	Cybersecurity and Computer Networks	TESP	4	72	View	Update	Delete
TESP-DWM	Web and Multimedia Development	TESP	4	52	View	Update	Delete
TESP-PSI	Information System Programming	TESP	4	71	View	Update	Delete
TESP-RSI	Computer Networks and Systems	TESP	4	10	View	Update	Delete
TESP-TI	IT Technologies	TESP	4	10	View	Update	Delete

40. Just to test if the builder is running correctly, change the following code within the layout view - file `resources/views/layout.blade.php`:

```
. . .
<body>
    <nav class="bg-indigo-400">
            . . .
```

41. If everything is installed and properly configured, the following URL: `http://yourdomain/courses` should look like this (the navigation bar background color has changed to indigo-400):



Courses
Disciplines

List of Courses
Create a new course

Abbreviation	Name	Type	Nº Semesters	Nº Places			
EI	Computer Engineering	Degree	6	150	View	Update	Delete
JDM	Digital Games and Multimedia	Degree	6	50	View	Update	Delete
MCD	Data Science	Master	4	50	View	Update	Delete
MCIF	Cybersecurity and Computer Forensics	Master	4	20	View	Update	Delete
MEI-CM	Computer Engineering - Mobile Computing	Master	4	40	View	Update	Delete
TESP-CRI	Cybersecurity and Computer Networks	TESP	4	72	View	Update	Delete
TESP-DWM	Web and Multimedia Development	TESP	4	52	View	Update	Delete
TESP-PSI	Information System Programming	TESP	4	71	View	Update	Delete
TESP-RSI	Computer Networks and Systems	TESP	4	10	View	Update	Delete
TESP-TI	IT Technologies	TESP	4	10	View	Update	Delete

42. At this point, our Laravel application has the TailwindCSS framework ready for use – it is installed and correctly configured.
43. Before continuing, let's analyze the resulting HTML to better understand how **vite** tool works. When opening our application while running "`npm run dev`" in the background, analyze the source code of the courses page (`http://yourdomain/course`).

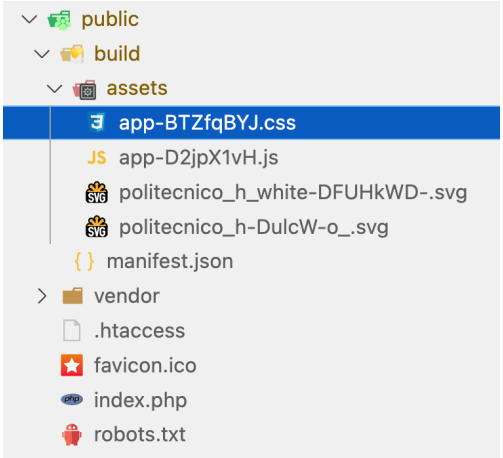
- It is possible to verify that there are several JavaScript and CSS references that point to the server `http://127.0.0.1:5173` - this server is implemented by **vite** and is running locally while developing. The CSS and JavaScript files are dynamically generated on the fly by vite – they do not exist as files.

44. Interrupt the build process (`npm run dev`) by pressing CTRL+C key combination. If we try to open the application with the browser we will get an error, because **vite** local web server is not running and therefore, CSS and JS files are not loaded.
45. When publishing the application, vite local web server will not be running, so we have to build static CSS and JS files for publishing. To build these files, execute the following command:

```
npm run build
```

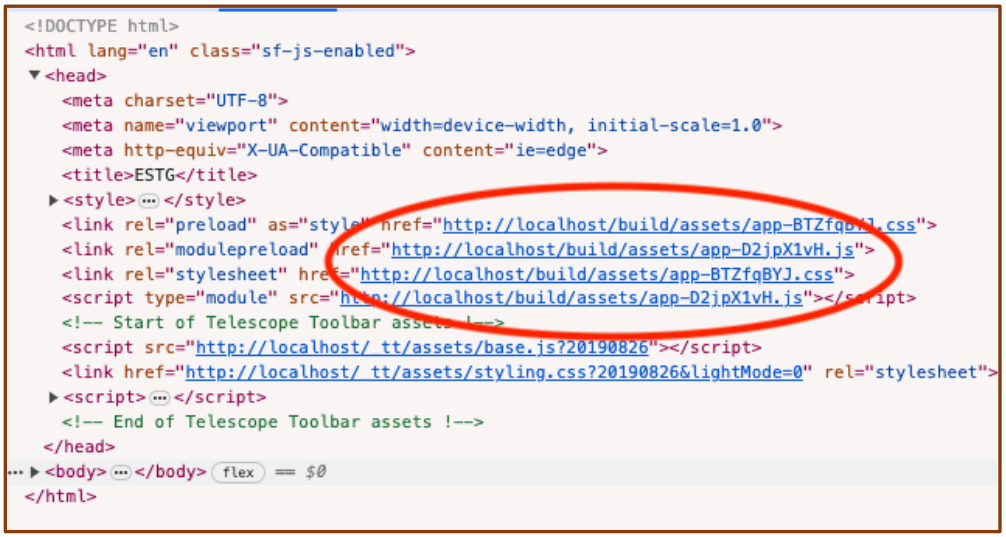
```
❯ npm run build

> build
> vite build

vite v5.2.6 building for production...
✓ 47 modules transformed.
public/build/manifest.json                          0.57 kB │ gzip:  0.21 kB
public/build/assets/politecnico_h-DulcW-o_.svg      4.86 kB │ gzip:  1.93 kB
public/build/assets/politecnico_h_white-DFUHkWD-.svg  5.19 kB │ gzip:  1.98 kB
public/build/assets/app-BTZfqBYJ.css               21.82 kB │ gzip:  4.80 kB
public/build/assets/app-D2jpX1vH.js                29.83 kB │ gzip: 11.98 kB
✓ built in 312ms
~/Documents/Internal/AI/Pratica/05.Laravel.2/ai-laravel-2
```

- Previous command also uses **vite** tool to build the CSS and JS. The difference is that this command (`npm run build`) will generate static CSS and JS files prepared for publishing. These files will be generated on the `public/build` folder and should be copied to the web server when publishing.
- Confirm that CSS and JS files were generated on the `public/build` folder:



- Note that the name of the compiled CSS and JS files are random unique name. This will ensure that every time we rebuild them (every time we execute `npm run build`), they will have a unique name and therefore, they will not be cached. If the compiled file name was the same every time, the browser might not load the new version of the file because it would have a file with the same name available on the browser local cache.

46. Open the courses page (`http://yourdomain/courses`) and analyze the source code again.



- When comparing current source code with the version that executed on vite local web server (`npm run dev`), we can now observe that the JavaScript and CSS references are pointing to the `build` folder of the Laravel project (`build` folder is within Laravel `public` folder), instead of pointing to the server `http://127.0.0.1:5173`. CSS and JS files are now static files, and therefore, can be copied to any web server for publishing.

47. Restart the build process in the background, by executing the following command again:

```
npm run dev
```

48. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.3.zip`").

4. TailwindCSS template

Laravel project is now prepared to use the TailwindCSS framework. In this section, we will integrate a Tailwind template (most of the template files are already included on the project) on our project. The template is based on the pages implemented during the TailwindCSS worksheet, with some adaptations.

49. First, we will change the configuration of the TailwindCSS theme – similar to what was implemented on the TailwindCSS worksheet. Change the configuration file `tailwind.config.js` to:

```
import defaultTheme from 'tailwindcss/defaultTheme';
import forms from '@tailwindcss/forms';
```

```
/** @type {import('tailwindcss').Config} */
export default {
    content: [
        "./resources/**/*.blade.php",
        "./resources/**/*.js",
        "./resources/**/*.vue",
  ],
  theme: {
    extend: {
        fontFamily: {
            sans: ['Figtree', ...defaultTheme.fontFamily.sans],
        },
    },
  },
  plugins: [forms],
}
```

- Ensure that the vite builder is running by executing the command:

```
npm run dev
```

50. The first view where we will test the new template is the `home` view (file `resources/views/home.blade.php` that includes the introduction text). The file `home.blade.php` of the `resources/views` folder, is already included in the project with the following code structure:

```
@extends('layout')

@section('header-title', 'Introduction')

@section('main')
<main>
    <div class="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
        . . .
    </div>
</main>
@endsection
```

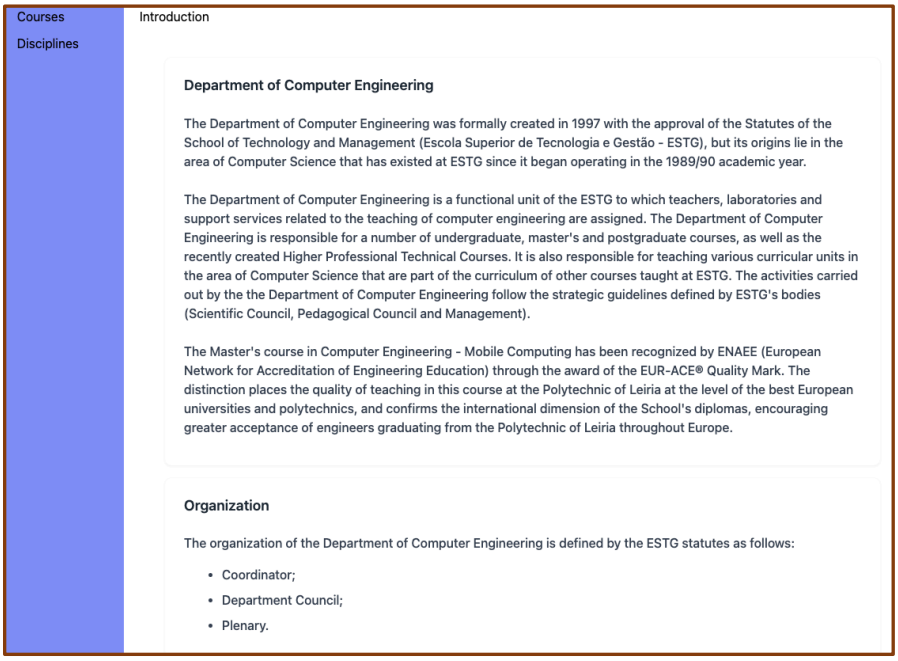
- Note that currently, the view uses the basic layout created previously on this worksheet.

51. Edit the routes configuration file (`routes/web.php`) and replace existing "/" route with the following code:

```php
<?php

use App\Http\Controllers\CourseController;
use App\Http\Controllers\DisciplineController;
use Illuminate\Support\Facades\Route;

// REPLACE THIS
// Route::get('/', function () {
//     return view('welcome');
// })->name('home');

//WITH THIS
Route::view('/', 'home')->name('home');
```

- Note that we can add routes associated to views (using the method `Route::view`) – this is useful when the content of the view is static (as is the case for the `home` view).

52. Open the application on the browser with the following url: `http://yourdomain/`



- If the error *"Cannot find module @tailwindcss/forms"* occurs, we need to install the package tailwindcss/forms with the command:

```
npm install @tailwindcss/forms
```

53. The layout file is already included on the project with the name `layouts.main`, which is defined by the file `resources/views/layouts/main.blade.php`. If we analyze the layout file, we can observe that the code is based on the solution for the TailwindCSS worksheet,

but with some minor adjustments and other relevant changes, which we will emphasize in the following code sections:

```
<!DOCTYPE html>
. . .
<head>
    . . .
    <!-- Scripts AND CSS Fileds -->
    @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
<body class="font-sans antialiased">
    . . .
    <div class="h-16 w-40 bg-cover bg-[url('../img/politecnico_h.svg')]
                        dark:bg-[url('../img/politecnico_h_white.svg')]">
    . . .
         <!-- Menu Item: Courses -->
         <x-menus.menu-item
            content="Courses"
            href="{{ route('courses.index') }}"
            selected="{{ Route::currentRouteName() == 'courses.index'}}"
         />
    . . .
         <!-- Menu Item: Curricula -->
         <x-menus.submenu-full-width
            content="Curricula"
            selectable="1"
            selected="0"
            uniqueName="submenu_curricula">
                <x-menus.submenu-item
                   content="Computer Engineering"
                   selectable="1"
                   selected="1"
                   href="#"/>
```

```
    . . .
        </x-menus.submenu-full-width>
        <x-menus.submenu
           selectable="0"
           uniqueName="submenu_user"
        >
           <x-slot:content>
              <div class="pe-1">
              <img src="{{ Vite::asset('resources/img/photos/photo_example.jpeg') }}"
                   class="w-11 h-11 min-w-11 min-h-11 rounded-full">
{{-- ATENÇÃO:
     ALTERAR FORMULA DE CALCULO DAS LARGURAS MÁXIMAS QUANDO O MENU FOR ALTERADO --}}
              <div class="ps-1 sm:max-w-[calc(100vw-39rem)]
                               md:max-w-[calc(100vw-41rem)]
                               lg:max-w-[calc(100vw-46rem)]
                               xl:max-w-[34rem]
                          truncate">
       . . .
<!-- Page Heading -->
<header class="bg-white dark:bg-gray-900 shadow">
    . . .
    <h2 class="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">
        @yield('header-title')
    </h2>
    . . .
</header>

<main>
    <div class="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
        @yield('main')
    </div>
</main>
. . .
```

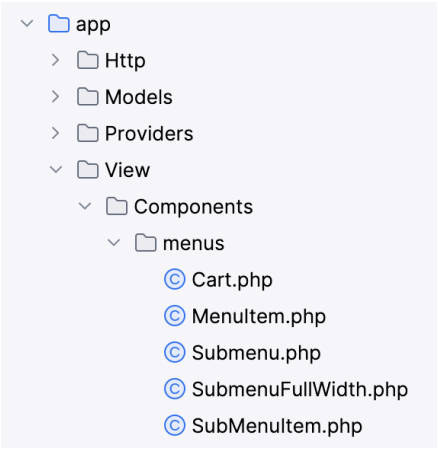
- **@yield('header-title')** and **@yield('main')** define the points where the view sections (`header-title` and `main`) will be "injected".
- The code **@vite(['resources/css/app.css', 'resources/js/app.js'])** ensures that the CSS and JS built by vite tool is included on the layout.
- We can add assets (image files or other static resources) to the Laravel "`resources`" folder, and **vite** tool will copy them as needed to the public folder (it changes the names to a random unique name).
  Vite::asset('resources/img/photos/photo_example.jpeg') will generate the "final" URL for the "compiled" asset.

- **bg-[url('../img/politecnico_h.svg')]** is specifying the background image using Tailwind CSS arbitrary values. Also, note that when using the "`url()`" function, **vite** recalculates the "final" URL for the "compiled" asset.
- The size of the user's name (on the top navigation bar) is calculated with Tailwind CSS arbitrary values – uses the `calc()` function to calculate the maximum width. The maximum width of the user's name depends on the width of the device (`100vw`) and changes for different breakpoints.

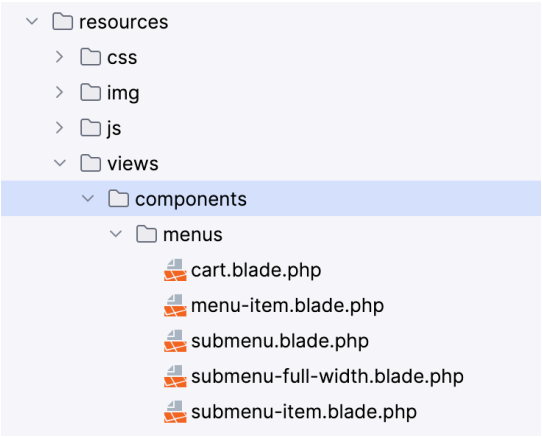
```
. . .    sm:max-w-[calc(100vw-39rem)] md:max-w-[calc(100vw-41rem)]
         lg:max-w-[calc(100vw-46rem)] xl:max-w-[34rem] . . .
```

If you want to **reuse the layout** (template) on a different context, with a different set of menu items and different elements, you must adapt the width of the user's name by **changing the classes of the user's name**

- The layout uses components, that are referred on the HTML code as tags with the prefix **`x-`**. Examples of components used by the layout are: <x-menus.menu-item …>, <x-menus.submenu-full-width …> and <x-menus.submenu …> and others (check the full code). These components will have properties (examples: `content`, `href`, `selectable`, `selected`, etc.), that affect their behavior or design.
- Each component is defined by:
  - A class with the component's code on the folder: `app/View/Components`



  - A view with the component's design (blade template) on the folder: `resources/views/components`



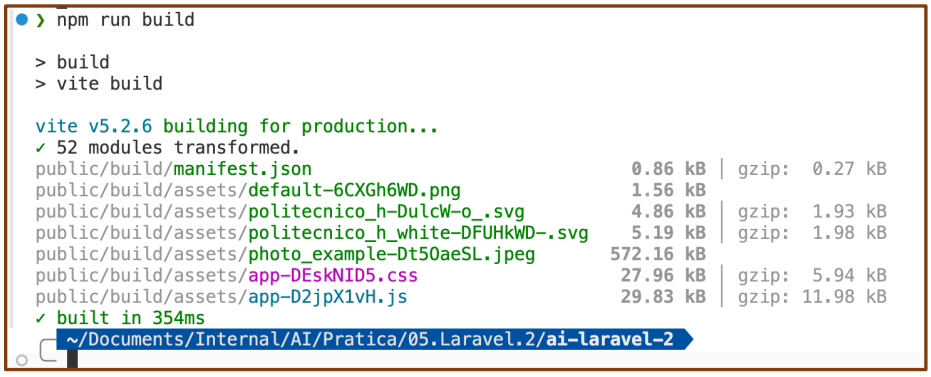
- Provided project includes 5 components: `menus.menu-item`; `menus.submenu`; `menus.submenu-full-width`; `menus.submenu-item` and `menus.cart`. These components were created to simplify the code of the layout design. Later, we will learn how to create our own components.

54. To ensure that **vite** will process all images inside the `resources/img` folder when compiling the application (it will copy -with a different name- the images to a publicly accessible folder), add the following code to the JavaScript entry point (`resources/js/app.js`):

```
import.meta.glob([
  '../img/**',
]);
```

55. Interrupt the execution of `npm run dev` and execute:

```
npm run build
```



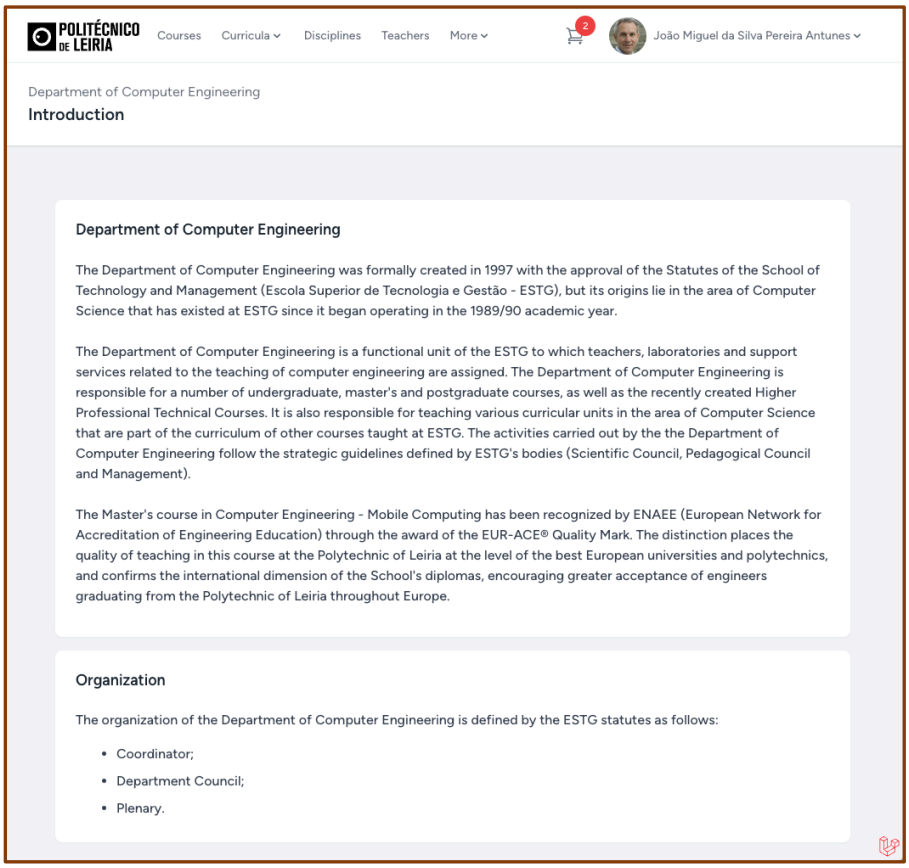
56. Re-run "`npm run dev`" by executing:

```
npm run dev
```

57. Change the layout used by the home view. Edit the file `resources/views/home.blade.php`:

```
@extends('layouts.main')

@section('header-title', 'Introduction')

@section('main')
<main>
    . . .
</main>
@endsection
```

58. Open the URL `http://yourdomain` on the browser.



59. The layout design is complete. Also, the submenus ("`Curricula`"; "`More`" and "`User's Menu`") are opened correctly. However, when we reduce the width of the browser the "hamburger" button does not work – when we click it, nothing happens. This is because the JavaScript `menu.js` file is not compiled yet.

60. On the JavaScript entry point - file `resources/js/app.js` – import the `menu.js` file. Complete code for the `app.js` file:

```
import './bootstrap';

import './menu'

import.meta.glob([
  '../img/**',
]);
```

61. Open the URL `http://yourdomain` again and verify if the behavior of the hamburger button is correct.
62. One final detail is required to complete the layout. The submenu of "Curricula" is composed of 10 static items. We will replace them with code that loads the items from the database.
63. First, we need to pass the variable "`$courses`" to the layout. We could do this with the following code: `return view('someView')->with('courses', $courses)`. However, this approach would require repeating this code for all the views that use the layout, i.e. most of the views. Instead, we will use the View share method that allows us to share data (variables) on all views (https://laravel.com/docs/views#sharing-data-with-all-views). Add the following code to the `App\Providers\AppServiceProvider.php` file:

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\View;
use App\Models\Course;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     */
    public function register(): void
    {
        //
    }
```

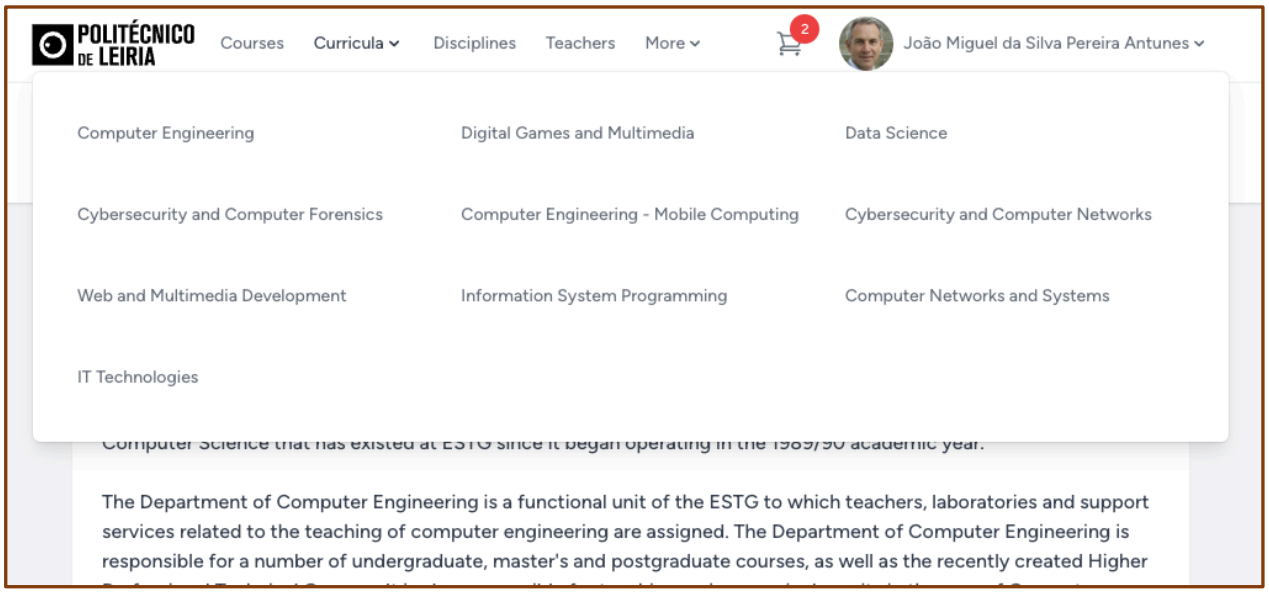
```
    /**
     * Bootstrap any application services.
     */
    public function boot(): void
    {
        // View::share adds data (variables) that are shared through all views
        View::share('courses', Course::all());
    }
}
```

- Previous code ensures that the variable `$courses` is available to all views. A similar pattern could be used to any variable that is required in all views.

64. Edit the code of the layout file `resources/views/layouts/main.blade.php` – change the section that defines the submenu of `curricula`:

```
. . .
<!-- Menu Item: Curricula -->
<x-menus.submenu-full-width
    content="Curricula"
    selectable="1"
    selected="0"
    uniqueName="submenu_curricula">
    @foreach ($courses as $course)
        <x-menus.submenu-item
        content="{{ $course->name }}"
        selectable="1"
        selected="0"
        href="#"/>
    @endforeach
</x-menus.submenu-full-width>
. . .
```

65. Open the `http://yourdomain` page and check the curricula submenu content:



66. The name of the courses is still missing the prefix "Master's in " or "TeSP – " for the master and TESP courses. Instead of adding an expression directly on the view, we will create a *"computed"* property on the Course model class. Add the method `getFillNameAttribute` to the Course class - file `app/Models/Course.php`:

```php
<?php
. . .
class Course extends Model
{
    . . .
    protected $keyType = 'string';

    public function getFullNameAttribute()
    {
        return match ($this->type) {
            'Master'    => "Master's in ",
            'TESP'      => 'TeSP – ',
            default     => ''
        } . $this->name;
    }
}
```

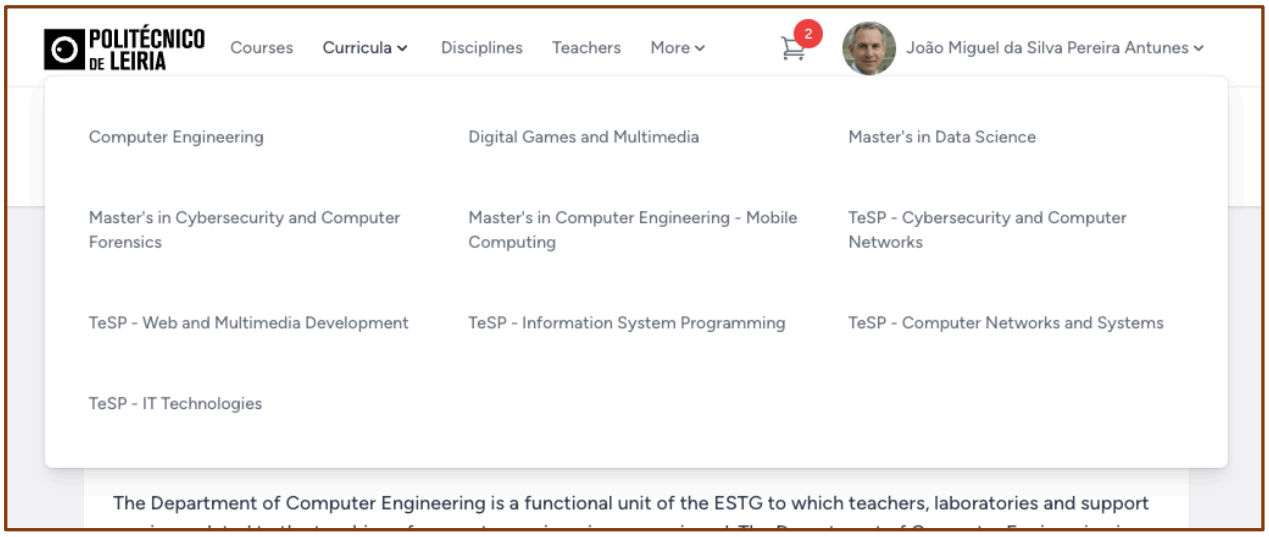
- The value returned by a method called `get`**`Xyz`**`Attribute()` of a model class can be accessed as the attribute with the name `xyz`. This means that the method `getFullNameAttribute` can be accessed as the attribute `fullName`.

67. Edit the code of the layout file `resources/views/layouts/main.blade.php` – replace the expression that writes the `name` property with an expression that writes the `fullName`:

```
. . .
<!-- Menu Item: Curricula -->
<x-menus.submenu-full-width
    content="Curricula"
    selectable="1"
    selected="0"
    uniqueName="submenu_curricula">
    @foreach ($courses as $course)
        <x-menus.submenu-item
        :content="$course->fullName"
        selectable="1"
        selected="0"
        href="#"/>
    @endforeach
</x-menus.submenu-full-width>
```

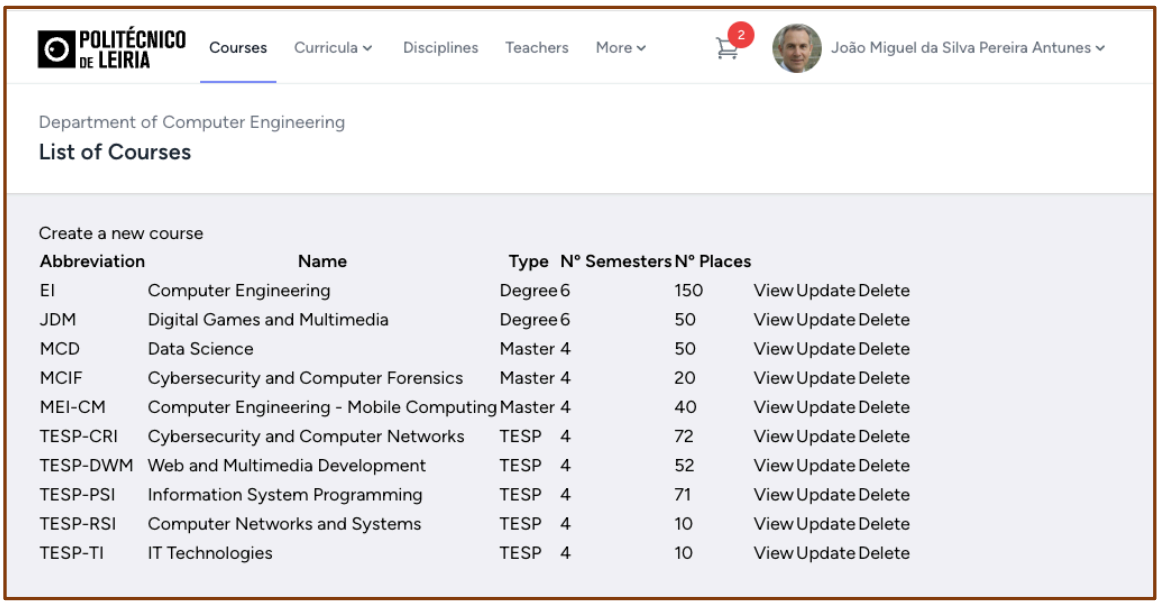
- Note that content property value is a PHP expression

68. Open the `http://yourdomain` page and check the curricula submenu content:



69. Redefine all remaining views (except subviews) relative to the courses and disciplines, so that all views use the new layout (`layouts.main`).

70. Try the application and verify that it maintains a consistent layout and visual appearance – for instance, the menu is present on all views.



71. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.4.zip`").

5. Courses index view & View Components

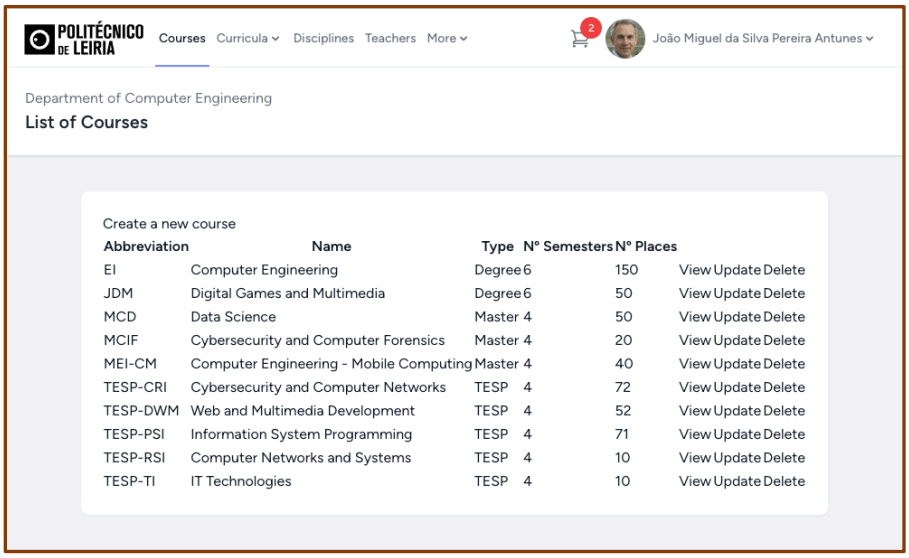
In this section we're adapting the "courses" index view (with the list of courses) to the template. Also, we're going to implement our first **View Components** that will be reused later – a hyperlink text button and 3 table icon buttons (view, edit or delete).

72. Edit the "`courses.index`" view (file `resources/views/courses/index.blade.php`) and add 2 elements that includes all the original view content, with the following classes:

```
@extends('layouts.main')

@section('header-title', 'List of Courses')
```

```
@section('main')
    <div class="flex justify-center">
        <div class="my-4 p-6 bg-white dark:bg-gray-900 overflow-hidden
                    shadow-sm sm:rounded-lg text-gray-900 dark:text-gray-50">
            <p>
            . . .
            </table>
        </div>
    </div>
@endsection
```

73. Open the courses page (`http://yourdomain/courses`):



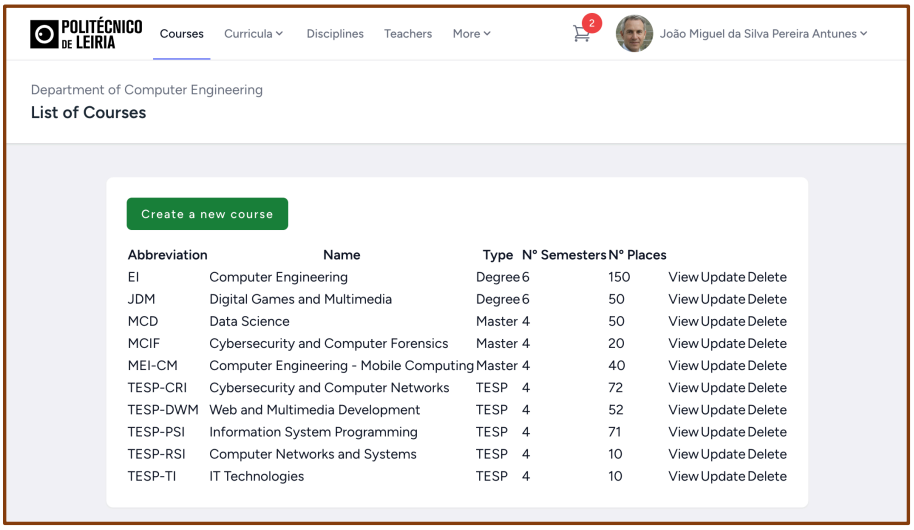
- The visual aspect of the page is not updated if the TailwindCSS builder in not running. Ensure that the following command is running:

```
npm run dev
```

74. The link to "`Create a new course`" will be transformed (visually) to a button by replacing the `<p>` and `<a>` elements with the following code:

```
. . .
@section('main')
    <div class="flex justify-center">
        <div class="my-4 p-6 bg-white dark:bg-gray-900 overflow-hidden
                    shadow-sm sm:rounded-lg text-gray-900 dark:text-gray-50">
```

```
<div class="flex items-center gap-4 mb-4">
    <a href="{{ route('courses.create') }}"
       class="px-4 py-2 inline-block border border-transparent rounded-md
              font-medium text-sm tracking-widest
              text-white dark:text-gray-900
              bg-green-700 dark:bg-green-200
              hover:bg-green-800 dark:hover:bg-green-100
              focus:bg-green-800 dark:focus:bg-green-100
              active:bg-green-900 dark:active:bg-green-100
              focus:outline-none focus:ring-2
              focus:ring-indigo-500 dark:focus:ring-indigo-400
              focus:ring-offset-2 transition ease-in-out duration-150">
        Create a new course
    </a>
</div>
<table>
. . .
```

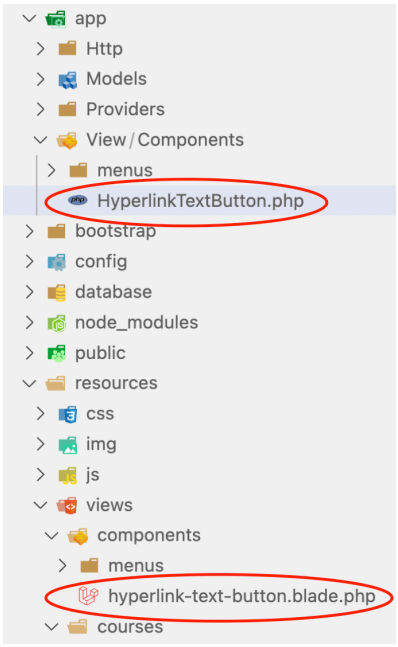


75. The hyperlink button (`<a>` element that is visually represented as a button with text) will be used throughout the application. To reduce code repetition, we will create a **View Component** for this type of button with the name: "`HyperlinkTextButton`". Start by creating the View Component files with the following command:

```
php artisan make:component HyperlinkTextButton
```

76. Previous command creates 2 files:

    - `app/View/Components/HyperlinkTextButton.php` – Class with the components' code
    - `resources/views/components/hyperlink-text-button.blade.php` – Blade with the components' design (HTML)



77. Our "`HyperlinkTextButton`" component will have 3 properties: "`text`", "`href`" and "`type`". Edit the code of `app/View/Components/HyperlinkTextButton.php` to add these 3 properties:

```php
<?php

namespace App\View\Components;

use Closure;
use Illuminate\Contracts\View\View;
use Illuminate\View\Component;

class HyperlinkTextButton extends Component
{
    public function __construct(
        public string $text = '',
        public string $href = '#',
        public string $type = 'dark',
    ) {
        $this->type = strtolower($type);
        if (!in_array($this->type, ['primary', 'secondary', 'success', 'danger',
                    'warning', 'info', 'light', 'dark', 'link'], true)) {
            $this->type = 'dark';
        }
        $this->text = trim($text) ?: ucfirst($this->type);
    }
```

```
    public function render(): View|Closure|string
    {
        return view('components.hyperlink-text-button');
    }
}
```

- Note that all 3 properties have a default value (`text` = '', `href` = '`#`' and `type` = '`dark`')
- The property `type` supports the following values: `primary`, `secondary`, `success`, `danger`, `warning`, `info`, `light`, `dark` and `link`.
- When `text` property is empty (empty string) it will assume the same value as the property `type`.

78. Next, we will change the design of the "`HyperlinkTextButton`" component. Edit the code of `resources/views/components/hyperlink-text-button.blade.php` file:

```
@php
    $colors = match($type) {
        'primary' => 'text-white dark:text-gray-900
                      bg-blue-600 dark:bg-blue-400
                      hover:bg-blue-700 dark:hover:bg-blue-300
                      focus:bg-blue-700 dark:focus:bg-blue-300
                      active:bg-blue-800 dark:active:bg-blue-200',
        'secondary' => 'text-white dark:text-gray-700
                      bg-gray-500 dark:bg-gray-400
                      hover:bg-gray-600 dark:hover:bg-gray-300
                      focus:bg-gray-600 dark:focus:bg-gray-300
                      active:bg-gray-700 dark:active:bg-gray-200',
        'success' => 'text-white dark:text-gray-900
                      bg-green-700 dark:bg-green-200
                      hover:bg-green-800 dark:hover:bg-green-100
                      focus:bg-green-800 dark:focus:bg-green-100
                      active:bg-green-900 dark:active:bg-green-100',
        'danger' => 'text-white dark:text-gray-900
                      bg-red-600 dark:bg-red-200
                      hover:bg-red-700 dark:hover:bg-red-100
                      focus:bg-red-700 dark:focus:bg-red-100
                      active:bg-red-800 dark:active:bg-red-100',
        'warning' => 'text-gray-900 dark:text-gray-200
                      bg-amber-400 dark:bg-amber-600
                      hover:bg-amber-300 dark:hover:bg-amber-700
                      focus:bg-amber-300 dark:focus:bg-amber-700
                      active:bg-amber-300 dark:active:bg-amber-700',
```

```
        'info' => 'text-gray-900 dark:text-gray-200
                        bg-cyan-400 dark:bg-cyan-600
                        hover:bg-cyan-300 dark:hover:bg-cyan-700
                        focus:bg-cyan-300 dark:focus:bg-cyan-700
                        active:bg-cyan-300 dark:active:bg-cyan-700',
        'light' => 'text-gray-900 dark:text-gray-200
                        bg-slate-50 dark:bg-slate-600
                        hover:bg-slate-200 dark:hover:bg-slate-700
                        focus:bg-slate-200 dark:focus:bg-slate-700
                        active:bg-slate-200 dark:active:bg-slate-700',
        'link' => 'text-blue-500
                        border-gray-200',
        default => ' text-white dark:text-gray-900
                        bg-gray-800 dark:bg-gray-200
                        hover:bg-gray-900 dark:hover:bg-gray-100
                        focus:bg-gray-900 dark:focus:bg-gray-100
                        active:bg-gray-950 dark:active:bg-gray-50',
    }
@endphp
<div>
    <a href="{{ $href }}"
       class="px-4 py-2 inline-block border border-transparent rounded-md
              font-medium text-sm tracking-widest
              focus:outline-none focus:ring-2
              focus:ring-indigo-500 dark:focus:ring-indigo-400
              focus:ring-offset-2 transition ease-in-out duration-150 {{ $colors }}">
        {{ $text }}
    </a>
</div>
```

- The variable `$colors` defines a different set of classes for the colors, that depends on the value of the `$type` variable.
- Usage example:
  `<x-hyperlink-text-button type="success">`

79. The "`HyperlinkTextButton`" component is complete. Now we just use it the view. Edit the "`courses.index`" view (file `resources/views/courses/index.blade.php`) and replace the "`Create a new course`" button with the "`HyperlinkTextButton`" component:

```
. . .
@section('main')
    <div class="flex justify-center">
        <div class="my-4 p-6 bg-white dark:bg-gray-900 overflow-hidden
                    shadow-sm sm:rounded-lg text-gray-900 dark:text-gray-50">
            <div class="flex items-center gap-4 mb-4">
                <x-hyperlink-text-button
                    href="{{ route('courses.create') }}"
                    text="Create a new course"
                    type="success"/>
            </div>
            . . .
```

- The visual aspect of the page should remain exactly the same.
- By convention, the View Component will be used as a **`tag`** with the prefix **`x-`**, and the component property values will be defined with tag **`attributes`**.

```
<x-name-of-component property_name="value">
      … some content …
</x-name-of-component>
```

When no content is required:

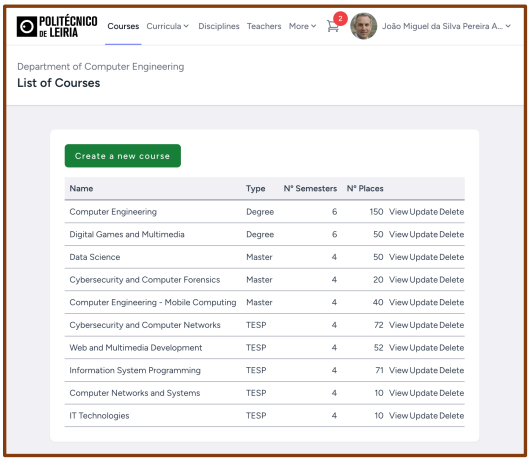
```
<x-name-of-component property_name="value"/>
```

80. Next, we will change the visual aspect of the `<table>` element and content. Replace the `<table>` code with the following:

```
. . .
@section('main')
    <div class="flex justify-center">
        <div class="my-4 p-6 . . . ">
            <div class="flex items-center gap-4 mb-4">
                <x-hyperlink-text-button
                    . . . />
            </div>
            <div class="font-base text-sm text-gray-700 dark:text-gray-300">
                <table class="table-auto border-collapse">
```

```
<thead>
<tr class="border-b-2 border-b-gray-400 dark:border-b-gray-500
           bg-gray-100 dark:bg-gray-800">
    <th class="px-2 py-2 text-left hidden lg:table-cell">Abbreviation</th>
    <th class="px-2 py-2 text-left">Name</th>
    <th class="px-2 py-2 text-left">Type</th>
    <th class="px-2 py-2 text-right hidden sm:table-cell">Nº Semesters</th>
    <th class="px-2 py-2 text-right hidden sm:table-cell">Nº Places</th>
    <th></th>
    <th></th>
    <th></th>
</tr>
</thead>
<tbody>
@foreach ($courses as $course)
    <tr class="border-b border-b-gray-400 dark:border-b-gray-500">
        <td class="px-2 py-2 text-left hidden lg:table-cell">
              {{ $course->abbreviation }}</td>
        <td class="px-2 py-2 text-left">{{ $course->name }}</td>
        <td class="px-2 py-2 text-left">{{ $course->type }}</td>
        <td class="px-2 py-2 text-right hidden sm:table-cell">
              {{ $course->semesters }}</td>
        <td class="px-2 py-2 text-right hidden sm:table-cell">
              {{ $course->places }} </td>
        <td>
            <a href="{{ route('courses.show', ['course' => $course]) }}">
                View</a>
        </td>
        <td>
            <a href="{{ route('courses.edit', ['course' => $course]) }}">
                 Update</a>
        </td>
        <td>
            <form method="POST"
            action="{{ route('courses.destroy', ['course' => $course]) }}">
                @csrf
                @method('DELETE')
                <button type="submit" name="delete">Delete</button>
            </form>
        </td>
    </tr>
@endforeach
</tbody>
```

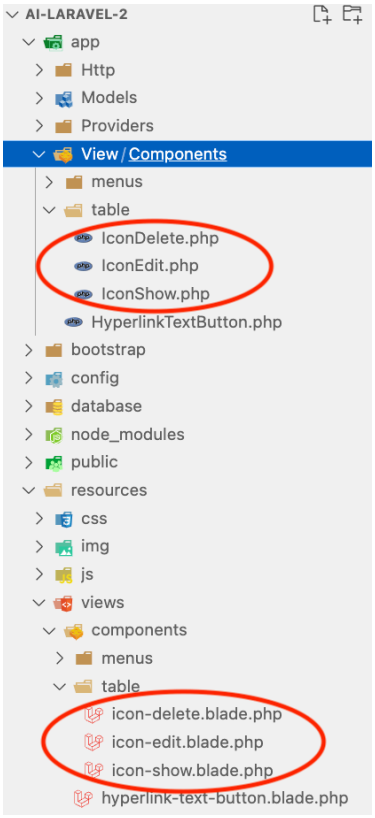
```
            </table>
        </div>
    </div>
</div>
@endsection
```



- Try to change the width of the browser. Some columns ("abbreviation", "Nº Semesters" and "Nº Places") are hidden when the browser width is smaller than specific breakpoints.

81. Next, we will create and use 3 components, to replace the options: "View"; "Update" and "Delete" of each table row. Create the 3 components by executing these 3 commands:

```
php artisan make:component table/IconShow
php artisan make:component table/IconEdit
php artisan make:component table/IconDelete
```



- Previous command generates 3 component code files, on the folder `app/View/Components/table` and 3 design (views) files on the folder `resources/views/components/table`

82. Change the `IconShow` constructor (file `app/View/Components/table/IconShow`):

```
. . .
    public function __construct(
        public string $href = '#',
    ) {
        //
    }
. . .
```

83. Change the `IconEdit` constructor (file `app/View/Components/table/IconEdit`):

```
. . .
    public function __construct(
        public string $href = '#',
    ) {
        //
    }
. . .
```

84. Change the `IconDelete` constructor (file `app/View/Components/table/IconDelete`):

```
. . .
    public function __construct(
        public string $action = '#',
    ) {
        //
    }
. . .
```

- While `IconShow` and `IconEdit` have the property `href`, the `IconDelete` has the property `action`. First 2 components will generate a `hyperlink` and the latter a `form`.

85. Add the following code to the `IconShow` view (file `resources/views/components/table/icon-show.blade.php`):

```
<div {{ $attributes->merge(['class' => 'hover:text-gray-900']) }}>
    <a href="{{ $href }}">
        <svg class="hover:stroke-2 w-6 h-6" xmlns="http://www.w3.org/2000/svg"
             fill="none" viewBox="0 0 24 24" stroke-width="1" stroke="currentColor">
            <path stroke-linecap="round" stroke-linejoin="round" d="M2.036
                  12.322a1.012 1.012 0 0 1 0-.639C3.423 7.51 7.36 4.5 12 4.5c4.638
                  0 8.573 3.007 9.963 7.178.07.207.07.431 0 .639C20.577 16.49 16.64
                  19.5 12 19.5c-4.638 0-8.573-3.007-9.963-7.178Z" />
            <path stroke-linecap="round" stroke-linejoin="round" d="M15 12a3 3 0 1
                  1-6 0 3 3 0 0 1 6 0Z" />
        </svg>
    </a>
</div>
```

- The method `$attributes->merge(['class' => …` will merge the value of the class attribute defined on the component, with the value of the class attribute when the component is used. This will allow us to customize the component when used, by adding specific classes to it.
    - Usage example:
      `<x-icon-show href="…" class="ml-3">`
    - Previous example would add the class `ml-3` (defined on the view) to the class `hover:text-gray-900` (defined on the component)

86. Add the following code to the `IconEdit` view

    (file `resources/views/components/table/icon-edit.blade.php`):

```
<div {{ $attributes->merge(['class' => 'hover:text-blue-600']) }}>
    <a href="{{ $href }}">
        <svg class="hover:stroke-2 w-6 h-6" xmlns="http://www.w3.org/2000/svg"
             fill="none" viewBox="0 0 24 24" stroke-width="1" stroke="currentColor" >
            <path stroke-linecap="round" stroke-linejoin="round" d="m16.862 4.487
                  1.687-1.688a1.875 1.875 0 1 1 2.652 2.652L10.582 16.07a4.5 4.5 0 0
                  1-1.897 1.13L6 18l.8-2.685a4.5 4.5 0 0 1 1.13-1.897l8.932-8.931Zm0
                  0L19.5 7.125M18 14v4.75A2.25 2.25 0 0 1 15.75 21H5.25A2.25 2.25 0 0
                  1 3 18.75V8.25A2.25 2.25 0 0 1 5.25 6H10" />
        </svg>
    </a>
</div>
```

87. Add the following code to the `IconDelete` view

    (file `resources/views/components/table/icon-delete.blade.php`):

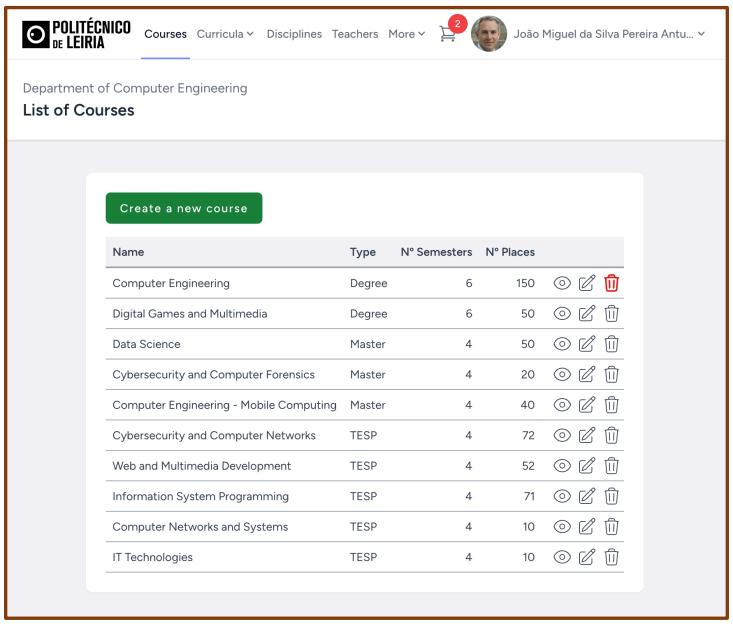
```
<div {{ $attributes->merge(['class' => 'hover:text-red-600']) }}>
    <form method="POST" action="{{ $action }}"  class="w-6 h-6">
        @csrf
        @method('DELETE')
        <button type="submit" name="delete" class="w-6 h-6">
            <svg class="hover:stroke-2 w-6 h-6" xmlns="http://www.w3.org/2000/svg"
               fill="none" viewBox="0 0 24 24" stroke-width="1" stroke="currentColor">
            <path stroke-linecap="round" stroke-linejoin="round" d="m14.74 9-.346
                  9m-4.788 0L9.26 9m9.968-3.21c.342.052.682.107
                  1.022.166m-1.022-.165L18.16 19.673a2.25 2.25 0 0 1-2.244
                  2.077H8.084a2.25 2.25 0 0 1-2.244-2.077L4.772 5.79m14.456 0a48.108
                  48.108 0 0 0-3.478-.397m-12 .562c.34-.059.68-.114 1.022-.165m0
                  0a48.11 48.11 0 0 1 3.478-.397m7.5
                  0v-.916c0-1.18-.91-2.164-2.09-2.201a51.964 51.964 0 0 0-3.32
                  0c-1.18.037-2.09 1.022-2.09 2.201v.916m7.5 0a48.667 48.667 0 0
                  0-7.5 0" />
            </svg>
        </button>
    </form>
</div>
```

- Note that the `<path>` elements define the vectors for the icons.

88. Replace the last 3 cells of each row of the table in the `courses.index`, (file `resources/views/courses/index.blade.php`) so that they use the 3 newly created components:

```
. . .
@section('main')
    . . .
    @foreach ($courses as $course)
       <tr class=". . . ">
            . . .
            <td>
                 <x-table.icon-show class="ps-3 px-0.5"
                    href="{{ route('courses.show', ['course' => $course]) }}"/>
            </td>
            <td>
                 <x-table.icon-edit class="px-0.5"
                    href="{{ route('courses.edit', ['course' => $course]) }}"/>
            </td>
            <td>
                <x-table.icon-delete class="px-0.5"
                    action="{{ route('courses.destroy', ['course' => $course]) }}"/>
            </td>
       </tr>
    @endforeach
    . . .
@endsection
```

89. Open the courses page (`http://yourdomain/courses`) and try to change the width of the browser and to hover over the table icons. Visual aspect of the page should be similar to:



90. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.5.zip`").

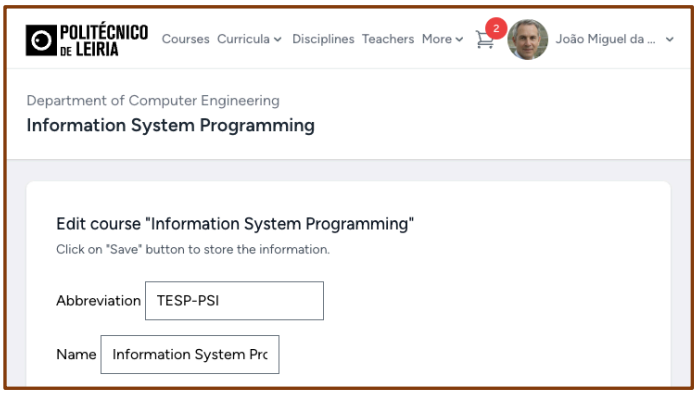
6. Courses form & View Components

In this section we're adapting to the template, the form to update, create and show a "course". Also, we're going to implement **View Components** to help us create consistent form fields through the application.

91. Edit the "`courses.edit`" view (file `resources/views/courses/edit.blade.php`):

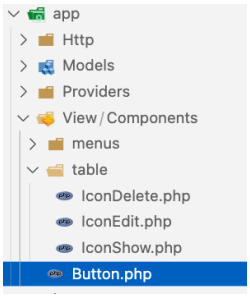
```
@extends('layouts.main')

@section('header-title', $course->name)

@section('main')
<div class="flex flex-col space-y-6">
    <div class="p-4 sm:p-8 bg-white dark:bg-gray-900 shadow sm:rounded-lg">
        <div class="max-full">
            <section>
                <header>
                    <h2 class="text-lg font-medium text-gray-900 dark:text-gray-100">
                        Edit course "{{ $course->name }}"
                    </h2>
                    <p class="mt-1 text-sm text-gray-600 dark:text-gray-300  mb-6">
                        Click on "Save" button to store the information.
                    </p>
                </header>

                <form method="POST"
                      action="{{ route('courses.update', ['course' => $course]) }}">
                    @csrf
                    @method('PUT')
                    <div class="mt-6 space-y-4">
                         @include('courses.shared.fields')
                    </div>
                    <div>
                        <button type="submit" name="ok">Save course</button>
                    </div>
                </form>
            </section>
        </div>
    </div>
</div>
@endsection
```

92. Open the page to edit a course. It should be similar to:



93. The visual aspect of the "Save" button will be similar to the hyperlink text button previously created (Create a new course). However, the component we have created before only supports hyperlink (`<a>`) elements. We're going to refactor the component, so that it also supports button (`<button>`) elements.

94. Change the name of the file `app/View/Components/`**`HyperlinkTextButton`**`.php` to `app/View/Components/`**`Button`**`.php`.



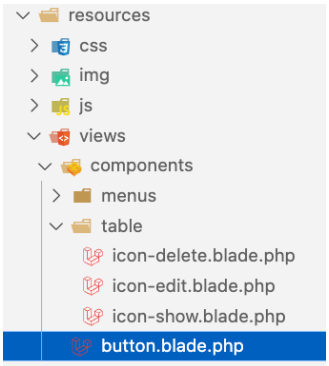
95. Change the code of the file `app/View/Components/`**`Button`**`.php`:

```
<?php

namespace App\View\Components;

use Closure;
use Illuminate\Contracts\View\View;
use Illuminate\View\Component;

class Button extends Component
{
```

```
    public function __construct(
        public string $element = 'a',
        public string $buttonName = '',
        public string $text = '',
        public string $href = '#',
        public string $type = 'dark',
    ) {
        $this->element = strtolower($element);
        if (!in_array($this->element, ['a', 'button', 'submit', 'reset'], true)) {
            $this->element = 'a';
        }
        $this->type = strtolower($type);
        if (!in_array($this->type, ['primary', 'secondary', 'success', 'danger',
                    'warning', 'info', 'light', 'dark', 'link'], true)) {
            $this->type = 'dark';
        }
        $this->buttonName = trim($buttonName);
        $this->text = trim($text) ?: ucfirst($this->type);
    }

    public function render(): View|Closure|string
    {
        return view('components.button');
    }
}
```

- The view to render the component will be changed to "`components.button`". Check the method `render()`.
- The component refactoring adds 2 new properties: `element` and `buttonName`:
    - `element` – defines which HTML element to use. Possible values are:
        - `a` - element `<a>`
        - `button` – element `<button type="button">`
        - `submit` – element `<button type="submit">`
        - `reset` – element `<button type="reset">`
    - `buttonName`– defines the name when the element is a `<button>`:
        - `<button … name="buttonName">`

96. Change the name of the file `resources/views/components/`**`hyperlink-text-button`**`.blade.php` to `resources/views/components/`**`button`**`.blade.php`.



97. Change the code of the file `resources/views/components/`**`button`**`.blade.php`:

```
@php
    $colors = match($type) {
        'primary' => 'text-white dark:text-gray-900
                      bg-blue-600 dark:bg-blue-400
                      hover:bg-blue-700 dark:hover:bg-blue-300
                      focus:bg-blue-700 dark:focus:bg-blue-300
                      active:bg-blue-800 dark:active:bg-blue-200',
        'secondary' => 'text-white dark:text-gray-700
                      bg-gray-500 dark:bg-gray-400
                      hover:bg-gray-600 dark:hover:bg-gray-300
                      focus:bg-gray-600 dark:focus:bg-gray-300
                      active:bg-gray-700 dark:active:bg-gray-200',
        'success' => 'text-white dark:text-gray-900
                      bg-green-700 dark:bg-green-200
                      hover:bg-green-800 dark:hover:bg-green-100
                      focus:bg-green-800 dark:focus:bg-green-100
                      active:bg-green-900 dark:active:bg-green-100',
        'danger' => 'text-white dark:text-gray-900
                      bg-red-600 dark:bg-red-200
                      hover:bg-red-700 dark:hover:bg-red-100
                      focus:bg-red-700 dark:focus:bg-red-100
                      active:bg-red-800 dark:active:bg-red-100',
        'warning' => 'text-gray-900 dark:text-gray-200
                      bg-amber-400 dark:bg-amber-600
                      hover:bg-amber-300 dark:hover:bg-amber-700
                      focus:bg-amber-300 dark:focus:bg-amber-700
                      active:bg-amber-300 dark:active:bg-amber-700',
        'info' => 'text-gray-900 dark:text-gray-200
                      bg-cyan-400 dark:bg-cyan-600
                      hover:bg-cyan-300 dark:hover:bg-cyan-700
                      focus:bg-cyan-300 dark:focus:bg-cyan-700
                      active:bg-cyan-300 dark:active:bg-cyan-700',
```

```
        'light' => 'text-gray-900 dark:text-gray-200
                        bg-slate-50 dark:bg-slate-600
                        hover:bg-slate-200 dark:hover:bg-slate-700
                        focus:bg-slate-200 dark:focus:bg-slate-700
                        active:bg-slate-200 dark:active:bg-slate-700',
        'link' => 'text-blue-500
                        border-gray-200',
        default => 'text-white dark:text-gray-900
                        bg-gray-800 dark:bg-gray-200
                        hover:bg-gray-900 dark:hover:bg-gray-100
                        focus:bg-gray-900 dark:focus:bg-gray-100
                        active:bg-gray-950 dark:active:bg-gray-50',
    }
@endphp
<div {{ $attributes }}>
    @if ($element == 'a')
        <a href="{{ $href }}"
            class="px-4 py-2 inline-block border border-transparent rounded-md
               font-medium text-sm tracking-widest
               focus:outline-none focus:ring-2
               focus:ring-indigo-500 dark:focus:ring-indigo-400
               focus:ring-offset-2 transition ease-in-out duration-150 {{ $colors }}">
            {{ $text }}
        </a>
    @else
        <button type="{{ $element }}" {{ $buttonName ? "name='$buttonName'" : '' }}
            class="px-4 py-2 inline-block border border-transparent rounded-md
               font-medium text-sm tracking-widest
               focus:outline-none focus:ring-2
               focus:ring-indigo-500 dark:focus:ring-indigo-400
               focus:ring-offset-2 transition ease-in-out duration-150 {{ $colors }}">
            {{ $text }}
        </button>
    @endif
</div>
```

- Note that there are 2 alternative designs – the first uses a `<a>` element and the second uses a `<button>` element.

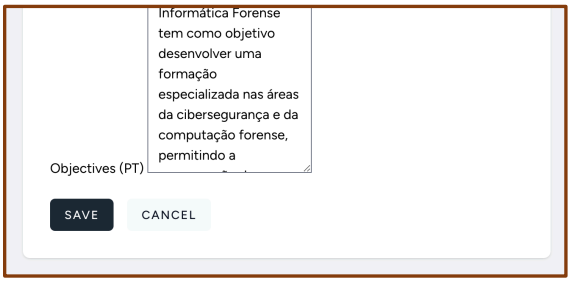
98. We've completed our "button" view component (`<x-button>` element). Now, we have to replace the old component with the new component. Change the `courses.index` view (file `resources/views/courses/index.blade.php`):

```
. . .
@section('main')
    <div class="flex justify-center">
        <div class="my-4 p-6 bg-white dark:bg-gray-900 overflow-hidden
                    shadow-sm sm:rounded-lg text-gray-900 dark:text-gray-50">
            <div class="flex items-center gap-4 mb-4">
                <x-button
                    href="{{ route('courses.create') }}"
                    text="Create a new course"
                    type="success"/>
            </div>
```

99. Now we're going use our new component to create the save button on the `courses.edit` view, and create a new button to "cancel" course editing – the latter is just an hyperlink that reloads the page without saving it. Change the `courses.edit` view (file `resources/views/courses/edit.blade.php`):

```
. . .
<form method="POST" action="{{ route('courses.update', ['course' => $course]) }}">
    @csrf
    @method('PUT')
    <div class="mt-6 space-y-4">
         @include('courses.shared.fields')
    </div>
    <div class="flex mt-6">
        <x-button element="submit" type="dark" text="Save" class="uppercase"/>
        <x-button element="reset" type="light" text="Cancel" class="uppercase ms-4"/>
    </div>
</form>
. . .
```

- Bottom of the courses edit page:



100. Our next step is to create components for the form fields. Execute the following commands:

```
php artisan make:component field/Input
php artisan make:component field/Select
php artisan make:component field/CheckBox
php artisan make:component field/RadioGroup
php artisan make:component field/TextArea
```

101. The components' code (Components' Classes) is available on the provided file `05-Laravel.2.code.txt`. Check the section `101)`.

102. The components' design code (Components' Vews) is available on the provided file `05-Laravel.2.code.txt`. Check the section `102)`.

103. Use the provided code to complete the components' class and view code. Analyze the provided code.

- All 5 components include the following properties:
  - `name` – mandatory property with the name of the field.
  - `value` – the value of the field (Boolean for the CheckBox and string for all other components)
  - `label` – the label of the field. On the checkbox it represents the textual description for the true value. On all other components it is the field title.
  - `readonly` – field is read-only (true) or writable (false - the default). Field is disabled when read-only.
  - `required` – field value is mandatory (true) or optional (false – the default)
  - `width` = the width of the field. Possible values are `full` (the default, that represents 100% of the parent), `xs`, `sm`, `md`, `lg` and `xl` (fixed width values) and the following relative (to the parent) values: `1/3`, `2/3`, `1/4`, `2/4`, `3/4`, `1/5`, `2/5`, `3/5` and `4/5`. The purpose of including a width property in the components is to ensure some consistency between them – it is possible to redefine the width with our own classes.
- The `x-field.input` creates an `<input>` element and includes the following extra property:
  - `type` – type of field. Possible values are `text` (the default), `password`, `number`, `email`, `date`, `time`, `datetime-local`, `month`, `week`, `range` and `color`

- The `x-field.select` creates an `<select>` element and includes the following extra properties:
    - `options` – an array with all options of the select dropdown list. Array keys will translate to the value of the `<option>`, and the array value to the "readable" text of the of the `<option>`.
    - `defaultValue` – if the `value` does not exist on the `options` array (there is no key with the value), the `defaultValue` will be used.
- The `x-field.radio-group` creates a set of radio (`<input type="radio">`) elements that work together as a group of values. The component `radio-group` and `select` do the same (provide a set of values for the end-user to choose from) but with different types of inputs. This component includes the following extra properties:
    - `options` – an array with the set of options available as radio elements. Array keys will translate to the value of the radio element, and the array value to the "readable" text associated to the radio element.
    - `defaultValue` – if the `value` does not exist on the `options` array (there is no key with the value), the `defaultValue` will be used.
- The `x-field.check-box` creates a check box element (`<input type="check">`) that represents a Boolean value. It does not include any extra property, but the property value is different from the value of other components:
    - `value` – the value of this component is a Boolean.
- The `x-field.text-area` creates a `<textarea>` element that represents a string value that can occupy multiple lines (versus an input text that is a single line input). Includes the following extra properties:
    - `height` = the height of the field. Possible values are `xs`, `sm`, `md` (the default value), `lg` and `xl`. They are all fixed height values. The purpose of including a height property in the component is to ensure some consistency.
    - `resizable` – a Boolean property that defines if the height of the element is resizable (true) or fixed (false – default value).

104. Our next step is to use the newly created components to define the fields of the courses form. Change the file `resources/views/courses/shared/fields.blade.php` to:

```
@php
    $mode = $mode ?? 'edit';
    $readonly = $mode == 'show';
@endphp
<x-field.input name="abbreviation" label="Abbreviation" width="md"
               :readonly="$readonly || ($mode == 'edit')"
               value="{{ $course->abbreviation }}"/>
<x-field.select name="type" label="Type of course" width="md" :readonly="$readonly"
               value="{{ $course->type }}"
               :options="[
                    'Degree' => 'Degree',
                    'Master' => 'Master',
                    'TESP' => 'TESP'
               ]"/>
<x-field.input name="name" label="Name" :readonly="$readonly"
              value="{{ $course->name }}"/>
<x-field.input name="name_pt" label="Name (Portuguese)" :readonly="$readonly"
              value="{{ $course->name_pt }}"/>
<div class="flex space-x-4">
    <x-field.input name="semesters" label="Nº Semesters" width="sm"
                   :readonly="$readonly"
                   value="{{ $course->semesters }}"/>
    <x-field.input name="ECTS" label="Nº ECTS" width="sm" :readonly="$readonly"
                   value="{{ $course->ECTS }}"/>
    <x-field.input name="places" label="Nº Places" width="sm" :readonly="$readonly"
                   value="{{ $course->places }}"/>
</div>
<x-field.input name="contact" label="Contact" :readonly="$readonly"
                   value="{{ $course->contact }}"/>
<x-field.text-area name="objectives" label="Objectives" :readonly="$readonly"
                   value="{{ $course->objectives }}"/>
<x-field.text-area name="objectives_pt" label="Objectives (Portuguese)"
                   :readonly="$readonly"
                   value="{{ $course->objectives_pt }}"/>
```

- `fields.blade.php` expect the variable `$mode` to be filled with the value "`edit`", "`create`" or "`show`" – it will define what the form is used for (editing; creating; showing).
- When the mode is "`show`" (form is showing data), all fields will be "read-only".
- If the mode is "`edit`", then the "`abbreviation`" field will also be "read-only". All other fields will be "writeable". If mode is "`create`" all fields will be "writeable".
- Note that when a property of a component is not a string or does not convert from a string, we should use the following syntax:

```
<x-component :property="expression" …>
```

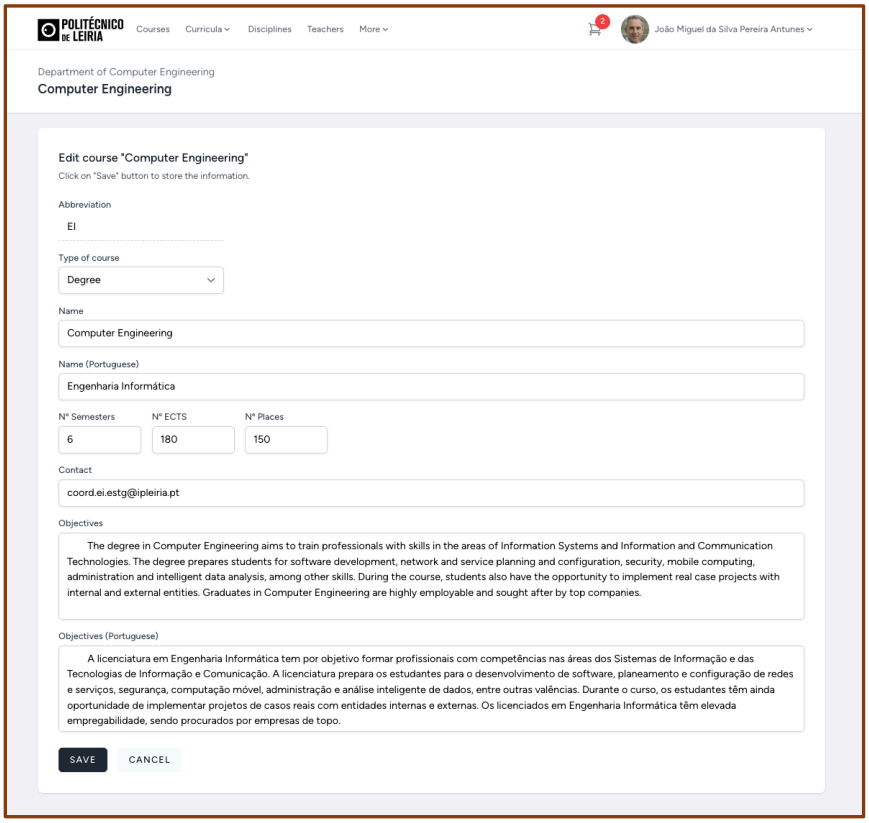
- For example, the property `readonly` expect a Boolean:

```
<x-field.input … :readonly="$readonly || ($mode == 'edit')" …/>
```

- Another example: the property `options` expect an array:

```
<x-field.select … :options="[
                    'Degree' => 'Degree',
                    'Master' => 'Master',
                    'TESP' => 'TESP'
                ]" …/>
```

105. Open the page to edit a course (e.g. `http://yourdomain/courses/MEI-CM/edit`)



- Confirm that abbreviation field is readonly.

106. Change the `courses.edit` view (`resources/views/courses/edit.blade.php`), so that it passes the `$mode` variable with "edit" value:

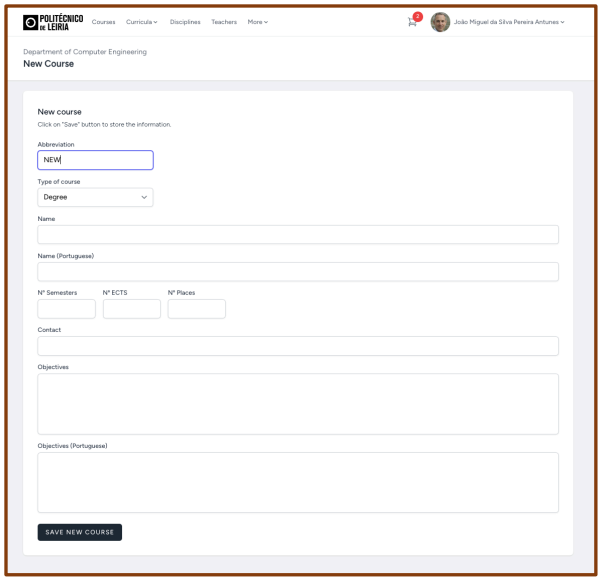
```
<form method="POST" action="{{ route('courses.update', ['course' => $course]) }}">
    @csrf
    @method('PUT')
    <div class="mt-6 space-y-4">
        @include('courses.shared.fields', ['mode' => 'edit'])
    </div>
    <div class="flex mt-6">
        <x-button element="submit" type="dark" text="Save" class="uppercase"/>
        <x-button element="reset" type="light" text="Cancel" class="uppercase ms-4"/>
    </div>
</form>
```

107. Change the `courses.create` view (`resources/views/courses/create.blade.php`):

```
@extends('layouts.main')

@section('header-title', 'New Course')

@section('main')
<div class="flex flex-col space-y-6">
    <div class="p-4 sm:p-8 bg-white dark:bg-gray-900 shadow sm:rounded-lg">
        <div class="max-full">
            <section>
                <header>
                    <h2 class="text-lg font-medium text-gray-900 dark:text-gray-100">
                        New course
                    </h2>
                    <p class="mt-1 text-sm text-gray-600 dark:text-gray-300  mb-6">
                        Click on "Save" button to store the information.
                    </p>
                </header>

                <form method="POST" action="{{ route('courses.store') }}">
                    @csrf
                    <div class="mt-6 space-y-4">
                        @include('courses.shared.fields', ['mode' => 'create'])
                    </div>
                    <div class="flex mt-6">
                        <x-button element="submit" type="dark" text="Save new course"
                                  class="uppercase"/>
                    </div>
                </form>
```

```
            </section>
        </div>
    </div>
</div>
@endsection
```

108. Open the page to create a new course (`http://yourdomain/courses/create`)

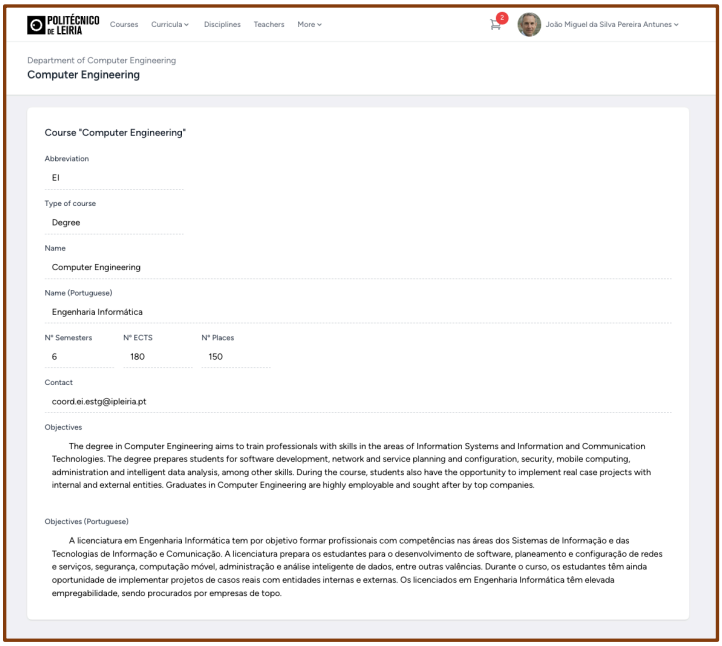


109. Change the `courses.show` view (`resources/views/courses/show.blade.php`):

```
@extends('layouts.main')

@section('header-title', $course->name)

@section('main')
<div class="flex flex-col space-y-6">
    <div class="p-4 sm:p-8 bg-white dark:bg-gray-900 shadow sm:rounded-lg">
        <div class="max-full">
            <section>
                <header>
                    <h2 class="text-lg font-medium text-gray-900 dark:text-gray-100">
                        Course "{{ $course->name }}"
                    </h2>
                </header>
                <div class="mt-6 space-y-4">
                    @include('courses.shared.fields', ['mode' => 'show'])
                </div>
```

```
            </section>
        </div>
    </div>
</div>
@endsection
```

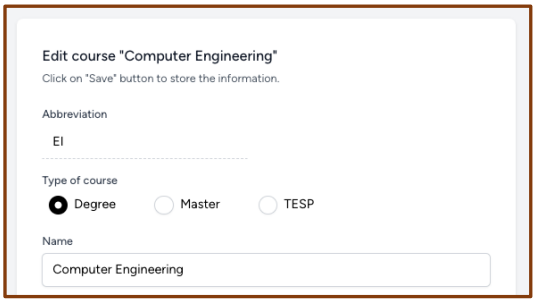
110. Open the page to show a course (e.g: `http://yourdomain/courses/ei`)



111. We'll implement a final detail on our course form. We'll replace the select field ("Type of course") with a radio group field. Edit the view file `resources/views/courses/shared/fields.blade.php`:

```
…
<x-field.input name="abbreviation" … />
<x-field.radio-group name="type" label="Type of course" :readonly="$readonly"
                value="{{ $course->type }}"
                :options="[
                     'Degree' => 'Degree',
                     'Master' => 'Master',
                     'TESP' => 'TESP'
                ]"/>
<x-field.input name="name" …/>
…
```

112. Open the page to edit a course (e.g. `http://yourdomain/courses/MEI-CM/edit`) and confirm that the "Type of course" field is now a radio group:



113. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.6.zip`").

7. Pagination

The courses page includes a `<table>` element with few rows (around 10 / 20 rows). However, there are cases where the number of rows may grow, and it maybe cumbersome to show all rows at once. Instead, we can paginate the dataset, so that the end-user only views a small subset (one page) of the rows at once. This is easily implemented using Laravel **pagination mechanism**, that handles all the code to paginate data from the database, and easily creates a pagination navigation bar that allows end-users to navigate through the pages of data.

In this section we'll create a pagination mechanism for the list of courses.

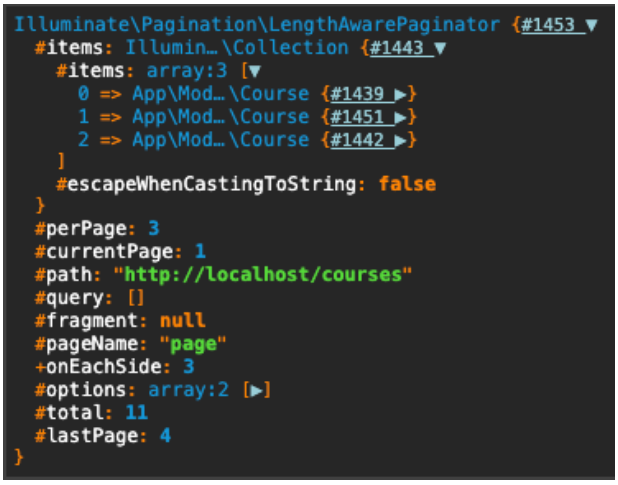
For academic purposes, each page will have only 3 lines (because the total number of courses is small).

114. First, we'll modify the controller so that instead of passing a regular (not paginated) dataset (`Eloquent Collection`) to the view, it passes a paginated dataset. Change the method `index` of the CourseController (file: `app/Http/Controllers/CourseController.php`):

```
public function index(): View
{
    $allCourses = Course::paginate(3);
    debug($allCourses);
    return view('courses.index')->with('courses', $allCourses);
}
```

- Variable `$allCourses` type is `Illuminate\Pagination\LengthAwarePaginator` instead of `Illuminate\Database\Eloquent\Collection`.

- `$allCourses` includes only 3 rows of data (1 page), but it includes also information about the pagination –current page, total rows per page, number of pages, etc.



115. Before generating the Pagination Navigation Bar on the view, we must configure TailwindCSS, so that it builds the classes of the navigation bar included with Laravel. Edit the file `tailwind.config.js` on the project root folder. Add a row to the "`content`" array:

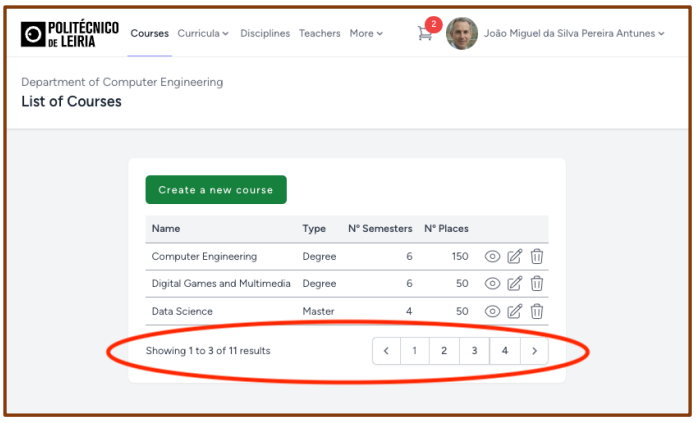
```
import defaultTheme from 'tailwindcss/defaultTheme';
import forms from '@tailwindcss/forms';
/** @type {import('tailwindcss').Config} */
export default {
    content: [
    "./resources/**/*.blade.php",
    "./resources/**/*.js",
    "./resources/**/*.vue",
    "./vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php",
  ],
  theme: {
    extend: {
        fontFamily: {
            sans: ['Figtree', ...defaultTheme.fontFamily.sans],
        },
    },
  },
  plugins: [forms],
}
```

116. Now, we just have to edit the file "`resources/views/courses/index.blade.php`" (`courses.index` view). Add the following code to generate the pagination navigation bar:

```
. . .
@section('main')
    <div …>
        <div …>
            <div …>
                <table …>
                 …
                </table>
            </div>
            <div class="mt-4">
                {{ $courses->links() }}
            </div>
        </div>
    </div>
@endsection
```

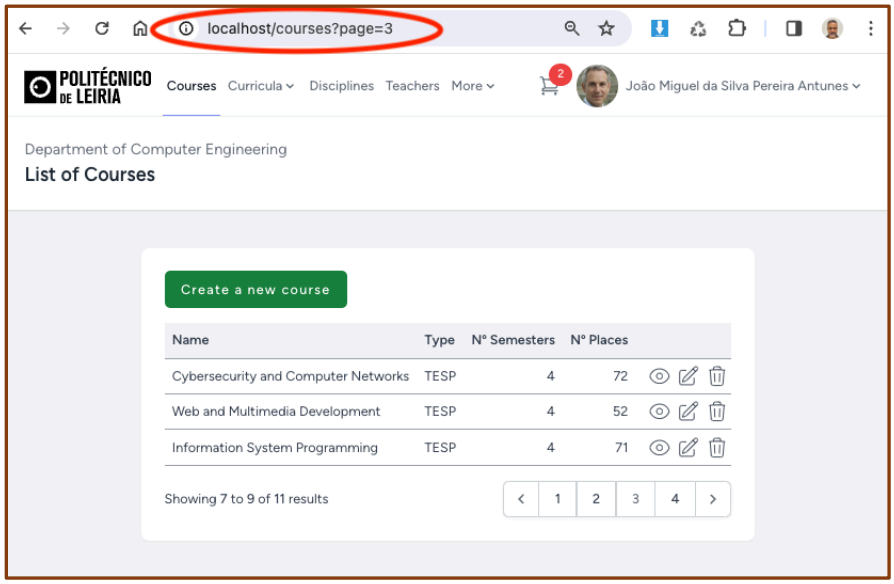
- The `links()` method of the `LengthAwarePaginator` object generates the pagination navigation bar.

117. Open the list of courses (`http://yourdomain/courses`).



- The Pagination Navigation Bar is added to the view.

118. Click on the link "3" to jump to the third page. This is a hyperlink (element `<a>`) to the current page but passing the query string parameter "`page`" with the value "3" (the number of the page) - e.g. `http://yourdomain/courses?page=3`.

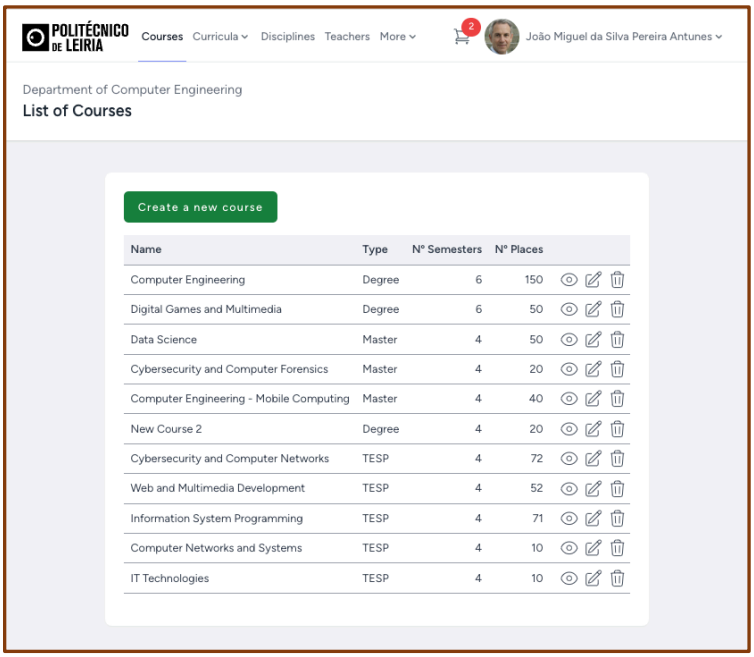


119. On the controller (`CourseController`) action method (`index`), the method `paginate()` of the eloquent model "`Course`" will verify if the query string parameter "`page`" is present on the Http Request. It will load the page specified by that parameter, or the first page otherwise.

120. Change the rows per page to 20 – file `app/Http/Controllers/CourseController.php`:

```
. . .
    public function index(): View
    {
        $allCourses = Course::paginate(20);
        debug($allCourses);
        return view('courses.index')->with('courses', $allCourses);
    }
. . .
```

121. Open the list of courses (`http://yourdomain/courses`).



- The pagination navigation bar is not visible because the dataset has only 1 page.

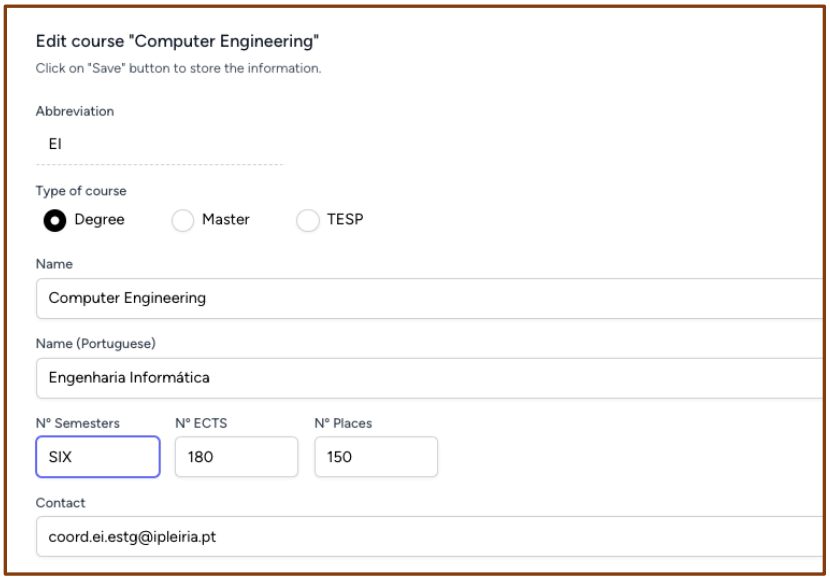
122. For more information about pagination with Laravel, check: `https://laravel.com/docs/pagination`

123. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.7.zip`").

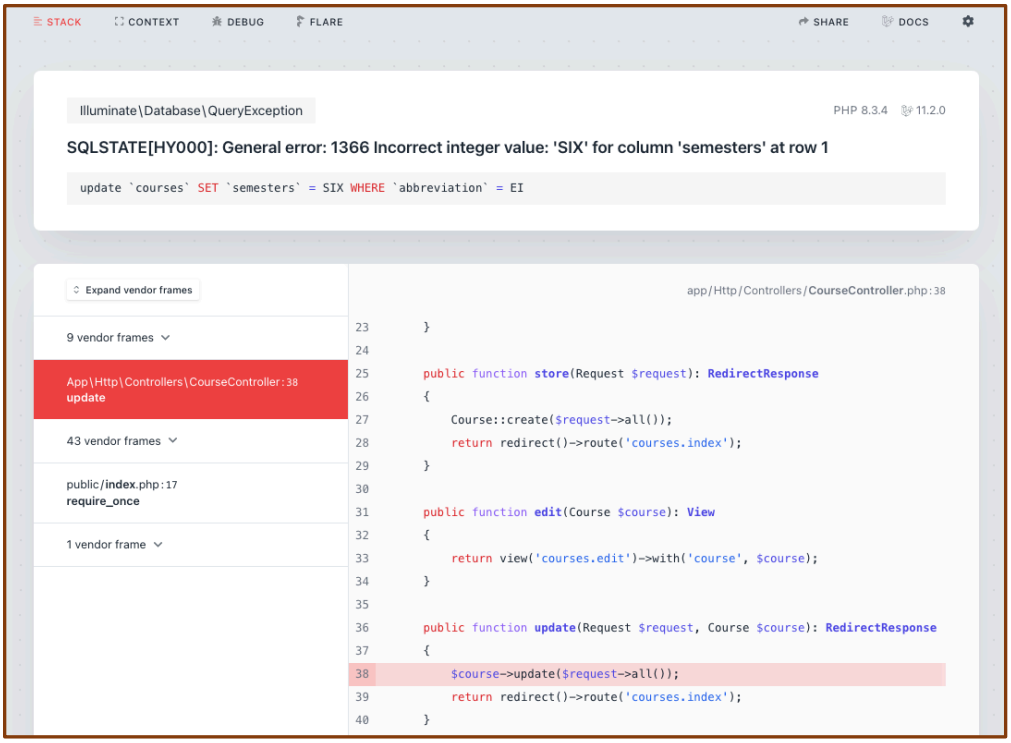
8. Validation

When creating or updating courses, the data inputted by the end-user is not validated, which is bad for the user experience and may cause some security problems. Laravel includes a validation mechanism that allows us to validate most of the values, using only **validation rules**. Check https://laravel.com/docs/validation for more information about that mechanism.

124. Open the page to edit a course (e.g: `http://yourdomain/courses/ei/edit`), and change the value of the field "`semesters`" to a non-numeric value (e.g. `six`):



125. Try to save the changes (click on button “Save”). The following error occurs:



126. The SQL error "Incorrect integer value: 'SIX' for column 'semesters'…" happens because we are trying to update a `course` with a string as the value of `semesters` column (instead of an integer).

127. We’ll start by implementing the validation of the `courses` update operation. This will ensure that the values inputted by the end-user when updating a course are valid according to the database structure and business model. Edit the method `update` of the `CourseController` – file `app/Http/Controllers/CourseController.php`:

```php
public function update(Request $request, Course $course): RedirectResponse
{
    $validated = $request->validate([
        'name' => 'required|string|max:255',
        'name_pt' => 'required|string|max:255',
        'type' => 'required|in:Degree,Master,TESP',
        'semesters' => 'required|integer|between:1,10',
        'ECTS' => 'required|integer|min:1',
        'places' => 'required|integer|min:0',
        'contact' => 'required|email',
        'objectives' => 'required|string',
        'objectives_pt' => 'required|string',
    ]);
    // Use the $validated array instead of $request->all() array
    //$course->update($request->all());
    $course->update($validated);
    return redirect()->route('courses.index');
}
```

- The argument of the method `$request->validate()` is an array with a set of **validation rules**.
- Check https://laravel.com/docs/validation#available-validation-rules for the complete list of available validation rules.
- To add custom rules, check https://laravel.com/docs/validation#custom-validation-rules validation rules.

128. Open the application and try to edit one course again and change the value of the field semesters to a non-numeric value. Save the changes. What happens now?

- The course is not updated on the database and the user is redirected to the edit page again (to the same page as it was before) – note that after the redirect, the value of the `semesters` is the same as the value on the database, because when we are redirected the page is reloaded and filled with the current values on the database (that were not changed because the operation has failed).
- This is what happens internally:
    - When the method `$request->validate()` executes, it will verify if all the values in the request (form fields) comply will all validation rules.
    - If all values are valid (all values comply will all validation rules), then the method `$request >validate()` returns an array with the valid values

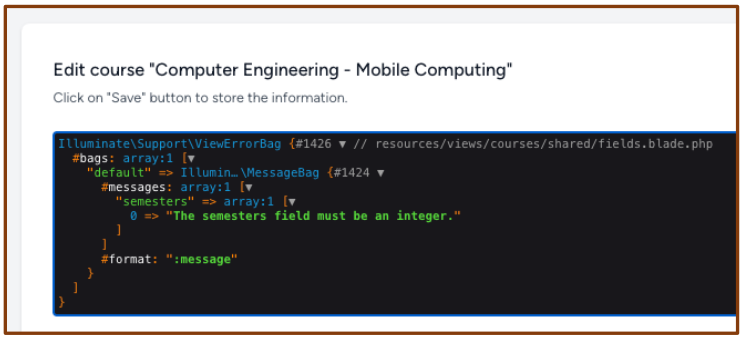
(`$validated`) and the controller continues to run as usual (it updates the corresponding table row with the `$validated` values).

- If at least one of the values does not comply with the validation rules, then the **execution is interrupted** (the update method is no longer executed) and the user is redirected to the original page (in this case, it is redirected to the form to update the course).
- When redirecting to the original page, Laravel "injects" a variable (`$errors`) with all error messages related to the "broken" validation rules (these error messages are not yet used on the view)

129. The validation is already working, but the end-user cannot understand what has just happened because there was no feedback for the user. Our next step is to modify the view so that it shows error messages for all invalid values. For now, let's just show the content of the variable **`$errors`** that is injected by the validation mechanism to the view. Add the following code to the file `resources/views/courses/shared/fields.blade.php`:

```
@php
    $mode = $mode ?? 'edit';
    $readonly = $mode == 'show';
@endphp
@dump($errors)
<x-field.input name="abbreviation" …/>
…
```

130. Open the page to edit a course (`http://yourdomain/courses/mei-cm/edit`), change the value of the field "`semesters`" to a non-numeric value (e.g. `six`) and save the data. The `@dump($errors)` will show the validation error message relative to the semesters field:



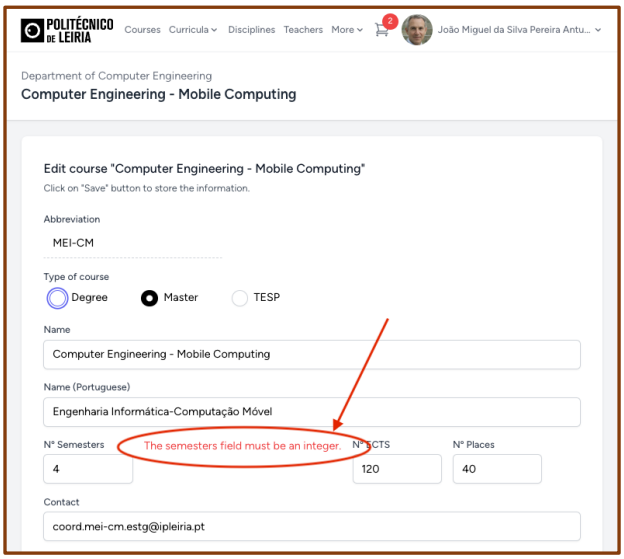
131. Instead of showing the dump of the $errors variable, we will provide validation error messages next to each field. For academic purposes, we will handle error messages for the

`semesters` field only. Remove the `@dump($errors)` and add the following code to the file `resources/views/courses/shared/fields.blade.php`:

```
@php
    $mode = $mode ?? 'edit';
    $readonly = $mode == 'show';
@endphp
<x-field.input name="abbreviation" .../>
. . .
    <x-field.input name="semesters" label="Nº Semesters" width="sm"
                   :readonly="$readonly"
                   value="{{ $course->semesters }}"/>
    @error('semesters')
        <div class="text-red-500">{{ $message }}</div>
    @enderror
. . .
```

- The blade directive `@error('FieldName') ... @enderror` defines a section that is available when the specified field (`'FieldName'`) is invalid.
- The variable `$message` inside the `@error` directive, has the error message relative to the validation rule that was broken for that specific field (*it has the first error message, because a single field can break multiple validation rules, each with its own message*)

132. Open the page to edit a course (`http://yourdomain/courses/mei-cm/edit`), change the value of the field `semesters` to a non-numeric value (e.g. `six`) and save the data. A validation error message will be visible next to the field `semesters`.



133. Although the feedback for validation (validation error messages) is working (currently, just for the field `semesters`), there is still a problem with the previous form page. When there is a validation error and the user is redirected to the form, the values of the fields are restored to the current values of the database. This is a problem because the error messages are now inconsistent with the field values. For instance, on the previous example, the `semesters` value was "*six*" when the user submitted the form. However, when the form page is reopened with the error messages, the value of `semesters` is no longer "*six*" – the value is set to the value on the database (e.g. "4").

    Also, when an error occurs, all other field values that the user has changed are lost. If the form has a lot of fields, this is very bad for the user experience (imagine filling up 20 field values, submitting the form, and because one of the values is considered invalid, all the 20 fields values inputted are lost).

134. To solve the problem described previously, we must guarantee that our form is able to **maintain the state** – that means that the values of the fields are maintained when, for some reason, the form is redirected (reopened) after the user has submitted data. This is implemented by the function **`old('FieldName', initialValue)`**.

    - When the function `old` is executed for the first time (when the form is opened for the first time), it returns the `initialValue` (the second argument of the function).
    - When the function `old` is executed after a form submit (through a redirect), it returns the latest value of the field with the specified name (first argument) – the value that the field had before submitting the form.
    - For the form to maintain state, we must replace (on all fields) this:

```
<x-field.input name="semesters" … value="{{ $course->semesters }}"/>
```

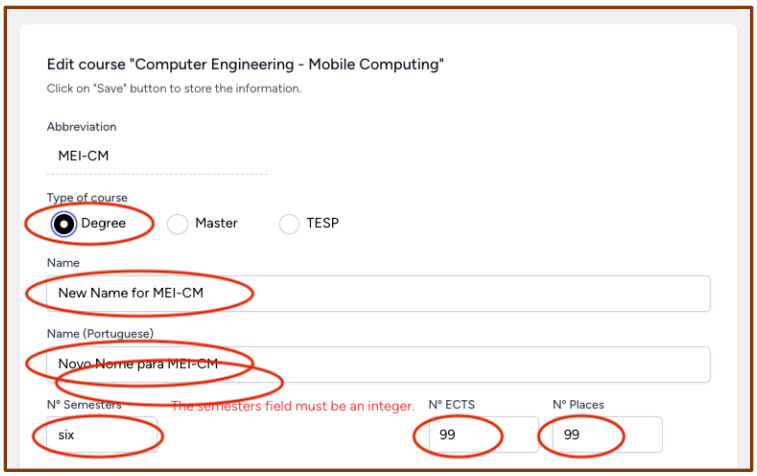
    with this:

```
<x-field.input name="semesters" … value="{{ old('semesters', $course->semesters) }}"/>
```

135. Apply the same pattern (that uses the old function) to all fields of `courses.shared.fields` view - file `resources/views/courses/shared/fields.blade.php`. Copy the full source code from the file "`05-Laravel.2.txt`" – section 135).

136. Open the page to edit a course (`http://yourdomain/courses/mei-cm/edit`), change the value of the field `semesters` to a non-numeric value (e.g. `six`) and save the data. A validation error message will be visible next to the field `semesters` and all fields

maintain the state – field values are the latest values edited by the end-user, not the values currently on the database:



137. Using the old() function to maintain the state creates a minor *glitch* on our form to edit a course: when the end-user clicks on the Cancel button, field values are reset to the initial state of the page (the values that the fields have when the page is opened), which now have the values returned by old() function - these values are different from the values on the database.

We want the Cancel button to "cancel" all changes made by the end-user and to show the values currently on the database (cancel will show the real values on the database, not the editing values).

To solve this glitch, replace the code of the Cancel button on the `courses.edit` view (`resources/views/courses/edit.blade.php`) with the following:

```
<x-button element="a" type="light" text="Cancel" class="uppercase ms-4"
          href="{{ url()->full() }}"/>
```

- `url()->full()` returns the current URL. `<x-button …>` component creates a hyperlink for the current page. When the end-user clicks on it, the browser will make an HTTP request for the page to update the course (method = `get`, url = `http://yourdomain/courses/MEI-CM/edit`) but without any session information with the old() state – it just opens the edit page as it was the first time (not a redirect).

138. The solution is complete for the field `semesters`, but none of the other fields show any validation error message. We could replicate the same pattern to all fields in the form, but instead, we're going to integrate the error message code on the field view components

(`field.input`; `field.select`; `field.check-box`; `field.radio-group` and `field.text-area`).

139. Change the view of the component `field.input` (file: `resources/views/components/field/input.blade.php`). Add `@error($name)` sections to modify the classes of the input element (add a red border to the `<input>` element) and to add an error message after the `<input>` element.

```
. . .
<div {{ $attributes->merge(['class' => "$widthClass"]) }}>
    <label ... >
    <input id="id_{{ $name }}" name="{{ $name }}" type="{{ $type }}"
           value="{{ $value }}"
        class="appearance-none block
            mt-1 w-full
            bg-white dark:bg-gray-900
            text-black dark:text-gray-50
            @error($name)
                border-red-500 dark:border-red-500
            @else
                border-gray-300 dark:border-gray-700
            @enderror
            focus:border-indigo-500 dark:focus:border-indigo-400
            focus:ring-indigo-500 dark:focus:ring-indigo-400
            rounded-md shadow-sm
            disabled:rounded-none disabled:shadow-none
            disabled:border-t-transparent disabled:border-x-transparent
            disabled:border-dashed
            disabled:opacity-100
            disabled:select-none"
            {{ $required ? 'required' : ""}}
            {{ $readonly ? 'disabled' : ""}}
            autofocus="autofocus"
            @error( $name ) is-invalid @enderror
        >
    @error( $name )
        <div class="text-sm text-red-500">
            {{ $message }}
        </div>
    @enderror
</div>
```

140. Apply the same pattern to other fields components (`field.select`; `field.check-box`; `field.radio-group` and `field.text-area`). Full code for all 5 components' view is available on `05-Laravel.2.code.txt`. Check the section `140)`.

141. Finally, remove the semesters error message from the `courses.edit` view (file `resources/views/courses/shared/fields.blade.php`):

```
@php
    $mode = $mode ?? 'edit';
    $readonly = $mode == 'show';
@endphp
<x-field.input name="abbreviation" label="Abbreviation" width="md"
               :readonly="$readonly || ($mode == 'edit')"
               value="{{ old('abbreviation', $course->abbreviation) }}"/>
<x-field.radio-group name="type" label="Type of course" :readonly="$readonly"
               value="{{ old('type', $course->type) }}"
               :options="[
                    'Degree' => 'Degree',
                    'Master' => 'Master',
                    'TESP' => 'TESP'
               ]"/>
<x-field.input name="name" label="Name" :readonly="$readonly"
               value="{{ old('name', $course->name) }}"/>
<x-field.input name="name_pt" label="Name (Portuguese)" :readonly="$readonly"
              value="{{ old('name_pt', $course->name_pt) }}"/>
<div class="flex space-x-4">
    <x-field.input name="semesters" label="Nº Semesters" width="sm"
                   :readonly="$readonly"
                   value="{{ old('semesters', $course->semesters) }}"/>
    <x-field.input name="ECTS" label="Nº ECTS" width="sm" :readonly="$readonly"
                   value="{{ old('ECTS', $course->ECTS) }}"/>
    <x-field.input name="places" label="Nº Places" width="sm" :readonly="$readonly"
                   value="{{ old('places', $course->places) }}"/>
</div>
<x-field.input name="contact" label="Contact" :readonly="$readonly"
                   value="{{ old('contact', $course->contact) }}"/>
<x-field.text-area name="objectives" label="Objectives" :readonly="$readonly"
                   value="{{ old('objectives', $course->objectives) }}"/>
<x-field.text-area name="objectives_pt" label="Objectives (Portuguese)"
                  :readonly="$readonly"
                  value="{{ old('objectives_pt', $course->objectives_pt) }}"/>
```

142. Open the page to edit a course (`http://yourdomain/courses/mei-cm/edit`) and leave some fields empty or add invalid values to them. Analyze the behavior of the page.

- The course update form is now validating all fields and maintaining the state when there are validation errors.
- All 5 field view components (`field.input`; `field.select`; `field.check-box`; `field.radio-group` and `field.text-area`) handle error messages – they are prepared to show the error messages.

143. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.8.zip`").

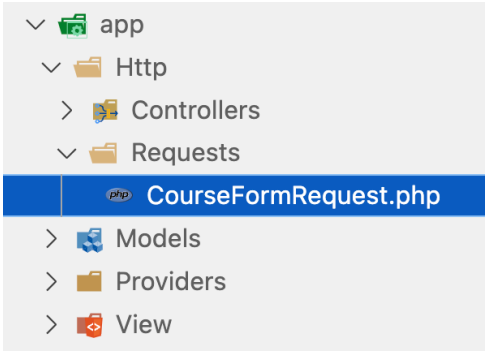
9. Form Request

The update course implements the validation mechanism and gives visual feedback for the end-user to fill up the form. However, validation rules are integrated on the action controller method, which will force us to duplicate code every time we want to implement validation rules and is not good for the separation of concerns principle. Instead of using the controller code (the method `$request->validate()` that accepts the validation rules) to define the validation rules, we will define **FormRequest** classes that centralizes all the server-side validation relative to a specific type of request. Check this content for more information about FormRequests: https://laravel.com/docs/validation#form-request-validation

144. Execute the following command:

```
php artisan make:request CourseFormRequest
```

145. This creates the file `app/Http/Requests/CourseFormRequest`



146. Change the `CourseFormRequest` code to:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
```

```php
class CourseFormRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     */
    public function rules(): array
    {
        return [
            'name' => 'required|string|max:255',
            'name_pt' => 'required|string|max:255',
            'type' => 'required|in:Degree,Master,TESP',
            'semesters' => 'required|integer|between:1,10',
            'ECTS' => 'required|integer|min:1',
            'places' => 'required|integer|min:0',
            'contact' => 'required|email',
            'objectives' => 'required|string',
            'objectives_pt' => 'required|string',
        ];
    }

    public function messages(): array
    {
        return [
            'ECTS.required' => 'ECTS is required',
            'ECTS.integer' => 'ECTS must be an integer',
            'ECTS.min' => 'ECTS must be equal or greater that 1',
        ];
    }
}
```

- Form requests are custom request classes that encapsulate their own validation and authorization logic.
- A FormRequest is a class that extends from the class Request, so we can think of it as a custom Request that encapsulate their own validation and authorization logic – it represents a HTTP Request whose data (payload/fields) must follow a custom set of validation rules (as well as a custom authorization process).

- The **authorize** method is responsible for determining if the currently authenticated user can perform the action represented by the request.
    - Please note that the authorize method returns true. If you forget to change this method (authorize), then it would always return false, meaning that the user would never be authorized to execute the associated action, therefore, the request would never be accepted.
- The **rules** method returns the validation rules that should apply to the request's data.
    - The array returned by the method "rules" is exactly the same as the validation rules array used by the `$request->validate()` of the `update` method in the `CourseController`.
- The **messages** method is an optional method that returns an array with custom messages for the specified validation rules. All validation rules have a standard message, so we only use this method (messages) if we want to change the standard message of any validation rule. Also, we only need to specify the custom messages that are different from the standard messages.

147. After defining the Form Request, it is very simple to use it on the controller. Replace previous `update` method of the `CourseController` – file `app/Http/Controllers/CourseController.php`:

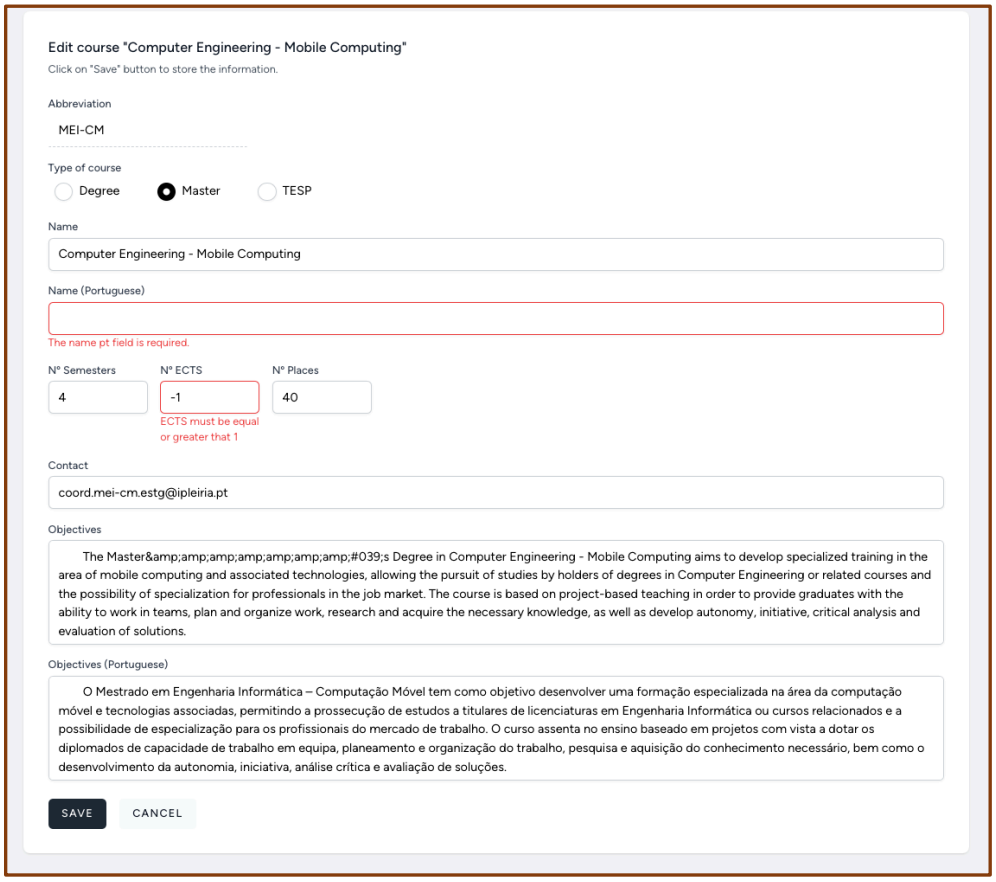
```
public function update(Request $request, Course $course): RedirectResponse
{
    $validated = $request->validate([
        'name' => 'required|string|max:255',
        'name_pt' => 'required|string|max:255',
        'type' => 'required|in:Degree,Master,TESP',
        'semesters' => 'required|integer|between:1,10',
        'ECTS' => 'required|integer|min:1',
        'places' => 'required|integer|min:0',
        'contact' => 'required|email',
        'objectives' => 'required|string',
        'objectives_pt' => 'required|string',
    ]);
    // Use the $validated array instead of $request->all() array
    //$course->update($request->all());
    $course->update($validated);
    return redirect()->route('courses.index');
}
```

with this:

```
. . .
use App\Http\Requests\CourseFormRequest;
. . .
public function update(CourseFormRequest $request, Course $course): RedirectResponse
{
    $course->update($request->validated());
    return redirect()->route('courses.index');
}
```

- Note that we have to declare an alias for the Form Request (`CourseFormRequest`) with a use statement: `use App\Http\Requests\CourseFormRequest;`
- The generic Request argument (first argument of the `update` function) is replaced by our custom Form Request (`CourseFormRequest`).
- Laravel will automatically apply the validation and authorization logic defined on the `CourseFormRequest`, before entering the controller method.
- The validation mechanism works as previously – if one or more validation rule is broken, then the execution is **interrupted**, and the user is redirected to the original page (usually the page with the form). In this case (when data is invalid), the method of the controller (example: "update" method) **is not** even **executed**.
- The method of the controller (example: "update" method) is **executed only** when all authorization and validation logic was executed successfully.
- To access the validated data, we can use the method **validated()** of the form request.

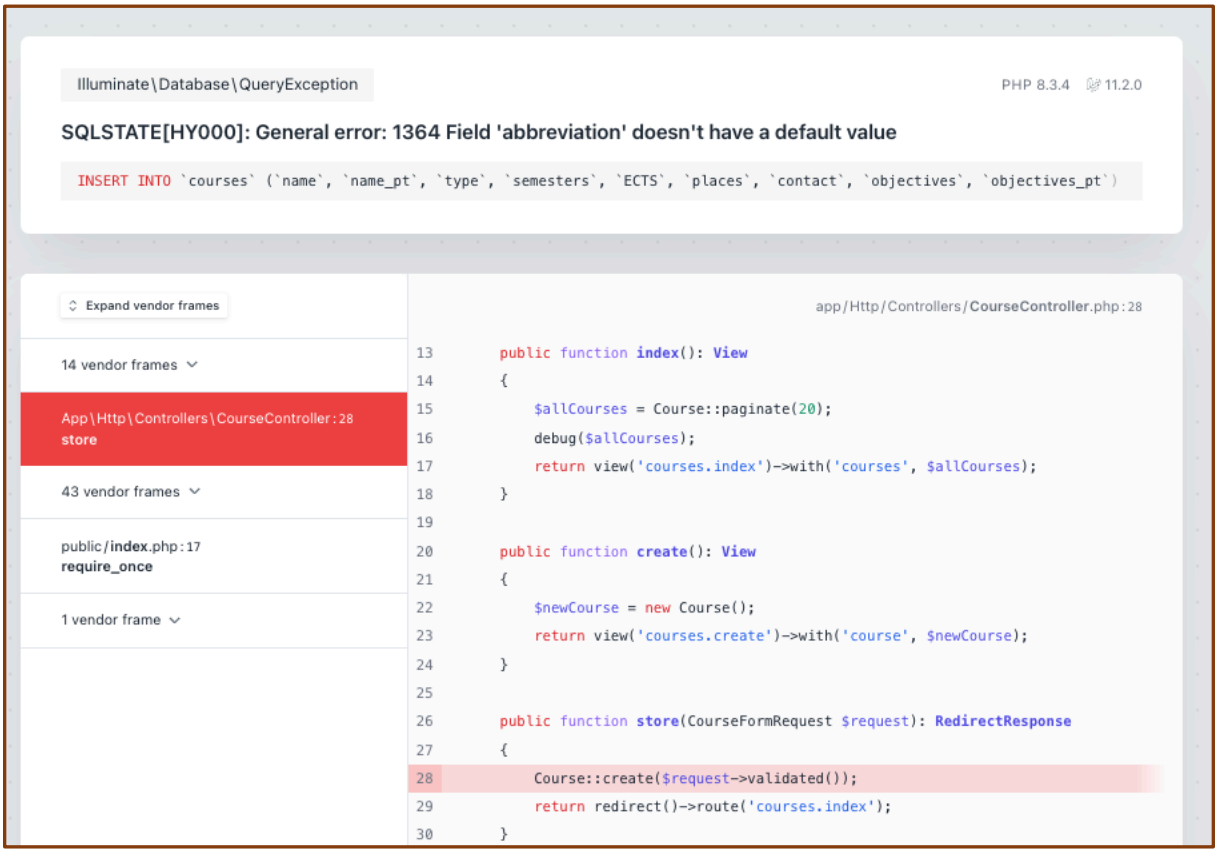
148. Test the application and try to update a course – either with or without errors. Everything should be working as before, except for the ECTS error messages that will use our custom messages defined on the Form Request.



149. After adapting the update validation to use the Form Request, we will try to reuse the same FormRequest to validate the `store` method. Change the method `store` of the `CourseController` – file app/Http/Controllers/CourseController.php:

```
. . .
public function store(CourseFormRequest $request): RedirectResponse
{
    Course::create($request->validated());
    return redirect()->route('courses.index');
}
. . .
```

150. Test the application and try to create a new course – either with or without errors. When the user clicks on the "Save new course" button to submit the form, the error "…field 'abbreviation' doesn't have a default value …" occurs:



- This error happens because the `abbreviation` field is not included in the validation rules, and therefore, the array returned by `$request->validated()` does not include the abbreviation field – the field is included on the HTTP request payload, but it is ignored by the validation process.
- The form request works perfectly when updating a course, because the abbreviation of that course is not updatable – the abbreviation does not change. However, when creating a new course, the abbreviation is fundamental – it will be the course's primary key.

151. To solve previous problem, we just have to guarantee that the Form Request used by the store method (that creates a new course) includes the abbreviation field. However, we don't want to include that field when we're updating the course. In conclusion, we have 2 sets of validation rules that are almost equal – the only difference is the presence of one extra field. To create these 2 sets of validation rules, we could:

- Create 2 independent Form Request classes (e.g. `CourseUpdateFormRequest` and `CourseStoreFormRequest`).
- Create one Form Request class for the update (e.g. `CourseUpdateFormRequest`), and another Form Request (e.g. `CourseStoreFormRequest`) for the store that extends from the first Form Request.

- Use just one Form Request (the current Form Request: `CourseFormRequest`) and adapt it to support distinct set of rules.

To minimize the number of different Form Requests and considering that both sets of validation rules are almost identical, we'll implement the third option – all validation (update and store validations) will be implemented on the same Form Request.

152. Edit the file `app/Http/Requests/CourseFormRequest.php` (Form Request class: `CourseFormRequest`) and change the **rules** method so that it returns a different array (set of validation rules) that depends of the HTTP method:

```
. . .
public function rules(): array
{
    $rules = [
        'name' => 'required|string|max:255',
        'name_pt' => 'required|string|max:255',
        'type' => 'required|in:Degree,Master,TESP',
        'semesters' => 'required|integer|between:1,10',
        'ECTS' => 'required|integer|min:1',
        'places' => 'required|integer|min:0',
        'contact' => 'required|email',
        'objectives' => 'required|string',
        'objectives_pt' => 'required|string',
    ];
    if (strtolower($this->getMethod()) == 'post') {
        // This will merge 2 arrays:
        // (adds the "abbreviation" rule to the $rules array)
        $rules = array_merge($rules, [
            'abbreviation' => 'required|string|max:20|unique:courses,abbreviation',
        ]);
    }
    return $rules;
}
. . .
```

- Previous code guarantees that when the HTTP method is post (instead of put), an extra element is added to the array of rules – that extra element includes the validation rules for the abbreviation field.
- Note that the rule: `unique:courses,abbreaviation`, ensures that the abbreviation is unique – we cannot add a course with an abbreviation that already exists.

153. Test the application and try to create and update a course – either with or without errors. Everything should be working correctly.

154. Laravel validation includes several more features, such as the ability to performing additional validation with the **after()** function or to create your own validation rules, among many other. Check https://laravel.com/docs/validation for more details about Laravel validations.

155. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.9.zip`").

10. Alert messages with flash data

The visual feedback provided by the validation error messages is fundamental to understand what happens when an entity is not stored or updated correctly. The same should happen when they are stored or updated correctly – an alert message should be shown to the end-user. This will be implemented through a technique called "flash data".

Flash data technique use sessions (check https://laravel.com/docs/session for more information about Laravel Sessions) to pass data between 2 pages. In this case, we will pass "messages" between the page with the form and the page where the user is redirected after an operation has been successfully executed. After the data has been passed on to the second page, it is removed automatically from the session – thus the name "flash".

As we are using this technique to pass messages (alert messages), sometimes it is also known as "flash messages".

156. Change the code of the methods `update` and `store` of `CourseController` (file: `app/Http/Controllers/CourseController.php` to:

```
. . .
public function store(CourseFormRequest $request): RedirectResponse
{
    $newCourse = Course::create($request->validated());
    $url = route('courses.show', ['course' => $newCourse]);
    $htmlMessage = "Course <a href='$url'><strong>{$newCourse->abbreviation}</strong>
                    - '{$newCourse->name}'</a> has been created successfully!";
    return redirect()->route('courses.index')
        ->with('alert-type', 'success')
        ->with('alert-msg', $htmlMessage);
}
```

```
public function update(CourseFormRequest $request, Course $course): RedirectResponse
{
    $course->update($request->validated());
    $url = route('courses.show', ['course' => $course]);
    $htmlMessage = "Course <a href='$url'><strong>{$course->abbreviation}</strong> -
                    '{$course->name}'</a> has been updated successfully!";
    return redirect()->route('courses.index')
        ->with('alert-type', 'success')
        ->with('alert-msg', $htmlMessage);
}
. . .
```

- When redirecting to a new page we can send flash data (session variables) using the method **with** applied to the `redirect()->route()`. Previous example sends 2 "flash" variables (`alert-msg` and `alert-type`) when redirecting to the `course.index` route.
- Note that the "`alert-msg`" is a string with an HTML section that has a message and includes a hyperlink.

157. Now we just show the flash data (session variables) on the view as an alert message. Since alert messages are used in several contexts and multiple views, we will create a view component to draw the alert messages.

158. Execute the following command:

```
php artisan make:component Alert
```

159. Change the code of the component class – file `app/View/Components/Alert.php` :

```
<?php

namespace App\View\Components;

use Closure;
use Illuminate\Contracts\View\View;
use Illuminate\View\Component;

class Alert extends Component
{
    public string $randomId = '';
```

```
    public function __construct(
        public string $type = 'dark',
        public string $message = '',
    )
    {
        $this->type = strtolower($type);
        if (!in_array($this->type, ['primary', 'secondary', 'success', 'danger',
                                    'warning', 'info', 'light', 'dark'], true)) {
            $this->type = 'dark';
        }
        $this->randomId = 'alert-' . rand(10000, 99999);
    }

    public function render(): View|Closure|string
    {
        return view('components.alert');
    }
}
```

160. Change the view of the component– file:

`resources/views/components/alert.blade.php` :

```
@php
    $colors = match($type) {
        'primary' => 'text-blue-900 dark:text-blue-500
                      bg-blue-200 dark:bg-gray-800
                      border-blue-500 dark:border-blue-800',
        'secondary' => 'text-gray-900 dark:text-gray-400
                    bg-gray-200 dark:bg-gray-800
                    border-gray-500 dark:border-gray-600',
        'success' => 'text-green-800 dark:text-green-300
                      border-green-300 dark:border-green-700
                      bg-green-50 dark:bg-gray-800',
        'danger' => 'text-red-800 dark:text-red-400
                      border-red-300 dark:border-red-700
                      bg-red-50 dark:bg-gray-800',
        'warning' => 'text-yellow-800 dark:text-yellow-500
                      border-yellow-300 dark:border-yellow-600
                      bg-yellow-50 dark:bg-gray-800',
        'info' => 'text-blue-800 dark:text-blue-400
                      bg-blue-50 dark:bg-gray-800
                      border-blue-300 dark:border-blue-900',
        'light' => 'text-gray-500 dark:text-gray-600
                bg-gray-50 dark:bg-gray-800
                border-gray-300 dark:border-gray-700',
```

```
            default => 'text-white dark:text-gray-900
                        bg-gray-800 dark:bg-gray-200
                        border-gray-950 dark:border-gray-50',
    }
@endphp

<div id="{{ $randomId }}"
    {{ $attributes->merge(['class' =>
            'flex items-center p-4 ps-8 mb-2
                text-sm font-medium
                border rounded-lg ' . $colors]) }}>
    <div>
        @if ($slot->isEmpty())
            {{ $message }}
        @else
            {{ $slot }}
        @endif
    </div>
    <button type="button" class="ms-auto -mx-1.5 -my-1.5 p-1.5 rounded-lg
                                 inline-flex items-center justify-center h-8 w-8"
onclick="document.getElementById('{{ $randomId }}').style.display = 'none'">
        <svg class="w-3 h-3" aria-hidden="true" xmlns="http://www.w3.org/2000/svg"
                  fill="none" viewBox="0 0 14 14">
            <path stroke="currentColor" stroke-linecap="round" stroke-linejoin="round"
                  stroke-width="2" d="m1 1 6 6m0 0 6 6M7 7l6-6M7 7l-6 6"/>
        </svg>
  </button>
</div>
```

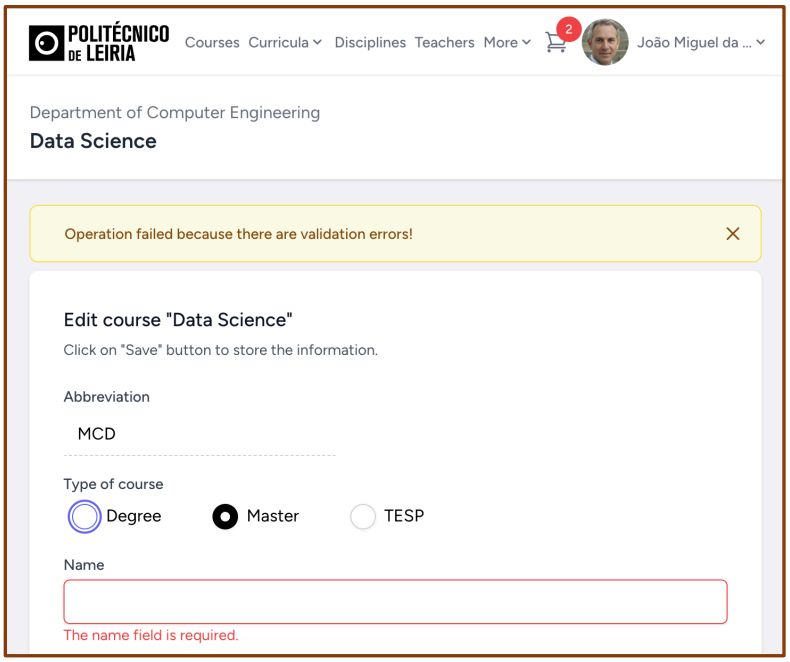
161. Next, we must include the component that we have created (alert), on our template (layout). We will include the alert component if the `alert-msg` exists and is not empty. We'll also take the opportunity to include an alert when there is any validation error. Change the view `layouts.main` – file `resources/views/layouts/main.blade.php` and add the following code (or copy the full source code from the file `05-Laravel.2-code.txt`):

```
<main>
    <div class="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
        @if (session('alert-msg'))
            <x-alert type="{{ session('alert-type') ?? 'info' }}">
                {!! session('alert-msg') !!}
            </x-alert>
        @endif
        @if (!$errors->isEmpty())
            <x-alert type="warning" message="Operation failed because there are
                                             validation errors!"/>
        @endif
```

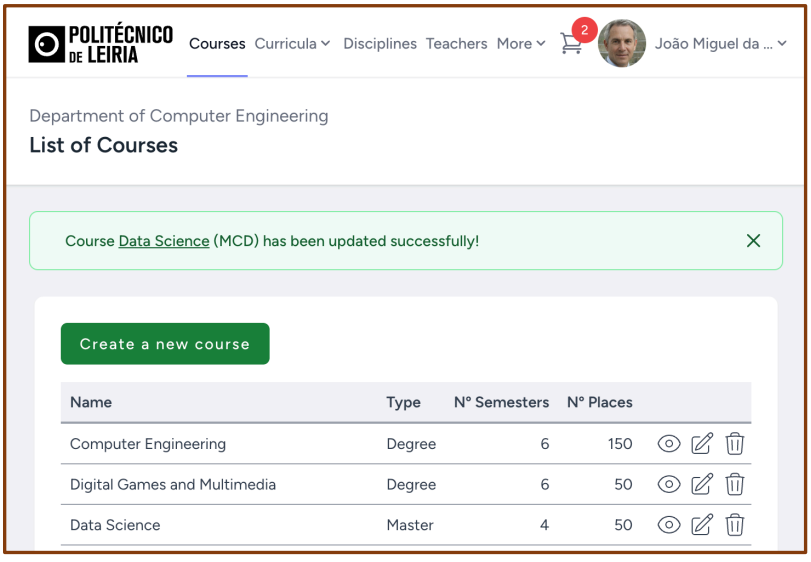
```
        @yield('main')
    </div>
</main>
```

- The interpolation defined by `{!! ... !!}` allows us to write HTML directly – it can be unsecure – use it only exceptionally and when you are sure that its content is safe.

162. Try to update a course with some validation errors. In addition to the error messages associated with the fields, a general alert will appear with the following message: " Operation failed because there are validation errors!"



163. Try to update a course successfully. The user is redirected to the list of courses with an alert message at the top of the page:



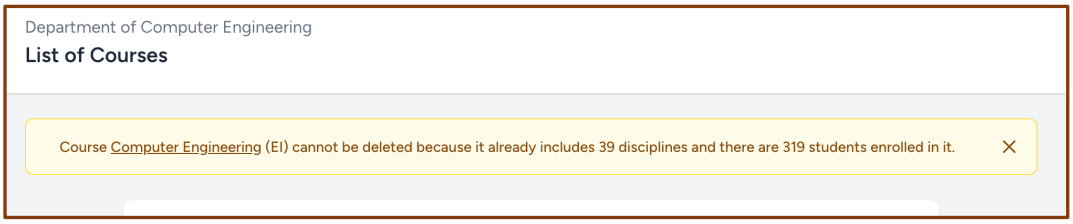
164. We have implemented a generic alert message system. From now on, every time we need to alert the end-user of something, we can reuse this mechanism. Let's use it to show a message when we try to delete a course – success message if course is deleted, or an error message if course cannot be deleted (usually due to database constraints). Change the code of the method `destroy` of `CourseController` - file: `app/Http/Controllers/CourseController.php` to:

```
. . .
use Illuminate\Support\Facades\DB;
. . .
public function destroy(Course $course): RedirectResponse
{
    try {
        $url = route('courses.show', ['course' => $course]);
        $totalStudents = DB::scalar(
            'select count(*) from students where course = ?',
            [$course->abbreviation]);
        $totalDisciplines = DB::scalar(
            'select count(*) from disciplines where course = ?',
            [$course->abbreviation]);
        if ($totalStudents == 0 && $totalDisciplines == 0) {
            $course->delete();
            $alertType = 'success';
            $alertMsg = "Course {$course->name} ({$course->abbreviation})
                         has been deleted successfully!";
        } else {
            $alertType = 'warning';
            $studentsStr = match(true) {
                $totalStudents <= 0 => "",
                $totalStudents == 1 => "there is 1 student enrolled in it",
                $totalStudents > 1 => "there are $totalStudents students enrolled in it",
            };
            $disciplinesStr = match(true) {
                $totalDisciplines <= 0 => "",
                $totalDisciplines == 1 => "it already includes 1 discipline",
                $totalDisciplines > 1 => "it already includes $totalDisciplines disciplines",
            };
            $justification = $studentsStr && $disciplinesStr
                ? "$disciplinesStr and $studentsStr"
                : "$disciplinesStr$studentsStr";
            $alertMsg = "Course <a href='$url'><u>{$course->name}</u></a>
                         ({$course->abbreviation}) cannot be deleted because $justification.";
        }
```

```
    } catch (\Exception $error) {
        $alertType = 'danger';
        $alertMsg = "It was not possible to delete the course
                        <a href='$url'><u>{$course->name}</u></a> ({$course->abbreviation})
                        because there was an error with the operation!";
    }
    return redirect()->back()
        ->with('alert-type', $alertType)
        ->with('alert-msg', $alertMsg);
}
```

- Instead of redirecting to a specific route, the controller returns to `back()` – to the same page where the delete form was submitted (the courses index).
- Note that the class **DB** is responsible for the interaction with the database without using the Eloquent models. It allows us to, among other things, send SQL command directly to the database.
- To use the DB class, we've added the use instruction:
  `use Illuminate\Support\Facades\DB;`
- `DB::select(…)` sends a Select SQL command that returns rows and columns
- `DB::scalar(…)` sends a Select SQL command that returns a single row and column - example: `select count(*) from table`
- For more information about the **DB** class:
  - https://laravel.com/docs/database
  - https://laravel.com/docs/queries

165. Try to delete a preexisting course (ex: "Computer Engineering"). The application should present an alert message similar to the following:



166. Add a new course and then delete it. The application should present an alert message similar to the following:

167. A partial resolution is available with the full project up until this exercise (file "`ai-laravel-2.partial.resolution.10.zip`").

11. Autonomous Work

Using the same design patterns that were used during the implementation of the worksheet, add all CRUD for departments, adapt disciplines CRUD and add page to view the courses with cards.

168. Create all that is necessary to implement CRUD operations, including validations and alert messages, for the entity **Departments**.

- **Validation rules** for departments:
  - `abbreviation` – required string with maximum size of 20 characters.
  - `name` – required string with maximum size of 255 characters.
  - `name_pt` – required string with maximum size of 255 characters.
- Recommendations:
  - Start by executing the shell command:

```
php artisan make:model Department --resource
```

  - This will create the Department model and a resource controller for that model that includes all routes for a typical CRUD.
  - Verify if all eloquent model conventions are correct for the departments table. If that's not the case, configure the model accordingly.
  - Use the same pattern for the signature of the controllers' methods.
  - Use "*Route-Model Binding*" when working with route parameters (the same as used for the courses route parameters)
- Don't forget to change the layout (`layouts.main`) menu so that the option "`More/Departments`" opens the page with the list of all departments.

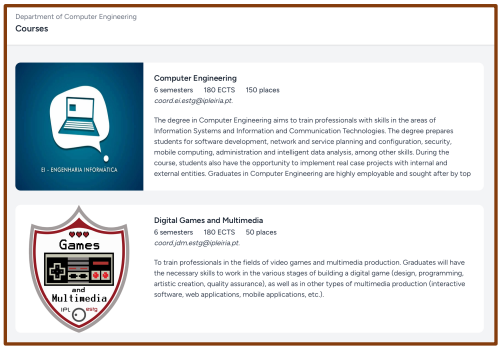
169. Create or change all that is necessary to implement CRUD operations, including validations and alert messages, for the entity **Disciplines**.

- **Validation rules** for disciplines:
  - `course` – required string with maximum size of 20 characters, and the value must exist on the table `courses` (column `abbreviation`).
  - `year` – required integer with values between 1 and 3.

- `semester` – required integer with the value 0 (annual), 1 or 2.
- `abbreviation` – required string with maximum size of 20 characters. with the value 0 (annual), 1 or 2.
- `name` – required string with maximum size of 255 characters.
- `name_pt` – required string with maximum size of 255 characters.
- `ECTS` – required integer with values between 1 and 100.
- `hours` – required integer with values between 1 and 1000.
- `optional` – required boolean.

- For selecting the list of courses of one discipline (create or update a discipline), create a `<select>` field (component `field.select`) with a dynamic list of options (obtain that list from the database).
- Create a component for the disciplines <table>, with the following properties:
  - `disciplines` – Eloquent collection with the data to show on the table (the disciplines to show)
  - `showCourse` – whether to show (value true) or hide, the course column.
  - `showView` – whether to show (value true) or hide, the view icon hyperlink column.
  - `showEdit` – whether to show (value true) or hide, the edit icon hyperlink column.
  - `showDelete` – whether to show (value true) or hide, the delete icon hyperlink column.
- Don't forget to change the layout (`layouts.main`) menu so that the option "`Disciplines`" opens the page with the list of all disciplines.

170. Create all that is necessary to implement a new page, with the url `http://yourdomain/courses/showcase` that shows the courses using cards. The visual aspect of the page should be similar to the courses page of TailwindCSS worksheet:



- Course images are located on the folder: "`storage/app/public/courses`".
- To use the images (or other types of files) inside the storage folder (Laravel File Storage), check:
    - https://laravel.com/docs/filesystem
    - https://laravel.com/docs/filesystem#the-public-disk
- When the course does not have an image, use the image: "`no_course.png`".
- Change the layout (`layouts.main`) menu so that the option "`courses`" opens the newly created page (`courses/showcase`) and the menu option "`More/Course Management`" opens the courses page (with the courses table).

Summary

Summary of features, implementations, technologies, and concepts applied during the worksheet:

```
Blade templates
      Layouts
      @yields
      @extends and @sections
      View Components
TailwindCSS package & integration
Vite
      Generating client assets with Vite
      Generating CSS with vite
      Generating JS with vite
      "npm run dev" versus "npm run build"
Tailwind template and styling
      Tables
      Forms and Inputs
      Buttons
      Styling hyperlinks as buttons
      Validation errors - input and messages
      Alert messages
```

Routes and URLs

- Generate URL from route name
- Current route with Route::currentRouteName()

Forms

- Hidden fields for CheckBox - “technique to define value for non-checked”
- View components to abstract fields
- Reusing fields to design create, update and "show" forms

Pagination

- Eloquent paginate() method
- "page" query string parameter
- Navigation bar – generated with links() method
- Configure navigation bar for Tailwind

Validation

- Validation rules
- Validation algorithm
- Maintain form state with "old" function
- FormRequest definition and usage
- Custom error messages
- Integrating validation messages on TailwindCSS templates
- $errors
- @error blade directive and $message variable

Alert messages

- Flash data
- Redirect with flash data
- Alert messages with Tailwind

DB class

- DB::select
- DB::scalar