DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

APLICAÇÕES PARA A INTERNET

Engenharia Informática

Marco Monteiro

Laravel - 4

Objectives:

(1) Use and comprehend Laravel Framework;

Note the following:

- Every page MUST comply to the HTML5 standard and it must be properly validated as such"
- The PHP code written MUST follow the "coding style guide" PSR [PSR-12];
- **Source code** referenced during these exercises is also available on a **text file**.

Setting up the environment:

- If Laragon environment is not installed and configured, follow the instructions of worksheet 2.
- Copy the zip file “08-Laravel.4-start.zip” with the provided base project, to the Laragon root folder and decompress it on that folder.
- The provided project folder should be available on “C:\<laragon_www_root\worksheet8”, for example: “C:\ainet\worksheet8” or “C:\laragon\www\worksheet8” (it depends of the Laragon root folder)
- Run laragon and start all services (in ESTG computers, before starting the services, it is necessary to **stop vmware services**).
- Execute the following command on the project folder, to rebuild the “vendor” folder:

```
composer update
```

- Execute the following two commands on the project folder, to install client packages (rebuilds the “node_modules” folder) and to compile the client assets:

```
npm install
```

```
npm run build
```

- Use previous database (worksheet5, worksheet6 or worksheet7) or create a new database (worksheet8). Configure .env file accordingly.
- If you are using a new database, or if you want to reset the database to its initial status, execute:

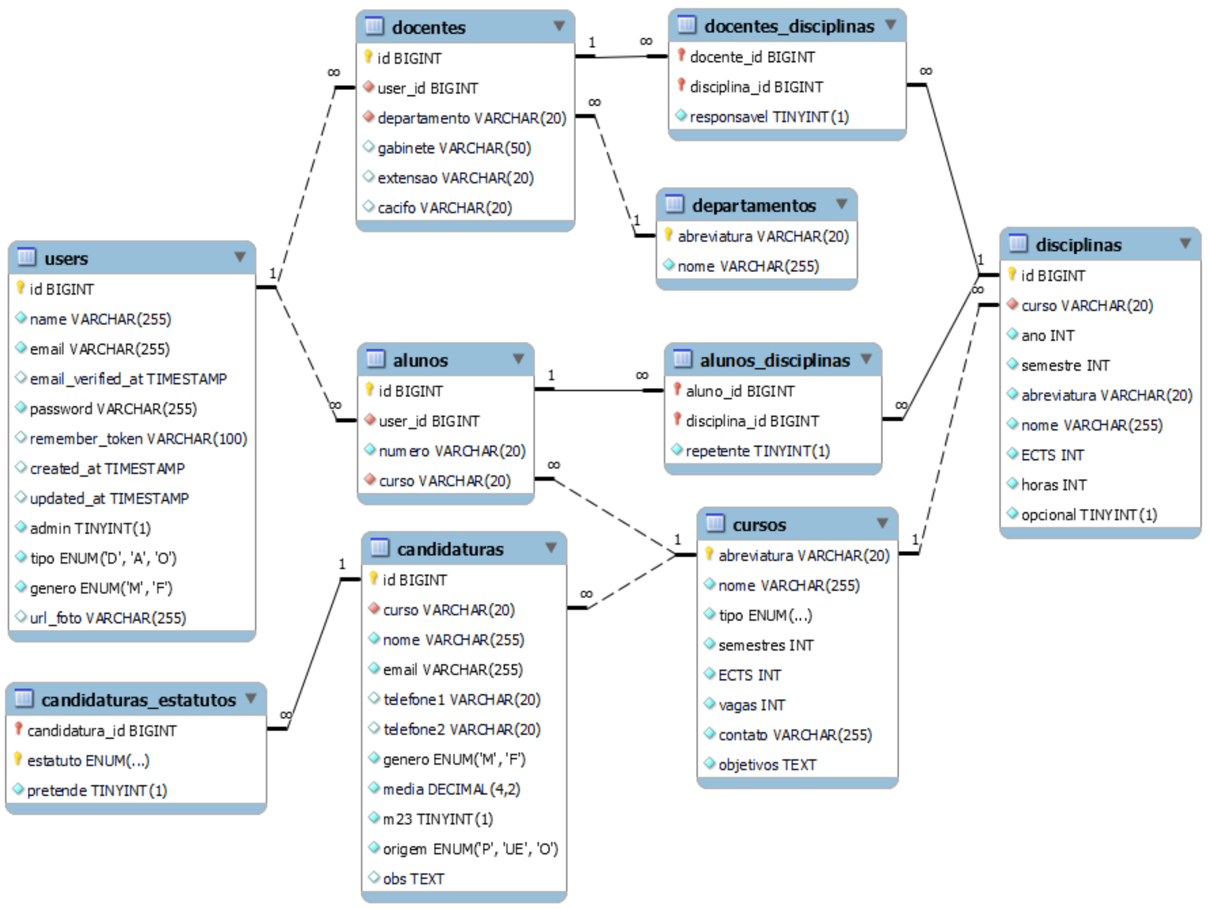
```
php artisan migrate:fresh
```

```
php artisan db:seed
```

- Use the “http://worksheet8.test” URL to access the content – test the application – operations for “cursos”, “disciplinas”, “departamentos”, “docentes”, “alunos”, “candidaturas” and “planos curriculares” should be working correctly.
- If necessary, check the “worksheet 6” and “worksheet 7” for further details on how to configure the environment and the database.

Database diagram

The structure of the database created by the migration process is the following:



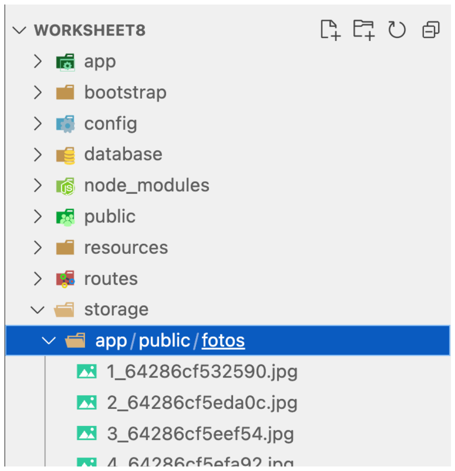
Scenery

This worksheet will continue the development of the Web Application started on worksheet 5 and continued through worksheet 6 and worksheet7. In this worksheet we will use Laravel Storage to keep the "docentes" and "alunos" photo files, and we will use file upload to keep these files synchronized with the database. We will also implement a confirmation dialog with Bootstrap (modal dialog) and a feature similar to a shopping cart using Laravel Sessions.

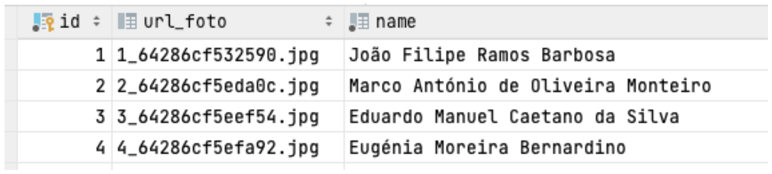
1. Storage

Currently, the photo of teachers ("docentes") and students ("alunos") shows a generic avatar image that should be used when they don't have a photo. However, most teachers and some students have a photo on the storage folder. In this section we will configure the Laravel File Storage and modify the code so that the teacher and students' photo is visible.
Laravel integrates the Flysystem PHP package which provides simple drivers for working with local filesystems, SFTP, and Amazon S3 for storing and accessing files. This package abstracts the storage, allowing us to use the same code for any type of storage and to change the type of storage just by modifying the configuration.
Check https://laravel.com/docs/filesystem for more information about Laravel file storage.

1. Using the text editor, windows explorer, or similar tool, check the content of the folder "<Your_Project_Folder>/storage/app/public/fotos". It should include random photo images. These images were copied to this folder during the database seeding process.



2. Also, open the database table "users" with HeidiSQL or any other database client, and check if the values of the column "url_foto" have a correspondent file on the "fotos" folder.



id	url_foto	name
1	1_64286cf532590.jpg	João Filipe Ramos Barbosa
2	2_64286cf5eda0c.jpg	Marco António de Oliveira Monteiro
3	3_64286cf5eef54.jpg	Eduardo Manuel Caetano da Silva
4	4_64286cf5efa92.jpg	Eugénia Moreira Bernardino

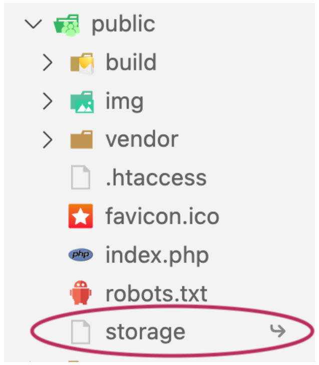
If there is no correspondence between file names and "url_foto" column values, execute the database migration and seeder again:

```
php artisan migrate:fresh
php artisan db:seed
```

3. The content of the storage folder is not public by default, because the "storage" folder is not within the "public" folder – this means that the file images are not yet available on the browser. To make the folder "<Your_Project_Folder>/storage/app/public" accessible to the public, we will create a link to that storage folder on the "public". To create the link, execute the following command on the root of the project:

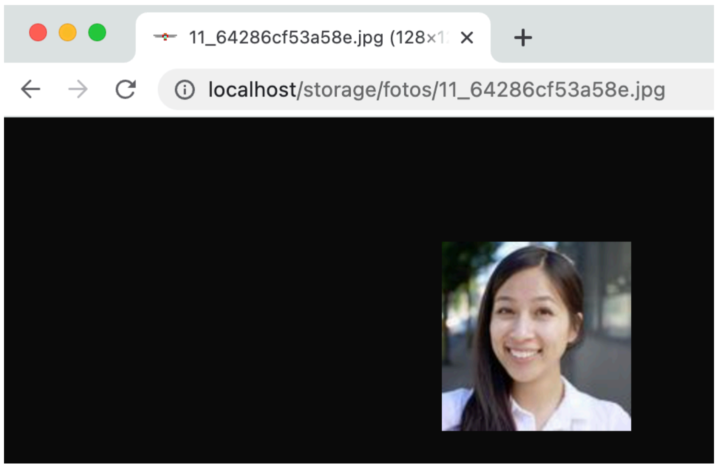
```
php artisan storage:link
```

4. Previous command creates a link on the "public" folder (with the name "storage").



```
public
  build
  img
  vendor
  .htaccess
  favicon.ico
  index.php
  robots.txt
  storage
```

5. If the "php artisan storage:link" does not work correctly, usually it's because the link already exists. If that's the case, delete the "storage" link on the "public" folder and repeat the "php artisan storage:link" command.
6. Previous link ("storage" link on the "public" folder) is associated to the folder "<Your_Project_Folder>/storage/app/public". This means that any file or folder inside the later should be available as http://worksheet8.test/**storage**/.
7. Check if storage public disk is working by copying the file name of one of the images file, and use it in the URL (http://worksheet8.test/**storage/fotos/image_name_copied.jpg**). Open previous URL on the browser – this will open the image in the browser:


11_64286cf53a58e.jpg (128×1
localhost/storage/fotos/11_64286cf53a58e.jpg

8. To show the photo of any user - teacher ("docente") or student ("aluno") - we must create the image URL by concatenation. Also, use the function **asset** to generate the full URL of the image file:

```
asset('storage/fotos/' . $url_foto_from_the_database)
```

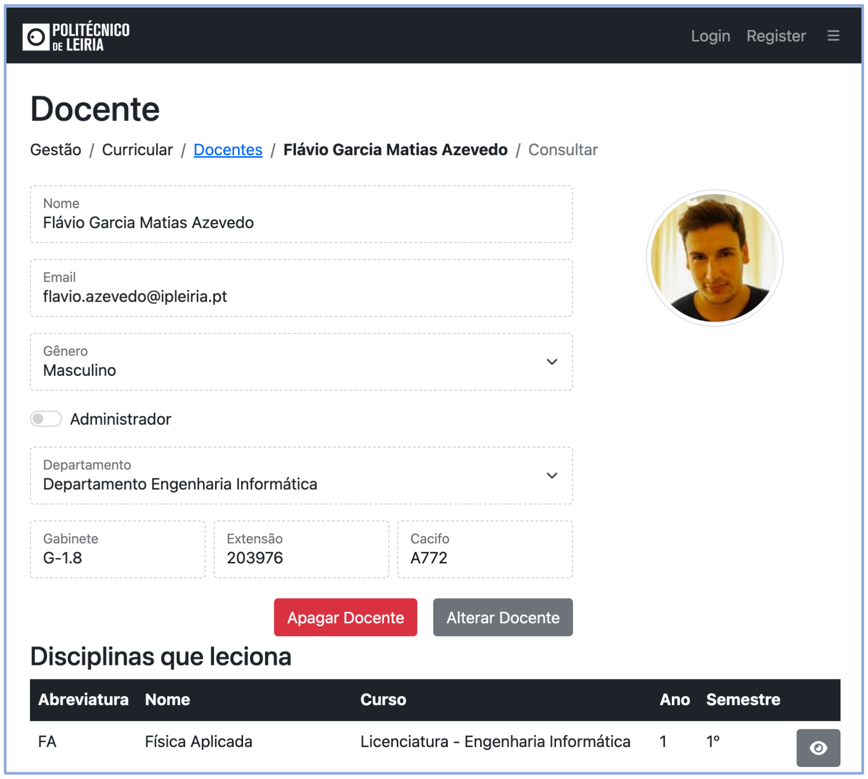
9. Open the model "User" and add the "fullPhotoUrl" accessor attribute:

```
protected function fullPhotoUrl(): Attribute
{
    return Attribute::make(
        get: function () {
            return $this->url_foto ? asset('storage/fotos/' . $this->url_foto) :
                                     asset('/img/avatar_unknown.png');
        },
    );
}
```

10. Edit the User partial view users.shared.fields_foto (file "resources/views/users/shared/.fields_foto.blade.php") so that the URL foto uses the previously create accessor:

```
<img src="{{ $user->fullPhotoUrl }}" alt="Avatar"
     class="rounded-circle img-thumbnail">
. . .
```

11. Try the application and check the edit and show pages of "docentes" and "alunos".



Docente

Gestão / Curricular / Docentes / Flávio Garcia Matias Azevedo / Consultar

Nome: Flávio Garcia Matias Azevedo

Email: flavio.azevedo@ipleiria.pt

Gênero: Masculino

Administrador

Departamento: Departamento Engenharia Informática

Gabinete: G-1.8

Extensão: 203976

Cacifo: A772

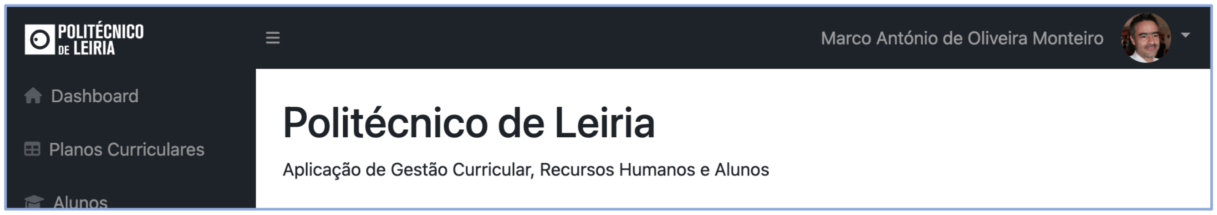
Disciplinas que leciona

Abreviatura	Nome	Curso	Ano	Semestre
FA	Física Aplicada	Licenciatura - Engenharia Informática	1	1º

12. Edit the template view (“template.layout”) – file “resources/views/template/layout.blade.php” - so that the template shows the logged in user photo (if available).

```
. . .
<a class="nav-link dropdown-toggle" id="navbarDropdown" href="#" role="button"
    data-bs-toggle="dropdown" aria-expanded="false">
    <img src="{{ Auth::user()->fullPhotoUrl }}" alt="Avatar"
         class="bg-dark rounded-circle" width="45" height="45">
</a>
. . .
```

13. Try the application and make a valid login - check the photo of the logged in user in the template:



Politécnico de Leiria

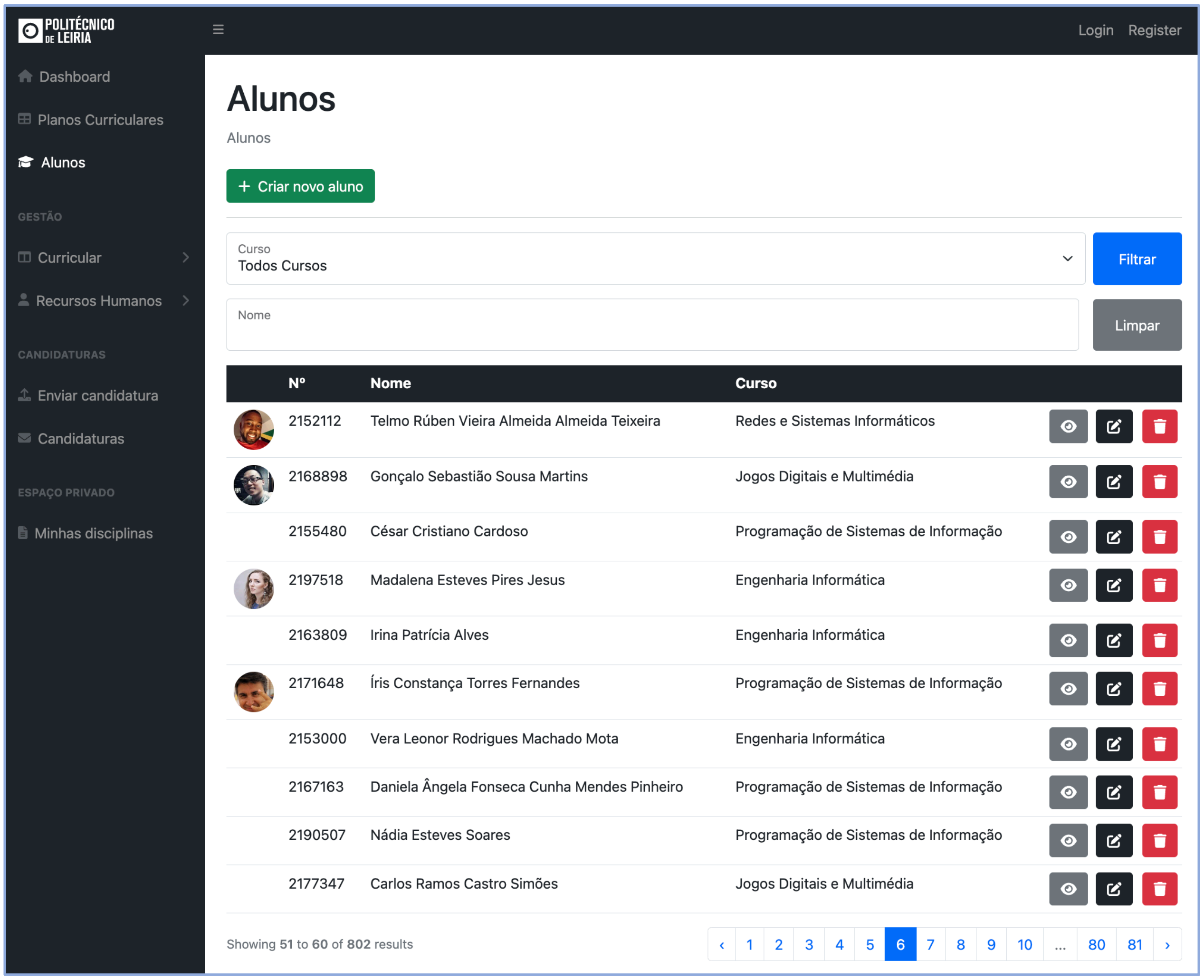
Aplicação de Gestão Curricular, Recursos Humanos e Alunos

14. Edit these 2 views (“alunos.shared.table” and “docentes.shared.table”) – files “resources/views/alunos/shared/table.blade.php” and “resources/views/docentes/shared/table.blade.php”:

```
@if ($showFoto)
    <td width="45">
        @if ($aluno->user->url_foto)
            <img src="{{ $aluno->user->fullPhotoUrl }}" alt="Avatar"
                 class="bg-dark rounded-circle" width="45" height="45">
        @endif
    </td>
@endif
```

```
@if ($showFoto)
    <td width="45">
        @if ($docente->user->url_foto)
            <img src="{{ $docente->user->fullPhotoUrl }}" alt="Avatar"
                 class="bg-dark rounded-circle" width="45" height="45">
        @endif
    </td>
@endif
```

15. Try the application and check the pages of “docentes” and “alunos”:



Alunos

Alunos

Curso: Todos Cursos

Nome:

Nº	Nome	Curso
2152112	Telmo Rúben Vieira Almeida Almeida Teixeira	Redes e Sistemas Informáticos
2168898	Gonçalo Sebastião Sousa Martins	Jogos Digitais e Multimédia
2155480	César Cristiano Cardoso	Programação de Sistemas de Informação
2197518	Madalena Esteves Pires Jesus	Engenharia Informática
2163809	Irina Patrícia Alves	Engenharia Informática
2171648	Íris Constança Torres Fernandes	Programação de Sistemas de Informação
2153000	Vera Leonor Rodrigues Machado Mota	Engenharia Informática
2167163	Daniela Ângela Fonseca Cunha Mendes Pinheiro	Programação de Sistemas de Informação
2190507	Nádia Esteves Soares	Programação de Sistemas de Informação
2177347	Carlos Ramos Castro Simões	Jogos Digitais e Multimédia

Showing 51 to 60 of 802 results

‹ 1 2 3 4 5 6 7 8 9 10 ... 80 81 ›

2. File upload

Photo images are presented throughout the application. To edit these images, we must support file upload on the “edit” and “create” forms of both “docentes” and “alunos”.

16. Because we have used partial views, all these 4 forms (edit and create docentes and alunos) use the same code file to define the field to upload. Check the view “users.shared.fields_foto” (file “resources/views/users/shared/fields_foto.blade.php”). Check the code for the field to upload the photo file and the button to delete the photo:

```
<img src="{{ $user->fullPhotoUrl }}" . . .>
@if ($allowUpload)
    <div class="mb-3 pt-3">
        <input type="file" class="form-control id="inputFileFoto"
               @error('file_foto') is-invalid @enderror" name="file_foto">
        @error('file_foto')
            <div class="invalid-feedback">
                {{ $message }}
            </div>
        @enderror
    </div>
@endif
@if ($formToDelete ?? false)
    <button type="submit" class="btn btn-danger" name="deletefoto"
            form="{{ $formToDelete }}">
        Apagar Foto
    </button>
@endif
```

- The name of the field to upload the photo file is **file_foto**.
- The button to delete a photo will submit the form $formToDelete. This is defined by the view that uses this partial view. Check the view “docentes.edit”:

```
. . .
                @include('users.shared.fields_foto', [
                    'user' => $docente->user,
                    'allowUpload' => true,
                    'formToDelete' => 'form_delete_photo',
                ])
. . .
    <form id="form_delete_photo" action="#" method="POST" class="d-none">
        @csrf
        @method('DELETE')
    </form>
. . .
```

- For instance, on the edit “docentes” page the button to delete the photo will submit the form “form_delete_photo”.

17. We already have the field to upload the file (<input type="file" …), but we have to add the attribute **enctype** with value = "**multipart/form**" to the form, so that it supports file uploading. On the views "docentes.edit", "docentes.create", "alunos.edit" and "alunos.create" add the attribute enctype. Change the files:

- "resources/views/docentes/edit.blade.php"
- "resources/views/docentes/create.blade.php"
- "resources/views/alunos/edit.blade.php"
- "resources/views/alunos/create.blade.php"

```
. . .
    <form . . . method="POST" . . . enctype="multipart/form-data">
. . .
```

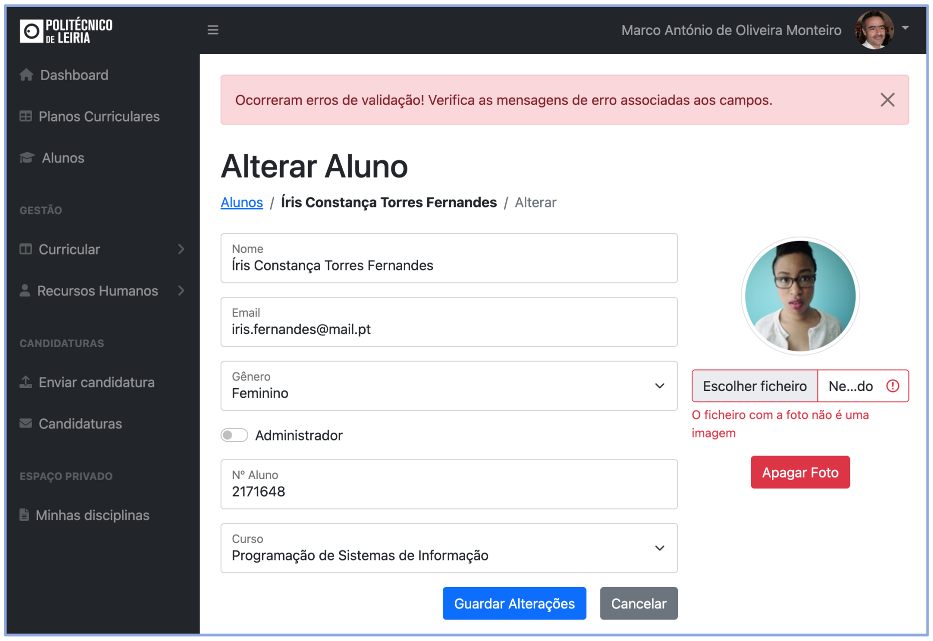
18. The field "file_foto" is included when submitting any of these 4 forms, but not considered on the form validation. Add the following validation rule to the DocenteRequest and AlunoRequest. Change the files:

- "Http/Requests/DocenteRequest.php"
- "Http/Requests/AlunoRequest.php"

```
. . .
public function rules(): array
. . .
'file_foto' =>          'sometimes|image|max:4096', // maxsize = 4Mb
. . .
public function messages(): array
. . .
  'file_foto.image' => 'O ficheiro com a foto não é uma imagem',
  'file_foto.size' => 'O tamanho do ficheiro com a foto tem que ser inferior a 4 Mb',
. . .
```

- Validation rule for the field "file_foto" dictates that this field is not always required – sometimes it is present, sometimes it is not present. The file to upload is only required when we want to upload a photo for the first time or when we want to change the user's photo. When we create or update a user without changing the photo, the field should not be present (we don't need to upload a photo every time we change something)
- When the file is uploaded, then it should be an image (jpg, jpeg, png, bmp, gif, svg, or webp) file with the size smaller than 4096 Kb (4Mb).

19. Open the application on the edit "aluno" page and try to upload a PDF file, or an image larger that 4Mb. The following should happen:


POLITÉCNICO DE LEIRIA

Marco António de Oliveira Monteiro

Ocorreram erros de validação! Verifica as mensagens de erro associadas aos campos.

Alterar Aluno

Alunos / Íris Constança Torres Fernandes / Alterar

Nome
Íris Constança Torres Fernandes

Email
iris.fernandes@mail.pt

Gênero
Feminino

Administrador

Nº Aluno
2171648

Curso
Programação de Sistemas de Informação

Escolher ficheiro | Ne...do

O ficheiro com a foto não é uma imagem

20. We need to handle the incoming file on the "AlunoController" and "DocenteController". Edit the method "update" of the file: "app/Http/Controllers/AlunoController.php":

```
    use Illuminate\Support\Facades\Storage;
    . . .
    public function update(AlunoRequest $request, Aluno $aluno): RedirectResponse
    {
        $formData = $request->validated();
        $aluno = DB::transaction(function () use ($formData, $aluno, $request) {
            $aluno->curso = $formData['curso'];
            $aluno->numero = $formData['numero'];
            $aluno->save();
            $user = $aluno->user;
            $user->tipo = 'A';
            $user->name = $formData['name'];
            $user->email = $formData['email'];
            $user->admin = $formData['admin'];
            $user->genero = $formData['genero'];
            $user->save();
            if ($request->hasFile('file_foto')) {
                if ($user->url_foto) {
                    Storage::delete('public/fotos/' . $user->url_foto);
                }
                $path = $request->file_foto->store('public/fotos');
                $user->url_foto = basename($path);
                $user->save();
            }
            return $aluno;
        });
. . .
```

- Note that the field name on the form (file_foto) is completely different from the database field name (url_foto) – they refer to 2 completely different things. The form field is the file binary that was uploaded to the server, and the url_foto on the database refers to the relative URL name used to access the photo (in this case the relative name is the same as the file name on the storage/fotos folder.
- The code to handle the file upload is only executed when the image file was uploaded. If no image file is present on the request, then nothing (related to file upload) is executed:

```
if ($request->hasFile('file_foto')) {
    . . .
```

- If the "aluno" had a previous photo file, then the file is removed from the storage:

```
if ($user->url_foto) {
    Storage::delete('public/fotos/' . $user->url_foto);
}
```

- The uploaded file (the new file) is saved on the storage (folder "public/foto") with a random name. $path will have the full random name of the file:

```
. . .
$path = $request->file_foto->store('public/fotos');
```

- "basename" function extracts the file name only, from the full name of the file. The file name is saved on the "url_foto" column of the "users" table.

```
$user->url_foto = basename($path);
$user->save();
```

21. Apply the same pattern to the "store" method of the file: "app/Http/Controllers/AlunoController.php" and change the code of the "destroy" method to ensure that the photo file is deleted when the aluno is deleted from the database.

```
. . .
public function store(AlunoRequest $request): RedirectResponse
{
    $formData = $request->validated();
    $aluno = DB::transaction(function () use ($formData, $request) {
        $newUser = new User();
        . . .
        if ($request->hasFile('file_foto')) {
            $path = $request->file_foto->store('public/fotos');
            $newUser->url_foto = basename($path);
            $newUser->save();
        }
        return $newAluno;
    });
. . .
```

```
    public function destroy(Aluno $aluno): RedirectResponse
    {
        try {
            $totalDisciplinas = DB::scalar('select count(*) from alunos_disciplinas
where aluno_id = ?', [$aluno->id]);
            $user = $aluno->user;
            if ($totalDisciplinas == 0) {
                DB::transaction(function () use ($aluno, $user) {
                    $aluno->delete();
                    $user->delete();
                });
                if ($user->url_foto) {
                    Storage::delete('public/fotos/' . $user->url_foto);
                }
            . . .
```

22. Apply the same pattern to the "update", "store" and "destroy" methods of the DocenteController – file "app/Http/Controllers/DocenteController.php". Code also available on the file "08-Laravel.4-Code.txt"
23. One final detail is necessary to fully implement our "upload" file feature – the button to delete the photo. Let's start to specify 2 routes to delete the photos. Change the file "routes/web.php":

```
. . .
Route::resource('docentes', DocenteController::class);
Route::delete('docentes/{docente}/foto', [DocenteController::class, 'destroy_foto'])
       ->name('docentes.foto.destroy');
. . .
Route::resource('alunos', AlunoController::class);
Route::delete('alunos/{aluno}/foto', [AlunoController::class, 'destroy_foto'])
       ->name('alunos.foto.destroy');
. . .
```

24. There are 2 views that include the form to delete a photo. Change the "form to delete" code on the "docentes.edit" view – file "resources/views/docentes/edit.blade.php"

```
<form id="form_delete_photo"
      action="{{ route('docentes.foto.destroy', ['docente' => $docente]) }}"
      method="POST" class="d-none">
        @csrf
        @method('DELETE')
</form>
```

25. Apply the same pattern to change the "form to delete" code on the "alunos.edit" view – file "resources/views/alunos/edit.blade.php"

```
<form id="form_delete_photo"
      action="{{ route('alunos.foto.destroy', ['aluno' => $aluno]) }}"
      method="POST" class="d-none">
        @csrf
        @method('DELETE')
</form>
```

26. Add the “destroy_foto” method to the DocenteController – file “app/Http/Controllers/DocenteController.php”.

```
public function destroy_foto(Docente $docente): RedirectResponse
{
    if ($docente->user->url_foto) {
        Storage::delete('public/fotos/' . $docente->user->url_foto);
        $docente->user->url_foto = null;
        $docente->user->save();
    }
    return redirect()->route('docentes.edit', ['docente' => $docente])
        ->with('alert-msg', 'Foto do docente "' . $docente->user->name .
               '" foi removida!')
        ->with('alert-type', 'success');
}
```

27. Add the “destroy_foto” method to the AlunoController – file “app/Http/Controllers/AlunoController.php”.

```
public function destroy_foto(Aluno $aluno): RedirectResponse
{
    if ($aluno->user->url_foto) {
        Storage::delete('public/fotos/' . $aluno->user->url_foto);
        $aluno->user->url_foto = null;
        $aluno->user->save();
    }
    return redirect()->route('alunos.edit', ['aluno' => $aluno])
        ->with('alert-msg', 'Foto do aluno "' . $aluno->user->name .
               '" foi removida!')
        ->with('alert-type', 'success');
}
```

28. Try the application and verify if the “docentes” and “alunos” upload feature is running accordingly to the expected.

3. Confirmation modal dialog - bootstrap

When removing a “docente” or “user” photo, the changes that were made to the other fields of the form are lost forever – the button to delete a photo “calls” its own independent route from an independent form – the data of the other form (with all other fields) is not passed on to the server.

We could change the view and controller to maintain the form state, by using the same form to both update the “docente” or “aluno” and to delete the photo. This would be a cumbersome and complex solution.

Instead, we will create a “confirmation dialog” that informs the end user what is about to happen, giving him the chance to abort the “delete” photo operation.

29. Add a new partial view with the confirmation dialog (implemented with “**Bootstrap Modal**”). Check https://getbootstrap.com/docs/5.2/components/modal/ for more information about bootstrap modals. Create the view “shared.confirmationDialog” – file “resources/views/shared/confirmationDialog.blade.php”, with the following code (also available on 08-Laravel.4-code.txt):

```
<!-- Confirmation Modal-->
<div class="modal fade" id="confirmationModal" tabindex="-1" role="dialog"
     aria-labelledby="confirmationModalTitleLabel" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="confirmationModalTitleLabel">{{ $title ?? 'Confirmation' }}
        </h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
                aria-label="Close"></button>
      </div>
      <div class="modal-body">
       <p id="confirmationModalMsgLine1">{!! $msgLine1 ?? 'Do you confirm the operation?' !!}</p>
       <p id="confirmationModalMsgLine2">{!! $msgLine2 ?? '' !!}</p>
      </div>
      <div class="modal-footer">
        <button class="btn btn-secondary" type="button" data-bs-dismiss="modal">Cancelar</button>
        <a class="btn btn-primary" id="confirmationModalButton"
           onclick="event.preventDefault();
                    document.getElementById('confirmationModalForm').submit();">
           {{ $confirmationButton ?? 'OK' }}</a>
        <form id="confirmationModalForm" action="{{ $formAction ?? '#' }}" method="POST"
              style="display: none;">
          @csrf
          <input type="hidden" name="_method" value="{{ $formMethod ?? 'POST' }}"
                 id="confirmationModalFormMethod"></form>
      </div>
    </div>
  </div>
</div>
```

30. Modify the view “users.shared.fields_foto” – file “resources/views/users/shared/fields_foto.blade.php”

```
<img src="{{ $user->fullPhotoUrl }}" alt="Avatar"
     class="rounded-circle img-thumbnail">
@if ($allowUpload)
    <div class="mb-3 pt-3">
        <input type="file" id="inputFileFoto"
class="form-control @error('file_foto') is-invalid @enderror" name="file_foto">
        @error('file_foto')
            <div class="invalid-feedback">
                {{ $message }}
            </div>
        @enderror
    </div>
@endif
@if (($allowDelete ?? false) && $user->url_foto)
    @if ($user->docente)
        <button type="button" class="btn btn-danger" data-bs-toggle="modal"
     data-bs-target="#confirmationModal"
     data-action="{{ route('docentes.foto.destroy', ['docente' => $user->docente]) }}"
     data-msgLine2="Quer realmente apagar a fotografia do docente <strong>{{
                   $user->name }}</strong>?">
            Apagar Foto
        </button>
    @elseif ($user->aluno)
        <button type="button" class="btn btn-danger" data-bs-toggle="modal"
     data-bs-target="#confirmationModal"
     data-action="{{ route('alunos.foto.destroy', ['aluno' => $user->aluno]) }}"
     data-msgLine2="Quer realmente apagar a fotografia do aluno <strong>{{
                   $user->name }}</strong>?">
            Apagar Foto
        </button>
    @endif
@endif
```

- In the new version of the code, when the user clicks on the button “Apagar Foto”, instead of submitting a form the application will show the confirmation dialog (modal) with the id “confirmationModal”. This modal dialog was defined on the view “shared.confirmationDialog” – check the code of that view and find the element with the id = “confirmationModal”.
- It’s the code in the modal view (“shared.confirmationDialog”) that will submit the form when the end user clicks on the button that confirms the operation
- The @if expression was also changed to show the “Apagar Foto” button only when the $allowDelete is true and the user has a photo – if the user does not have a photo, the button will not be created.

- The form action will be defined by the custom attribute:
  - **data-action** – the action of the form

31. Create a new javascript file that will have the required code to redefine the form action as well as other confirmation dialog properties. Create the file “modal.js” on the “resources/js” folder (resources/js/modal.js). Add the following code to that file:

```
const cModal = document.getElementById('confirmationModal')
const cForm = document.getElementById('confirmationModalForm')
cModal.addEventListener('show.bs.modal', event => {
    let action = (event.relatedTarget.getAttribute("data-action") ?? "").trim()
    let method = (event.relatedTarget.getAttribute("data-formMethod") ?? "").trim()
    let title = (event.relatedTarget.getAttribute("data-title") ?? "").trim()
    let msgLine1 = (event.relatedTarget.getAttribute("data-msgLine1") ?? "").trim()
    let msgLine2 = (event.relatedTarget.getAttribute("data-msgLine2") ?? "").trim()
    let buttonText = (event.relatedTarget.getAttribute("data-confirmationButton") ??
"").trim()
    if (action) {
        cForm.action = action
    }
    if (method) {
        cForm._method.value = method
    }
    if (title) {
        document.getElementById('confirmationModalTitleLabel').textContent = title
    }
    if (msgLine1) {
        document.getElementById('confirmationModalMsgLine1').innerHTML = msgLine1
    }
    if (msgLine2) {
        document.getElementById('confirmationModalMsgLine2').innerHTML = msgLine2
    }
    if (buttonText) {
        document.getElementById('confirmationModalButton').textContent = buttonText
    }
})
```

32. Change the file “app.js” to import previous file (file: “resources/js/app.js”):

```
import './bootstrap';

import './sb-admin'

import './modal'
```

33. “Compile” javascript code by executing the following command:

```
npm run build
```

34. Change the view “docentes.edit” (file: “resources/views/docentes/edit.blade.php”) to:

```
@extends('template.layout')
. . .
  <div class="ps-2 mt-5 mt-md-1 d-flex mx-auto flex-column align-items-center
            justify-content-between" style="min-width:260px; max-width:260px;">
     @include('users.shared.fields_foto', [
              'user' => $docente->user,
              'allowUpload' => true,
              'allowDelete' => true,
             ])
  </div>
</div>
</form>
   @include('shared.confirmationDialog', [
       'title' => 'Quer realmente apagar a foto?',
       'msgLine1' => 'As alterações efetuadas aos dados do docente vão ser perdidas!',
       'msgLine2' => 'Clique no botão "Apagar" para confirmar a operação.',
       'confirmationButton' => 'Apagar fotografia',
       'formAction' => route('docentes.foto.destroy',
                             ['docente' => $docente->user->docente]),
       'formMethod' => 'DELETE',
    ])
@endsection
```

- Instead of passing the variable “formToDelete” to the partial view “users.shared.fields_foto”, we pass the variable “allowDelete” (value = true)
- We have removed the form to delete the photo (form_delete_photo) – it is now “inside” the modal.
- We include the partial view “shared.confirmationDialog” that draws the modal dialog (by default it is hidden) and pass to that partial view: the title (“**$title**”); messages (“**$msgLine1**” and ”**$msgLine2**”); button text (“**$confirmationButton**”), the form action (“**$formAction**”) and the form method (“**$formMethod**”). These values (passed on the partial view) will be the **default properties** for the confirmation modal. We can change the default properties by applying a custom attribute (format name for the attribute: “data-propertyName”) on the button that will invoke the confirmation dialog.
    - Open the “resources/views/users/shared/fields_foto.blade.php” file and check the value of:
        - **data-action** – this value will replace the $action defined in the “shared.confirmationDialog” partial view.
        - **data-msgLine2** – this value will replace the value of $msgLine2 defined in the “shared.confirmationDialog” partial view.

- Check the views “shared.confirmationDialog” and “users/shared/fields_foto” and the javascript file “resources/js/modal.js”. Try to understand how these values are used and passed on to the modal confirmation dialog.

35. Change the view “alunos.edit” (file: “resources/views/alunos/edit.blade.php”) by applying the same pattern (only changes the values that are passed on to the partial views):

```
@extends('template.layout')
. . .
     @include('users.shared.fields_foto', [
              'user' => $aluno->user,
              'allowUpload' => true,
              'allowDelete' => true,
             ])
  </div>
</div>
</form>
   @include('shared.confirmationDialog', [
        'title' => 'Quer realmente apagar a foto?',
        'msgLine1' => 'As alterações efetuadas ao dados do aluno vão ser perdidas!',
        'msgLine2' => 'Clique no botão "Apagar" para confirmar a operação.',
        'confirmationButton' => 'Apagar fotografia',
        'formMethod' => 'DELETE',
    ])
@endsection
```

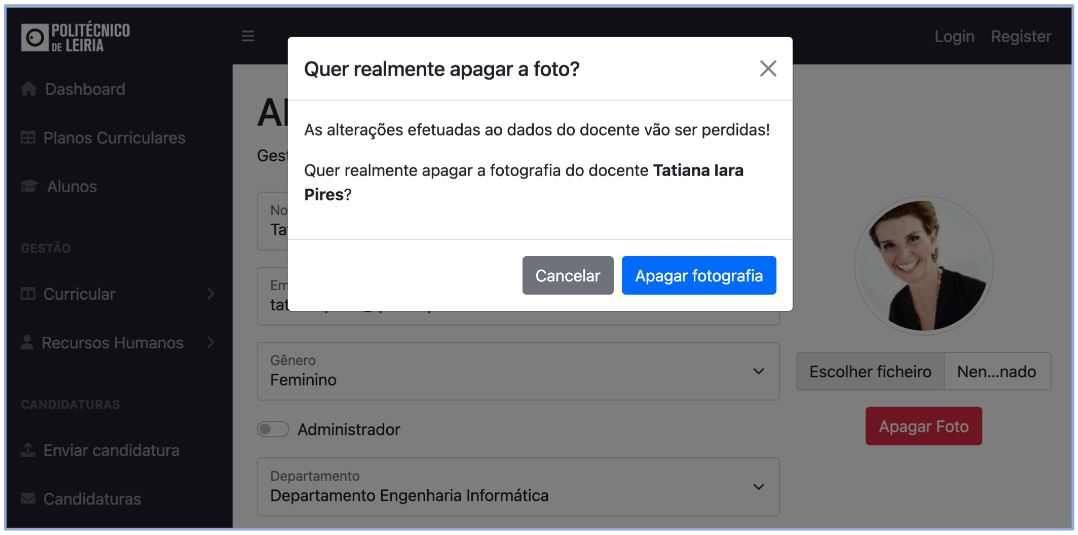
- In this view we do not pass the value of $formAction to the partial view, because that value is already filled by the custom attribute “data-action” of the button

36. All views that use (include) the partial view “users.shared.fields_foto” should be adjusted to pass the variable $allowDelete to the partial view. On these files:
    - “resources/views/alunos/show.blade.php”
    - “resources/views/alunos/create.blade.php”
    - “resources/views/docentes/show.blade.php”
    - “resources/views/docentes/create.blade.php”

    adjust the code that imports the partial view “users.shared.fields_foto” to:

```
    @include('users.shared.fields_foto', [
        'user' => . . . ,
        'allowUpload' => . . . ,
        'allowDelete' => false,
    ])
```

37. Open the application and try to remove a photo (“Apagar Foto”). The modal confirmation dialog should popup and the photo is only removed when the user confirms the operation.


Quer realmente apagar a foto?

As alterações efetuadas ao dados do docente vão ser perdidas!

Quer realmente apagar a fotografia do docente **Tatiana Iara Pires**?

4. Sessions and Shopping Carts

Most web applications technologies support the concept of **sessions**, that allows us to maintain data (usually in memory) associated to one session. The session is created (on the server) when the user accesses one of the pages of the web application for the first time, it is maintained (on the server) while the user is accessing pages of the web application and is terminated either explicitly or after a timeout – if the user does not access any page of the application for a predetermined amount of time.

When the session is created, the server creates a session ID and sends it in a cookie to the client (browser). On subsequent HTTP requests, the client (browser) send the session ID cookie to the server, which allows the server to identify the session for that specific request. This allows all requests from the same client (same browser instance) to be recognized as belonging to a specific session.

For each individual session, the web application server will maintain data (a set of session variables) that is specific for that session and is available throughout all pages of the web application – we can think of session variables like "global" variables for that specific session, in a sense that they are accessible on all pages and persist between multiple HTTP requests.

Laravel uses sessions to maintain the information about the authenticated user - Auth::user() -, to maintain the form state - function old(…) – or to implement the "flash data" mechanism. We can also add our own session variables. Check https://laravel.com/docs/session for more details about Laravel sessions.

One of the features that usually use sessions is the "**shopping cart**" – essential to online e-commerce applications. The shopping cart includes a set (array/collection) of products to buy, and it must be maintained through all pages of the web application – that makes it a perfect match for a session variable.

In this section, we will use Laravel sessions to create a feature similar to a “shopping cart”. Our project does not require a shopping cart, because it is not an e-commerce, and the database does not have products or orders. Instead, we will allow students to register “disciplinas” using a “shopping cart”. Students (“alunos”) will browse through the application and add “disciplinas” to the cart (as we would add products to a shopping cart). The cart will have multiple “disciplinas”, and then the student can confirm all “disciplinas” registration at once (as we would confirm the order of the products in a shopping cart).

38. First, let’s create the CartController. Execute the command:

```
php artisan make:controller CartController
```

39. Next, create 5 routes to:

- Add a “disciplina” to the cart.
- Remove a “disciplina” from the cart.
- View the cart.
- Confirm the cart (store the registrations on the database).
- Clear the cart.

Add the following routes to the “routes/web.php” file:

```
. . .
use App\Http\Controllers\CartController;
. . .
// Add a "disciplina" to the cart:
Route::post('cart/{disciplina}', [CartController::class, 'addToCart'])
       ->name('cart.add');
// Remove a "disciplina" from the cart:
Route::delete('cart/{disciplina}', [CartController::class, 'removeFromCart'])
       ->name('cart.remove');
// Show the cart:
Route::get('cart', [CartController::class, 'show'])->name('cart.show');
// Confirm (store) the cart and save disciplinas registration on the database:
Route::post('cart', [CartController::class, 'store'])->name('cart.store');
// Clear the cart:
Route::delete('cart', [CartController::class, 'destroy'])->name('cart.destroy');
```

40. For now, add 2 methods to the CartController to handle 2 routes: add a “disciplina” to the cart and show the cart. Edit the file app/Http/Controllers/CartController.php:

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

use Illuminate\Http\Request;
use Illuminate\Http\RedirectResponse;
```

```php
use Illuminate\View\View;
use App\Models\Disciplina;
use Illuminate\Support\Facades\DB;

class CartController extends Controller
{
    public function show(): View
    {
        $cart = session('cart', []);
        return view('cart.show', compact('cart'));
    }

    public function addToCart(Request $request, Disciplina $disciplina): RedirectResponse
    {
        try {
            $userType = $request->user()->tipo ?? 'O';
            if ($userType != 'A') {
                $alertType = 'warning';
                $htmlMessage = "O utilizador não é aluno, logo não pode adicionar disciplina ao carrinho";
            } else {
                $alunoId = $request->user()->aluno->id;
                $totalDisciplina = DB::scalar('select count(*) from alunos_disciplinas
                                    where aluno_id = ? and disciplina_id = ?', [$alunoId, $disciplina->id]);
                if ($totalDisciplina >= 1) {
                    $alertType = 'warning';
                    $url = route('disciplinas.show', ['disciplina' => $disciplina]);
                    $htmlMessage = "Não é possível adicionar a disciplina <a href='$url'>#{$disciplina->id}</a>
                        <strong>\"{$disciplina->nome}\"</strong> ao carrinho,
                        porque o aluno já está inscrito à mesma!";
                } else {
                    // We can access session with a "global" function
                    $cart = session('cart', []);
                    if (array_key_exists($disciplina->id, $cart)) {
                        $alertType = 'warning';
                        $url = route('disciplinas.show', ['disciplina' => $disciplina]);
                        $htmlMessage = "Disciplina <a href='$url'>#{$disciplina->id}</a>
                        <strong>\"{$disciplina->nome}\"</strong> não foi adicionada ao carrinho
                        porque já está presente no mesmo!";
                    } else {
                        $cart[$disciplina->id] = $disciplina;
                        // We can access session with a request function
                        $request->session()->put('cart', $cart);
                        $alertType = 'success';
                        $url = route('disciplinas.show', ['disciplina' => $disciplina]);
                        $htmlMessage = "Disciplina <a href='$url'>#{$disciplina->id}</a>
                        <strong>\"{$disciplina->nome}\"</strong> foi adicionada ao carrinho!";
                    }
                }
            }
        } catch (\Exception $error) {
            $url = route('disciplinas.show', ['disciplina' => $disciplina]);
            $htmlMessage = "Não é possível adicionar a disciplina <a href='$url'>#{$disciplina->id}</a>
                        <strong>\"{$disciplina->nome}\"</strong> ao carrinho, porque ocorreu um erro!";
            $alertType = 'danger';
        }
        return back()
            ->with('alert-msg', $htmlMessage)
            ->with('alert-type', $alertType);
    }
}
```

41. Create the view to show the cart. Start by creating the folder “cart” on the “resources/views” folder. Add the file “show.blade.php” to that folder, and add the following code to that file (“resources/views/cart/show.blade.php“):

```
@extends('template.layout')
@section('titulo', 'Carrinho')
@section('subtitulo')
    <ol class="breadcrumb">
        <li class="breadcrumb-item">Espaço Privado</li>
        <li class="breadcrumb-item active">Carrinho</li>
    </ol>
@endsection
@section('main')
    <div>
        <h3>Disciplinas no carrinho</h3>
    </div>
    @if ($cart)
        @include('disciplinas.shared.table', [
            'disciplinas' => $cart,
            'showCurso' => true,
            'showDetail' => true,
            'showEdit' => false,
            'showDelete' => false,
            'showRemoveCart' => true,
        ])
        <div class="my-4 d-flex justify-content-end">
            <button type="submit" class="btn btn-primary" name="ok" form="formStore">
                    Confirmar Inscrições</button>
            <button type="submit" class="btn btn-danger ms-3" name="clear" form="formClear">
                    Limpar Carrinho</button>
        </div>
        <form id="formStore" method="POST" action="{{ route('cart.store') }}" class="d-none">
            @csrf
        </form>
        <form id="formClear" method="POST" action="{{ route('cart.destroy') }}" class="d-none">
            @csrf
            @method('DELETE')
        </form>
    @endif
@endsection
```

42. Next, let’s create a button to add “disciplinas” to the cart. Change the view “disciplinas.shared.table” code – file “resources/views/disciplinas/shared/table.blade.php”:

```
<table class="table">
    <thead class="table-dark">
        . . .
            @if ($showAddCart ?? false)
                <th class="button-icon-col"></th>
            @endif
            @if ($showRemoveCart ?? false)
                <th class="button-icon-col"></th>
            @endif
        </tr>
    </thead>
    <tbody>
        @foreach ($disciplinas as $disciplina)
             . . .
                @if ($showAddCart ?? false)
                    <td class="button-icon-col">
                        <form method="POST" action="{{ route('cart.add', ['disciplina' => $disciplina]) }}">
                            @csrf
                            <button type="submit" name="addToCart" class="btn btn-success">
                                <i class="fas fa-plus"></i></button>
                        </form>
                    </td>
                @endif
                @if ($showRemoveCart ?? false)
                    <td class="button-icon-col">
                        <form method="POST" action="{{ route('cart.remove', ['disciplina' => $disciplina]) }}">
                            @csrf
                            @method('DELETE')
                            <button type="submit" name="removeFromCart" class="btn btn-danger">
                                <i class="fas fa-remove"></i></button>
                        </form>
                    </td>
                @endif
          </tr>
        @endforeach
    </tbody>
</table>
```

- The variables $showAddCart and $showRemoveCart will define whether the buttons to add or remove a “disciplina” from the cart are created – if the variable is not present, then the corresponding button is not created.

43. Change the view “resources/views/disciplinas/index.blade.php” to show the add button to the “disciplinas” (“showAddCart” = true):

```
    . . .
    @include('disciplinas.shared.table', [
        'disciplinas' => $disciplinas,
        'showCurso' => true,
        'showDetail' => true,
        'showEdit' => true,
        'showDelete' => true,
        'showAddCart' => true,
    ])
```

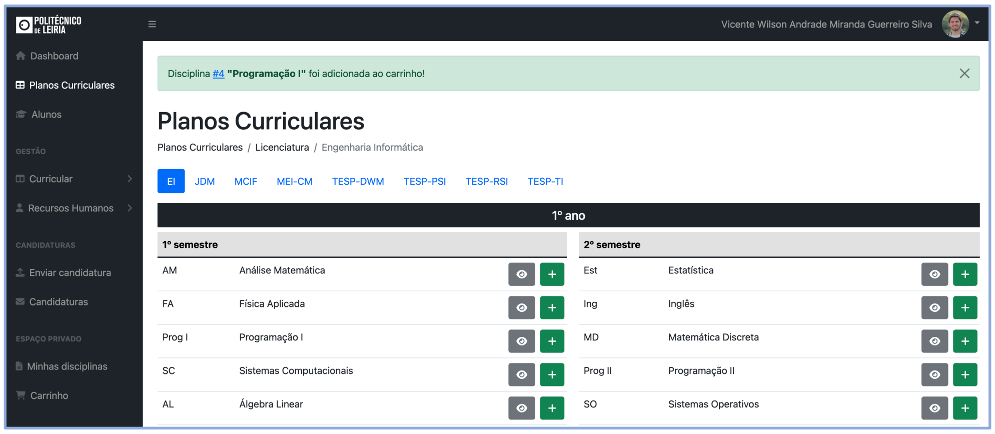
44. Also, create a button to add “disciplinas” to the cart on the view “planos_curriculares.shared.plano”. Change the file “resources/views/planos_curriculares/shared/plano.blade.php”:

```
. . .
    <th colspan="4">{{ $semestre }}º semestre</th>
. . .
@foreach ($disciplinas as $disciplina)
    <tr>
        <td>{{ $disciplina->abreviatura }}</td>
        <td>{{ $disciplina->nome }}</td>
        <td class="button-icon-col"><a class="btn btn-secondary"
                href="{{ route('disciplinas.show', ['disciplina' => $disciplina]) }}">
                <i class="fas fa-eye"></i></a></td>
        <td class="button-icon-col">
            <form method="POST"
                action="{{ route('cart.add', ['disciplina' => $disciplina]) }}">
                @csrf
                <button type="submit" name="addToCart" class="btn btn-success">
                    <i class="fas fa-plus"></i></button>
            </form>
        </td>
    </tr>
@endforeach
. . .
```

45. Add the menu option “Carrinho” to the template – file “resources/views/template/layout.blade.php”:

```
<div class="sb-sidenav-menu-heading">Espaço Privado</div>
<a class="nav-link {{ Route::currentRouteName() == 'disciplinas.minhas' ? 'active' : '' }}"
    href="{{ route('disciplinas.minhas') }}">
    <div class="sb-nav-link-icon"><i class="fas fa-file-text"></i></div>Minhas disciplinas</a>
<a class="nav-link {{ Route::currentRouteName() == 'cart.show' ? 'active' : '' }}"
    href="{{ route('cart.show') }}">
    <div class="sb-nav-link-icon"><i class="fas fa-shopping-cart"></i></div>Carrinho </a>
```

46. Open the application and login with the email of a student (“aluno”). Try to add “disciplinas” to the cart from the list of “disciplinas” or from the “plano curricular”:



Disciplina #4 **"Programação I"** foi adicionada ao carrinho!

Planos Curriculares

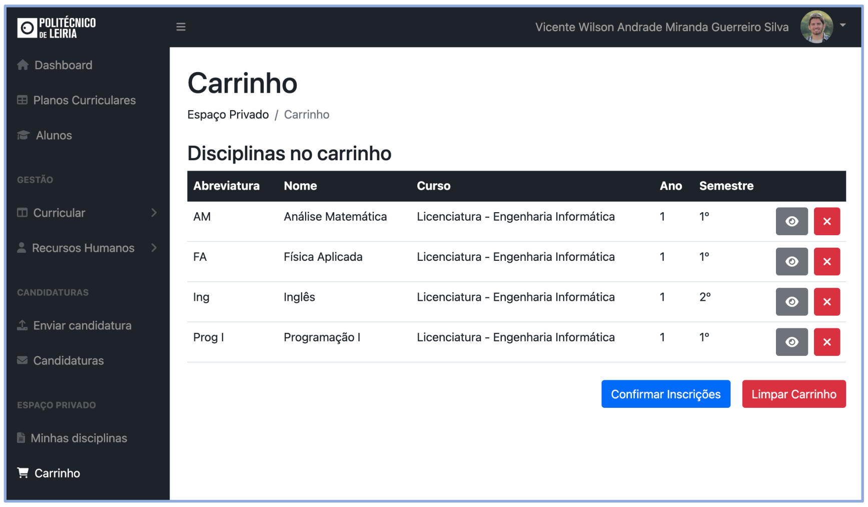
Planos Curriculares / Licenciatura / Engenharia Informática

EI JDM MCIF MEI-CM TESP-DWM TESP-PSI TESP-RSI TESP-TI

1º ano

1º semestre		2º semestre	
AM	Análise Matemática	Est	Estatística
FA	Física Aplicada	Ing	Inglês
Prog I	Programação I	MD	Matemática Discreta
SC	Sistemas Computacionais	Prog II	Programação II
AL	Álgebra Linear	SO	Sistemas Operativos

47. After adding some “disciplinas”, the “carrinho” (cart) page (menu option “Carrinho”) should be similar to:



Carrinho

Espaço Privado / Carrinho

Disciplinas no carrinho

Abreviatura	Nome	Curso	Ano	Semestre
AM	Análise Matemática	Licenciatura - Engenharia Informática	1	1º
FA	Física Aplicada	Licenciatura - Engenharia Informática	1	1º
Ing	Inglês	Licenciatura - Engenharia Informática	1	2º
Prog I	Programação I	Licenciatura - Engenharia Informática	1	1º

48. The options to remove a “disciplina” from the carrinho, “Confirmar Inscrições” and “Limpar Carrinho” are not working yet. Add 3 methods to the CartController, that will handle the 3 routes responsible for these option – file “app/Http/Controllers/CartController.php”

```
public function removeFromCart(Request $request, Disciplina $disciplina): RedirectResponse
{
    $cart = session('cart', []);
    if (array_key_exists($disciplina->id, $cart)) {
        unset($cart[$disciplina->id]);
    }
    $request->session()->put('cart', $cart);
    $url = route('disciplinas.show', ['disciplina' => $disciplina]);
    $htmlMessage = "Disciplina <a href='$url'>#{$disciplina->id}</a>
                     <strong>\"{$disciplina->nome}\"</strong> foi removida do carrinho!";
    return back()
```

```
        ->with('alert-msg', $htmlMessage)
        ->with('alert-type', 'success');
}
public function store(Request $request): RedirectResponse
{
    try {
        $userType = $request->user()->tipo ?? 'O';
        if ($userType != 'A') {
            $alertType = 'warning';
            $htmlMessage = "O utilizador não é aluno, logo não pode confirmar as inscrições
                            às disciplina do carrinho";
        } else {
            $cart = session('cart', []);
            $total = count($cart);
            if ($total < 1) {
                $alertType = 'warning';
                $htmlMessage = "Não é possível confirmar as inscrições porque não há disciplina no carrinho";
            } else {
                $aluno = $request->user()->aluno;
                DB::transaction(function () use ($aluno, $cart) {
                    foreach ($cart as $disciplina) {
                        $aluno->disciplinas()->attach($disciplina->id, ['repetente' => 0]);
                    }
                });
                if ($total == 1) {
                    $htmlMessage = "Foi confirmada a inscrição a 1 disciplina ao aluno
                                   #{$aluno->id} <strong>\"{$request->user()->name}\"</strong>";
                } else {
                    $htmlMessage = "Foi confirmada a inscrição a $total disciplinas ao aluno
                                   #{$aluno->id} <strong>\"{$request->user()->name}\"</strong>";
                }
                $request->session()->forget('cart');
                return redirect()->route('disciplinas.minhas')
                    ->with('alert-msg', $htmlMessage)
                    ->with('alert-type', 'success');
            }
        }
    } catch (\Exception $error) {
        $htmlMessage = "Não foi possível confirmar a inscrição das disciplinas do carrinho,
                        porque ocorreu um erro!";
        $alertType = 'danger';
    }
    return back()
        ->with('alert-msg', $htmlMessage)
        ->with('alert-type', $alertType);
}
public function destroy(Request $request): RedirectResponse
{
    $request->session()->forget('cart');
    $htmlMessage = "Carrinho está limpo!";
    return back()
```

```
        ->with('alert-msg', $htmlMessage)
        ->with('alert-type', 'success');
}
```

5. Final adjustments (Autonomous work)

On this section, students should implement, autonomously, **confirmation dialogs** to the following operations – only execute the operation if the user confirms it:

- When removing (deleting) a “disciplina”. There are buttons to delete a “disciplina” on these 2 views: “disciplinas.shared.table” and “disciplinas.show”
- When removing (deleting) a “departamento”. There are buttons to delete a “departamento” on these 2 views: “departamentos.index” and “departamentos.show”
- When removing (deleting) a “curso”. There are buttons to delete a “curso” on these 2 views: “cursos.index” and “cursos.show”
- When removing (deleting) a student (“aluno”). There are buttons to delete a “aluno” on these 2 views: “alunos.shared.table” and “alunos.show”
- When removing (deleting) a teacher (“docente”). There are buttons to delete a “docente” on these 2 views: “docentes.shared.table” and “docentes.show”

6. Solution analyzes (Autonomous work)

49. Compare your implementation with the provided solution.

Summary

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

```
Storage with Flysystem
      Storage folder
      Storage link
      Storage methods do delete and store files
asset() function
File upload
      <input type="file" …>
      enctype “multipart/form-data”
      File upload validation
      Storing files on Storage
basename() function
Bootstrap Modal dialogs
    Confirmation Dialog
```

Sessions

    Shopping Cart