UD9. Pt1 – Modelo Vista Controlador. Uso framework (1)

En esta unidad desarrollaremos aplicaciones con arquitectura Modelo-Vista-Controlador empleando el framework **Laravel** y el ORM que lleva incorporado, **Eloquent**.

En esta práctica haremos un CRUD *(Create, Read, Update and Delete*) de cuentas y clientes.

1) Creación del proyecto.
   - En el aula virtual se encuentran los documentos «Laravel - Creación de un proyecto» y «Laravel – Ejercicio inicial». Siguiendo las indicaciones de estos documentos, crea un proyecto **ud9_pt1**
2) Creación del modelo de datos.

Vamos a desarrollar un ejemplo con acceso a datos mediante Eloquent.

- Eloquent es un ORM (Object Relational Mapper) incorporado en Laravel.
  - ✗ Un ORM realiza un mapeo de las clases de la nuestra aplicación en relaciones (tablas) de una Base de datos.
  - ✗ La práctica se desarrollará con un servidor de BDD MySQL.
  - ✗ En Laravel trabajaremos con objetos de las clases Model (clases de PHP), pero todo lo que hagamos se traducirá en cambios en la BDD real.
- Crearemos las clases Model por terminal.
  - ✗ Después, crearemos un fichero de migración específico para cada tabla, indicando la estructura que tendrá (nombres y tipos de las columnas).
  - ✗ Una vez creada la estructura de la BDD, ejecutaremos una **migración** a la BDD real.
- Para utilizar Eloquent en nuestro proyecto, podemos seguir la documentación oficial:

https://laravel.com/docs/11.x/eloquent

- Se propone implementar un modelo de datos con cuentas y clientes similar al que hicimos con PHP.
  - ✗ Hay que personalizar el contenido del fichero **.env** con el nombre de la base de datos, la llamamos **bank2**
  - ✗ Indicamos también el usuario y el password para conectarnos al servidor MySQL.
- La manera más sencilla de crear un modelo es por terminal.
  - ✗ Con el comando que se nos indica en el tutorial de Eloquent, crearemos una clase Model.
  - ✗ Crea por terminal la clase **Cliente**
  - ✗ Podríamos crear el fichero de migración en el momento de crear la clase Model, pero lo haremos más adelante.
  - ✗ Podemos observar que se nos ha creado en el directorio *app/Models* el fichero Cliente.php con la declaración de la clase Cliente

- ✗ Los objetos de esta clase nos permitirán acceder, desde los controladores de la aplicación, a los datos de la tabla correspondiente de la BDD real.

- A continuación, crearemos por terminal la clase **Cuenta**
- El siguiente paso sería crear los ficheros de migración a la BDD. Podemos consultar el tutorial oficial:

https://laravel.com/docs/11.x/migrations

- ✗ Hay que crear un fichero de migración para cada tabla. En la documentación oficial nos indican cómo hay que llamar al fichero de la migración que se creará (lo ponemos en el comando de terminal).

El nombre de la tabla que se creará, se obtiene a partir del nombre del fichero de la migración.

Por defecto, Eloquent espera que el nombre de la tabla sea el nombre de la clase Model en minúscula y pluralizado. Así pues, en nuestro caso las tablas se tienen que llamar *clientes* y *cuentas*.

- ✗ Podemos comprobar que se han creado los ficheros de migración en el directorio *database/migrations*
- ✗ En este momento, para cada tabla sólo tenemos el campo predeterminado *id* y los «timestamps» que informarán de la fecha de creación / modificación de un registro: campos *created_at* y *updated_at*
- ✗ El campo predeterminado *id* será la clave primaria de la tabla, y será autoincremental.

- Para añadir el resto de campos de las tablas cuentas y clientes, tenemos que usar el método **up()** del fichero de la migración que hemos creado. En el tutorial oficial de Migraciones, nos explican la sintaxis para añadir columnas a las tablas.
  - ✗ El método Schema::create nos crea la tabla.
  - ✗ Si queremos modificar una tabla ya existente, disponemos del método Schema::table
  - ✗ Haremos que al crear las tablas se creen las columnas:

En la tabla clientes: **dni**, **nombre**, **apellidos** y **fechaN** (fecha de nacimiento).

En la tabla cuentas: **codigo** y **saldo**.

- ✗ Todos los campos son de tipo cadena de texto, excepto la fecha de nacimiento (date) y el saldo (integer).
- ✗ Además, tenemos que indicar el campo **cliente_id** que hará la relación entre las tablas *cuentas* y *clientes*. Consulta en el tutorial de migraciones la sección “Foreign Key Constraints” para ver cómo hacerlo.
- ✗ Haz también que el campo cliente_id admita valores nulos. Podemos conseguirlo con el método **nullable()**. Consulta la sección “Column Modifiers” del tutorial para ver ejemplos de su uso.
- ✗ Para ejecutar todas las migraciones pendientes, usamos el comando que nos indica el tutorial.
- ✗ Podemos comprobar a través de phpMyAdmin que se han creado las tablas *cuentas* y *clientes* en la BDD *bank2*.
- ✗ Las tablas están vacías, pero tienen la estructura de nuestras clases.
- ✗ Como era de esperar, en la tabla *cuentas* hay una Foreign Key *cliente_id* que hace referencia a la tabla *clientes*.

- ✗ Comprueba también que el campo *cliente_id* admite valores nulos. En cambio, el resto de campos que hemos añadido tienen la restricción NOT NULL de MySQL.

3) Ver / Crear / Eliminar cuentas.

- Se proporciona al final de este documento el código necesario para implementar estas funcionalidades.
- Además, haremos que la página de inicio de la aplicación muestre un template con un logo de la aplicación.
- Tienes que crear los ficheros en la carpeta que se indica:
- **public/css/tabla.css**

Es un ejemplo sencillo de CSS para dar estilo a las tablas de la aplicación.

- **routes/web.php**

Fichero de rutas.

Tenemos que poner un nombre a las rutas. Por ejemplo, a la ruta «/» de la aplicación le hemos puesto el nombre 'home'.

Esto lo hacemos para poder apuntar a las rutas desde los links con el método route(), que genera una URL a partir del nombre de la ruta. También lo usaremos en las action de los formularios. Más información en este enlace:

https://laravel.com/docs/11.x/routing#generating-urls-to-named-routes

- **app/Http/Controllers/DefaultController.php**

Contiene el método home() que corresponde a la ruta «/» de la aplicación.

- **app/Http/Controllers/CuentaController.php**

Controlador para implementar las funcionalidades ver cuentas, crear una cuenta y eliminar una cuenta.

En el método para ver cuentas, se usa la instrucción *all()* de Eloquent para obtener todas las cuentas bancarias de la BDD.

El método para crear una cuenta, si miramos el fichero de rutas, se ejecuta con método HTTP Get o Post. Esto es porque, el mismo método, hace 2 cosas:

- ✗ Si venimos de clicar al link «Nueva cuenta» (método HTTP Get), muestra el formulario de creación. Se tiene que hacer una consulta de los clientes para cargar el desplegable del formulario.
- ✗ Si venimos de hacer submit al formulario (método HTTP Post), recogemos en un objeto $cuenta los valores recibidos del formulario, y lo persistimos en la base de datos.

- **resources/views/layout.blade.php**

Todas las páginas de la aplicación heredarán el diseño de este template.

- **resources/views/navbar.blade.php**

Es la barra de navegación, que está incluida en el template layout, y por tanto se verá en todas las páginas de la aplicación.

- **resources/views/default/home.blade.php**

En la página de inicio o home se muestra el logo de la aplicación. Tienes que crear una carpeta *img*

dentro de public, y dentro un fichero con la imagen del logo (con el nombre de fichero que hemos indicado en el template *home.blade.php*).

- **resources/views/cuenta/new.blade.php**

Template donde se muestra el formulario de creación de la cuenta.

Para cargar el desplegable de clientes, usamos el parámetro $clientes que hemos recibido del controlador.

- **resources/views/cuenta/list.blade.php**

Listado de cuentas, donde se hace una iteración sobre el parámetro $cuentas recibido del controlador, mediante la directiva @foreach de las plantillas Blade.

4) Prueba de funcionamiento.

- Para probar las funcionalidades del apartado anterior (ver, crear y eliminar cuentas) primero necesitamos crear clientes de prueba.
- Como todavía no hemos implementado las funcionalidades del cliente, se propone crear dos o tres clientes de prueba desde phpMyAdmin:

BDD bank2 -> tabla clientes -> *Insertar*

public/css/tabla.css

```
table, th, td {
  border: 1px solid black;
}
th, td {
  padding: 15px;
}
```

routes/web.php

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\DefaultController;
use App\Http\Controllers\CuentaController;

/*
Route::get('/', function () {
    return view('welcome');
});
*/

Route::get('/', [DefaultController::class, 'home'])->name('home');

//// CUENTAS

Route::get('/cuenta/list', [CuentaController::class, 'list'])->name('cuenta_list');

Route::match(['get', 'post'], '/cuenta/new', [CuentaController::class,
'new'])->name('cuenta_new');

Route::get('/cuenta/delete/{id}', [CuentaController::class,
'delete'])->name('cuenta_delete');
```

app/Http/Controllers/DefaultController.php

```
<?php

namespace App\Http\Controllers;

class DefaultController extends Controller
{
    function home()
    {
      return view('default.home');
    }
}
```

app/Http/Controllers/CuentaController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Cuenta;
use App\Models\Cliente;

class CuentaController extends Controller
{
    function list()
    {
      $cuentas = Cuenta::all();

      return view('cuenta.list', ['cuentas' => $cuentas]);
    }

    function new(Request $request)
    {
      if ($request->isMethod('post')) {
        // recogemos los campos del formulario en un objeto cuenta

        $cuenta = new Cuenta;
        $cuenta->codigo = $request->codigo;
        $cuenta->saldo = $request->saldo;
        $cuenta->cliente_id = $request->cliente_id;
        $cuenta->save();

        return redirect()->route('cuenta_list')->with('status', 'Nueva cuenta
'.$cuenta->codigo.' creada!');
      }
      // si no venimos de hacer submit al formulario, tenemos que mostrar el
formulario

      $clientes = Cliente::all();

      return view('cuenta.new', ['clientes' => $clientes]);
    }

    function delete($id)
    {
      $cuenta = Cuenta::find($id);
```

```
      $cuenta->delete();

      return redirect()->route('cuenta_list')->with('status', 'Cuenta
'.$cuenta->codigo.' eliminada!');
    }
}
```

resources/views/layout.blade.php

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>UD9 Pt1 - @yield('title')</title>
        @section('stylesheets')
           <link rel="stylesheet" href="{{ asset('css/tabla.css') }}" />
        @show
    </head>
    <body>
        @include('navbar')

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

resources/views/navbar.blade.php

```
<!-- Navigation -->
<nav>
    <a href="{{ route('home') }}">Inicio</a>
       
    <a href="{{ route('cuenta_list') }}">Cuentas</a>
</nav>
```

resources/views/default/home.blade.php

```
@extends('layout')

@section('title', 'Home')

@section('stylesheets')
    @parent
@endsection

@section('content')
    <div>
      <img src="{{ asset ('img/logo.png') }}" alt="">
      <h2>UD9 Pt1</h2>
      <hr>
      <h3>Práctica para iniciarse en los conceptos básicos de Laravel</h3>
    </div>
@endsection
```

resources/views/cuenta/new.blade.php

```
@extends('layout')

@section('title', 'Nueva Cuenta')
```

```
@section('stylesheets')
    @parent
@endsection

@section('content')
    <h1>Nueva Cuenta</h1>
    <a href="{{ route('cuenta_list') }}">&laquo; Volver</a>
      <div style="margin-top: 20px">
        <form method="POST" action="{{ route('cuenta_new') }}">
            @csrf
            <div>
                <label for="codigo">Código</label>
                <input type="text" name="codigo" />
            </div>
            <div>
                <label for="saldo">Saldo</label>
                <input type="number" name="saldo" />
            </div>
            <div>
                <label for="cliente_id">Cliente</label>
                <select name="cliente_id">
                    @foreach ($clientes as $cliente)
                        <option value="{{ $cliente->id }}">{{ $cliente->nombre }}
{{ $cliente->apellidos }}</option>
                    @endforeach
                </select>
            </div>
            <button type="submit">Crear Cuenta</button>
        </form>
      </div>
@endsection
```

resources/views/cuenta/list.blade.php

```
@extends('layout')

@section('title', 'Listado de cuentas')

@section('stylesheets')
    @parent
@endsection

@section('content')
    <h1>Listado de cuentas</h1>
    <a href="{{ route('cuenta_new') }}">+ Nueva cuenta</a>

    @if (session('status'))
        <div>
            <strong>Success!</strong> {{ session('status') }}
        </div>
    @endif

    <table style="margin-top: 20px;margin-bottom: 10px;">
        <thead>
            <tr>
                <th>Código</th><th>Saldo</th><th>Cliente</th>
```

```
            </tr>
        </thead>
        <tbody>
            @foreach ($cuentas as $cuenta)
                <tr>

<td>{{ $cuenta->codigo }}</td><td>{{ $cuenta->saldo }}</td><td>{{ $cuenta->cliente_id
}}</td>
                    <td>
                        <a href="{{ route('cuenta_delete', ['id' =>
$cuenta->id]) }}">Eliminar</a>
                    </td>
                </tr>
            @endforeach
        </tbody>
    </table>

    <br>
@endsection
```

Entrega de la práctica: carpeta comprimida con los ficheros de código fuente SIN LA CARPETA VENDOR