

Curso: Desarrollo web BackEnd

Docente: David Alcolea

DETALLE Y DESCRIPCIÓN

Nombre: LARAVEL: CRUD con ORM: Consulta i alta de datos

Objetivos:

- Construir una aplicación siguiendo el patrón Modelo Vista Controlador
- Confección de controladores y modelos
- Realizar un CRUD con Eloquent ORM
- Pruebas unitarias con phpUnit

Competencias asociadas al PLA:

Competencias técnicas	Soft Skills
<ul style="list-style-type: none">• Construcción de aplicaciones MVC• Aprender el funcionamiento del framework Laravel• Uso de Eloquent ORM	<ul style="list-style-type: none">• Resolución de problemas• Interpretar requerimientos• Gestionar un proyecto• Búsqueda, gestión y uso de la información

Instrucciones que recibirá el alumno:

A partir de la actividad desarrollada en el PLA anterior, el alumno confeccionará la operativa de consulta y alta de persona y el acceso a la vista de cuenta puntos de la persona

En sucesivas actividades se irá completando el proyecto hasta construir una plataforma de mantenimiento de un producto asociado a la operativa de pagos con tarjeta de una Entidad financiera real (resumida por motivos obvios de duración del curso)

En este documento se describe, en el apartado 'Requerimientos de usuario' el funcionamiento de la operativa solicitada en este PLA:

- Confección de las rutas para los controladores de las operativas de consulta y alta de persona
- Confección de los controladores de la tabla personas
- Confección del modelo Personas
- Modificación de las vistas de consulta y alta de personas
- Confección del fichero de pruebas unitarias con phpunit

Requerimientos de usuario

Se habilitará la operativa para la pantalla de Gestión comercial y alta de personas.

Gestión comercial

La entrada a la plataforma se realizará desde la pantalla de **Gestión comercial**, en donde informaremos el nif de la persona a asociar al programa cuenta puntos o bien, en caso que la persona ya tenga una cuenta, realizar la consulta y mantenimiento de la misma

Datos de la pantalla

Identificación fiscal → Obligatorio. Se informará con el nif de la persona a consultar

Nombre y apellidos → Protegido. Se mostrará nombre y apellidos de la persona consultada

Acciones a realizar al entrar en la pantalla

- Si se accede a la pantalla desde 'alta personas' o 'cuenta puntos' con una consulta previa en esta pantalla, se mostrarán los datos de nif y nombre de la persona consultada

Acciones posibles en la pantalla

- Al completar el campo de identificación fiscal se realizará la consulta del nombre de la persona al abandonar el cursor el control del formulario.

- El nombre se mostrará en el control protegido situado a su derecha
- Si no se informa el documento se mostrará un error:

- Si el documento no existe en la base de datos e mostrará un error:

Nº Ident. Fiscal

nif no existe en la base de datos

- Si se accede a la pantalla desde otra se mostrará el nif y el nombre de la última persona consultada en caso que exista una. Por ejemplo, si consultamos una persona, accedemos a la pantalla de Cuenta puntos y, posteriormente, regresamos a gestión (ya sea utilizando el botón de 'abandonar' en cuenta puntos o la opción de menu 'Gestión comercial')

Pantalla de alta de personas

Pantalla para efectuar el alta de personas a asociar al programa de puntos

Chungobank Investments & Trusts

Gestión comercial

Cuenta Puntos Alta personas

Alta Personas

NIF:

NOMBRE:

APELLIDOS:

DIRECCION:

EMAIL:

TARJETA

Zona de mensajes

Acceso a la pantalla

Se accederá a partir de la opción de menú 'Alta personas'

Datos de la pantalla

- | | |
|------------------|--|
| <i>Nif</i> | → Obligatorio. Se informará con el nif de la persona a dar de alta |
| <i>Nombre</i> | → Obligatorio. Se informará con el nombre de la persona |
| <i>Apellidos</i> | → Obligatorio. Se informará con los apellidos de la persona |
| <i>Dirección</i> | → Obligatorio. Se informará con la dirección de la persona |
| <i>Email</i> | → Obligatorio. Se informará con el email de la persona |
| <i>Tarjeta</i> | → Protegido. Se mostrará el número PAN (<i>Personal Account Number</i>) que el sistema asignará a la tarjeta |

Acciones a realizar al entrar en la pantalla

- No es necesaria ninguna acción adicional:

Acciones posibles en la pantalla

Botón Alta

- Estado por defecto: Activo
- Función:
 - Al completar todos los controles del formulario y pulsar el botón se procederá al alta de la persona en la base de datos (tabla Personas)
 - Todos los datos son obligatorios de forma que se mostrará un error en caso que no se informen

Alta Abandonar

- El campo nif es obligatorio
- El campo nombre es obligatorio
- El campo apellidos es obligatorio
- El campo dirección es obligatorio
- El campo email es obligatorio

- SI el nif ya existe en la base de datos se mostrará un error

Alta Abandonar

- Es nif ya existe

- SI el alta es correcta se mostrará el número de tarjeta que el sistema ha asignado a la personas

NIF: 12999678K

NOMBRE: John

APELLIDOS: Rambo

DIRECCION: Trattoria street, 10

EMAIL: john@mail.com

TARJETA: 3944 6604 6344 2255

- Desactivación: Nunca

Botón Abandonar

- Estado por defecto: Activo
- Función: Retorno a la pantalla de *Gestión comercial*
- Desactivación: Nunca

Pantalla de alta y mantenimiento cuenta puntos

Pantalla para efectuar el alta, consulta y mantenimiento de la cuenta puntos de la persona consultada en la pantalla de Gestión Comercial

Acceso a la pantalla

Se accederá a partir de la opción de menú 'Cuenta Puntos'

Datos de la pantalla

- | | |
|---------------------------|--|
| <i>Contrato puntos</i> | → Protegido. Se mostrará entidad, oficina, dc y numero de cuenta puntos de la persona consultada (si ya tiene una) o el numero asignado por el sistema en caso de alta |
| <i>Titular</i> | → Protegido. Se mostrará nombre y apellidos de la persona seleccionada |
| <i>Código programa</i> | → Obligatorio. Mostrar/seleccionar el programa de puntos |
| <i>Descripción</i> | → Protegido. Se mostrará la descripción del programa seleccionado |
| <i>Renuncia extracto</i> | → Opcional. Se seleccionará si el cliente no desea recibir extracto mensual |
| <i>Renuncia obtención</i> | → Opcional. Se seleccionará si el cliente no desea acumular puntos |

Acciones a realizar al entrar en la pantalla

- Si se accede sin haber seleccionado una persona en gestión comercial se retornará a Gestión Comercial o se mostrará el mensaje:

- Si se accede con una persona que todavía no tiene cuenta puntos asociada se mostrará el mensaje;

- Si se accede con una persona que ya tiene asignada una cuenta puntos se mostrarán todos los datos de la cuenta

CONTRATO PUNTOS:	0001	0200	10	0200123456
TITULAR:	10000001A	Margaret Rose		
<hr/>				
CÓDIGO PROGRAMA:	PBS ▼			
DESCRIPCIÓN PROGRAMA:	Programa Puntos Básico			
RENUNCIA EXTRACTO:	<input type="checkbox"/>			
RENUNCIA OBTENCIÓN PUNTOS:	<input type="checkbox"/>			
<div>Alta Modificar Baja Consulta mvto Abandonar</div>				

Especificaciones estructura plataforma

Consideraciones técnicas a tener en cuenta en la realización de la actividad:

- Se creará un controlador para la operativa asociada a cada una de las tablas:

PersonasController

CuentasController

- Se creará un modelo para la operativa asociada a cada una de las tablas:

Personas

Cuentas

- Se creará un fichero de test para cada uno de los controladores

PersonasTests

CuentasTests

- Para los accesos a la base de datos se utilizará ***Eloquent ORM***
- Utilizaremos en las peticiones al servidor la terminología ***REST*** en cuanto al método de envío de datos a utilizar según la operativa:

GET para consultas

POST para altas

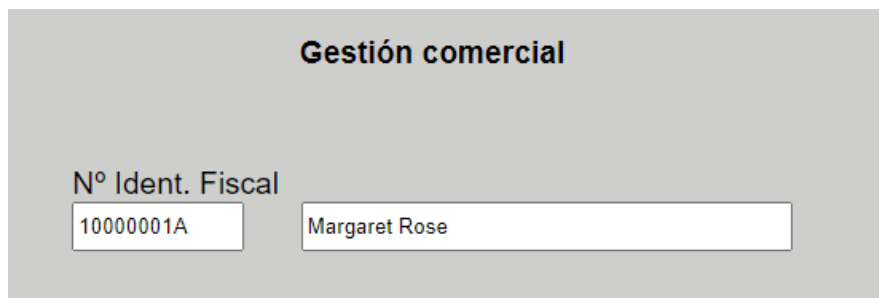
PUT para modificación de datos

DELETE para borrado de datos

Explicación técnica del Reto (propuesta de solución)

EJERCICIO 1: GESTION COMERCIAL: CONSULTA DE PERSONA

Añadiremos la operativa de consulta de persona a partir del documento identificativo en la pantalla de Gestión comercial:



1. Modificación fichero de rutas

Modificaremos el fichero de rutas **routes/web.php** para añadir la nueva ruta de la operativa de consulta de persona por nif. Por ejemplo:

```
Route::get('/personas/{nif?}', [PersonasController::class, 'consulta'])->name('personas.consulta');
```

NOTA: Vamos a seguir la convención REST en la que las consultas por GET se componen de la estructura 'recurso/id' (recurso a utilizar y clave del registro a consultar)

De esta forma esperamos recibir para el recurso personas un parámetro opcional nif (opcional porque el usuario puede tabular sobre el control de identificación fiscal sin informar ningún valor)

2. Creación del modelo *Personas*

Crearemos el modelo **Personas** en la carpeta **app/Http/models**

3. Creación del controlador *PersonasController*

Crearemos el controlador **PersonasController** en la carpeta **app/Http/controllers** con el método de consulta de persona por nif:

```
public function consulta($nif=null) { ... } //recordad que nif puede venir sin informar
```

- Incorporar la librería del modelo al inicio del controlador: `use App\Models\Personas;`
- Recuperar el nif de la vista con `$data['nif'] = $nif;`
- Validar que el nif se encuentre informado y que exista en la base de datos con la clase **Validator**:
 - Incorporar la librería al inicio del controlador:


```
use Illuminate\Support\Facades\Validator;
```
 - Definir las reglas de validación y los mensajes asociados a cada una de ellas


```
$rules = array(
                'nif' => ["required"]
            );
```



```
$messsages = array(
    'nif.required' => 'El campo nif es obligatorio'
);
```

- Ejecutar las validaciones con:

```
$validator = Validator::make($data, $rules, $messsages);
```

NOTA: ¿Podemos crear una regla personalizada para validar la existencia del nif?

Aunque podemos validar la existencia del nif más adelante cuando accedamos a la tabla **personas** para recuperar los datos, vamos a aprovechar la clase **Validator** para crear una regla personalizada que llamaremos **nifExists**

1. Desde la consola crearemos la nueva regla con `php artisan make:rule nifExists`
2. Editamos el nuevo fichero que se ha creado en **app/Rules/nifExists.php**
3. En el método **passes()** especificamos la validación a realizar (en este caso comprobar que el nif exista en la tabla personas):

```
public function passes($attribute, $value) {
    return Personas::where('nif', $value)->first();
}
```

Recordad incorporar el modelo **Personas** al inicio del fichero.

Si el nif no existe se retornará un valor **null** que se evaluará como false

4. en el método **message()** asociamos el mensaje de error a la regla de validación

```
public function message() {
    return ':attribute no existe en la base de datos';
}
```

Fijaos como asignaremos de forma dinámica el nombre del campo a validar con **:attribute**

5. Para utilizar la nueva regla en nuestro controlador primero tenemos que incorporar el fichero que contiene la regla con `use App\Rules\nifExists;`
6. Dentro del método de consulta, al definir las reglas, añadimos la nueva regla de validación de la siguiente forma (recordad que para el nif ya deberíamos tener la regla **required**)

```
$rules = array(
    'nif' => ["required", new nifExists]
);
```

- Una vez ejecutadas las validaciones comprobamos si se ha producido algún error:

```
if ($validator->fails()) {
    $datos['errors'] = $validator->messages();
}
```

NOTA 1: Guardaremos los errores obtenidos del método **messages()** en el array de datos que posteriormente enviaremos a la vista de gestión comercial

- Si no se han producido errores accederemos a la tabla de personas con el nif informado en la vista. Podemos utilizar:

`$persona = Personas::where('nif', $data['nif'])->first()` // \$data['nif'] es el nif que llega al controlador desde la vista

NOTA 1: Si el nif no existe, en el objeto de consulta `$persona` tendremos un valor `null`

NOTA 2: Importante utilizar los métodos **Eloquent** correctos para poder trabajar posteriormente con el resultado de la consulta en la vista.

La instrucción anterior nos devolverá una instancia del modelo **Personas** (que es lo que nos interesa para poder acceder a sus atributos de forma directa con, por ejemplo, `$persona->nombre`). Ejemplo:

```
=> App\Models\Personas {#4250
  id: 55,
  nif: "10000001A",
  nombre: "Margaret",
  apellidos: "Rose",
  direccion: "Av. Pignarelli, 56",
  email: "margaret@mail.com",
  tarjeta: "1234567890123456",
  created_at: "2021-11-16 09:36:18",
  updated_at: "2021-11-16 09:36:18",
}
```

```
>>> $personas->nombre
=> "Margaret"
```

Mientras que si utilizamos la instrucción:

`$persona = Personas::where('nif', $data['nif'])->get()`

Obtendremos una colección que no podremos utilizar de forma directa en la vista

```
=> Illuminate\Database\Eloquent\Collection {#4460
  all: [
    App\Models\Personas {#4252
      id: 55,
      nif: "10000001A",
      nombre: "Margaret",
      apellidos: "Rose",
      direccion: "Av. Pignarelli, 56",
      email: "margaret@mail.com",
      tarjeta: "1234567890123456",
      created_at: "2021-11-16 09:36:18",
      updated_at: "2021-11-16 09:36:18",
    },
  ],
}
```

```
>>> $personas->nombre
Exception with message 'Property [nombre] does not exist on this collection instance.'
```

- La clave primaria de la persona consultada en Gestión comercial la tendremos que guardar para poder acceder al resto de pantallas de la plataforma y poder consultar los datos asociados a ésta.

Tenemos varios sistemas pero el mejor de ellos es, sin duda, una variable de sesión. Laravel nos proporciona un útil helper para trabajar con variables de sesión de forma que crearemos una para guardar el id de la persona consultada:

`session(['idPersona' => $persona->id]);` // \$persona es el objeto de consulta de la tabla personas

- Sería deseable también que, en caso de producirse un error de validación, borremos el contenido de la variable de sesión ya que significará que el usuario está consultando un cliente distinto. Para ello dentro de `if ($validator->fails()) {...}` borraremos la sesión:

```
session()→forget('idPersona')
```

- Por último volvemos a cargar la vista de gestión comercial pasando como datos el objeto de la persona creado en la consulta de la base de datos, los errores que se hayan podido producir y cualquier otra información que necesite la página. Por ejemplo:

```
$datos['errors'] = $validator→messages() //mensajes de error de validación
```

```
$datos['persona'] = $persona; //objeto con los datos de la persona consultada
```

```
$datos['titulo'] = 'Gestión comercial' //título de la página
```

```
return view('gestion')→with($datos) //carga de la vista de gestión con los datos anteriores
```

4. Modificación en la vista

Modificaremos la vista **resources/views/gestion-blade.php** para adaptarla a la operativa de consulta

```
@extends('layout')
@section('titulo')
    {{ $titulo }}
@endsection
@section('contenido')
    <form>
        <label>Nº Ident. Fiscal</label><br>
        <input type="text" id="nif" required>
        <input type="text" id='nombre' readonly>
        <span id='mensajes'>Zona de mensajes</span>
        <br><br>
    </form>
@endsection
```

- En principio tendríamos que modificar la etiqueta **<form>** para indicar el método de envío del formulario y la ruta que apunte al controlador a utilizar (y que será el que hemos creado antes)

```
<form method='get' action='{{ url("personas/") }}'>
```

Pero ¡cuidado! Que la ruta que hemos definido espera recibir un parámetro que no podemos enviar dentro del atributo action ya que depende del valor que introduzca el usuario en el input nif del formulario

- Lo que vamos a hacer es detectar cuando el usuario tabula sobre el control del nif en la pantalla para lanzar la consulta utilizando el evento **onblur** el cual va a ejecutar una función javascript

```
<input type="text" id="nif" required onblur='enviar()>
```

- Y realizar la llamada a la ruta utilizando javascript. De esta forma nos aseguramos que estamos enviando el nif informado por el usuario como parte de la ruta:

```
<script type="text/javascript">
    function enviar() {
        let nif = document.querySelector('#nif').value
        window.location.href = "{{ url('personas/') }}" + '/' + nif
    }
</script>
```

- Si la consulta es correcta recibiremos del servidor un objeto Persona con los datos de la persona consultada. De este objeto necesitamos el nombre y los apellidos para mostrarlos en el control situado a la derecha del nif

```
<input type="text" id='nombre' readonly value="{{ $persona->nombre }}" {{ $persona->apellidos }}">
```

- Si ahora intentamos entrar a la pantalla de gestión comercial es probable que veamos el siguiente error:

```
ErrorException
Undefined property: stdClass::$nombre (View:
C:\xampp\htdocs\CIFO\PLA15\RESUELTO\chungobank\resources\views\gestion.blade.php)
http://localhost:8000/
```

¿Por qué?

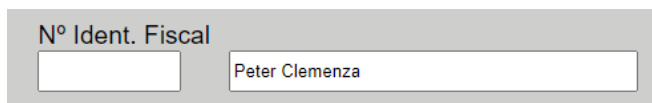
Pues porque cuando accedemos a la pantalla por primera vez todavía no hemos realizado la consulta y, por tanto, no tenemos ningún objeto `$persona->nombre` que mostrar.

¿Podemos hacer algo sencillo para solucionarlo?

Pues sí, solo basta indicar que se muestre el nombre y los apellidos si existe el objeto `persona` y que se muestre un `null` en caso contrario:

```
<input type="text" id='nombre' readonly value="{{ $persona->nombre ?? null }}"
{{ $persona->apellidos ?? null }}">
```

- Si ahora intentamos acceder a la pantalla y consultamos un nif válido veremos como nos debería aparecer el nombre y apellidos correctos pero desaparece el nif que hemos utilizado en la consulta



En este caso tenemos que mostrar también el nif del objeto `Persona` que nos devuelve la consulta en el control de captura del documento de identificación fiscal:

```
<input type="text" id="nif" name="nif" required value="{{ $persona->nif ?? null }}"
onblur='this.form.submit()'
```

- Si consultamos un nif inexistente o bien lanzamos la consulta sin informar ningún nif veremos como no aparecen los mensajes de error que habíamos asociado a las reglas de validación de la class `Validate`.

Vamos a incorporar en la vista la operativa necesaria para mostrar los errores del objeto **\$errors** que nos llegará a la vista en caso de producirse errores de validación:

```
<form method='get' action='{ url("personas/") }'>
    .../...
    @if ($errors->has('nif'))
        <span id='mensajes'>{{ $errors->first('nif') }}</span> //recuperamos el primer error para
        el campo nif
    @endif
</form>
```

Nº Ident. Fiscal

nif no existe en la base de datos

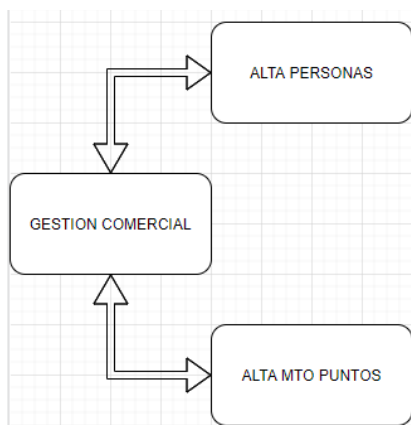
NOTA: Si mostramos con **dd()** el contenido del objeto **\$errors** podemos ver su contenido:

```
Illuminate\Support\MessageBag {#315 ▼
  #messages: array:1 [▼
    "nif" => array:1 [▼
      0 => "nif no existe en la base de datos"
    ]
  ]
  #format: ":message"
}
```

5. Modificación del controlador de carga inicial de la vista

Tal como tenemos la operativa hasta este momento hemos conseguido consultar la persona asociada al nif introducido en la pantalla de Gestión comercial y a mostrar los posibles errores de validación del nif.

Unos de los requisitos del usuario consiste en mantener los datos de la persona consultada cuando se accede a Gestión comercial de forma directa (mediante la opción de menú) o de forma indirecta (mediante la pulsación del botón 'abandonar' de las pantallas de 'alta persona' o 'alta y mantenimiento de cuenta puntos')



Si consultamos una persona y accedemos, por ejemplo, a la pantalla de alta de persona y volvemos a la pantalla anterior o, simplemente, actualizamos la pantalla de Gestión con F5 vemos como perdemos los datos de la persona consultada.

Nº Ident. Fiscal

Para evitarlo tendremos que modificar el método que carga la pantalla de 'Gestión comercial' para añadir una operativa adicional para consultar los datos de la persona que tenemos guardada en la variable de sesión (recordad que en la consulta guardamos el id de la persona en una variable de sesión que permanece mientras no consultemos otra persona)

El método a modificar lo indicará la ruta asociada al directorio raíz de nuestro proyecto (ya que Gestión comercial es la pantalla de entrada a la plataforma)

`Route::get('/', [CargaVistasController::class, 'gestion'])->name('gestion')`

Operativa a añadir en el método **gestion** de **CargaVistasController**

- Comprobar si existe la variable de sesión que hemos creado en la consulta por nif y, de ser así, recuperar su contenido (que será la clave primaria de la persona consultada) y acceder al modelo para recuperar los datos de la persona:

```
if (session()->has('idPersona')) {
    $id = session('idPersona');
    $persona = Personas::find($id);
    $datos['persona'] = $persona;
}
```

Nº Ident. Fiscal

EJERCICIO 2: ALTA DE PERSONAS

Añadiremos la operativa de alta de persona a la que accederemos desde la opción de menu 'Alta personas'

Alta Personas

NIF:

NOMBRE:

APELLIDOS:

DIRECCION:

EMAIL:

TARJETA:

1. Modificación fichero de rutas

Modificaremos el fichero de rutas **routes/web.php** para añadir la nueva ruta de la operativa de alta de persona. Por ejemplo:

```
Route::post('/personas', [PersonasController::class, 'alta'])->name('personas.alta');
```

2. Modificación del controlador *PersonasController*

Modificaremos el controlador **PersonasController** en la carpeta **app/Http/controllers** para añadir el método de alta de persona:

```
public function alta() { ... }
```

- Recuperar los datos de la vista con `$datos = request()→all();`

NOTA: Otra opción es utilizar `$datos = $request→input()` y recoger el objeto `$request` utilizando la Inyección de Dependencias de Laravel:

```
public function alta(Request $request) { ... }
```

- Validar los datos que recibimos de la vista con la clase **Validator**:
 - Las reglas de validación que necesitaremos serán las siguientes:

```
'nif'      => 'required|unique:personas,nif',
'nombre'   => 'required',
'apellidos' => 'required',
'direccion' => 'required',
'email'     => 'required|email',
```

- Asociar a cada regla su correspondiente mensaje:
- Una vez ejecutadas las validaciones comprobamos si se ha producido algún error:

```
if ($validator->fails()) {
    ...
}
```

- A diferencia de la operativa de tratamiento de errores que hemos visto en la pantalla de consulta de nif en Gestión comercial, aquí vamos a cargar también la vista de alta de personas pero utilizando el redireccionamiento con los datos que ya teníamos informados en el formulario y los mensajes de error:

```
if ($validator->fails()) {
    ...
    return redirect()->route('altaPersonas')
        ->withInput() //para mostrar de nuevo los input ya informados
        ->withErrors($validator); //para enviar los mensajes de validación
}
```

NOTA: Otra opción para retornar a la vista es utilizar el método **back()**

```
return back()->withInput()->withErrors($validator)
```

- Si no se han producido errores daremos de alta los datos de la persona en la tabla Personas. Para ello utilizaremos un método estático `alta()` que definiremos en el modelo (ver en el punto 3 de este apartado):

```
$datos['persona'] = Personas::alta($datos); //llamada al método alta() del modelo Personas
```

NOTA 1: La variable `$datos` es el array que hemos recuperado de la vista conteniendo todos los datos recogidos en el formulario

NOTA 2: El número de tarjeta (que no capturamos en el formulario) lo calcularemos concatenando cuatro números aleatorios entre 1000 y 9999:

```
$datos['tarjeta'] = rand(1000, 9000).rand(1000, 9000).rand(1000, 9000).rand(1000, 9000)
```

- La instrucción anterior nos devolverá en `$datos['persona']` una instancia del modelo **Personas** con todos los datos de la persona una vez se ha dado de alta en la base de datos.
- Vamos a guardar la clave primaria de la persona que acabamos de dar de alta en la variable de sesión que vimos en el ejercicio 1, de esta forma, cuando retornemos a la pantalla de Gestión, será la que tengamos seleccionada para otras operativas.

```
session(['idPersona' => $datos['persona']->id])
```

- Finalmente volvemos a cargar la vista de alta de personas

```
$datos['titulo'] = 'Alta Personas'
return view('alta-personas')->with($datos)
```

Control de excepciones

¿Y si necesitamos realizar un control personalizado de las excepciones que nos pueda retornar el modelo Eloquent en el alta para enviar a la vista un mensaje?

- Incluimos al inicio del fichero el namespace de la clase **QueryException**

```
use Illuminate\Database\QueryException;
```

- Incorporamos la operativa de alta dentro de un bloque try...catch

```
try {
    $datos['persona'] = Personas::alta($datos);
    .../...
} catch (QueryException $e) { ... }
```

- El objeto `$e` pertenece a la clase **QueryException** pero tiene un atributo **errorInfo** de tipo array que nos proporcionará toda la información que necesitamos para evaluar el tipo de error que se ha producido (es el mismo atributo **errorInfo** que ya conocemos de la librería PDO ya que Laravel está utilizando internamente esta librería)

```
$e->errorInfo[0]    ---> código SQLSTATE
$e->errorInfo[1]    ---> código error del sistema gestor (ejemplo: 1451, 1062, etc...)
$e->errorInfo[2]    ---> mensaje de error del sistema gestor
```

- Que podemos utilizar para enviar el error a la vista

```
$errores = ['errorbbdd' => $e->errorInfo[2]];
return back()->withErrors($errores)->withInput();
```



```
<form method='post' action='{{ url("personas/") }}'>
```

- Para evitar ataques **CSRF** (*Cross-Site Request Forgery*) Laravel nos obliga a incorporar un token que se enviará al servidor cada vez que se realice un alta. Este token lo incorporaremos dentro del formulario utilizando la función `csrf_field()` o la directiva `@csrf` indistintamente:

```
<form method='post' action='{{ url("personas/") }}'>
    {{ csrf_field() }} o @csrf
    ...
</form>
```

NOTA 1: Esta función o la directiva nos creará un `input` de tipo `hidden` con el token a enviar

- `<input type="hidden" name="_token" value="V0k0rRwcs1uQMgQ0kyVFMilZH8n0lxW7FkaaSaeE">`
- NOTA 2: Si no incorporamos el token es muy probable que nos aparezca el siguiente error cuando intentemos realizar la consulta de una persona en la pantalla:

419 | PAGE EXPIRED

- Añadir los atributos `name` a todos los controles que permiten introducción de datos y cambiar el tipo de botón de alta a `submit`:

```
<input type="text" id="nif" name="nif">
<input type="submit" id="alta" name="alta" value='Alta'>
```

- Si la consulta es correcta recibiremos del servidor un objeto **Persona** con los datos de la persona que hemos dado de alta. De este objeto necesitamos el número de la tarjeta que hemos asignado en el controlador para informar los cuatro controles protegidos `pan1` a `pan4`. Ejemplo para `pan1`

```
<input type="text" maxlength='4' id="pan1" disabled value='{{ substr($persona->tarjeta ?? null, 0, 4) }}'>
```

NOTA 1: La función php `substr()` la utilizamos para extraer de la cadena que especificamos en el primer parámetro, el carácter que ocupa la posición del segundo parámetro y, desde esta posición, los elementos que indiquemos en el tercer parámetro.

NOTA 2: Si accedemos a la pantalla de alta desde gestión comercial es obvio que todavía no tendremos una instancia del modelo `Personas` por lo que, para evitar que `$persona->tarjeta` de un error, utilizamos la expresión `$persona->tarjeta ?? null`

- Ya estamos en condiciones de probar nuestra pantalla de alta. Si informamos todos los campos obligatorios y pulsamos sobre el botón de alta es posible que veamos el siguiente error:

```
Illuminate\Database\Eloquent\MassAssignmentException
Add [nif] to fillable property to allow mass assignment on [App\Models\Personas].
```

<http://localhost:8000/personas>

La asignación masiva significa que estamos completando una fila con más de una columna utilizando una matriz de datos. (algo así como un acceso directo en lugar de construir manualmente la matriz) usando `request()->all()`.

Para evitar este error tendremos que indicar en el modelo **Personas** los atributos para los cuales permitimos asignación masiva de datos. Para ello en **app/Models/Personas.php** añadimos una propiedad **fillable** con la lista de atributos donde vamos a utilizar asignación masiva de valores:

```
protected $fillable = [
    'nif',
    'nombre',
    'apellidos',
    'direccion',
    'email',
    'tarjeta'
];
```

- Probamos de nuevo un alta de persona y deberíamos ver como ahora si que se inserta una fila en la base de datos, se muestra el número de tarjeta asignado a la persona pero perdemos el resto de valores del formulario.

NIF:	<input type="text"/>
NOMBRE:	<input type="text"/>
APELLIDOS:	<input type="text"/>
DIRECCION:	<input type="text"/>
EMAIL:	<input type="text"/>
TARJETA	<input type="text" value="1393"/> <input type="text" value="6335"/> <input type="text" value="4797"/> <input type="text" value="1389"/>

66 10000078K david alcolea Foscarelli avenue, 45 david@mail.com

Para conservar los valores tendremos que volver a refrescarlos cuando se carga la vista después del alta. Ejemplo para nif:

```
<input type="text" id="nif" name="nif" value="{{ $persona->nif ?? null }}">
```

NOTA: De nuevo tenemos que tener en cuenta que, cuando entramos en la página por primera vez no existirá el objeto **\$persona**

- Si pulsamos el botón de alta con algún campo obligatorio sin informar veremos como no aparecen los mensajes de error que habíamos asociado a las reglas de validación de la class **Validate**.

Vamos a incorporar en la vista la operativa necesaria para mostrar los errores del objeto `$errors` que nos llegará a la vista en caso de producirse errores de validación:

```
<form method='post' action='{{ url("personas/") }}'>
    .../...
    @if ($errors->any())
        <div id='mensajes'>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
</form>
```

NOTA: Si mostramos con `dd()` el contenido del objeto `$errors` podemos ver su contenido:

```
Illuminate\Support\MessageBag {#306 ▼
  #messages: array:5 [▼
    "nif" => array:1 [▼
      0 => "Es nif ya existe"
    ]
    "nombre" => array:1 [▼
      0 => "El campo nombre es obligatorio"
    ]
    "apellidos" => array:1 [▼
      0 => "El campo apellidos es obligatorio"
    ]
    "direccion" => array:1 [▼
      0 => "El campo dirección es obligatorio"
    ]
    "email" => array:1 [▼
      0 => "El campo email es obligatorio"
    ]
  ]
  #format: ":message"
}
```

- Vemos que ahora si aparecen los errores pero no se conservan los valores de los campos del formulario (ya que se están informando a partir del objeto `$personas` que todavía no se ha enviado a la vista al no haberse producido el alta de la persona)

- Es nif ya existe
- El campo dirección es obligatorio
- El campo email es obligatorio

Para solucionarlo solo tenemos que indicar en la vista que se vuelvan a restaurar los valores de los controles input que teníamos antes de pulsar el botón de alta. Para ello utilizaremos la función de laravel `old()`. Ejemplo para nif:

```
<input type="text" id="nif" name="nif" value="{{ old('nif') ?? $persona->nif ?? null }}">
```

- `old()` nos va a asegurar que los valores de los campos se mantengan en caso de error
- `$persona->nif` nos asegura que los valores se mantengan al completarse el alta
- `null` nos asegura que cuando entramos desde la pantalla de gestión no se produzcan errores de variables

5. Modificación del controlador de carga inicial de la vista

Al entrar en la vista de alta de personas no hay que realizar ninguna acción adicional

EJERCICIO 3: ALTA Y MANTENIMIENTO CUENTA PUNTOS: CONSULTA DATOS PERSONA

Añadiremos la operativa de consulta de datos de la persona al acceder a la pantalla de mantenimiento de cuenta puntos desde el menu.

La operativa asociada a esta pantalla la realizaremos en dos actividades:

1. En este PLA realizaremos únicamente la consulta de los datos de la persona en caso que accedamos a ella con una consulta previa en la pantalla de Gestión Comercial. También cumplimentaremos la combo de programas de forma dinámica
2. En el PLA siguiente habilitaremos la consulta de los datos de la cuenta puntos de la persona y el alta, modificación y baja de la cuenta puntos utilizando AJAX

1. Modificación fichero de rutas

En este PLA no hará falta modificar el fichero de rutas ya que las rutas para consulta, alta, modificación y baja de datos de la cuenta puntos la realizaremos en el PLA siguiente

2. Modificación del controlador de carga inicial de la vista

Modificaremos el controlador asociado a la carga inicial de la vista para realizar la consulta de los datos de la persona y de los programas que utilizaremos para cumplimentar la combo..

Uno de los requisitos del usuario consiste en mostrar los datos de la persona consultada cuando se accede a esta pantalla de forma directa (mediante la opción de menú) o de forma indirecta (mediante la pulsación del botón 'abandonar' de la pantalla de 'consulta movimientos')



Para mostrar los datos de la persona tendremos que modificar el método que carga la pantalla de 'mantenimiento de puntos' para añadir la operativa adicional para consultar los datos de la persona que tenemos guardada en la variable de sesión

El método a modificar lo indicará la ruta asociada al controlador que se encarga de la carga de la pantalla al pulsar sobre el menú

`Route::get('/altaMtoPuntos', [CargaVistasController::class, 'altaMtoPuntos'])->name('gestionCtaPuntos')`

Operativa a añadir en el método **altaMtoPuntos** de **CargaVistasController**

- Comprobar si existe la variable de sesión que hemos creado en la consulta por nif y, de ser así, recuperar su contenido (que será la clave primaria de la persona consultada) y acceder al modelo para recuperar los datos de la persona.

En este caso, vamos a realizar una variante utilizando la estructura **try...catch**:

1. Si no existe la variable de sesión lanzamos una excepción

```

if (!session()->has('idPersona')) {
    throw new Exception("Error Processing Request", 1);
}
  
```

2. Si no existe la persona en la base de datos lanzamos una excepción

```

$id = session('idPersona');
if (!$datos['persona'] = Personas::find($id)) {
    throw new Exception("Error Processing Request", 1);
}
  
```

3. En el **catch** de la excepción redireccionamos a la pantalla de gestión comercial

```

return redirect()->route('gestion');
  
```

NOTA: Si vemos que la excepción no funciona no olvideis de incorporar el fichero de Excepciones al inicio del controlador con **use Exception**;

- Si la consulta es correcta enviamos el array datos a la vista de mantenimiento de puntos
- ```

return view('alta-mto-puntos')->with($datos);

```

### 3. Modificación en la vista

Modificaremos la vista **resources/views/alta-mto-puntos-blade.php** para adaptarla a la operativa de consulta de persona

```
@section('contenido')
<form id='formulario'>
<label>CONTRATO PUNTOS:</label>
<input type="text" id="entidad" disabled>
<input type="text" id="oficina" disabled>
<input type="text" id="digito" disabled>
<input type="text" id="cuenta" disabled>

<label>TITULAR:</label>
<input type="text" id="nif" value='' disabled>
<input type="text" id="titular" value='' disabled>

<hr>

<label>CÓDIGO PROGRAMA:</label>
<select id='codigo'>
<option disabled selected value=''>Seleccione código</option>
<option>PBS</option>
<option>PAV</option>
<option>PPR</option>
</select>

<label>DESCRIPCIÓN PROGRAMA:</label>
<input type="text" id='descripcion' disabled>

<label>RENUNCIA EXTRACTO:</label>
<input type="checkbox" name="extracto" value='si'>

<label>RENUNCIA OBTENCIÓN PUNTOS:</label>
<input type="checkbox" name="renuncia" value='si'>

<input type="button" id="altapuntos" value='Alta'>
<input type="button" id="modifpuntos" value='Modificar'>
<input type="button" id="bajapuntos" value='Baja'>
<input type="button" id="movimientos" value='Consulta mvotos' onclick="window.location.href = '{{ url('/ consultaMovimientos') }}'">
<input type="button" id="salir" value='Abandonar' onclick="window.location.href = '{{ url('/') }}'">
Zona de mensajes
</form>
@endsection
```

Tendremos que realizar estrictamente las modificaciones que tengan que ver con la operativa a realizar al cargar la página desde el controlador ya que el resto de operativas asociadas a los botones de la página las realizaremos con peticiones AJAX

- Incorporar el token `csrf_field()` o la directiva `@csrf` indistintamente:
- Informaremos los controles de nif, nombre y apellidos y el campo oculto de id de la persona a partir del objeto Persona que nos llegará del controlador que hemos modificado en el punto anterior. Ejemplo para nif:

```
<input type="text" id="nif" value='{{ $persona->nif ?? null }}' disabled>
```

**Alta y Mantenimiento Cta Puntos**

CONTRATO PUNTOS:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
TITULAR:	<input type="text" value="12345678K"/>	<input type="text" value="Peter Clemenza"/>		

#### 4. Carga dinámica de la combo de programas puntos

Puesto que la combo de programas de puntos es un dato fijo que no depende de la persona o cuenta consultada sería conveniente tenerla ya confeccionada al cargar la pagina.

En la vista de recursos del PLA veréis que las opciones se encuentran confeccionadas de forma estática pero vamos a modificarlas para que se confeccionen a partir de los datos que tenemos en la tabla Programas

- Creamos un nuevo modelo **Programas** en **app/Models**
- En **App/Http/Controllers** modificamos el controlador que carga la página de alta y mantenimiento de puntos para acceder al modelo y recuperar todos los programas  

```
$datos['programas'] = Programas::all(['codigo', 'descripcion']); //solo recuperamos dos columnas
```

NOTA: Recordad incorporar el modelo con `use App\Models\Programas;`
- En la vista **alta-mto-puntos.blade** utilizamos la directiva `@foreach` para confeccionar la combo

```
<select id='programa'>
 <option disabled selected value="">Seleccione código</option>
 @foreach ($programas as $programa)
 <option>{{ $programa->codigo }}</option>
 @endforeach
</select>
```

- Ya deberíamos ver la combo de programas confeccionada correctamente

*PROPUESTA: Para la actividad siguiente sería interesante que, cada vez que se seleccione un programa de la combo, aparezca su descripción en el control 'Descripción programa' situado debajo.*

Podríamos realizar una llamada ajax al servidor para consultar la descripción cada vez que seleccionamos un programa, pero sería más eficiente (al ser un dato fijo con muy poca variabilidad) que tengamos un array de objetos en javascript para poder recuperar la descripción a partir del código del programa sin realizar ninguna llamada al servidor.

```
PAV: "Programa Puntos Avanzado"
PBS: "Programa Puntos Básico"
PPR: "Programa Puntos Premium"
```



Tenemos dos opciones:

1. Confeccionar el objeto javascript con los programas de puntos recorriendo el objeto del modelo `$programas` que recibimos del controlador:

```
<script type="text/javascript">
 var programas = {
 @foreach ($programas as $programa)
 {{ $programa->codigo }} : "{{ $programa->descripcion }}",
 @endforeach
 }
</script>
```

2. Enviar el objeto de programas de puntos en formato json desde el controlador:

```
$datos['programas_json'] = $datos['programas']->toJson();
```

Recogerlo desde javascript sin escapar las comillas para convertirlo a array de objetos javascript

```
var arrayProgramas = {!! $programas_json !!}
```

```
▼ (4) [{...}, {...}, {...}, {...}] 1
 ▶ 0: {codigo: 'PBS', descripcion: 'Programa Puntos Básico'}
 ▶ 1: {codigo: 'PAV', descripcion: 'Programa Puntos Avanzado'}
 ▶ 2: {codigo: 'PPR', descripcion: 'Programa Puntos Premium'}
 ▶ 3: {codigo: 'PPP', descripcion: 'Programa Puntos Prueba'}
 length: 4
```

Confeccionar el objeto javascript de forma dinámica

```
var programas = {}
for (i in arrayProgramas) {
 programas[arrayProgramas[i].codigo] = arrayProgramas[i].descripcion
}
```

3. O, todavía más fácil, usando la directiva `@json` de Laravel para la que no necesitamos enviar el json desde el controlador (utilizaremos directamente el objeto `Programas`)

```
var arrayProgramas = @json($programas)
```

En todos los casos para, posteriormente, recuperar la descripción del programa a partir de su código:

```
let codigo = 'PBS' //código recuperado de la combo del formulario
```

```
let descripcion = programas[codigo]
```

## EJERCICIO 4: ELABORACIÓN DEL FICHERO DE PRUEBAS UNITARIAS

Como último paso de esta actividad, vamos a elaborar el fichero de pruebas unitarias en la carpeta **tests/Feature** utilizando phpUnit con el objetivo de probar:

- La operativa de consulta de personas en gestión comercial
- Alta de persona
- Carga de la vista de alta y mantenimiento de cuenta puntos con consulta de datos de la persona
- Carga de la vista de alta y mantenimiento de cuenta puntos con la combo de programas.

### 1.- Creación de la base de datos de pruebas

Puesto que en estas pruebas necesitaremos la base de datos vamos a crearnos una segunda base de datos copia de la principal para utilizar en las pruebas automatizadas y, de esta forma, no contaminaremos la base de datos principal:

- Desde phpMyAdmin creamos una base de datos de prueba **chungobank\_test**
- Editamos el archivo **phpunit.xml** situado en la carpeta raíz de nuestro proyecto y añadimos una nueva variable para indicar la base de datos a utilizar en las pruebas unitarias  

```
<server name="DB_DATABASE" value="chungobank_test"/>
```
- Vamos a indicar en los ficheros de pruebas unitarias en **tests/feature/** que, antes de cada ciclo de pruebas, se reconstruya la base de datos **chungobank\_test** (recordad que es la que hemos indicado en el fichero **phpunit.xml**) a partir de los ficheros de migración que tenemos en la carpeta **database/migrations**

Incorporar dentro de la clases de prueba el trait **RefreshDatabase** (observar que ya está incorporado al inicio del archivo)

```
class PersonasTest extends TestCase {
 use RefreshDatabase;
 .../...
}
```

### 2.- Creación de los datos de prueba

Vamos a ver dos ejemplos de casos de prueba para ver como podemos crear datos de forma automatizada para aquellas pruebas que los necesiten y algunas aserciones interesantes:

Gestión comercial: test para validar nif sin informar

Nº Ident. Fiscal	<input type="text"/>	<input type="text"/>	nif no existe en la base de datos
------------------	----------------------	----------------------	-----------------------------------

- Vamos a crear una variable de sesión para guardar un id de una persona que exista en la base de datos. ¿Por qué?. Pues porque una de las especificaciones técnicas consiste en que, si informamos un nif no válido se borre la variable de sesión que tuviéramos de una consulta anterior:

Utilizamos las **model factories** de Laravel para crear un usuario (\* ver abajo)

```
$persona = Personas::factory()→create();
```

Creamos la variable de sesión:

```
session(['idPersona' => $persona→id]);
```

- Llamamos a la ruta que corresponde al controlador de consulta de persona pero con el nif sin informar

```
$response = $this->get('/personas', [
 'nif' => "",
]);
```

- En la aserción esperamos ver el error de nif sin informar en la pantalla

```
$response->assertSee('El campo nif es obligatorio')
```

- En otra aserción esperamos que la variable de sesión se haya borrado

```
$response->assertSessionMissing('idPersona');
```

(\*) NOTA:

Para crear una model factory utilizaremos en la linea de comandos:

```
php artisan make:factory Personas
```

Editamos el fichero que se ha creado en **database/factories/PersonasFactory.php** para definir los tipos de datos que queremos crear en la tabla Personas

```
public function definition() {
 return [
 'nif' => $this->faker->unique()->regexify('[A-Za-z0-9]{9}'),
 'nombre' => $this->faker->firstName(),
 'apellidos' => $this->faker->lastName(),
 'direccion' => $this->faker->text(30),
 'email' => $this->faker->unique()->safeEmail(),
 'tarjeta' => $this->faker->regexify('[0-9]{16}')
];
}
```

El objeto **faker** nos genera una secuencia pseudoaleatoria que corresponda con el tipo de dato que especificamos en el método que invocamos a continuación.

En el fichero de pruebas nos aseguramos que incorporamos el namespace del objeto **Faker**

```
use Illuminate\Foundation\Testing\WithFaker;
```

### Alta persona: test de alta de persona con datos válidos

NIF:	56434434G
NOMBRE:	Amerigo
APELLIDOS:	Bonasera
DIRECCION:	Foscarelli avenue, 45
EMAIL:	amerigo@mail.com
TARJETA	<input type="text" value="2834"/> <input type="text" value="5477"/> <input type="text" value="6097"/> <input type="text" value="7599"/>

- Vamos a crear el usuario llamando al controlador encargado del alta de persona

```
$this->post('/personas', [
 'nif' => '56434434G',
 'nombre' => 'Amerigo',
 'apellidos' => 'Bonasera',
 'direccion' => 'Foscarelli avenue, 45',
 'email' => 'amerigo@mail.com',
])
```

- Con una aserción comprobamos que se ha creado la variable de sesión

```
$this->assertSessionHas('idPersona');
```

- Y comprobamos que efectivamente la persona se encuentre en la base de datos

```
$this->assertDatabaseHas('personas', [
 'nif' => '56434434G',
 'nombre' => 'Amerigo',
 'apellidos' => 'Bonasera',
 'direccion' => 'Foscarelli avenue, 45',
 'email' => 'amerigo@mail.com',
])
```

### 3.- Confección de los ficheros de pruebas

Los ficheros de pruebas deberían ser:

- CargaVistasTest

```
PASS Tests\Feature\CargaVistasTest
 1 cargo vista gestion comercial
 2 cargo vista gestion comercial con sesion
 3 cargo vista gestion comercial sin sesion
 4 cargo vista alta personas
 5 cargo vista alta mto cta puntos sin sesion
 6 cargo vista alta mto cta puntos con sesion
 7 cargo vista alta mto cta puntos con sesion no valido
 8 cargo vista consulta mvto cta puntos
 9 cargo vista detalle mvto cta puntos
 10 cargo vista alta mvto cta puntos
```

- PersonasTest

```
❏ buscar nif vacio gestion comercial
❏ buscar nif inexistente gestion comercial
❏ buscar nif valido gestion comercial
❏ alta persona nif sin informar
❏ alta persona nif duplicado
❏ alta persona nombre sin informar
❏ alta persona apellidos sin informar
❏ alta persona direccion sin informar
❏ alta persona email sin informar
❏ alta persona email no valido
❏ alta persona datos validos
```