Se puede ver el funcionamiento de la web en http://51.254.116.159:8004 (Los datos de sesión se encuentran en los comentarios de la práctica)

Ejercicio 1. Añadir y editar clientes

Tenemos que añadir las rutas que nos van a hacer falta para recoger los datos al enviar los formularios. Para esto editamos el fichero de rutas y **añadimos** dos rutas (que deben estar protegidas por el filtro auth):

```
Route::get('/catalog/create', 'CatalogController@getCreate');
Route::post('/catalog/create', 'CatalogController@postCreate');
Route::get('/catalog/edit/{id}', 'CatalogController@getEdit');
Route::put('/catalog/edit/{id}', 'CatalogController@putEdit');
Route::put('/catalog/edit/{id}', 'CatalogController@putEdit');
```

A continuación vamos a editar la vista catalog/edit.blade.php con los siguientes cambios:

 Tenemos que modificar los *input*s para que en el formulario se carguen los valores correspondientes al cliente que vamos a editar. Por ejemplo, en el primer *input* tendríamos que añadir value="{{\$cliente->nombre}}".

Por último tenemos que actualizar el controlador CatalogController con los dos nuevos métodos. En ambos casos tenemos que usar la inyección de dependencias para añadir la clase Request como parámetro de entrada (revisa la sección "Datos de entrada" de GitBook) que nos permitirá acceder a los datos del formulario:

```
39
          public function postCreate(Request $request){
40
            $c = new Cliente;
41
            $c->nombre = $request->input('title');
43
            if ($request->hasFile('file')) {
45
46
               $fichero = $request->file('file');
               $c->imagen = $fichero->getClientOriginalName();
48
               $fichero->storeAs('img',$c->imagen);
              $c->fecha_nacimiento = $request->input('fecha_nacimiento');
51
              $c->correo = $request->input('email');
              $c->save();
              return redirect('/catalog');
```

Realizamos la prueba

Crearemos a trinity

N	ombre
	Trinity
U	rl de la imágen
	Seleccionar archivo trinity.jpg
Fe	echa Nacimiento
	04/07/2006
Er	nail
Г	trinity@gmail.es

Vemos como se ha creado.



Creamos el método putEdit()

```
public function putEdit(Request $request, $id){
    $c = Cliente::findOrFail($id);

    //$c = new Cliente;

    $c->nombre = $request->input('title');

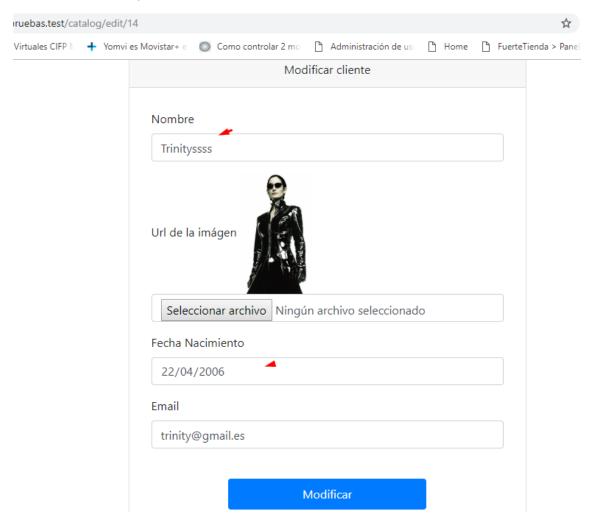
if ($request->hasFile('file')) {

    //obtenemos el campo file definido en el formulario
        $fichero = $request->file('file');
        $c->imagen = $fichero->getClientOriginalName();
        $fichero->storeAs('img',$c->imagen);
}

    $c->fecha_nacimiento = $request->input('fecha_nacimiento');
    $c->c->correo = $request->input('email');
    $c->save();

    return redirect('/catalog');
}
```

Modificamos a Trinity



Consultamos a trinity



Ejercicio 2

Para finalizar nuestro prototipo de aplicación vamos a añadir un botón de borrar cuando estemos en la vista de detalle de un cliente.

Tendremos que:

- 1. Crear una nueva ruta,
- 2. Crear un nuevo método en el controlador
- 3. Actualizar el botón en la vista y redireccionar a la vista catálogo.

En la siguiente tabla se muestra un resumen de la ruta:

Ruta Tipo Controlador / Acción

/catalog/delete/{id} DELETE CatalogController@putDelete

En el método putDelete obtendremos el registro del cliente en cuestión y llamaremos al método delete().

Tenemos que editar la vista detalle de clientes para añadir el botón de borrado. Dado que el borrado se realiza usando una petición HTTP tipo DELETE no podemos poner un enlace normal (ya que sería tipo GET). Para solucionarlo tenemos que crear un formulario alrededor del botón y asignar al formulario el método correspondiente, por ejemplo:

Creamos el método putDelete

Ojo! para usar Storage::delete hay que añadir : use Illuminate\Support\Facades\Storage;

```
public function putDelete($id){

    $cliente = Cliente::find($id);
    $img = $cliente["imagen"];
    Storage::delete('img/'.$img);
    $cliente->delete();

    return redirect('/catalog');
}
```