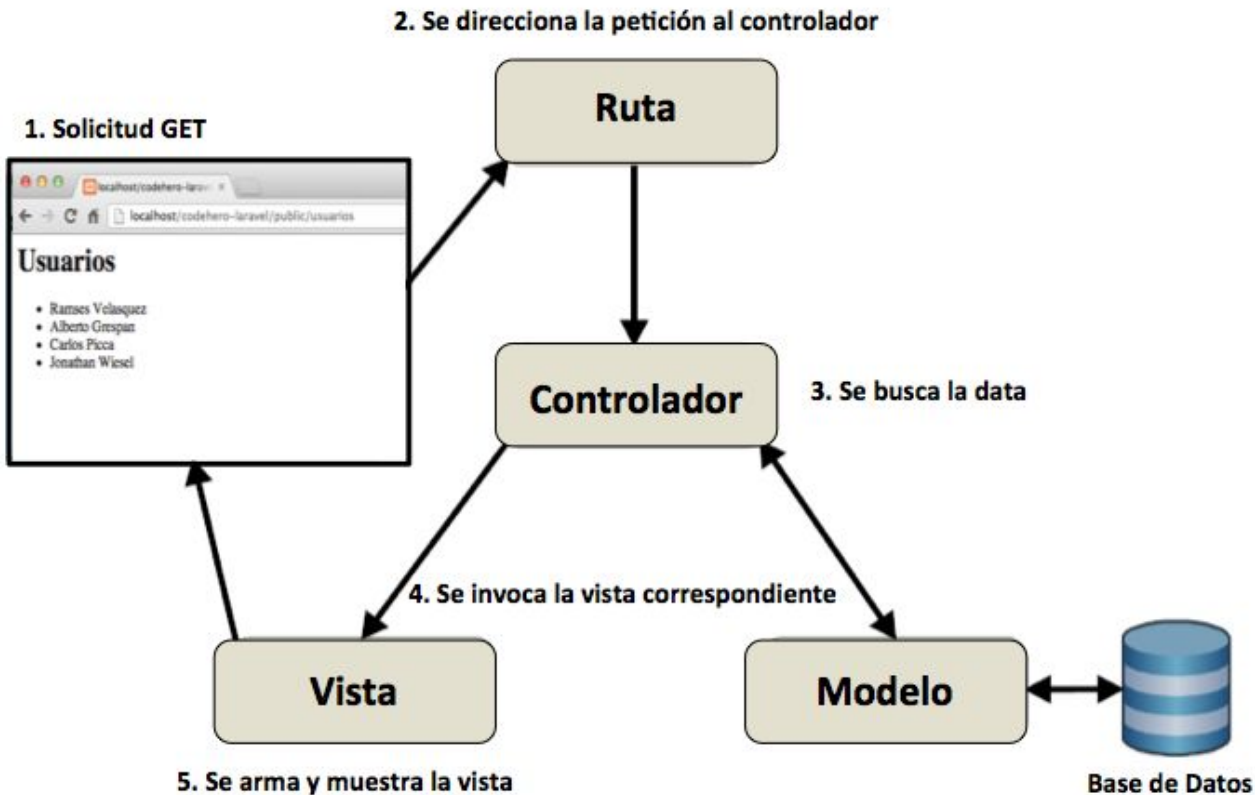


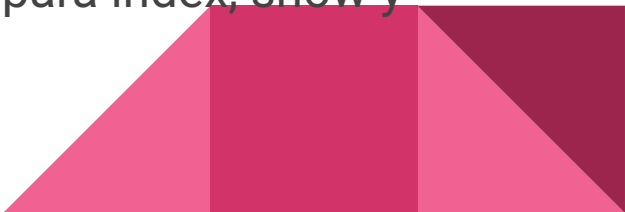
Laravel: Modelo-Vista-Controlador

Crear aplicación básica MVC con datos persistentes en Laravel

Recordatorio: esquema MVC en Laravel



Pasos para generar aplicación con diseño MVC

1. Crear proyecto de Laravel
 2. Configurar la conexión con la base de datos (editar .env)
 3. Crear **migration** con estructura de tablas (esquema) necesaria y generar tablas en base de datos.
 4. Crear **Modelo** para cada tabla.
 5. Crear **Controlador** de recursos para cada tabla.
 6. Crear las **rutas** necesarias para poder acceder al servicio (al controlador).
 7. Crear **vistas** para mostrar datos de las tablas y con formularios para trabajar con registros. Se requieren, mínimo, vistas para index, show y create/edit (formulario).
- 

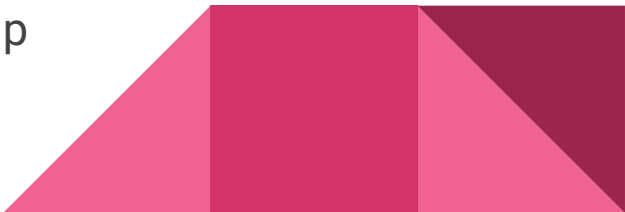
Pasos para generar MVC - migration, modelo y controlador

Con Artisan es posible **crear un migration, modelo y controlador de recursos simultáneamente** (pasos 3, 4 y 5).

Por ejemplo, al crear el modelo se puede indicar que añada controlador y migration:

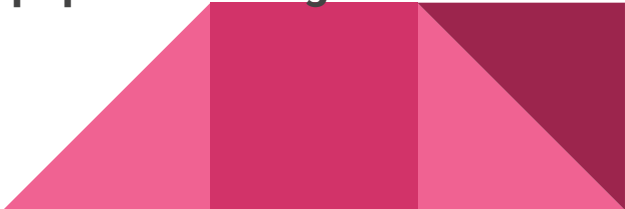
```
php artisan make:model Product --migration --controller --resource
```

El anterior comando generará los siguientes ficheros:

- **migration:** database/migrations/xxxx_xx_xx_create_products_table.php (x es la fecha de creación)
 - **Modelo:** app/Models/Product.php
 - **Controlador:** app/Http/Controllers/ProductController.php
- 

Pasos para generar MVC - migration

Los ficheros de **migration**, **modelo** y **controlador de recursos** creados con artisan NO están completos. Falta:

- **migration** - database/migrations/xxxx_xx_xx_create_products_table.php:
 1. Completar las **acciones del metodo *up()*** introduciendo el esquema de la tabla a crear. Si Laravel ha interpretado correctamente que se trata de una nueva tabla, bastará con indicar las columnas de la tabla
 2. Completar las **acciones del metodo *down()***: debe ser opuesto a lo que realiza *up()*. En este caso, drop de la nueva tabla (posiblemente Laravel ya lo ha rellenado).
 3. Crear la tabla en base de datos ejecutando el comando: **php artisan migrate**
- 

Pasos para generar MVC - Modelo

Los ficheros de **migration, modelo y controlador de recursos** creados con artisan NO están completos. Falta:

- **Modelo:** app/Models/Product.php:
 - Si tenemos un **nombre de tabla que no sigue el convenio**; añadir a la clase Modelo creada el atributo ***protected \$table = 'Nombre_tabla'***
 - Si tenemos una **primary key con nombre de columna diferente de 'id'** (no sigue el convenio), añadir a la clase Modelo creada el atributo ***protected \$primaryKey = 'Nuestro_id'***
 - Si se deben **permitir crear o editar registros de forma masiva**; (pe. pasando un array o emplear el método Product::create()), definir el atributo ***protected \$fillable = ['columna1', columna2];***

Pasos para generar MVC - controlador de recursos

Los ficheros de **migration, modelo y controlador de recursos** creados con artisan NO están completos. Falta:

- **Controlador:** app/Http/Controllers/ProductController.php:
 1. Asegurar que el controlador hace **uso del Modelo**. Laravel incluye este use si se crea en conjunto.
 2. **Completar los 7 métodos RESTful** para conseguir CRUD con nuestra aplicación web.
 3. Incluir el controlador en routes/web.php y **crear las rutas** (paso 6). Estas rutas se pueden definir automáticamente indicando *Route::resource()*. Por ejemplo:

```
use App\Http\Controllers\ProductController;  
Route::resource('products', ProductController::class);
```



Recordatorio: Controladores de Recursos

Los controladores de recursos están orientados a ofrecer **servicios CRUD** sobre datos persistentes. Por convenio, contiene 7 métodos (llamados RESTful) que deben realizar tareas específicas sobre registros:

- **index()** - Mostrar listado de registros
- **show()** - Mostrar detalles de un registro
- **create()** - Carga formulario para crear un registro
- **store()** - Recibe POST de formulario con datos de un nuevo registro y guarda en base de datos
- **edit()** - Carga formulario para modificar un registro
- **update()** - Recibe PUT de formulario con datos de un registro existente y actualiza en base de datos
- **destroy()** - Recibe DELETE y elimina un registro de base de datos

Nota: create() / store(), edit() / update() trabajan en conjunto.



Recordatorio: Rutas del Controlador de Recursos

Solicitud	URI	Método	Route Name
GET	/products	index	products.index
GET	/products/create	create	products.create
POST	/products	store	products.store
GET	/products/{product}	show	products.show
GET	/products/{product}/edit	edit	products.edit
PUT/PATCH	/products/{product}	update	products.update
DELETE	/products/{product}	destroy	products.destroy

{product} representa un paso de parámetro al método del controlador llamado.

Route::resource('products', ProductController::class) crea internamente las rutas definidas en la tabla, con los nombres indicados.

Ejemplo MVC - Gestión de productos (products)

Pasos para generar MVC - migration

- Vía comando, ejecutar:

```
php artisan make:model Product --migration --controller --resource
```

- Dentro del migration creado (X_create_products_table), completar el método *up()*:

```
Schema::create('products', function (Blueprint $table) {  
    $table->id();  
    $table->string('nombre');  
    $table->decimal('precio', 8, 2);  
    $table->timestamps();  
});
```

Nota: método *down()* ya está creado con drop de la tabla products.

- Vía comando, ejecutar: *php artisan migrate*

Pasos para generar MVC - modelo


Para poder trabajar con **inserciones o actualizaciones masivas en bases de datos** (pasar inserts y updates de múltiples registros a Laravel **vía arrays** y métodos `::create()` y `::update()`), conviene añadir el atributo protected `$fillable` al modelo `Producto.php`:

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class Product extends Model {

    protected $fillable = ['nombre','precio'];

}
```



Pasos para generar MVC - controlador


- En `app/Http/Controllers/ProductController.php`, definir los métodos REST:

```
use App\Models\Product;  
use Illuminate\Http\Request;
```

```
class ProductController extends Controller {
```

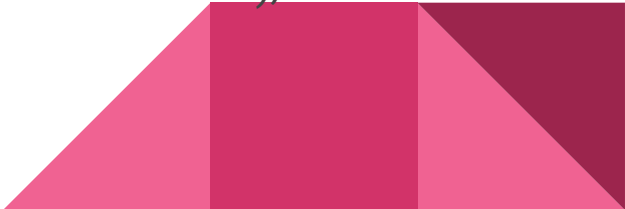
```
    public function index() {  
        $productos = Product::all();      // equivale a: select * from products  
        return view('products.index', ['productos' => $productos]); // ALTERNATIVA: compact('productos')  
    }
```

```
    public function show($id) {  
        $producto = Product::findOrFail($id);    // Localiza en base de datos un registro por su id  
        //dd($producto);                        // para debuggear - se puede eliminar (recordatorio)  
        return view('products.show', ['producto' => $producto]);  
    }
```



Pasos para generar MVC - controlador (continuación)

```
public function create() {  
    return view('products.create');           // devuelve vista create.blade.php  
}  
  
public function store(Request $request) {  
    $request->validate([                       // validación del formulario en el servidor (revisa solicitud entrante)  
        'nombre' => 'required',               // debe haber dato con clave nombre  
        'precio' => 'required|numeric'        // debe haber dato con clave precio Y debe ser numérico  
    ]);                                       // Localiza en base de datos un registro por su id  
    Product::create($request->all());         // Update en base de datos, "all()" pasa los inputs en array asociativo  
    // Redirección a products/index.blade.php con dato en session, clave "success"  
    return redirect()->route('products.index')->with('success', '¡Producto creado!');  
}
```




Pasos para generar MVC - controlador (continuación)

```
public function edit($id) {  
    $producto = Product::findOrFail($id);  
    return view('products.edit', ['producto' => $producto]);  
}  
  
public function update(Request $request, $id) {  
    $request->validate([  
        'nombre' => 'required',           // validación del formulario en el servidor (revisa solicitud entrante)  
                                           // debe haber dato con clave nombre  
        'precio' => 'required|numeric'    // debe haber dato con clave precio Y debe ser numérico  
    ]);  
                                           // Si validate falla, devuelve error (y recarga última vista)  
    $producto = Product::findOrFail($id); // Localiza en base de datos un registro por su id  
    $producto->update($request->all());    // Update en base de datos, "all()" pasa los inputs en array asociativo  
    return redirect()->route('products.index')->with('success', '¡Producto actualizado!'); // Redirección  
}  
  
public function destroy($id) {  
    Product::findOrFail($id)->delete(); // Delete en base de datos del registro con id dada.  
    return redirect()->route('products.index')->with('success', '¡Producto eliminado!');  
}  
}
```

Pasos para generar MVC - Crear las Vistas (Blade Templates)

- Crear la **carpeta en resources/views dónde guardar las vistas**. En este caso: *resources/views/products/*
- Dentro de la nueva carpeta, crear las **vistas**:
 - ❑ **index.blade.php** - Para mostrar la lista de productos
 - ❑ **show.blade.php** - Para ver detalles de un producto existente
 - ❑ **create.blade.php** - Para mostrar formulario de creación de producto
 - ❑ **edit.blade.php** - Para mostrar formulario de modificación de un producto existente

Nota: La distribución de los formularios de create.blade.php y edit.blade.php debería ser la misma.



Pasos para generar MVC - Crear las Vistas - index

```
<h1>Lista de Productos</h1>
@if(session('success'))
    <p style="color: green;">{{ session('success') }}</p>
@endif

<a href="{{ route('products.create') }}">Agregar Producto</a>

@foreach($productos as $producto)
    <p>
        {{ $producto->nombre }} - ${{ $producto->precio }}
        <a href="{{ route('products.show', $producto->id) }}">Ver</a>
        <a href="{{ route('products.edit', $producto->id) }}">Editar</a>
        <form action="{{ route('products.destroy', $producto->id) }}" method="POST" style="display:inline;">
            @csrf
            @method('DELETE')
            <button type="submit">Eliminar</button>
        </form>
    </p>
@endforeach
```

Pasos para generar MVC - Crear las Vistas - show

```
<h1>{{ $producto->nombre }}</h1>
```

```
<p>Precio: ${{ $producto->precio }}</p>
```

```
<a href="{{ route('products.edit', $producto->id) }}">Editar</a>
```

```
<form action="{{ route('products.destroy', $producto->id) }}" method="POST">
```

```
    @csrf
```

```
    @method('DELETE')
```

```
    <button type="submit">Eliminar</button>
```

```
</form>
```

```
<a href="{{ route('products.index') }}">Volver</a>
```

Pasos para generar MVC - Crear las Vistas - create

```
<h1>Crear Producto</h1>
@if($errors->any())
    <ul>
        @foreach($errors->all() as $error)
            <li style="color: red;">{{ $error }}</li>
        @endforeach
    </ul>
@endif

<form action="{{ route('products.store') }}" method="POST">
    @csrf
    <label>Nombre:</label>
    <input type="text" name="nombre">
    <label>Precio:</label>
    <input type="text" name="precio">
    <button type="submit">Guardar</button>
</form>

<a href="{{ route('products.index') }}">Volver</a>
```

Pasos para generar MVC - Crear las Vistas - edit

```
<h1>Editar Producto</h1>
@if($errors->any())
    <ul>
        @foreach($errors->all() as $error)
            <li style="color: red;">{{ $error }}</li>
        @endforeach
    </ul>
@endif

<form action="{{ route('products.update', $producto->id) }}" method="POST">
    @csrf
    @method('PUT')
    <label>Nombre:</label>
    <input type="text" name="nombre" value="{{ $producto->nombre }}">
    <label>Precio:</label>
    <input type="text" name="precio" value="{{ $producto->precio }}">
    <button type="submit">Actualizar</button>
</form>

<a href="{{ route('products.index') }}">Volver</a>
```

Pasos para generar MVC - Definir las rutas

- En routes/web.php definir la siguiente ruta:

```
use App\Http\Controllers\ProductController;
```

```
Route::resource('products', ProductController::class);
```

Una vez completados todos los puntos anteriores, **ya está** creado un servicio CRUD que sigue el patrón de diseño Modelo-Vista-Controlador en Laravel.




Añadir Login al MVC - Breeze

Para asegurar el **acceso al servicio creado mediante login** establecido por el paquete **Breeze**, basta con modificar la ruta al controlador de recursos **añadiendo un middleware de “auth”** sobre la agrupación de `route::resource()`:

// Las 7 rutas del controlador de recursos están protegidas con middleware 'auth'

```
Route::middleware(['auth']->group(function () {  
    Route::resource('products', ProductController::class);  
});
```

// Requiere rutas de autenticación Breeze, ya incluido en web.php
`require __DIR__.'/auth.php';`



Añadir Login al MVC - Breeze (alternativa)

Otra forma para **asegurar que los métodos CRUD quedan protegidos detrás del login de Breeze** es definir el middleware en el constructor del controlador de recursos (productoController.php):

```
public function __construct() {  
  
    $this->middleware('auth');  
  
}
```



Añadir Login al MVC - Breeze - Añadir enlaces

Dado que ahora se requiere autenticación, conviene **añadir los enlaces a iniciar sesión** y registrarse al principio de la vista index.blade.php para que los clientes puedan realizar login:

```
@if(Auth::check())  
    <p>Bienvenido, {{ Auth::user()->name }} | <a href="{{ route('logout') }}"  
    onclick="event.preventDefault(); document.getElementById('logout-form').submit();">  
    Cerrar sesión  
</a></p>  
    <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">  
    @csrf  
    </form>  
@else  
    <a href="{{ route('login') }}">Iniciar sesión</a>  
    <a href="{{ route('register') }}">Registrarse</a>  
@endif
```

También es buena idea añadir este código al resto de vistas.



