



BLOG CON LARAVEL

Adrián Peña Carnero



20 DE ENERO DE 2025

IES AGL

Contenido

1. Configuración del proyecto	2
<i>Instalar Laravel</i>	2
2. Generación de modelos y migraciones	4
3. Creación de los controladores	7
4. Configuración de rutas	10
5. Creación de las Vistas	11
6. Estilos	14
7. Seeders	17
8. Pruebas	20

1. Configuración del proyecto

Instalar Laravel

Asegúrate de tener Composer instalado y crea un nuevo proyecto Laravel:

```
composer create-project laravel/laravel blog2
```

en mi caso he utilizado el commando: **laravel new – nombre Proyecto**

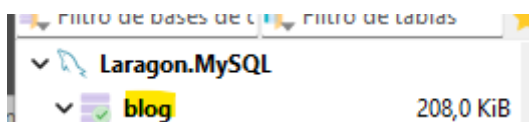
Configuración BD

En el archivo .env, configura tu conexión a la base de datos:

```
.env x PostFactory.php
APP_DEBUG=true
APP_TIMEZONE=UTC
APP_URL=http://localhost
APP_LOCALE=en
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=en_US
APP_MAINTENANCE_DRIVER=file
# APP_MAINTENANCE_STORE=database
PHP_CLI_SERVER_WORKERS=4
BCRYPT_ROUNDS=12
LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=blog
DB_USERNAME=root
DB_PASSWORD=
SESSION_DRIVER=database
SESSION_LIFETIME=120
SESSION_ENCRYPT=false
SESSION_PATH=/
SESSION_DOMAIN=null
BROADCAST_CONNECTION=log
```

Creación de la base de datos

Para crear la bd nos dirigimos a nuestro gestor de base de datos y creamos una base de datos con el nombre que queramos:



2. Generación de modelos y migraciones

Para generar los modelos con su migración correspondiente utilizaremos los siguientes comandos:

```
php artisan make:model Post -m
```

```
php artisan make:model Comment -m
```

Esto creará el modelo Post y el modelo Comment y dos migraciones en database/migrations.

Los archivos deberían los he configurado de esta manera:

Modelo **Post**:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;
13 references | 0 implementations
class Post extends Model
{
    use HasFactory;

    0 references
    protected $fillable = ['titulo', 'contenido'];

    1 reference | 0 overrides
    public function comments(): HasMany{
        return $this->hasMany(related: Comment::class);
    }
}
```

Modelo **Comment**:

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

2 references | 0 implementations
class Comment extends Model

{

    use HasFactory;

    0 references
    protected $fillable = ['autor', 'contenido', 'post_id'];

    0 references | 0 overrides
    public function post(): HasMany{
        return $this->hasMany(related: Post::class);
    }
}
```

Migración **Post**:

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'posts', callback: function (Blueprint $table): void {
            $table->id();
            $table->string(column: 'titulo');
            $table->string(column: 'contenido');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists(table: 'posts');
    }
};
```

Migración **Comments**:

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'comments', callback: function (Blueprint $table): void {
            $table->id();
            $table->unsignedBigInteger(column: 'post_id');
            $table->string(column: 'autor');
            $table->string(column: 'contenido');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists(table: 'comments');
    }
};
```

Para ejecutar las migraciones introduciremos el siguiente comando:

php artisan migrate

te migrara las tablas donde hayas enlazado la bd , si no existe te sugerirá crearla, y también te sugerirá si deseas crear las tablas por defecto de laravel.

3. Creación de los controladores

Generar el controlador

Usa el siguiente comando para generar un controlador con recursos:

php artisan make:controller PostController --resource

php artisan make:controller CommentController --resource

Editamos **PostController**:

```
namespace App\Http\Controllers;

use App\Models\Post;
use Illuminate\Http\Request;

3 references | 0 implementations
class PostController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    1 reference | 0 overrides
    public function index(): View
    {
        $posts = Post::latest()->paginate(perPage: 5);
        return view('posts.index', data: compact('posts'));
    }

    /**
     * Show the form for creating a new resource.
     */
    0 references | 0 overrides
    public function create(): View
    {
        return view('posts.create');
    }

    /**
     * Store a newly created resource in storage.
     */
    0 references | 0 overrides
    public function store(Request $request): RedirectResponse
    {
        $request->validate(rules: [
            'titulo' => 'required|max:255',
            'contenido' => 'required'
        ]);

        Post::create(attributes: $request->all());
        return redirect()->route('posts.index')->with(key: 'success', value: 'Post creado exitosamente');
    }

    /**
     * Display the specified resource.
     */
    0 references | 0 overrides
    public function show(Post $post): View
    {
        return view('posts.show', data: compact('post'));
    }

    /**
     * Show the form for editing the specified resource.
     */
    0 references | 0 overrides
    public function edit(Post $post): View
    {
        return view('posts.edit', data: compact('post'));
    }
}
```

```
/**
 * Update the specified resource in storage.
 */
0 references | 0 overrides
public function update(Request $request, Post $post): RedirectResponse
{
    $request->validate(rules: [
        'titulo' => 'required|max:255',
        'contenido' => 'required' ,
    ]);

    $post->update(attributes: $request->all());
    return redirect()->route(route: 'posts.index')->with(key: 'sucess',value: 'Post actualizado exitosamente');
}

/**
 * Remove the specified resource from storage.
 */
0 references | 0 overrides
public function destroy(Post $post): RedirectResponse
{
    $post->delete();
    return redirect()->route(route: 'posts.index')->with(key: 'success',value: 'Post eliminado exitosamente');
}
```

Editamos **CommentController**:

```

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Post;
use App\Models\Comment;

2 references | 0 implementations
class CommentController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    0 references | 0 overrides
    public function index(): void
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     */
    0 references | 0 overrides
    public function create(): void
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     */
    1 reference | 0 overrides
    public function store(Request $request, Post $post): RedirectResponse
    {
        $validated = $request->validate(rules: [
            'autor' => 'required|string|max:255',
            'contenido' => 'required|string|min:5'
        ]);

        $post->comments()->create(attributes: $validated);

        return redirect()->route(route: 'posts.show', parameters: $post)->with(key: 'success',value: 'comentario añadido correctamente');
    }

    /**
     * Display the specified resource.
     */
    0 references | 0 overrides
    public function show(string $id): void
    {
    }
}

```

4. Configuración de rutas

Definir rutas en **routes/web.php**:

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PostController;
use App\Http\Controllers\CommentController;

Route::get(uri: '/', action: function () { View {
    return view(view: 'posts.index');
}});

Route::resource(name: 'posts',controller: PostController::class);

Route::post(uri: 'posts/{post}/comments',action: [CommentController::class,'store']->name(name: 'comments.store');

Route::get(uri: '/', action: [PostController::class, 'index']->name(name: 'posts.index');

```

5. Creación de las Vistas

Crea las vistas en resources/views/posts/:

Index (Listado de posts)

resources/views/posts/index.blade.php:

```
@extends('layout')

@section('title',content: 'Listado de Posts')

@section('content')

<h1>Listado de Posts</h1>
<a href="{{ route(name: 'posts.create') }}" class="button">Crear nuevo post</a>
<ul>

@foreach ($posts as $post)
    <li>
        <a href="{{ route(name: 'posts.show', parameters: $post) }}">{{ $post->titulo }}</a>
        <a href="{{ route(name: 'posts.edit', parameters: $post) }}">Editar</a>

        <form action="{{ route(name: 'posts.destroy', parameters: $post) }}" method="POST">
            @csrf
            @method('DELETE')
            <button type="submit">Eliminar</button>
        </form>
    </li>
@endforeach

</ul>

<div class="pagination">
    {{ $posts->links() }}
</div>

@endsection
```

Create (Formulario de creación)

resources/views/posts/create.blade.php:

```

@extends(view: 'layout') <!--Extendemos de una vista padre-->

@section(section: 'title', content: 'Crear Post') <!--En la sección titulo el contenido se va a llamar Crear Post-->

@section(section: 'content') <!--Le indicamos que en la sección content se va a inyectar este código-->

<h1>Crear Post</h1>

<form action="{{route(name: 'posts.store')}}" method="POST">

    @csrf
    <label for="titulo">Titulo:</label>
    <input type="text" name="titulo" id="titulo" required>
    <label for="contenido">Contenido:</label>
    <textarea name="contenido" id="contenido" required></textarea>
    <button type="submit">Guardar</button>

</form>

@endsection

```

Show (Ver detalles de un post)

resources/views/posts/show.blade.php:

```

@extends(view: 'layout')

@section(section: 'title', content: $post->titulo)

@section(section: 'content')

<h1>{{ $post->titulo }}</h1>
<p>{{ $post->contenido }}</p>

<h2>Comentarios</h2>

<ul>
    @foreach ($post->comments as $comment)
        <li>
            {{ $comment->autor }} dijo:
            <p>{{ $comment->contenido }}</p>
            <small>Publicado el {{ $comment->created_at->format('d/m/Y H:i')}}</small>
        </li>
    @endforeach
</ul>

<h3>Añadir comentario</h3>

@if (session(key: 'success'))
    <p class="success">{{ session(key: 'success') }}</p>
@endif

<form action="{{ route(name: 'comments.store', parameters: $post) }}" method="POST">
    @csrf
    <label for="autor">Nombre:</label>
    <input type="text" name="autor" id="autor" required>
    <label for="contenido">Comentario:</label>
    <textarea name="contenido" id="contenido" required></textarea>
    <button type="submit">Añadir un comentario</button>
</form>

<a href="{{ route(name: 'posts.index') }}" class="button">Volver al listado</a>

@endsection

```

Edit (Formulario de edición)

resources/views/posts/edit.blade.php:

```
@extends('layout')

@section('title',content: 'Editar Post')

@section('content')

<h1>Editar Post</h1>

<form action="{{route(name: 'posts.update',parameters: $post)}}" method="POST"> <!--Obtenemos el id del post pasandole como parametro la ruta-->
@csrf
@method('PUT')
<label for="titulo">Titulo:</label>
<input type="text" name="titulo" id="titulo" value="{{ $post->titulo }}" required>
<label for="contenido">Contenido:</label>
<textarea name="contenido" id="contenido" required>{{ $post->contenido }}</textarea>
<button type="submit">Actualizar</button>

</form>

@endsection
```

Layout de la que se extienden todas las clases anteriores:

resources/views/layout.blade.php:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@yield(section: 'title', default: 'Blog')</title>
    <link rel="stylesheet" href="{{asset(path: 'css/styles.css')}}">
</head>

<body>
    <div class="container">
        @yield(section: 'content')
    </div>
</body>

</html>
```

6. Estilos

He modificado los estilos para generar una apariencia más llamativa:

```
/* General */
body {
  font-family: 'Comic Sans MS', cursive, sans-serif;
  margin: 0;
  padding: 0;
  background: radial-gradient(circle, #ff6ec7, #ff7b00);
  color: #fff;
  text-align: center;
  animation: backgroundAnim 3s infinite alternate;
}

@keyframes backgroundAnim {
  0% {
    background: radial-gradient(circle, #ff6ec7, #ff7b00);
  }

  100% {
    background: radial-gradient(circle, #00bfff, #ff1493);
  }
}

/* Cabeceras */
h1,
h2,
h3 {
  font-family: 'Impact', sans-serif;
  text-transform: uppercase;
  color: #ff0;
  text-shadow: 2px 2px 8px rgba(0, 0, 0, 0.5);
  margin-top: 20px;
}

/* Enlaces */
a {
  color: #ff0;
  font-weight: bold;
  text-decoration: none;
  text-transform: uppercase;
  letter-spacing: 2px;
  border: 2px solid #fff;
  padding: 5px 15px;
  transition: transform 0.3s ease, color 0.3s ease;
}

a:hover {
  transform: scale(1.1);
  color: #ff00ff;
  text-decoration: underline;
  background-color: #fff;
  color: #ff1493;
}

/* Contenedor principal */
.container {
  width: 85%;
  margin: 20px auto;
  background-color: #222;
  padding: 40px;
  border-radius: 15px;
  box-shadow: 0 0 30px rgba(0, 0, 0, 0.7);
  box-sizing: border-box;
  border: 3px solid #ff00ff;
}
```



```
/* Formularios */
form {
  margin-top: 30px;
  padding: 20px;
  background: linear-gradient(to right, #ff00ff, #00bfff);
  border-radius: 10px;
  box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
}

label {
  font-size: 18px;
  color: #ff1493;
  font-weight: bold;
  display: block;
  margin-bottom: 10px;
}

input[type="text"],
textarea {
  width: 100%;
  padding: 15px;
  margin-bottom: 20px;
  border: 3px solid #ff0;
  border-radius: 8px;
  background-color: #222;
  color: #fff;
  font-size: 16px;
  font-family: 'Comic Sans MS', cursive, sans-serif;
  box-sizing: border-box;
}

input[type="text"]:focus,
textarea:focus {
  border-color: #ff1493;
  background-color: #333;
}

/* Botones */
button {
  background-color: #ff00ff;
  color: white;
  border: none;
  padding: 15px 25px;
  border-radius: 8px;
  cursor: pointer;
  font-size: 18px;
  letter-spacing: 1px;
  transition: background-color 0.3s ease, transform 0.3s ease;
}

button:hover {
  background-color: #ff1493;
  transform: scale(1.05);
}

/* Lista de contenidos */
ul {
  list-style-type: none;
  padding: 0;
  font-size: 18px;
  line-height: 2;
}
```

```
li {  
  margin-bottom: 15px;  
  padding: 20px;  
  background-color: #444;  
  border-radius: 10px;  
  border: 2px solid #ff1493;  
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);  
}  
  
/* Mensajes de éxito */  
.success {  
  color: #00ff00;  
  font-weight: bold;  
  font-size: 20px;  
  text-shadow: 2px 2px 8px rgba(0, 0, 0, 0.7);  
  margin-top: 20px;  
}  
  
svg.w-5.h-5 {  
  width: 20px;  
  height: 20px;  
}  
  
svg.w-5.h-5 {  
  fill: #ff6f61;  
}  
  
svg.w-5.h-5 {  
  margin-right: 5px;  
  margin-left: 5px;  
}
```

Con estos estilos te quedara una web muy llamativa.

7. Seeders

Seeders

Los **seeders** en Laravel se utilizan para llenar tu base de datos con datos de ejemplo o prueba. Esto es muy útil cuando estás desarrollando o probando una aplicación.

Aquí tienes una explicación paso a paso para crear un **seeder** que genere muchos posts automáticamente:

1. Crear un Seeder

Genera un nuevo seeder usando el comando Artisan:

php artisan make:seeder PostSeeder

Esto creará un archivo llamado **PostSeeder.php** en el directorio database/seeds.

2. Configurar el Seeder

Edita el archivo **PostSeeder.php** para que cree posts en la base de datos:

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Post;

0 references | 0 implementations
class PostSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    0 references | 0 overrides
    public function run(): void
    {
        Post::factory()->count(50)->create();
    }
}
```

3. Crear una Fábrica para Generar Posts

Las **fábricas** en Laravel permiten generar datos ficticios. Si no tienes una fábrica para Post, crea una con el comando:

php artisan make:factory PostFactory --model=Post

Esto generará un archivo llamado PostFactory.php en el directorio database/factories.

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Post>
 */
0 references | 0 implementations
class PostFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    0 references | 0 overrides
    public function definition(): array
    {
        return [
            'titulo' => $this->faker->sentence(),
            'contenido' => $this->faker->paragraph(nbSentences: 3, variableNbSentences: true),
        ];
    }
}
```

Laravel usa la biblioteca FakerPHP para generar datos ficticios, como textos, nombres y fechas.

4. Ejecutar el Seeder

Ejecuta el seeder para llenar tu base de datos con posts:

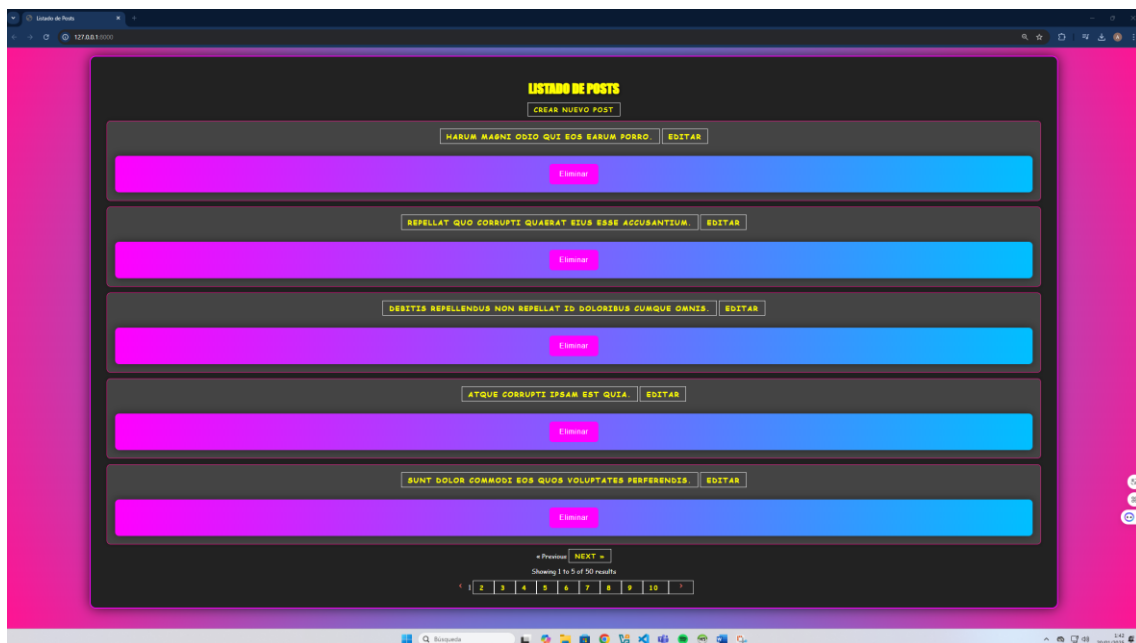
php artisan db:seed --class=PostSeeder

Esto generará 50 posts en la base de datos.

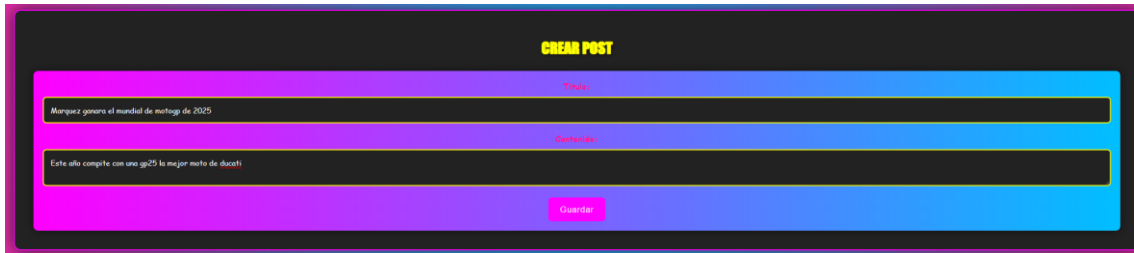
8. Pruebas

Página de inicio:

Como podemos observar se han añadido correctamente los posts con los seeders:



Creación de post:



CREAR POST

Titulo:

Marquez gana el mundial de motogp de 2025

Contenido:

Este año compete con una gp25 la mejor moto de ducati

Guardar

Como podemos ver se ha añadido correctamente:



LISTADO DE POSTS

CREAR NUEVO POST

MARQUEZ GANARA EL MUNDIAL DE MOTOGP DE 2025 EDITAR

Eliminar

HABUM MAGNI ODIO QUI EOS EABUM PORRO. EDITAR

Eliminar

REPELLAT QUO CORRUPTI QUARRAT EIUS ESSE ACCUSANTIUM. EDITAR

Eliminar

DEBITIS REPELLENDUS NON REPELLAT ID DOLORIBUS CUMQUE OMNIS. EDITAR

Eliminar

ATQUE CORRUPTI IPSAM EST QUIA. EDITAR

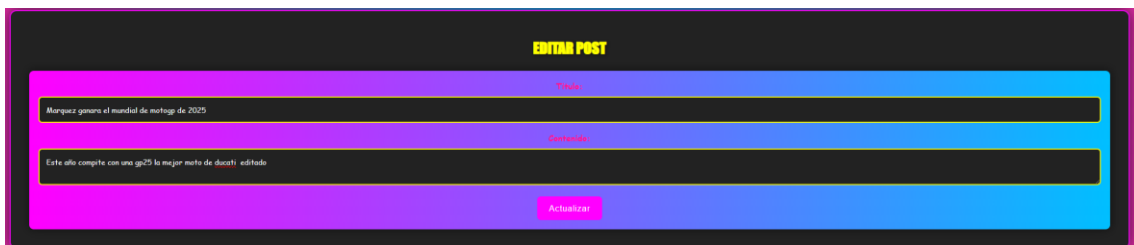
Eliminar

« Previous NEXT »

Showing 1 to 5 of 51 results

< 2 3 4 5 6 7 8 9 10 11 >

Ahora vamos a editar este post:



EDITAR POST

Titulo:

Marquez gana el mundial de motogp de 2025

Contenido:

Este año compete con una gp25 la mejor moto de ducati: editado

Actualizar

Se ha editado correctamente:

MARQUEZ GANARA EL MUNDIAL DE MOTOCP DE 2025

Este año compete con uno gp25 la mejor moto de Ducati [exhibida](#)

COMENTARIOS

añade comentarios

Nombre:

Comentario:

Añadir un comentario

VOLVER AL LISTADO

Ahora vamos a comentar en el post:

Nombre:

Comentario:

como modo de post? :o

Añadir un comentario

VOLVER AL LISTADO

Se añade correctamente:

MARQUEZ GANARA EL MONDIAL DE MOTOGP DE 2025

Este año compete con una gp25 la mejor moto de ducati editada

COMENTARIOS

edición dija:

como mala el post ha

Publicado el 20/01/2025 00:47

¡Bueno comentario

comentario añadido correctamente

Responde:

Compartir:

Añadir un comentario

VOLVER AL LISTADO

Ahora vamos a eliminar el post:

LISTADO DE POSTS

CREAR NUEVO POST

HARUM MAGNI ODIO QUI EOS SARUM PORRO

EDITAR

Eliminar

REPELLAT QUO CORRUPTI QUAEAT EIUS ESSE ACCUSANTIUM

EDITAR

Eliminar

DEBITIS REPELLENDUS NON REPELLAT ID DOLORIBUS CUMQUE OMNIS

EDITAR

Eliminar

ATQUE CORRUPTI IPSAM EST QUIA

EDITAR

Eliminar

SUNT DOLOR COMMUDI EOS QUOS VOLUPTATES PERFERENDIS

EDITAR

Eliminar

« Previous

NEXT »

Showing 1 to 5 of 50 results

1

2

3

4

5

6

7

8

9

10

>