Escrituras de

Prueba en:

Laravel

Creación de proyecto

C:\

C:\> laravel **new <name>** #Ejecuta este
C:\> echo “Debes elegir la configuracion que necesites para tu proyecto ”
C:\> echo “Modifica tu archivo .env a tus necesidades”
C:\> php artisan **migrate** #Ejecuta este
C:\> echo “Para hacer el comando de arriba debes estar en la carpeta del projecto
C:\> echo “Ya tendrias tu proyecto inicial con su respectiva conexión”

PHPUnit?

Al momento de instalar laravel se incluye asi que solo debemos comprobar si lo trae, de lo contrario debemos instalarlo

C:\

C:\> **./vendor/bin/phpunit** **--version** **#Ejecuta esto**
C:\> echo “Debe salir esto: ⇩”
PHPUnit xx.x.x by Sebastian Bergmann and contributors.
C:\> echo “Entonces si tienes instalado PHPUnit, si no pues instalalo con: ”
C:\> **composer** **require** **--dev** **phpunit/phpunit ^x** **#Ejecuta esto**
C:\> echo “Acuérdate que todo debe ser en la carpeta del proyecto”

Estructura basica de test en PHPUnit

Cada prueba en Laravel se encuentra dentro del directorio tests/ y puede estar en Feature/ o Unit/, dependiendo del tipo de prueba.

```
use Tests\TestCase;     // TestCase proporciona metodos de prueba

class ExampleTest extends TestCase //Debe comenzar con test_ o estar anotado con @test.
{

        public function test_basic_example() //Debe comenzar con test_ o estar con @test.
        {

                $this->assertTrue(true); /*Se utilizan para comprobar resultados esperados, como
                                                assertTrue, assertEquals*/
        }

}
```

Pruebas unitarias

Las pruebas unitarias se enfocan en probar una sola unidad de código, como una función, método o clase, de forma aislada. No interactúan con la base de datos ni con otros componentes del sistema.

📌 **Ubicación: tests/Unit/**

✅ **Objetivo: Asegurar que cada unidad de código funcione correctamente por sí sola**

✅ **Ejemplo: Probar que un modelo devuelve un atributo correctamente.**

Codigo de ejemplo

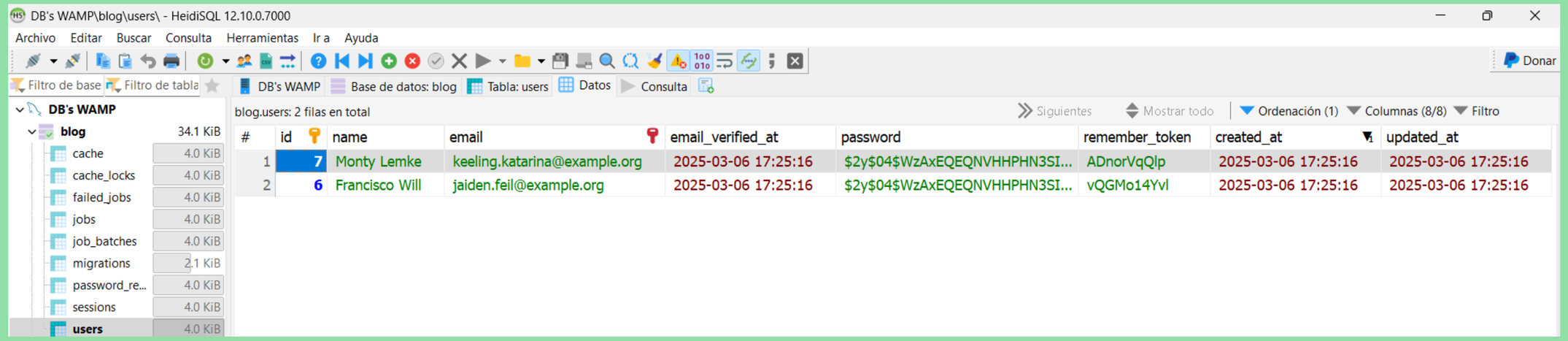
UserTest.php

tests > Unit > UserTest.php > UserTest > test_user_full_name

```php
<?php

namespace Tests\Unit;

use Tests\TestCase;
use App\Models\User;

class UserTest extends TestCase
{
    /**
     * Verificar que un usuario se crea correctamente.
     *
     * @return void
     */
    public function test_user_creation()
    {
        // Creamos un usuario de prueba
        $user = User::factory()->create();

        // Verificamos que el usuario se ha creado correctamente
        $this->assertDatabaseHas('users', [
            'email' => $user->email,
        ]);
    }

    /**
     * Verificar que un usuario tiene un nombre completo.
     *
     * @return void
     */
    public function test_user_full_name()
    {
        // Creamos un usuario de prueba usando el factory
        $user = User::factory()->create([
            'name' => 'Monty Lemke',
        ]);

        // Verificamos que el nombre completo sea igual al nombre del usuario
        $this->assertEquals('Monty Lemke', $user->name);
    }
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\wamp64\www\blog> php artisan test

   PASS  Tests\Unit\UserTest
   ✓ user creation                                                0.15s
   ✓ user full name                                               0.04s

   Tests:    2 passed (2 assertions)
   Duration: 0.32s
```



Pruebas de caracteristicas

Las pruebas de integración verifican que varios componentes del sistema funcionan bien juntos. Pueden incluir la interacción entre modelos, controladores, bases de datos y servicios.

📌 **Ubicación: tests/Feature/**

✅ **Objetivo: Probar que los módulos de la aplicación trabajan correctamente en conjunto.**

✅ **Ejemplo: Probar que un servicio de pedidos calcula correctamente el total con impuestos y se guarda en la base de datos.**

Codigo de ejemplo

PostControllerTest.php

tests > Feature > PostControllerTest.php > ...

```php
<?php
namespace Tests\Feature;

use App\Models\Post;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class PostControllerTest extends TestCase
{
    use RefreshDatabase; // Utiliza la funcionalidad de base de datos para refrescar la base de datos entre pruebas

    /**
     * Prueba de la ruta 'posts.index' que muestra la lista de posts.
     */
    public function test_index()
    {
        // Realiza una solicitud GET a la ruta 'posts.index' (controlador PostController, método index)
        $response = $this->get(route('posts.index'));

        // Verifica que la respuesta sea exitosa (código de estado 200)
        $response->assertStatus(200);

        // Verifica que la vista mostrada sea 'posts.index' (la vista que lista los posts)
        $response->assertViewIs('posts.index');
    }

    /**
     * Prueba de la ruta 'posts.create' que muestra el formulario para crear un nuevo post.
     */
    public function test_create()
    {
        // Realiza una solicitud GET a la ruta 'posts.create' (controlador PostController, método create)
        $response = $this->get(route('posts.create'));

        // Verifica que la respuesta sea exitosa (código de estado 200)
        $response->assertStatus(200);

        // Verifica que la vista mostrada sea 'posts.create' (la vista que contiene el formulario para crear un post)
        $response->assertViewIs('posts.create');
    }

```

PostControllerTest.php

tests > Feature > PostControllerTest.php > ...

```php
class PostControllerTest extends TestCase

    /**
     * Prueba de la ruta 'posts.store' que almacena un nuevo post en la base de datos.
     */
    public function test_store()
    {
        // Define los datos para el nuevo post
        $postData = [
            'title' => 'Test Post',  // Título del post
            'body' => 'This is the body of the post.',  // Cuerpo del post
        ];

        // Realiza una solicitud POST a la ruta 'posts.store' (controlador PostController, método store)
        // Se envían los datos del nuevo post
        $response = $this->post(route('posts.store'), $postData);

        // Verifica que la respuesta sea una redirección a la ruta 'posts.index' (donde se listan los posts)
        $response->assertRedirect(route('posts.index'));

        // Verifica que el mensaje de éxito se haya agregado a la sesión
        $response->assertSessionHas('success', 'Post was created');

        // Verifica que los datos del nuevo post estén presentes en la base de datos
        $this->assertDatabaseHas('posts', $postData);
    }
}
```

```
PS C:\wamp64\www\project1> php artisan test --filter PostControllerTest

   PASS  Tests\Feature\PostControllerTest
  ✓ index                                                              0.27s
  ✓ create                                                             0.03s
  ✓ store                                                              0.04s

  Tests:    3 passed (8 assertions)
  Duration: 0.55s
```