Pruebas

Clases

Clase actividades

Tabla antes de los cambios:



Crear:

Método:

```
// Crear una nueva actividad 
public static function crearActividad($titulo, $descripcion, $duracion, $ubicacion, $precio, $categoria, $fecha, $hora) {

// Validaciones
| (empty ($titulo) || empty ($descripcion) || $duracion <= 0 || $precio < 0 || empty ($ubicacion) || empty ($categoria) || empty ($fecha) || empty ($hora)) {

return false;
}

// Comprobar si la actividad ya existe
| (!sel:: comprobarActividadExistente(null, $titulo, $fecha, $hora)) {

// Si no existe, proceder con la creación
$conexion = Conexion:: conectar();
$query = "INSERT INTO actividades (titulo, descripcion, duracion, ubicacion, precio, categoria, fecha, hora)
$stmt = $conexion->prepare($query);
$stmt->bindParam("titulo", $titulo);
$stmt->bindParam("titulo", $titulo);
$stmt->bindParam("titulo", $titulo);
```

```
// Crear una actividad correctamente
public function testCrearActividad()
{

$resultado = Actividades:: crearActividad("Spinning", "Clase activa", 2, "Sala A", 20, "Bienestar", "2024-12-10", "10:00");
$this->assertTrue($resultado); // Se espera que la actividad sea creada con éxito
}

// Intentar crear actividad con datos inválidos
public function testCrearActividadConDatosInvalidos()
{

$resultado = Actividades:: crearActividad("", "", -1, "", -5, "", "2024-12-10", "10:00");
$this->assertFalse($resultado); // Se espera que falle debido a los datos no válidos
}
```

Editar:

Método:

Prueba:

```
// Editar una actividad existente

public function testEditarActividad()
{

// Editar la actividad

$resultado = Actividades:: editarActividad(30, "Clase de Natación", "Clase más intensa", 2, "Sala B", 25, "Bienestar", "2024-12-10", "12:00");

$this->assertTrue($resultado); // Se espera que la edición sea exitosa
}

// Intentar editar una actividad inexistente

public function testEditarActividad(nexistente())
{

$resultado = Actividades:: editarActividad(999, "Clase de Yoga", "Clase relajante", 2, "Sala A", 20, "Bienestar", "2024-12-10", "10:00");

$this->assertFalse($resultado); // Se espera que falle, no existe la actividad con ID 999
}
```

Eliminar:

Método:

```
// Eliminar una actividad
public static function eliminarActividad($id) {
    // Comprobar si la actividad con el ID existe
    if (!self:: comprobarActividadExistente($id)) {
        return false; // La actividad no existe
    }

    // Procedemos a eliminar la actividad
    $conexion = Conexion:: conectar();
    $query = "DELETE FROM actividades WHERE id = :id";
    $stmt = $conexion->prepare($query);
    $stmt->bindParam(':id', $id);
    return $stmt->execute();
}
```

Prueba:

```
// Eliminar una actividad existente
public function testEliminarActividad()
{
    // Crear una actividad
    $resultado = Actividades:: eliminarActividad(31);
    $this->assertTrue($resultado); // Se espera que la actividad sea eliminada correctamente
}

// Intentar eliminar una actividad inexistente
public function testEliminarActividad(nexistente())
{
    $resultado = Actividades:: eliminarActividad(999);
    $this->assertFalse($resultado); // Se espera que falle, no existe la actividad con ID 999
}
```

Obtener todas las actividades:

Método:

```
// Obtener todas las actividades
public static function obtenerActividades() {
    $conexion = Conexion:: conectar();
    $query = "SELECT * FROM actividades";
    $stmt = $conexion->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

Obtener por id:

Método:

```
// Obtener una actividad por ID
public static function obtenerActividadPorld($id) {
    $conexion = Conexion:: conectar();
    $query = "SELECT * FROM actividades WHERE id = :id";
    $stmt = $conexion->prepare($query);
    $stmt->bindParam(':id', $id);
    $stmt->execute();
    $actividad = $stmt->fetch(PDO::FETCH_ASSOC);
    if ($actividad) {
        return $actividad;
    }
    return false;
}
```

```
// Obtener actividad por ID
public function testObtenerActividadPorId()
{

$actividad = Actividades::obtenerActividadPorId(30);

// Verifica que la actividad devuelta tenga el id 1
$this->assertNotFalse($actividad); // Se espera que no devuelva false
$this->assertEquals(30, $actividad['id']); // Se espera que devuelva la actividad con ID 1
}
```

Ejecución pruebas:



Como se puede comprobar se ha creado el registro 32 (spinning), se ha eliminado el 31 (clase de pilates) y se ha editado el 30 a clase de natación

Clase categorías

Tabla antes de los cambios:



Crear:

Método:

```
public static function crearCategoria ($nombre) {

// Validaciones

if (empty ($nombre)) {

return false; // Nombre vacio no es válido
}

// Comprobar si la categoria ya existe

if (self::comprobarCategoriaExistente(null, $nombre)) {

return false; // La categoria ya existe
}

// Si no existe, proceder con la creación

$conexion = Conexion::conectar();

$stmt = $conexion->prepare("INSERT INTO categorias (Nombre) VALUES (:nombre)");

$stmt->bindParam(':nombre', $nombre);

return $stmt->execute();
}
```

```
// Test para la creación de una categoría
public function testCrearCategoria()
{
    // Crear la categoría
    $resultado = Categorias::crearCategoria('test4');

    // Verificar que la categoría se haya creado correctamente
    $this->assertTrue($resultado);
}

// Test para intentar crear una categoría con un nombre duplicado
public function testCrearCategoriaDuplicada()
{
```

Editar:

Método:

```
// Modificar una categoria existente
public static function modificarCategoria($id, $nombre) {
    // Validaciones
    if (empty($nombre)) {
        return false; // Nombre vacio no es válido
    }

    // Comprobar si la categoria con el ID existe
    if (tself::comprobarCategoriaExistente($id)) {
        return false; // La categoria no existe
    }

    // Procedemos a modificar la categoria
    $conexion = Conexion::conectar();
    $stmt = $conexion->prepare("UPDATE categorias SET nombre = :nombre WHERE id = :id");
    $stmt->bindParam(:id', $id);
    return; $stmt->execute();
}
```

```
// Test para modificar una categoria existente
public function testModificarCategoria()
{
    // Modificar la categoria
    $resultado = Categorias::modificarCategoria(35, 'Prueba categoria');
    // Verificar que la modificación fue exitosa
    $this->assertTrue($resultado);
}

// Test para intentar modificar una categoria que no existe
public function testModificarCategoriaNoExistente()
{
    $resultado = Categorias::modificarCategoria(999, 'Categorialnexistente');
    // Verificar que no se puede modificar una categoria inexistente
    $this->assertFalse($resultado);
}
```

Eliminar:

Método:

```
// Eliminar una categoria por ID
public static function eliminarCategoria($id) {
    // Comprobar si la categoria con el ID existe
    if (!self::comprobarCategoriaExistente($id)) {
        return false; // La categoria no existe
    }

    // Procedemos a eliminar la categoria
    $conexion = Conexion::conectar();
    $stmt = $conexion->prepare("DELETE FROM categorias WHERE id = :id");
    $stmt->bindParam(':id', $id);
    return $stmt->execute();
}
```

```
// Test para eliminar una categoria existente
public function testEliminarCategoria()
{
    // Eliminar la categoria
    $resultado = Categorias::eliminarCategoria(37);

    // Verificar que la categoria fue eliminada correctamente
    $this->assertTrue($resultado);
}

// Test para intentar eliminar una categoria que no existe
public function testEliminarCategoriaNoExistente()
{
    $resultado = Categorias::eliminarCategoria(999); // ID no existente
    // Verificar que no se puede eliminar una categoria inexistente
    $this->assertFalse($resultado);
}
```

Obtener todas las categorías:

Método:

```
// Obtener todas las categorias
public static function obtenerCategorias() {
    $conexion = Conexion::conectar();
    $stmt = $conexion->prepare("SELECT * FROM categorias");
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

Prueba:

```
// Obtener todas las categorias
public function testObtenerCategoria()
{
    $resultado = Categorias::obtenerCategorias();

// Verifica que se devuelvan exactamente dos actividades
    $this->assertCount(8, $resultado);
}
```

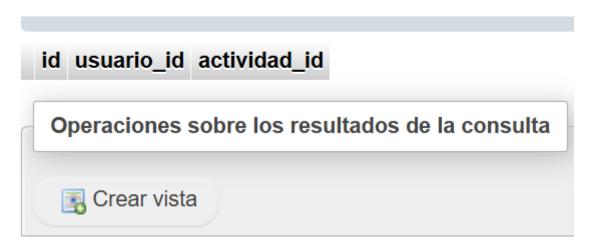
Ejecución pruebas:



Como se puede comprobar se ha creado el registro 38 (test4), se ha eliminado el 37 (test 3) y se ha editado el 35 a clase de natación

Clase favoritos

Tabla antes de los cambios:



Crear:

método:

```
// Agregar una actividad a los favoritos
public static function agregarFavorite ($usuario_id, $actividad_id)
{
    // Comprobar si el usuario y la actividad existen y son favoritos
    if (!seli::comprobarUsuarioExistente($usuario_id)||!seli::comprobarActividadExistente($actividad_id)|| seli::comprobarFavoritoExistente($usuario_id, $actividad_id)) {
        return false;
    }

    // Si no es favorito, proceder a agregarlo
$conexion = Conexion::conectar();
$query = "INSERT INTO favoritos (usuario_id, actividad_id) VALUES (:usuario_id, :actividad_id)";
$stmt = $conexion->prepare($query);
$stmt:->bindParam(':usuario_id, $usuario_id);
$stmt:->bindParam(':usuario_id, $usuario_id);
*stmt:->bindParam(':actividad_id', $actividad_id');
*return $stmt:->execute();
}
```

```
// Test para agregar un favorito correctamente
public function testAgregarFavorito()
{

// Agregar a favoritos
$resultado = Favoritos::agregarFavorito(5, 1);

// Verificar que se ha agregado correctamente
$this->assertTrue($resultado);
}

// Test para intentar agregar un favorito que ya existe
public function testAgregarFavoritoExistente()
{

// Intentar agregar el mismo favorito de nuevo
$resultado = Favoritos::agregarFavorito(5, 1);

// Verificar que no se ha agregado porque ya existe
$this->assertFalse($resultado);
}
```

Eliminar:

Método:

```
// Eliminar una actividad de los favoritos
public static function eliminarFavorito($usuario_id, $actividad_id)
{
    // Comprobar si el usuario y la actividad existen
    // Comprobar si el usuario y la actividad existen y son favoritos

    // (!sel::comprobarFavoritoExistente($usuario_id, $actividad_id)) {
    return false;
}

// Si es un favorito, proceder a eliminarlo
$conexion = Conexion::conectar();
$query = "DELETE FROM favoritos WHERE usuario_id = :usuario_id AND actividad_id = :actividad_id";
$stmt = $conexion->prepare($query);
$stmt->bindParam(':usuario_id', $usuario_id);
$stmt->bindParam(':actividad_id', $actividad_id);
return $stmt->execute();
}
```

Prueba:

```
public function testEliminarFavorito()
{

// Agregar a favoritos
Favoritos::agregarFavorito(5, 2);

// Eliminar el favorito
$resultado = Favoritos::eliminarFavorito(5, 2);

// Verificar que el favorito fue eliminado
$this->assertTrue($resultado);
}
```

Ver si una actividad está en favoritos:

Método:

```
// Comprueba si una actividad es favorita para un usuario
public static function esFavorito ($usuario_id, $actividad_id)
{
    // Comprobar si el usuario y la actividad existen
    if (!self::comprobarUsuarioExistente($usuario_id) || !self::comprobarActividadExistente($actividad_id)) {
        return false; // Si no existe el usuario o la actividad, no puede ser favorito
    }
    return self::comprobarFavoritoExistente($usuario_id, $actividad_id);
}
```

Prueba:

```
// Test para comprobar si una actividad es favorita
public function testEsFavorito()
{

// Verificar que ahora es favorito
$this->assertTrue(Favoritos::esFavorito(5, 1));
}
```

Resultado:



A partir de este punto voy a realizar menos comentarios en los métodos referente a la inserción, obtención y eliminación de datos, ya que es la misma metodología a la existente en las 3 clases vistas y mi proyecto cuenta con 5 clases más. Dicho esto prodigo con la clase comentarios:

Comentarios

Esta tabla al igual que la anterior estaba vacia antes de realizar las pruebas

Creación:

```
// Test para crear un comentario correctamente
public function testCrearComentario()
  $usuario_id = 5;
  $actividad id = 1;
  $comentario = "Este es un comentario de prueba";
  $puntuacion = 5;
  // Crear el comentario
  $resultado = Comentarios::crearComentario($usuario_id, $actividad_id, $comentario, $puntuacion);
  // Verificar que el comentario fue creado correctamente
  $this->assertTrue($resultado);
public function testCrearComentarioConUsuarioInexistente()
  $usuario id = 999; // ID de usuario no existente
  $actividad id = 100;
  $comentario = "Este es un comentario de prueba";
  $puntuacion = 5;
  // Intentar crear un comentario con un usuario que no existe
  $resultado = Comentarios:: crearComentario($usuario_id, $actividad_id, $comentario, $puntuacion);
  // Verificar que no se puede crear el comentario
  $this->assertFalse($resultado);
```



Eliminación:



```
// Test para obtener los comentarios de una actividad
public function testObtenerComentariosPorActividad()
{
// Definir una actividad de prueba que tenga comentarios
$actividad_id = 1;

// Obtener los comentarios asociados a esta actividad
$comentarios = Comentarios::obtenerComentariosPorActividad($actividad_id);

// Verificar que se ha devuelto un arreglo
$this->assertlsArray($comentarios);

// Verificar que los comentarios corresponden a la actividad y tienen datos
if (count($comentarios) > 0)
{
$this->assertArrayHasKey('usuario', $comentarios[0]); // Verifica que el nombre del usuario está presente
$this->assertArrayHasKey('comentario', $comentarios[0]); // Verifica que el comentario está presente
}

// Test para abtenes los comentarios de una actividad que no aviete

// Test para abtenes los comentarios de una actividad que no aviete

// Test para abtenes los comentarios de una actividad que no aviete

// Test para abtenes los comentarios de una actividad que no aviete

// Test para abtenes los comentarios de una actividad que no aviete
```

```
// Test para obtener la puntuación promedio de una actividad
public function testObtenerPuntuacionPromedio ()
{
// Definir una actividad de prueba con comentarios
$actividad_id = 1;

// Obtener la puntuación promedio de la actividad
$puntuacionPromedio = Comentarios::obtenerPuntuacionPromedio($actividad_id);

// Verificar que la puntuación promedio es un número
$this->assertIsFloat($puntuacionPromedio);
$this->assertGreaterThanOrEqual(0, $puntuacionPromedio); // Asegurarse de que la puntuación es positiva o cero
}
```

```
public function testObtenerPuntuacionPromedioActividadSinComentarios()
// Actividad sin comentarios
    $actividad id = 999;
// Obtener la puntuación promedio de la actividad
    $puntuacionPromedio = Comentarios::obtenerPuntuacionPromedio($actividad id);
// Verificar que la puntuación promedio es 0
    $this->assertEquals(0, $puntuacionPromedio);
// Test para obtener un comentario especifico por su ID
  public function testObtenerComentarioPorId()
// Definir un comentario que existe en la base de datos
    comentario id = 2;
// Obtener el comentario por su ID
    $comentario = Comentarios::obtenerComentarioPorld($comentario id);
// Verificar que el comentario ha sido encontrado
    $this->assertNotNull($comentario);
    $this->assertArrayHasKey('comentario', $comentario); // Verifica que el campo 'comentario' está presente
 // Test para intentar obtener un comentario que no existe
    public function testObtenerComentarioPorldNoExistente()
    {
```

```
// Test para intentar obtener un comentario que no existe

public function testObtenerComentarioPorIdNoExistente()
{

// Comentario que no existe

$comentario_id = 999;

// Intentar obtener el comentario por su ID

$comentario = Comentarios::obtenerComentarioPorId($comentario_id);

// Verificar que no se ha encontrado el comentario

$this->assertFalse($comentario);

}
```

Como se puede observar las pruebas han ido bien, es decir, esta clase al igual que el resto funciona como se esperaba

Clase notificaciones:

```
SELECT * FROM `notificaciones`
id usuario_id mensaje fecha leida
Operaciones sobre los resultados de la consulta
 Crear vista
```

Crear:

```
// Test para agregar una notificación
public function testAgregarNotificacion()
  $usuario_id = 5;
  $mensaje = "Prueba de notificación";
  $resultado = Notificaciones::agregarNotificacion($usuario_id, $mensaje);
  $this->assertTrue($resultado); // Verificar que la inserción fue exitosa
```

```
// Test para marcar una notificación como leida
  public function testMarcarComoLeida()
     $notificacion id = 1; // Asume que esta notificación existe en la base de datos
     $resultado = Notificaciones::marcarComoLeida($notificacion_id);
     $this->assertTrue($resultado); // Verificar que la operación fue exitosa
\leftarrow T \rightarrow

▼ id usuario_id mensaje

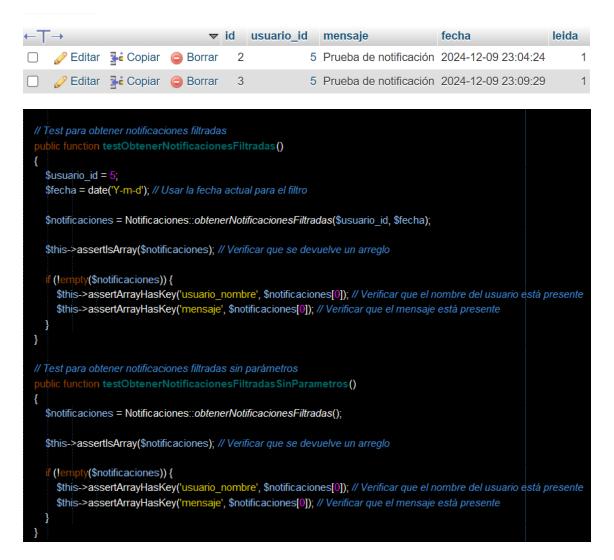
                                                                   fecha
                                                                                      leida
☐ / Editar 3 Copiar ☐ Borrar 1 5 Prueba de notificación 2024-12-09 23:03:14
 Test para marcar todas las notificaciones como leidas
  olic function testMarcarTodasComoLeidas()
 $usuario_id = 5;
 $resultado = Notificaciones::marcarTodasComoLeidas($usuario_id);
 $this->assertTrue($resultado); // Verificar que todas las notificaciones fueron marcadas como leidas
\leftarrow T \rightarrow

▼ id usuario_id mensaje

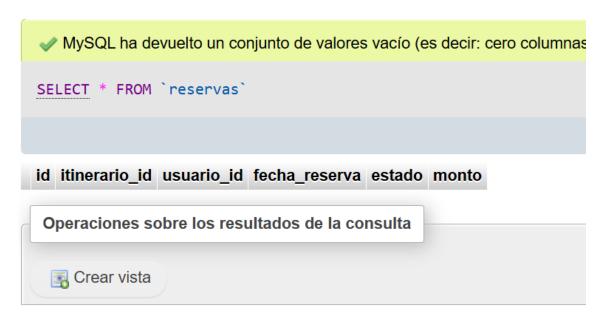
                                                                   fecha
                                                                                     leida
☐ / Editar → Copiar | Borrar 1 5 Prueba de notificación 2024-12-09 23:03:20
5 Prueba de notificación 2024-12-09 23:04:24
☐ Ø Editar ♣ Copiar 	 Borrar 3
                                              5 Prueba de notificación 2024-12-09 23:09:29
// Test para contar las notificaciones no leidas
public function testContarNoLeidas()
 susuario id = 5;
  $conteo = Notificaciones::contarNoLeidas($usuario_id);
  $this->assertIsInt($conteo); // Verificar que se devuelve un número entero
  $this->assertGreaterThanOrEqual(0, $conteo); // Asegurarse de que el conteo es positivo
```

Eliminar:

```
// Test para eliminar una notificación
public function testEliminarNotificacion()
{
    $notificacion_id = 1; // Asume que esta notificación existe en la base de datos
    $resultado = Notificaciones::eliminarNotificacion($notificacion_id);
    $this->assertTrue($resultado); // Verificar que la notificación fue eliminada
}
```



Clase reservas



ID	Método Probado	Descripción	Datos	Resultado Esperado	
		Caso	Entrada		
1	crearReserva	Crear una reserva con datos válidos.	itinerario_id=20, usuario_id=7, monto=100.50	Devuelve un ID numérico.	
2	crearReserva	Crear una reserva Con itinerario_id nulo.	itinerario_id=null, usuario_id=7, monto=100.50	Devuelve false.	
3	obtenerReservas PorUsuario	Obtener reservas de un usuario que tiene reservas.	usuario_id=7	Devuelve un arreglo de reservas.	
4	obtenerReservas PorUsuario	Obtener reservas de un usuario inexistente.	usuario_id=9999	Devuelve un arreglo vacío.	

			Reserva		
_	obtenerReserva	Obtener una reserva	id=último ID	Devuelve los	
5	Porld	existente por su ID.	creado	datos de la reserva.	
			dinámicamente		
6	obtenerReserva Porld	Obtener una reserva inexistente por su ID.	reserva_id=9999	Devuelve null.	
			reserva_id=		
	Actualizar	Actualizar el estado de	último ID creado		
7	Estado	una reserva con un estado válido.	dinámicamente, nuevoEstado=	Devuelve true.	
			'confirmada'		
			reserva_id=		
	una re	Actualizar el estado de	último ID creado		
8		una reserva con un estado inválido.	dinámicamente, nuevoEstado=	Devuelve false.	
9	Cancelar Reserva	Cancelar reserva existente.	reserva_id= último ID creado dinámicamente	Devuelve true.	
	Cancelar	Cancelar reserva			
10	Reserva	inexistente.	reserva_id=9999	Devuelve false.	
11	Eliminar Reserva	Eliminar reserva existente.	reserva_id= último ID creado dinámicamente	Devuelve true.	
12	Eliminar	Intentar eliminar	reserva_id=9999	Devuelve false.	

	Reserva	reserva inexistente.		
13	Obtener CosteTotal Actividades	Obtener el costo total actividades asociadas a un itinerario existente con actividades.	itinerario_id=20	Devuelve un número flotante mayor que 0.
14	Obtener CosteTotal Actividades	Obtener el costo total actividades asociadas a un itinerario sin actividades.	itinerario_id=9999	Devuelve 0.
16	Actualizar Presupuesto	Actualizar presupuesto de itinerario existente.	itinerario_id=20, monto=50.00	Devuelve true.
17	Actualizar Presupuesto	Intentar actualizar presupuesto de itinerario inexistente.	itinerario_id=9999, monto=30.00	Devuelve false.
18	Obtener Presupuesto	Obtener presupuesto de itinerario existente.	itinerario_id=20	Devuelve un número flotante mayor o igual que 0.
19	Obtener Presupuesto	Intentar obtener presupuesto de itinerario inexistente.	itinerario_id=9999	Devuelve 0.

Ejecución pruebas:

Test Crear Reserva

```
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter testCrearRe
serva tests\Modelo\ReservasTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
.
    1 / 1 (100%)
Time: 00:00.022, Memory: 4.00 MB
OK (1 test, 2 assertions)
```

Como se puede ver se ha creado solo la reserva cuyo id no era nulo:



Test Obtener Reservas Por Usuario

Se puede apreciar que ha creado la reserva en cuestión y la ha encontrado



Test Obtener Reserva Por Id

Encuentra la reserva con el id proporcionado, no encuentra la reserva inexistente

Test actualizar estado



Actualiza a confirmada por que es un estado existente

Cancelar reserva:

```
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter testCancela
rReserva tests\Modelo\ReservasTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
.
    1 / 1 (100%)
Time: 00:00.021, Memory: 4.00 MB
OK (1 test, 2 assertions)
```

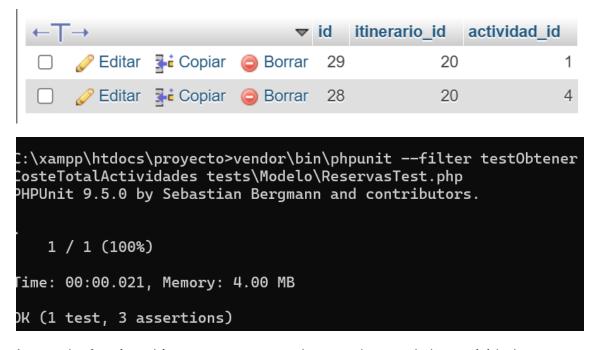


Test actualizar presupuesto:



Como se puede comprobar suma correctamente el monto al presupuesto

Test obtener coste total actividades:



La prueba funciona bien porque comprueba que el coste de las actividades asociadas a un itinerario es mayor a 0

Test obtener Presupuesto:

La prueba funciona bien al igual que en el caso anterior, ya que también comprueba que el presupuesto de un itinerario existente es mayor a 0:

```
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter testObtener
Presupuesto tests\Modelo\ReservasTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
.
    1 / 1 (100%)
Time: 00:00.023, Memory: 4.00 MB
OK (1 test, 4 assertions)
C:\xampp\htdocs\proyecto>
```

Test Eliminar Reserva:

Eliminacion de la reserva id 21:



```
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter testElimina
rReserva tests\Modelo\ReservasTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
.
    1 / 1 (100%)
Time: 00:00.021, Memory: 4.00 MB
OK (1 test, 2 assertions)
```



```
// Confirma una reserva
public static function confirmarReserva($reserva_id)
{
    return self::actualizarEstado($reserva_id, 'confirmada');
}

// Marca una reserva como pagada
public static function pagarReserva($reserva_id)
{
    return self::actualizarEstado($reserva_id, 'pagada');
}
```

Estos 2 métodos no los pruebo por que son exactamente iguales a cancelar reserva, el cual se ha visto que funciona correctamente

Clase usuario:

Estado actual:



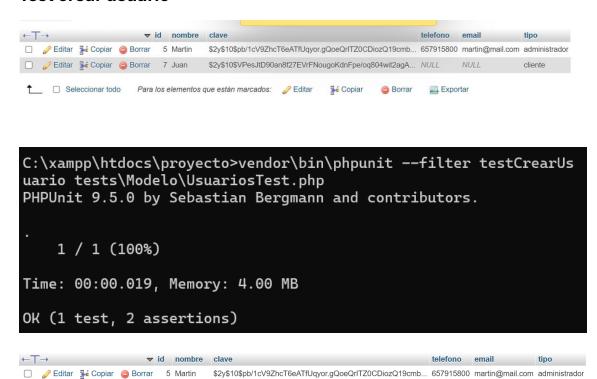
ID	Método Probado	Descripción	Caso	Datos Entrada	Resultado Esperado
1	testCrearUsuario	Crear un usuario con datos válidos.	Crear un usuario válido con nombre, clave y tipo.	nombre = 'Carlos', clave = 'contraseña_segura', tipo_usuario = 'cliente'	Devuelve true, el usuario se crea correctam ente.
2	testCrearUsuario	Intentar crear un usuario con nombre vacío.	Intentar crear un usuario con nombre vacío.	nombre = '', clave = 'contraseña_segura', tipo_usuario = 'cliente'	Devuelve false, ya que el nombre es vacío.
3	testObtenerUsuario s	Obtener todos los usuarios.	Obtener la lista de usuarios registrado s en la base de datos.	Ninguno (sin parámetros)	Devuelve un array con los usuarios existentes.
4	testObtenerUsuario PorNombre	Obtener un usuario por nombre que existe.	Buscar un usuario llamado 'Carlos' que existe en la base de datos.	nombre = 'Carlos'	Devuelve el usuario con nombre 'Carlos'.

ID	Método Probado	Descripción	Caso	Datos Entrada	Resultado Esperado
5	testObtenerUsuario PorNombre	Intentar obtener un usuario por nombre que no existe.	Buscar un usuario llamado 'NoExisten te' que no está en la base de datos.	nombre = 'NoExistente'	Devuelve null, no existe un usuario con ese nombre.
7	testActualizarUsuar io	Actualizar los datos de un usuario existente.	Actualizar el nombre, email y teléfono de un usuario con ID 8.	id = 8, nuevo_nombre = 'Antonio', nuevo_email = 'antonio@mail.com', nuevo_telefono = '657915800'	Devuelve true, el usuario se actualiza correctam ente.
8	testActualizarUsuar io	Intentar actualizar los datos de un usuario inexistente.		<pre>id = 9999, nuevo_nombre = 'NoExiste', nuevo_email = 'noexiste@mail.com', nuevo_telefono = '000000000'</pre>	Devuelve false, el usuario no existe para actualizar.
9	testActualizarContr asena	Actualizar la contraseña de un usuario existente.	Actualizar la contraseñ a del usuario con ID 8.	usuario_id = 8, nueva_contrasena = 'nueva_contrasena_seg ura'	Devuelve true, la contraseña se actualiza correctam ente.
10	testActualizarContr asena	Intentar actualizar la contraseña de un usuario inexistente.	Intentar actualizar la contraseñ a de un usuario con ID	usuario_id = 9999, nueva_contrasena = 'nueva_contrasena_seg ura'	Devuelve false, no se puede actualizar la contraseña de un

ID	Método Probado	Descripción	Caso	Datos Entrada	Resultado Esperado
			9999 que no existe.		usuario inexistente.
11	testEliminarUsuario	Eliminar un usuario existente.	Eliminar un usuario con ID 8.	id = 8	Devuelve true, el usuario se elimina correctam ente.
12	testEliminarUsuario	Intentar eliminar un usuario inexistente.	Intentar eliminar un usuario con ID 9999 que no existe.	id = 9999	Devuelve false, no se puede eliminar un usuario inexistente.

Ejecución pruebas:

Test crear usuario



Crea un usuario cuando se pasan los valores correctos, si falta alguno no lo crea

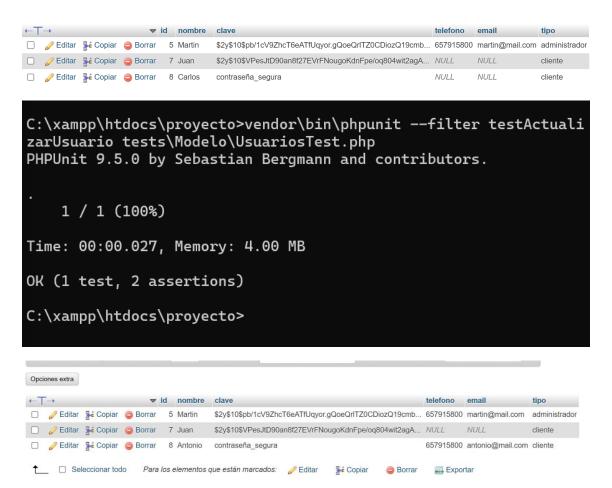
☐ 🔗 Editar 💃 Copiar 🥚 Borrar 7 Juan \$2y\$10\$VPesJtD90an8f27EVrFNougoKdnFpe/oq804wit2agA... NULL NULL cliente

Test obtener usuarios

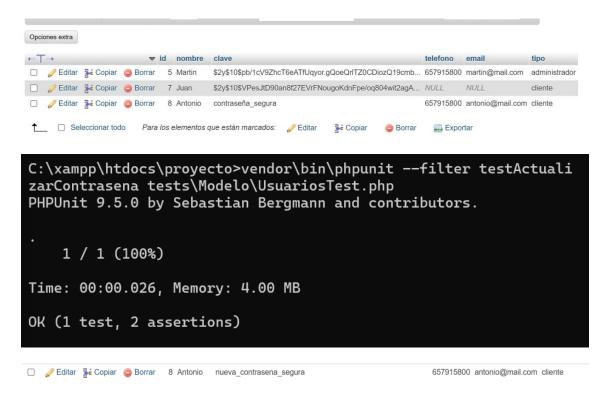
☐ Ø Editar ♣ Copiar ⑤ Borrar 8 Carlos contraseña_segura

Test obtener usuario por nombre:

Test actualizar usuario:



Test actualizar contraseña:



Test eliminar usuario:

```
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter testElimina
rUsuario tests\Modelo\UsuariosTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
.
    1 / 1 (100%)
Time: 00:00.027, Memory: 4.00 MB
OK (1 test, 2 assertions)
```



Clase Itinerarios



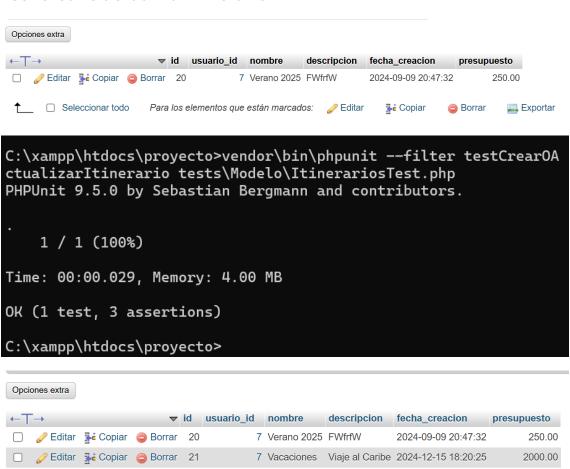
Pruebas

ID	Método Probado	Descripción	Caso	Datos Entrada	Resultado Esperado
1	testCrearOActualizar Itinerario	Crear un itinerario con datos válidos.	Crear un itinerario nuevo.	usuariold = 1, nombre = 'Vacaciones', descripción = 'Viaje a la playa', presupuesto = 1000	correctamen
2	testCrearOActualizar Itinerario	Actualizar un itinerario existente.	Actualizar un itinerario existente con el mismo nombre.	usuariold = 1, nombre = 'Vacaciones', descripción = 'Viaje al Caribe', presupuesto = 2000	correctamen
3	testCrearOActualizar Itinerario	Intentar crear un itinerario con datos incompletos.	Crear un itinerario sin presupue sto.	usuarioId = 1, nombre = 'Aventura', descripción = 'Excursión', presupuesto = null	Devuelve false, el itinerario no se crea debido a datos incompletos
4	testObtenerItinerario PorId	Obtener un itinerario por su ID.	Obtener el itinerario con ID 2.	id = 2	Devuelve el itinerario con ID 2.

ID	Método Probado	Descripción	Caso	Datos Entrada	Resultado Esperado
5	testObtenerActivida desPorItinerario	Obtener actividades asociadas a un itinerario.	Obtener actividad es del itinerario con ID 2.	itinerario_id = 2	Devuelve un array con las actividades asociadas al itinerario.
6	testAgregarActividad	Agregar una actividad a un itinerario existente.	Agregar la actividad con ID 5 al itinerario con ID 2.	itinerario_id = 2, actividad_id = 5	Devuelve true, la actividad se agrega correctamen te al itinerario.
7	testEliminarActivida d	Eliminar una actividad de un itinerario.	Eliminar la actividad con ID 5 del itinerario con ID 2.	itinerario_id = 2, actividad_id = 5	Devuelve true, la actividad se elimina correctamen te del itinerario.
8	testEliminarItinerario	Eliminar un itinerario existente.	Eliminar el itinerario con ID 3.	id = 3	Devuelve true, el itinerario se elimina correctamen te.
9	testEliminarItinerario	Intentar eliminar un itinerario inexistente.	Intentar eliminar el itinerario con ID 9999.	id = 9999	Devuelve false, el itinerario no se puede eliminar porque no existe.
10	testContarActividade s	Contar el número de actividades	Contar actividad es del	itinerario_id = 2	Devuelve un número

ID	Método Probado	Descripción	Caso	Datos Entrada	Resultado Esperado
		asociadas a un itinerario.	itinerario con ID 2.		mayor o igual a 0.
11	testObtenerItinerario sFiltrados	Obtener itinerarios filtrados por usuario y fecha de creación.	Filtrar itinerarios del usuario con ID 1 y fecha = '2024-12-	usuario_id = 1, fecha_creacion = '2024-12-01'	Devuelve un array con los itinerarios filtrados por usuario y fecha.
			01'.		

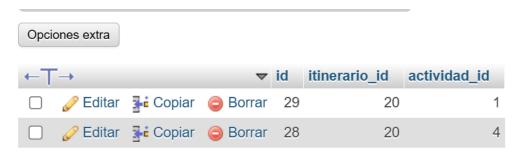
Test crear o actualizar itinerario

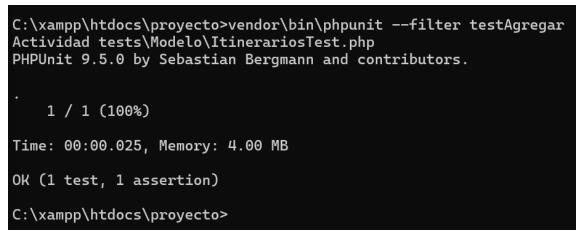


Crea el itinerario, lo actualiza después, pero no crea el itinerario al que le faltan datos

Test Obtener itinerario por id

Test agregar Actividad







Test eliminar actividad:

Test contar actividades:

```
public function testContarActividades()
{
    $itinerarioId = 21;
    $resultado = Itinerarios::contarActividades($itinerarioId);
    $this->assertIsInt($resultado);
    $this->assertGreaterThanOrEqual(0, $resultado);
}
```

Comprueba que el itinerario actualmente no tiene actividades:

```
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter testContarA
ctividades tests\Modelo\ItinerariosTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
.
. 1 / 1 (100%)
Time: 00:00.029, Memory: 4.00 MB
OK (1 test, 3 assertions)
```

En efecto, funciona correctamente

Test Obtener Itinerarios Filtrados

Obtiene el itinerario creado en esa fecha y por ese usuario:

```
public function testObtenerItinerariosFiltrados()
   Susuariold = 7;
   $fechaCreacion = '2024-09-09':
   $resultado = Itinerarios::obtenerItinerariosFiltrados($usuariold, $fechaCreacion);
   $this->assertIsArray($resultado);
 public function testObtenerItinerariosFiltrados()
   susuariold = 7;
   $fechaCreacion = '2024-09-09';
   $resultado = Itinerarios::obtenerItinerariosFiltrados($usuariold, $fechaCreacion);
   $this->assertIsArray($resultado);
C:\xampp\htdocs\proyecto>vendor\bin\phpunit --filter test0btener
ItinerariosFiltrados tests\Modelo\ItinerariosTest.php
PHPUnit 9.5.0 by Sebastian Bergmann and contributors.
    1 / 1 (100%)
Time: 00:00.025, Memory: 4.00 MB
OK (1 test, 1 assertion)
```

Test eliminar Itinerario:



He estado intentando realizar pruebas en los diferentes controladores durante varios días, pero me he encontrado con conflictos relacionados con las cabeceras al utilizar sesiones en las pruebas con PHPUnit. No estoy seguro de cómo proceder, ya que nunca hemos trabajado con PHPUnit para probar cabeceras y sesiones. He estado investigando sobre el tema, pero aún no he logrado encontrar una solución.

De todas formas, se puede comprobar que los métodos funcionan correctamente, y en la web todo está operando sin problemas.