CODIFICAR PRIMERO LA PRUEBA.

PU1.Validación de credenciales de usuario	
Se diseñó una prueba unitaria para validar que el sistema pueda identificar credenciales correctas e incorrectas.	
Justificación	El inicio de sesión es un punto de entrada clave para los usuarios. Garantizar su correcto funcionamiento es crucial para la experiencia del cliente y la seguridad del sistema.

PU2.Cálculo de totales en el carrito	
Se implementaron pruebas unitarias para verificar el cálculo del subtotal, IVA y total general del carrito de compras.	
Justificación	El cálculo correcto de totales asegura que los usuarios tengan claridad sobre los costos, evitando discrepancias que puedan generar desconfianza o errores en el cobro.

PU3.Simulación de colocación de pedidos	
Se simularon interacciones con la base de datos para validar la creación de pedidos en el sistema.	
Justificación	Garantizar que los pedidos se procesen correctamente es esencial para el flujo del sistema. Esta prueba asegura la integridad de los datos y la persistencia en la base de datos.

CODIFICAR PRIMERO LA PRUEBA.

PU4.Validación del manejo de errores y redirección en el inicio de sesión	
Se implementaron pruebas en el frontend para verificar el manejo de respuestas del servidor, mostrando mensajes de error para credenciales incorrectas o redireccionando tras un inicio de sesión exitoso.	
Justificación	Estas pruebas aseguran que los usuarios reciban retroalimentación clara e inmediata sobre sus acciones, mejorando la usabilidad y la experiencia general del sistema.

PRUEBAS UNITARIAS

PU1.Validación de credenciales de usuario

```php
function validateUserCredentials($email, $password, $dbConnection) {
    $stmt = $dbConnection->prepare("SELECT * FROM users WHERE email =
?");
    $stmt->execute([$email]);
    $user = $stmt->fetch();

    if ($user && password_verify($password, $user['password'])) {
        return true; // Credenciales correctas
    }
    return false; // Credenciales incorrectas
}
```

CODIFICAR PRIMERO LA PRUEBA.

PHPUNIT

```php
<?php

use PHPUnit\Framework\TestCase;

class LoginTest extends TestCase
{
    public function testValidateUserCredentials()
    {
        $mockDb = $this->createMock(PDO::class);
        $mockStmt = $this->createMock(PDOStatement::class);

        $mockDb->method('prepare')->willReturn($mockStmt);
        $mockStmt->method('execute')->willReturn(true);
        $mockStmt->method('fetch')->willReturn([
            'email' => 'test@example.com',
            'password' => password_hash('password123', PASSWORD_DEFAULT),
        ]);

        $result = validateUserCredentials('test@example.com',
'password123', $mockDb);
        $this->assertTrue($result);

        $result = validateUserCredentials('test@example.com',
'wrongpassword', $mockDb);
        $this->assertFalse($result);
    }
}
?>
```

PU2.Cálculo de totales en el carrito

```php
function calculateOrderTotal($cartItems, $taxRate = 0.15) {
    $subtotal = 0;
    foreach ($cartItems as $item) {
        $subtotal += $item['price'] * $item['quantity'];
    }
    $tax = $subtotal * $taxRate;
    $total = $subtotal + $tax;

    return [
        'subtotal' => round($subtotal, 2),
        'tax' => round($tax, 2),
        'total' => round($total, 2),
    ];

}
```

CODIFICAR PRIMERO LA PRUEBA.

PHPUNIT

```php
<?php

use PHPUnit\Framework\TestCase;

require_once 'path/to/your/functions.php';

class CheckoutTest extends TestCase
{
    public function testCalculateOrderTotal()
    {
        $cartItems = [
            ['price' => 100.00, 'quantity' => 2],
            ['price' => 50.00, 'quantity' => 1],
        ];
        $result = calculateOrderTotal($cartItems);

        $this->assertEquals(250.00, $result['subtotal']);
        $this->assertEquals(37.50, $result['tax']);
        $this->assertEquals(287.50, $result['total']);
    }
}
?>
```

PU3.Simulación de colocación de pedidos

```php
function placeOrder($clientId, $orderDetails, $dbConnection) {

    $stmt = $dbConnection->prepare("INSERT INTO orders (client_id,
total_amount, delivery_address) VALUES (?, ?, ?)");
    return $stmt->execute([$clientId, $orderDetails['total'],
$orderDetails['delivery_address']]);
}
```

CODIFICAR PRIMERO LA PRUEBA.

PHPUNIT

```php
<?php

use PHPUnit\Framework\TestCase;

class PlaceOrderTest extends TestCase
{
    public function testPlaceOrder()
    {
        $mockDb = $this->createMock(PDO::class);
        $mockStmt = $this->createMock(PDOStatement::class);

        $mockDb->method('prepare')->willReturn($mockStmt);
        $mockStmt->method('execute')->willReturn(true);

        $orderDetails = [
            'total' => 287.50,
            'delivery_address' => '123 Calle Principal, Ciudad, País',
        ];

        $result = placeOrder(1, $orderDetails, $mockDb);
        $this->assertTrue($result);
    }
}
?>
```

ENVIO DE EMAIL DE CONFIRMACION

```php
function getEmailProvider($email) {
    $email = strtolower($email);
    if (strpos($email, '@gmail.com') !== false) {
        return 'gmail';
    } elseif (strpos($email, '@outlook.com') !== false || strpos($email,
'@hotmail.com') !== false) {
        return 'outlook';
    }
    return 'default';
}
```

CODIFICAR PRIMERO LA PRUEBA.

PHPUNIT

```php
<?php

use PHPUnit\Framework\TestCase;

class EmailProviderTest extends TestCase
{
    public function testGetEmailProvider()
    {
        $gmail = "user@gmail.com";
        $outlook = "user@outlook.com";
        $default = "user@customdomain.com";
        $this->assertEquals('gmail', getEmailProvider($gmail));
        $this->assertEquals('outlook', getEmailProvider($outlook));
        $this->assertEquals('default', getEmailProvider($default));
    }
}
?>
```

PU4.Validación del manejo de errores y redirección en el inicio de sesión

JEST

```javascript
escribe("Login Form", () => {
    test("Handles successful login", () => {
        const mockAjax = jest.fn().mockImplementation((options) => {
            options.success({ status: "success" });
        });

        // Simular el redireccionamiento
        global.location = { reload: jest.fn() };

        // Usar la función simulada
        $.ajax = mockAjax;

        const loginForm = $("#login-form");
        loginForm.trigger("submit");

        expect(mockAjax).toHaveBeenCalled();
        expect(global.location.reload).toHaveBeenCalled();
    });

    test("Handles incorrect credentials", () => {
        const mockAjax = jest.fn().mockImplementation((options) => {
            options.success({ status: "incorrect" });
        });

        $.ajax = mockAjax;
```

CODIFICAR PRIMERO LA PRUEBA.

```
        const loginForm = $("#login-form");
        loginForm.trigger("submit");

        // Verifica que muestra el mensaje de error
        expect($(".err-msg").length).toBe(1);
        expect($(".err-msg").text()).toBe("Incorrect Credentials.");
    });
});
```