**INFORME DE PRUEBAS 3**

**Versión 1.0**

**Lima, junio 2023**

**Historial de revisiones**

| **Fecha** | **Revisión** | **Descripción** | **Autor** |
| --- | --- | --- | --- |
| 15/06/2023 | 1.0 | Primera versión del informe de pruebas 3 | Consultora HomeSkill S.A. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**ÍNDICE**

[**1. Introducción 4**](#_73spztqnwa75)

[1.1. Propósito del Documento 4](#_yco4zsf4p4d4)

[1.2. Alcance del Informe 4](#_24867z2724rq)

[**2. Plan de Pruebas 4**](#_xe7vc3su4d2i)

[2.1. Tipos de Pruebas 4](#_r0qmse83vxn8)

[2.2. Recursos Necesarios 19](#_j4vtqijag92u)

[2.3. Estrategias de Pruebas 20](#_betp4nxhci6l)

[**3. Casos de Prueba 20**](#_1r12el1bu0zi)

[3.1. Escenario de prueba 20](#_5ayqrnywxihr)

[3.1.1. Prueba de Requerimiento 1 20](#_o8y3sv2kehny)

[3.1.2. Prueba de Requerimiento 2 20](#_6bq7hcpjuc6m)

[3.1.3. Prueba de Requerimiento 3 21](#_sx7aqeeqyxbp)

[3.1.4. Prueba de Requerimiento 4 21](#_p7r87iyf77x8)

[3.1.5. Prueba de Requerimiento 5 22](#_j7xtwaqq23y9)

[3.1.6. Prueba de Requerimiento 6 22](#_g2echtoc4ywn)

[3.1.7. Prueba de Requerimiento 7 22](#_mipnsor48bln)

[**4. Resultados de Pruebas 22**](#_v9agp3i4p24h)

[**5. Conclusiones 22**](#_nkcpktyakvhq)

# **1. Introducción**

## 

## **1.1. Propósito del Documento**

El propósito principal es comprobar las actualizaciones de las pruebas que se han hecho, además de probar los nuevos test de los requisitos 6 y 7, qué resultados se obtuvieron, qué incidencias se encontraron, las áreas que necesitan mejorar y las recomendaciones para futuras pruebas.

## 

## **1.2. Alcance del Informe**

El informe de pruebas proporciona una evaluación completa y detallada del software del proyecto GoShop en términos de su calidad y capacidad para satisfacer los requisitos del usuario. El informe incluye información sobre los requisitos funcionales que se han evaluado, y cualquier problema que se haya encontrado durante las pruebas y cómo se han abordado.

# **2. Plan de Pruebas**

## **2.1. Tipos de Pruebas**

**Tabla 1**

*Prueba de Requerimiento 1*

| **Vista de registro de usuario** | |
| --- | --- |
| **Resumen:** Este caso se utiliza para probar el comportamiento de la vista de registro de usuario en una aplicación Django. | |
| **Pasos:**   1. Configuración:  * Crear una instancia del cliente (self.client). * Obtener las URL necesarias para la prueba (self.signup\_url, self.signin\_url, self.signout\_url).  1. Prueba de vista de registro con solicitud GET:  * Realizar una solicitud GET a la URL de registro (self.signup\_url) utilizando el cliente. * Verificar que el código de estado de la respuesta sea 200. * Verificar que se utilice la plantilla 'register.html'.  1. Prueba de vista de registro con solicitud POST válida:  * Crear un diccionario de datos con los campos necesarios para el registro. * Realizar una solicitud POST a la URL de registro (self.signup\_url) con los datos proporcionados. * Verificar que el código de estado de la respuesta sea 302 (redirección). * Verificar que se redirija a la URL 'shop' utilizando assertRedirects(). * Verificar que se haya creado un objeto User en la base de datos con el nombre de usuario 'testuser'.  1. Prueba de vista de inicio de sesión con solicitud GET:  * Realizar una solicitud GET a la URL de inicio de sesión (self.signin\_url) utilizando el cliente. * Verificar que el código de estado de la respuesta sea 200. * Verificar que se utilice la plantilla 'login.html'.  1. Prueba de vista de inicio de sesión con solicitud POST:  * Crear un usuario en la base de datos utilizando User.objects.create\_user(). * Crear un diccionario de datos con el nombre de usuario y la contraseña del usuario. * Realizar una solicitud POST a la URL de inicio de sesión (self.signin\_url) con los datos proporcionados. * Verificar que el código de estado de la respuesta sea 302 (redirección). * Verificar que se redirija a la URL 'shop' utilizando assertRedirects(). * Verificar que el objeto User devuelto por response.wsgi\_request.user sea igual al usuario creado.  1. Prueba de vista de cierre de sesión:  * Crear un usuario en la base de datos utilizando User.objects.create\_user(). * Iniciar sesión con el usuario creado utilizando self.client.login(). * Realizar una solicitud GET a la URL de cierre de sesión (self.signout\_url). * Verificar que el código de estado de la respuesta sea 302 (redirección). * Verificar que se redirija a la URL 'shop' utilizando assertRedirects(). * Verificar que el usuario en response.wsgi\_request.user sea anónimo utilizando response.wsgi\_request.user.is\_anonymous. | |
| **Resultados Esperados:**  **Método ‘test\_signup\_view\_with\_get\_request’**   * Esperamos que la vista de registro de usuario esté disponible y responda con un código de estado HTTP 200 (OK). * Esperamos que la plantilla ‘register.html’ se utilice para mostrar la vista de registro.   **Método ‘test\_signup\_view\_with\_valid\_post\_request’**   * Esperamos que se cree un nuevo usuario en la base de datos con éxito. * Esperamos que el usuario sea redirigido a la página "shop" después de un registro válido.   **Método ‘test\_signin\_get’**   * response.status\_code: Se espera que el código de estado de la respuesta sea 200, lo que indica una respuesta exitosa. * assertTemplateUsed(response, 'login.html'): Se espera que se utilice la plantilla 'login.html' para renderizar la respuesta.   **Método ‘test\_signin\_post’**   * El código de estado de la respuesta debe ser 302, indicando una redirección exitosa, lo cual se verifica utilizando self.assertEqual(response.status\_code, 302). * La respuesta debe redirigir a la URL 'shop', lo cual se verifica utilizando self.assertRedirects(response, reverse('shop')). * El objeto User devuelto por response.wsgi\_request.user debe ser igual al usuario creado previamente, lo cual se verifica utilizando self.assertEqual(str(response.wsgi\_request.user), str(user))   **Método ‘test\_signout\_view’**   * self.assertEqual(response.status\_code, 302): Se espera que el código de estado de la respuesta sea 302, que indica una redirección * self.assertEqual(response.url, reverse('shop')): Se espera que la URL de redirección sea igual a la URL inversa de 'shop'. Esto significa que después de cerrar sesión, el usuario debería ser redirigido a la página de la tienda. * self.assertTrue(response.wsgi\_request.user.is\_anonymous): Se espera que la propiedad is\_anonymous del usuario en response.wsgi\_request.user sea verdadera. | |

*Nota*. Elaboración Propia.

**Tabla 2**

Prueba de Requerimiento 2

| **Vista de búsqueda y visualización de productos** |
| --- |
| **Resumen:** El caso de prueba usa el cliente de prueba de Django para simular una solicitud a la vista y luego verifica si la respuesta recibida contiene los productos esperados y si el estado de respuesta es 200 (OK). |
| **Pasos:**   1. Importar las clases necesarias del módulo Django 2. Crear una clase de prueba llamada ProductTests que herede de TestCase 3. Definir el método setUp() para configurar los datos necesarios antes de ejecutar las pruebas. 4. Definir el método de prueba test\_products\_view() para verificar la vista de productos.    * Este método prueba la vista de productos al realizar una solicitud HTTP GET a la URL generada con el nombre "get\_products" y el argumento de texto "Kitchen".    * response.status\_code == 200 (verificación de código de estado HTTP)    * response contiene la cadena "Kitchen Products" (verificación de contenido en la respuesta) 5. Definir el método de prueba test\_products\_search() para realizar una búsqueda de productos    * Este método prueba la búsqueda de productos al realizar una solicitud HTTP GET a la URL generada con el nombre "get\_products" y el argumento de texto "Product".    * Se crean dos productos con nombres, categorías y descripciones específicas.    * response.status\_code == 200 (verificación de código de estado HTTP)    * response contiene la cadena "Product 1" (verificación de contenido en la respuesta)    * response contiene la cadena "Product 2" (verificación de contenido en la respuesta) |
| **Resultados Esperados:**  **Método ‘test\_products\_view’**   * Este método prueba la vista de productos al realizar una solicitud HTTP GET a la URL generada con el nombre "get\_products" y el argumento de texto "Kitchen". * El resultado esperado es que la respuesta tenga un código de estado HTTP 200 (indicando una respuesta exitosa) y que el cuerpo de la respuesta contenga la cadena "Kitchen Products".   **Método ‘test\_products\_search’**   * El resultado esperado es que la respuesta tenga un código de estado HTTP 200 (indicando una respuesta exitosa) y que el cuerpo de la respuesta contenga las cadenas "Product 1" y "Product 2". * response.status\_code == 200 (verificación de código de estado HTTP) * response contiene la cadena "Product 1" (verificación de contenido en la respuesta) * response contiene la cadena "Product 2" (verificación de contenido en la respuesta) |

*Nota*. Elaboración Propia.

**Tabla 3**

*Prueba de Requerimiento 3*

| **Vista de descripción de productos** | |
| --- | --- |
| **Resumen:** Se centra en probar la vista que muestra la descripción de un producto en la tienda en línea. Se utiliza el módulo de pruebas de Django TestCase y el objeto Client para hacer solicitudes HTTP simuladas y realizar comprobaciones en la respuesta recibida. | |
| **Pasos:**   1. Importar los módulos necesarios para el caso de prueba. 2. Configurar el objeto Client y crear instancias de ProductCategory y Product para utilizar en las pruebas. 3. Crear un método de prueba para verificar que la vista de descripción de productos funciona correctamente con un ID de producto válido:    1. Hacer una solicitud GET a la vista de descripción de productos con el ID del producto creado anteriormente.    2. Verificar que la respuesta tenga un código de estado 200.    3. Verificar que se utiliza la plantilla correcta ('product\_description.html').    4. Verificar que el contexto de la respuesta incluye el producto creado 4. Crear un segundo método de prueba para verificar que la vista de descripción de productos funciona correctamente con un ID de producto incorrecto    1. Hacer una solicitud GET a la vista de descripción de productos con un ID de producto incorrecto (100 en este caso).    2. Verificar que la respuesta tenga un código de estado 200.    3. Verificar que se utiliza la plantilla correcta ('product\_description.html').    4. Verificar que el contexto de la respuesta no incluye ningún producto (ya que el ID es incorrecto), pero incluye la vista anterior ('Cocina') y la vista actual ('kitchen'). | |
| **Resultados Esperados:**  **Método ‘test\_product\_description\_get’**   * Se espera que la solicitud GET a la vista de descripción de productos con el ID del producto creado anteriormente retorne un código de estado 200 (éxito) * Se espera que la plantilla utilizada sea ‘product\_description.html’   **Método ‘test\_product\_description\_get\_wrong\_product\_id’**   * Se espera que la solicitud GET a la vista de descripción de productos con un ID de producto incorrecto (100 en este caso) retorne un código de estado 200 (éxito). * Se espera que la plantilla utilizada sea ‘product\_description.html’ | |

*Nota*. Elaboración propia

**Tabla 4**

Prueba de Requerimiento 4

| **Vista de perfil de usuario** |
| --- |
| **Resumen:** Una vista de perfil de usuario que permita a los usuarios ver y editar su información personal y configuraciones de cuenta |
| **Pasos:**   1. Importar las clases y funciones necesarias de Django. 2. Crear una clase de prueba llamada ProfileViewTest que herede de TestCase. 3. Definir el método setUp() para configurar los datos necesarios antes de ejecutar las pruebas.    * Crear una instancia del cliente de pruebas Client().    * Crear un usuario de prueba utilizando el modelo User.objects.create\_user().    * Obtener o crear una instancia del perfil de usuario utilizando el modelo UserProfile.objects.get\_or\_create() 4. Definir el método de prueba test\_user\_profile\_view() para verificar la vista de perfil de usuario.    * Iniciar sesión en el cliente de pruebas con las credenciales del usuario de prueba utilizando client.login().    * Realizar una solicitud HTTP GET a la URL inversa llamada 'profile' utilizando client.get(reverse('profile')).    * Verificar que el código de estado de la respuesta sea 200 (indicando una respuesta exitosa) utilizando self.assertEqual(response.status\_code, 200). 5. Definir el método de prueba test\_user\_config\_view() para verificar la vista de configuración de usuario.    * Iniciar sesión en el cliente de pruebas con las credenciales del usuario de prueba utilizando client.login().    * Realizar una solicitud HTTP GET a la URL inversa llamada 'config' utilizando client.get(reverse('config')).    * Verificar que el código de estado de la respuesta sea 200 (indicando una respuesta exitosa) utilizando self.assertEqual(response.status\_code, 200). 6. Crear una clase de prueba separada llamada PasswordChangeViewTest que también heredé de TestCase. 7. Definir el método setUp() para configurar los datos necesarios antes de ejecutar las pruebas en la clase PasswordChangeViewTest.    * Crear una instancia del cliente de pruebas Client().    * Crear un usuario de prueba utilizando el modelo User.objects.create\_user().    * Obtener o crear una instancia del perfil de usuario utilizando el modelo UserProfile.objects.get\_or\_create(). 8. Definir el método de prueba test\_change\_password() para verificar el cambio de contraseña del usuario.    * Iniciar sesión en el cliente de pruebas con las credenciales del usuario de prueba utilizando client.login().    * Enviar una solicitud HTTP POST a la URL inversa llamada 'config' + "/change/password" con los datos de nueva contraseña proporcionados utilizando client.post().    * Verificar que el código de estado de la respuesta sea 200 (indicando una respuesta exitosa) utilizando self.assertEqual(response.status\_code, 200).    * Obtener una nueva instancia del usuario actualizado utilizando User.objects.get(id=self.user.id).    * Verificar que la contraseña del usuario se haya actualizado correctamente utilizando self.assertTrue(updated\_user.check\_password('newpassword123')). |
| **Resultados Esperados:**  **Método ‘test\_user\_profile\_view’**  ● El resultado esperado es que la respuesta tenga un código de estado HTTP 200, lo que indica una respuesta exitosa.  **Método ‘test\_user\_config\_view’**  ● El resultado esperado es que la respuesta tenga un código de estado HTTP 200, lo que indica una respuesta exitosa.  **Metodo ‘test\_change\_password()’**  ● El resultado esperado es que la respuesta tenga un código de estado HTTP 200, lo que indica una respuesta exitosa.  ● El resultado esperado adicional es que la contraseña del usuario se haya actualizado correctamente y se pueda autenticar con la nueva contraseña. |

*Nota*. Elaboración Propia.

**Tabla 5**

Prueba de Requerimiento 5

| **Método de pago** | |
| --- | --- |
| **Resumen:** Visualizar sus tarjetas registradas, y por otro, agregar más tarjetas para realizar sus compras en la tienda virtual. | |
| **Pasos:**   1. Configuración:  * Crear una instancia de RequestFactory (self.factory) para simular solicitudes HTTP. * Crear un usuario en la base de datos utilizando User.objects.create\_user(). * Crear una instancia de Client (self.client) para simular solicitudes HTTP. * Crear una categoría de producto en la base de datos utilizando ProductCategory.objects.create(). * Crear un producto en la base de datos utilizando Product.objects.create().  1. Prueba del carrito de compras:  * Crear una solicitud GET a la URL del carrito de compras (reverse('cart')) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función shopping\_cart(request) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando el método self.assertEqual(response.status\_code, 200).  1. Prueba de obtener la cantidad de un producto en el carrito:  * Crear una solicitud GET a la URL de obtener la cantidad de un producto en el carrito (reverse('get\_product\_quantity\_cart', args=[1])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función get\_product\_quantity\_cart(request, 1) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba de agregar un producto al carrito:  * Crear una solicitud POST a la URL de agregar un producto al carrito (reverse('add\_product', args=[1])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función add\_product(request, 1) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba de eliminar una unidad de un producto en el carrito:  * Crear una solicitud POST a la URL de eliminar una unidad de un producto en el carrito (reverse('remove\_product\_unit', args=[1])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función remove\_product\_unit(request, 1) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba de eliminar un producto del carrito:  * Crear una solicitud POST a la URL de eliminar un producto del carrito (reverse('remove\_product', args=[1])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función remove\_product(request, 1) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200). | |
| **Resultados Esperados:**  **Método ‘test\_shopping\_cart’**   * La vista del carrito de compras responda con un código de estado 200. Si la aserción self.assertEqual(response.status\_code, 200) es verdadera, se considera que el método de prueba ha sido exitoso.   **Método ‘test\_get\_product\_quantity\_cart’**   * Se espera que el código de estado de la respuesta sea 200. Esto indica que la solicitud para obtener la cantidad del producto en el carrito se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_add\_product’**   * Se espera que la operación de agregar el producto al carrito se realice correctamente y el código de estado de la respuesta sea 200. Si la aserción es verdadera, se considera que el método de prueba ha sido exitoso.   **Método ‘test\_remove\_product\_unit’**   * Se espera que el código de estado de la respuesta sea 200. Esto indica que la solicitud de eliminación de una unidad de un producto fue exitosa y se procesó correctamente.   **Método ‘test\_remove\_product’**   * Se espera que el código de estado de la respuesta sea 200. Esto indica que la solicitud para eliminar el producto del carrito se procesó correctamente y se obtuvo una respuesta exitosa | |

*Nota*. Elaboración propia

**Tabla 6**

Prueba de Requerimiento 6

| **Historial de compras** | |
| --- | --- |
| **Resumen: Visualización del historial de compras** | |
| **Pasos:**   1. Importar las clases y funciones necesarias de Django 2. Crear una clase de prueba llamada UserViewsTestCase que herede de TestCase 3. Definir el método setUp() para configurar los datos necesarios antes de ejecutar las pruebas.    * Crear una instancia del objeto RequestFactory() y asignarla a self.factory.    * Crear un usuario de prueba utilizando el modelo User.objects.create\_user().    * Crear una instancia del cliente de pruebas Client() y asignarla a self.client.    * Crear una categoría de producto de prueba utilizando el modelo ProductCategory.objects.create() y asignarla a self.category.    * Crear un producto de prueba utilizando el modelo Product.objects.create() y asignarlo a self.product 4. Definir el método de prueba test\_shopping\_history() para verificar el historial de compras del usuario.    * Establecer un estado de carrito de compras en la sesión del cliente utilizando self.client.session.    * Asignar un diccionario de productos al carrito de compras en la sesión utilizando session['cart'].    * Realizar una solicitud HTTP GET a la URL inversa llamada 'history' utilizando self.factory.get(reverse('history')).    * Asignar el usuario de prueba a la propiedad user de la solicitud utilizando request.user = self.user.    * Asignar la sesión del cliente a la propiedad session de la solicitud utilizando request.session = session.    * Llamar a la vista shopping\_history() pasando la solicitud y obtener la respuesta.    * Verificar que el código de estado de la respuesta sea 200 (indicando una respuesta exitosa) utilizando self.assertEqual(response.status\_code, 200) | |
| **Resultados Esperados:**  **Método ‘test\_shopping\_history’**   * El resultado esperado es que la respuesta tenga un código de estado HTTP 200, lo que indica una respuesta exitosa y pueda visualizar su historial de compras.   . | |

*Nota*. Elaboración propia

**Tabla 7**

Prueba de Requerimiento 7

| **Carrito de compras** | |
| --- | --- |
| **Resumen:** Verifica el funcionamiento de varias vistas relacionadas con el carrito de compras, incluyendo agregar y eliminar productos, obtener información sobre el carrito, ver el historial de compras y configurar el perfil de usuario. | |
| **Pasos:**   1. Configuración:  * Crear una instancia de RequestFactory (self.factory) para simular solicitudes HTTP. * Crear un usuario en la base de datos utilizando User.objects.create\_user(). * Crear una instancia de Client (self.client) para simular solicitudes HTTP. * Crear una categoría de producto en la base de datos utilizando ProductCategory.objects.create(). * Crear un producto en la base de datos utilizando Product.objects.create().  1. Prueba del carrito de compras:  * Crear una solicitud GET a la URL del carrito de compras (reverse('cart')) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función shopping\_cart(request) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando el método self.assertEqual(response.status\_code, 200).  1. Prueba de obtener la cantidad de un producto en el carrito:  * Crear una solicitud GET a la URL de obtener la cantidad de un producto en el carrito (reverse('get\_product\_quantity\_cart', args=[self.product.id])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función get\_product\_quantity\_cart(request, self.product.id) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba de agregar un producto al carrito:  * Crear una solicitud GET a la URL de agregar un producto al carrito (reverse('add\_product', args=[self.product.id])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función add\_product(request, self.product.id) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba de eliminar una unidad de un producto en el carrito:  * Crear una solicitud GET a la URL de eliminar una unidad de un producto en el carrito (reverse('remove\_product\_unit', args=[self.product.id])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función remove\_product\_unit(request, self.product.id) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba de eliminar un producto del carrito:  * Crear una solicitud GET a la URL de eliminar un producto del carrito (reverse('remove\_product', args=[self.product.id])) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función remove\_product(request, self.product.id) para procesar la solicitud y obtener la respuesta. * Verificar que el código de estado de la respuesta sea 200 utilizando self.assertEqual(response.status\_code, 200).  1. Prueba del perfil de usuario:  * Crear una solicitud GET a la URL del perfil de usuario (reverse('profile')) utilizando self.factory. * Establecer el usuario en la solicitud utilizando request.user = self.user. * Llamar a la función user\_profile(request) para procesar la | |
| **Resultados Esperados:**  **Método ‘test\_shopping\_cart’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para obtener el carrito de compras se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_get\_product\_quantity\_cart’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para obtener la cantidad del producto en el carrito se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_add\_product’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para agregar el producto al carrito se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_remove\_product\_unit’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para eliminar una unidad del producto del carrito se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_remove\_product’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para eliminar el producto del carrito se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_user\_profile’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para obtener el perfil de usuario se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_user\_config’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para acceder a la configuración del usuario se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_confirm\_user\_config’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para confirmar la configuración del usuario se procesó correctamente y se obtuvo una respuesta exitosa.   **Método ‘test\_shopping\_history’**   * Código de estado de la respuesta sea 200. Esto indica que la solicitud para acceder al historial de compras se procesó correctamente y se obtuvo una respuesta exitosa. | |

*Nota*. Elaboración propia

## **2.2. Recursos Necesarios**

● **Hardware necesario**

Se necesitará un equipo con suficiente capacidad de procesamiento y memoria RAM para ejecutar el servidor de desarrollo de Django, el navegador web y cualquier otra aplicación necesaria para realizar las pruebas.

● **Software necesario**

Se necesitará lo siguiente:

➢ Python 3.8 y las dependencias necesarias para ejecutar el proyecto de Django.

➢ Un entorno virtual para el proyecto de Django (Visual Studio).

➢ Un servidor de desarrollo de Django para ejecutar el proyecto y las pruebas.

➢ Un navegador web para realizar las pruebas.

● **Base de datos**

Se necesitará una base de datos para el proyecto de Django. Si se usa SQLite para el proyecto, no se necesitará configurar nada adicional, pero si se usa otro motor de base de datos, se necesitará configurarlo y asegurarse de que esté en funcionamiento.

● **Recursos adicionales**: Para las pruebas específicas del proyecto de Django mencionadas anteriormente, se necesitará lo siguiente:

➢ Vista de descripción de productos: se necesitará al menos un objeto de categoría de producto y un objeto de producto para realizar las pruebas. Se pueden crear estos objetos en la función setUp del TestCase.

➢ Vista de registro de usuario: no se necesitan recursos adicionales, ya que se puede utilizar la función de registro de Django para crear nuevos usuarios.

➢ Vista de búsqueda y visualización de productos: se necesitará al menos un objeto de categoría de producto y varios objetos de producto para realizar las pruebas. Se pueden crear estos objetos en la función setUp del TestCase.

## **2.3. Estrategias de Pruebas**

Las estratégicas de prueba que han sido usadas en los tests 6 y 7 son de tipo unitaria y de integración, las pruebas unitarias son pruebas a nivel código para verificar su comportamiento y validarlo, mientras que las de integración son para evaluar la interacción entre diferentes requisitos y cómo funcionan en conjunto. Además de las actualizaciones de los requisitos 1 al 5.

# **3. Casos de Prueba**

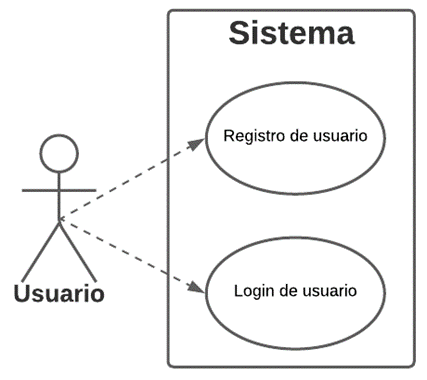
## **3.1. Escenario de prueba**

### ***3.1.1. Prueba de Requerimiento 1***

El propósito del caso de prueba es verificar el correcto funcionamiento de la vista de registro de usuario en una aplicación web, cuando se reciben solicitudes GET y POST, se redirecciona al usuario a la página correcta y se crea el usuario en la base de datos sin problemas. Para lograrlo, se configuran los elementos necesarios, se realizan solicitudes GET y POST, se comprueba la respuesta HTTP de la vista del perfil de usuario, se verifica que la redirección se efectúe adecuadamente y se confirma que el usuario se haya creado sin errores en la base de datos mediante la ejecución de diversas afirmaciones y validaciones.

**Figura 1**

*Diagrama de caso de uso 01*

**

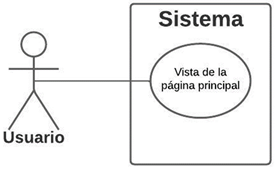
*Nota.* Elaboración propia

### ***3.1.2. Prueba de Requerimiento 2***

El propósito de las pruebas fue verificar el correcto funcionamiento de la vista de búsqueda y visualización de productos en una aplicación, para lo cual se emplearon pruebas funcionales e integración. El objetivo era asegurar que los productos se mostrarán de manera adecuada en la página de búsqueda, que la funcionalidad de búsqueda funcionara correctamente y que la información del producto fuera correcta en la página de detalle del producto. Asimismo, se evaluó la funcionalidad de paginación y de ordenamiento de los productos. Tras realizar todas las pruebas, se confirmó que la aplicación funcionaba sin problemas, ya que no se encontraron errores o problemas durante las pruebas.

**Figura 2**

*Diagrama de caso de uso 02*



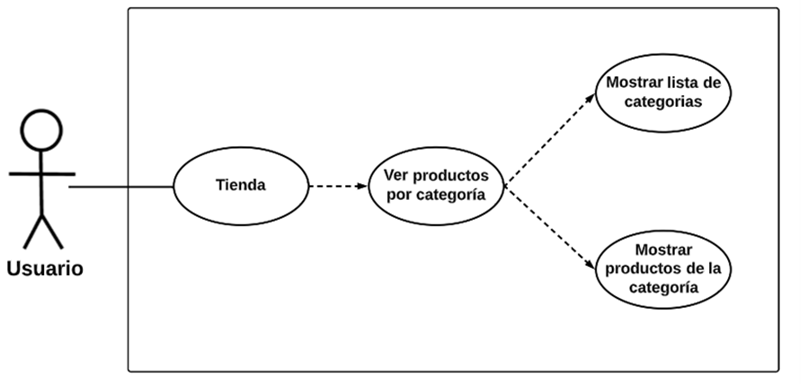
*Nota.* Elaboración propia

### ***3.1.3. Prueba de Requerimiento 3***

El propósito de las pruebas fue verificar el correcto funcionamiento de la página de descripción de productos en una aplicación, mediante la realización de pruebas funcionales y de regresión. El objetivo era asegurarse de que la página de descripción de cada producto se mostrará de manera adecuada, tanto para diferentes productos como en diferentes situaciones. Se evaluó que el producto correcto se mostrará en la página de descripción, que la plantilla y la información del producto fueran correctas, y que se visualizarán las vistas de la categoría anterior y actual de manera adecuada. Tras realizar todas las pruebas, se confirmó que la aplicación funcionaba sin problemas, ya que no se encontraron errores o problemas durante las pruebas.

**Figura 3**

*Diagrama de caso de uso 03*



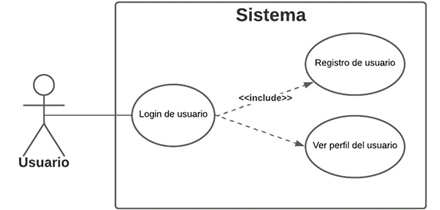
*Nota*. Elaboración propia

### ***3.1.4. Prueba de Requerimiento 4***

El propósito del caso de prueba es verificar el correcto funcionamiento de la vista de registro de usuario en una aplicación web, cuando se reciben solicitudes GET y POST, se re direcciona al usuario a la página correcta y se crea el usuario en la base de datos sin problemas. Para lograrlo, se configuran los elementos necesarios, se realizan solicitudes GET y POST, se comprueba la respuesta HTTP de la vista del perfil de usuario, se verifica que la redirección se efectúe adecuadamente y se confirma que el usuario se haya creado sin errores en la base de datos mediante la ejecución de diversas afirmaciones y validaciones.

**Figura 4**

*Diagrama de caso de uso 04*

****

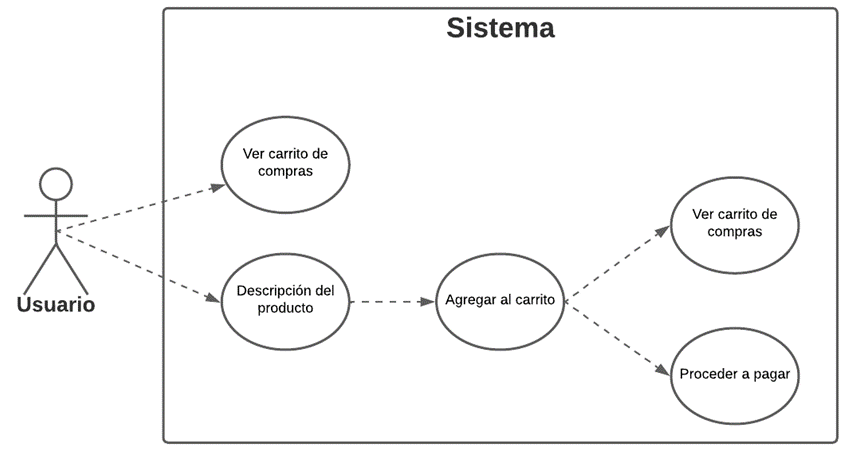
*Nota*. Elaboración propia

### ***3.1.5. Prueba de Requerimiento 5***

El propósito del caso de prueba es verificar el correcto funcionamiento de la vista de pago de usuario en una aplicación web, cuando se reciben solicitudes GET y POST, se redirecciona al usuario a la página correcta y se crea el usuario en la base de datos sin problemas. Para lograrlo, se configuran los elementos necesarios, se realizan solicitudes GET y POST, se comprueba la respuesta HTTP de la vista del perfil de usuario, se verifica que la redirección se efectúe adecuadamente y se confirma que el usuario se haya creado sin errores en la base de datos mediante la ejecución de diversas afirmaciones y validaciones. Además, se realizan solicitudes REVERSE y EXISTS, para retornar la lista o base de datos a la cual se llama y comprobar que existe la data en la base de datos.

**Figura 5**

*Diagrama de caso de uso 05*

**

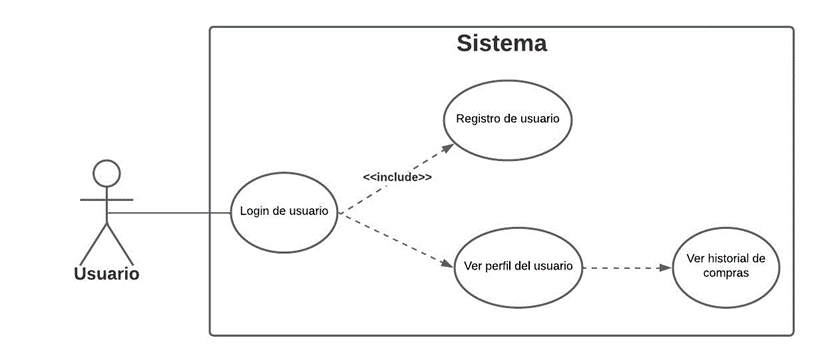
*Nota*. Elaboración propia

### ***3.1.6. Prueba de Requerimiento 6***

El propósito del caso de uso "Ver historial de compras" es permitir que los usuarios registrados accedan y visualicen los detalles de todas las compras que han realizado en la plataforma de compras en línea. Además, proporciona al usuario la capacidad de consultar información relevante, como la fecha de la compra, los productos adquiridos, la cantidad, el precio y el estado de la orden.

**Figura 6**

*Diagrama de caso de uso 06*



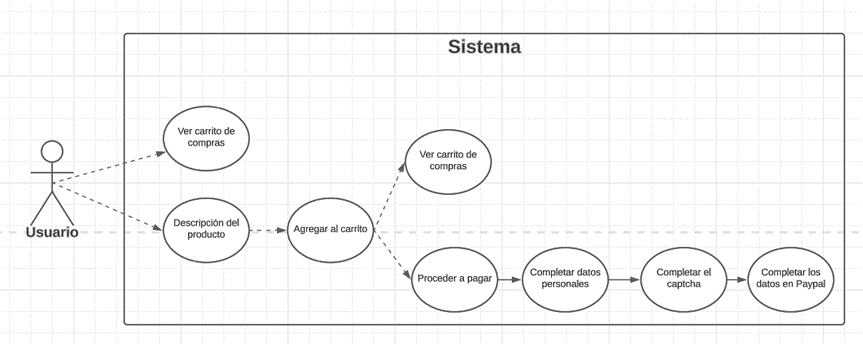
*Nota*. Elaboración propia

### ***3.1.7. Prueba de Requerimiento 7***

El propósito del caso de uso "Realizar compras" es describir el proceso que sigue un cliente registrado en el sistema con el objetivo de llevar a cabo una compra en el sitio web de la plataforma de compras. El caso de uso abarca desde la navegación del cliente por el sitio y la selección de productos hasta la confirmación y procesamiento del pago. El sistema registra la compra, actualiza el inventario y proporciona una confirmación al cliente. Este caso de uso garantiza que el cliente pueda realizar transacciones exitosas, contribuyendo así a la funcionalidad y utilidad del sistema de compras en línea.

**Figura 07**

*Diagrama de caso de uso 07*



*Nota*. Elaboración propia

# **4. Resultados de Pruebas**

# **5. Conclusiones**