

## **DOCUMENTACION DE PRUEBAS**

### **Aprendiz**

Luis Fernando Escobar Llanten

## **SENA CENTRO DE LA INDUSTRIA, LA EMPRESA Y LOS SERVICIOS**

Doris Gonzales Martinez

10 de agosto de 2024

## Introducción

El diseño de pruebas es una etapa crucial en el proceso de ejecución de pruebas de software, asegurando que todas las funcionalidades del sistema se prueben adecuadamente y que el producto final cumpla con los requisitos y estándares de calidad esperados.

Estas pruebas realizadas a la página de comercio electrónico “Construmole” fueron diseñadas y ejecutadas conforme al proceso conocido en la industria del software como Ciclo de Vida de las Pruebas de Software (STLC), proporcionando un enfoque estructurado y sistemático para la validación del software.

### 1. Alcance de las pruebas

Las pruebas realizadas al sistema aseguran su correcto funcionamiento y confiabilidad en diversas áreas, incluyendo la operación de sus componentes clave, la gestión de datos y la interacción con la API.

#### Excluido del alcance:

- Pruebas de interfaz de usuario (UI).

#### Tipos de prueba

Las pruebas que realizaremos se clasifican en tres tipos:

##### Pruebas unitarias

- Funcionalidad básica de los modelos.
- Funcionamiento correcto de las vistas principales.
- Navegación a través de las urls principales.

##### Pruebas de aceptación

- **Inicio de sesión:** Garantizar que el sistema maneje correctamente tanto las credenciales válidas como las inválidas.
- **Cesta de la compra:** Validación de la adición/eliminación de artículos, cálculo del precio total, y la persistencia de la cesta.

##### Pruebas API

- **API de productos:** Garantizar la integridad de los datos.
- **API de productos:** Comprobar los puntos de conexión de la API mediante una evaluación del código de estado de las respuestas de la API.

## 2. Enfoque General de las Pruebas

Se llevó a cabo una combinación de pruebas manuales y automatizadas. Se utilizaron herramientas como Selenium para las pruebas de aceptación automatizadas, Postman para las pruebas de la API, y TestCase, una subclase de unittest, el paquete nativo de Python para realizar pruebas unitarias.

## 3. Planificación y Preparación

La planificación de las pruebas incluyó la preparación de casos de prueba específicos para cada área evaluada, la configuración de un entorno de pruebas y la creación de un cronograma de pruebas.

## 4. Ejecución de las pruebas

### 4.1 Pruebas unitarias

#### Elementos Por Probar

##### Modelos:

- **Category:** Verificar la creación, actualización, y eliminación de categorías.

##### Vistas:

- **index:** Mostrar productos en la página principal.
- **product\_detail:** Mostrar los detalles de un producto seleccionado.
- **category\_list:** Mostrar los productos que pertenecen a una categoría específica.

##### Urls:

- **/:** Accede a la vista index.
- **/product/<slug>/:** Accede a la vista product\_detail.
- **/category/<slug>/:** Accede a la vista category\_list.

#### 4.1.1 Estrategia de las pruebas unitarias

##### Objetivo

Asegurar la funcionalidad correcta de los modelos, vistas, y urls del proyecto a través de pruebas unitarias exhaustivas.

##### Categorización

Las pruebas se clasifican como pruebas unitarias y se ejecutarán en un entorno de desarrollo controlado.

##### Recursos Utilizados

- **Unittest/TestCase:** Framework de pruebas de Django para ejecutar las pruebas unitarias y validar la lógica del backend.

##### Entornos de Ejecución

- Las pruebas se ejecutarán en un entorno de desarrollo local utilizando la base de datos MySQL.

#### 4.1.2 Explicación de las Pruebas

##### Pruebas Unitarias para el Modelo Category

**Objetivo:** Verificar que el modelo Category funcione correctamente en la creación, actualización, y eliminación de categorías.

- **Prueba:** Creación de Categoría:

**Descripción:** Esta prueba verifica que se puede crear una nueva categoría en la base de datos.

**Verificación:** Se asegura que la categoría creada existe en la base de datos y que sus atributos (e.g., name, slug) se han guardado correctamente.

**Importancia:** Asegura que el administrador puede añadir nuevas categorías, lo cual es fundamental para organizar los productos en la tienda.

- **Prueba:** Actualización de Categoría:

**Descripción:** Esta prueba verifica que se puede actualizar una categoría existente.

**Verificación:** Se asegura que los cambios realizados a una categoría (e.g., cambio de nombre) se reflejan correctamente en la base de datos.

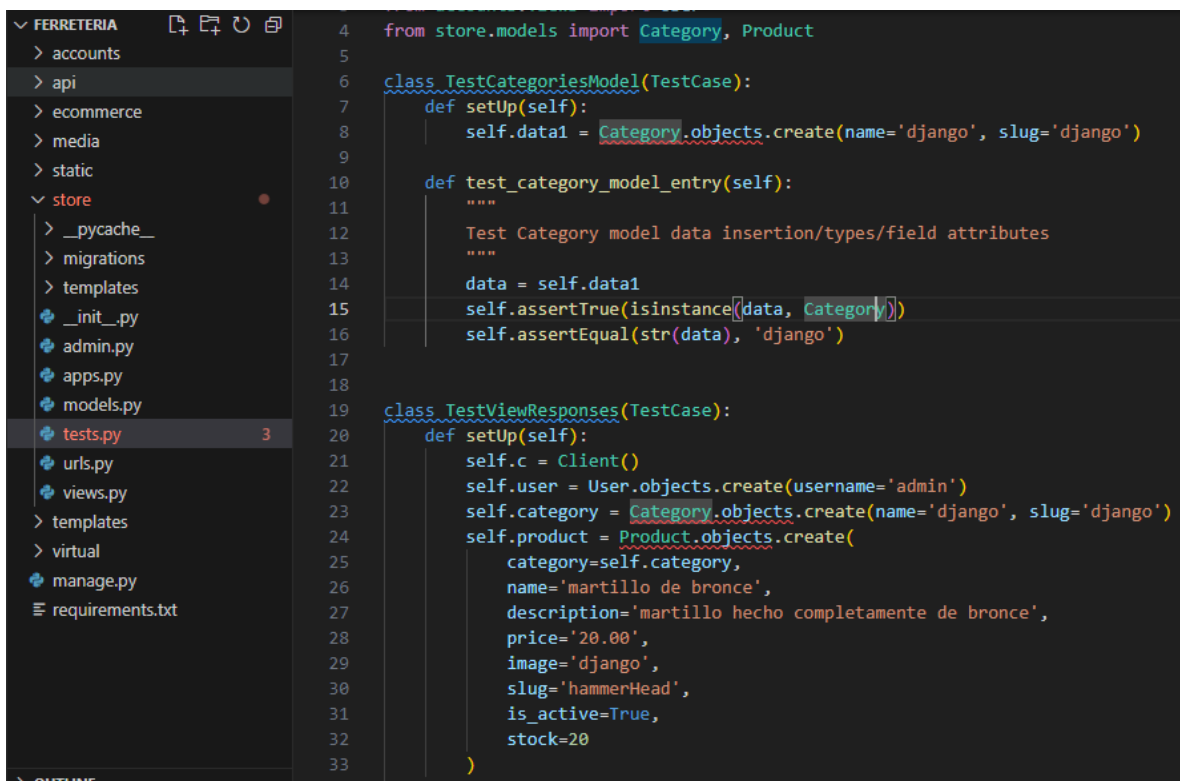
**Importancia:** Permite al administrador modificar las categorías existentes, manteniendo la tienda actualizada.

- **Prueba:** Eliminación de Categoría:

**Descripción:** Esta prueba verifica que se puede eliminar una categoría de la base de datos.

**Verificación:** Se asegura que la categoría eliminada ya no existe en la base de datos.

**Importancia:** Garantiza que el administrador puede eliminar categorías que ya no son necesarias, manteniendo la tienda limpia y organizada.



```
4 from store.models import Category, Product
5
6 class TestCategoriesModel(TestCase):
7     def setUp(self):
8         self.data1 = Category.objects.create(name='django', slug='django')
9
10    def test_category_model_entry(self):
11        """
12        Test Category model data insertion/types/field attributes
13        """
14        data = self.data1
15        self.assertTrue(isinstance(data, Category))
16        self.assertEqual(str(data), 'django')
17
18
19 class TestViewResponses(TestCase):
20     def setUp(self):
21         self.c = Client()
22         self.user = User.objects.create(username='admin')
23         self.category = Category.objects.create(name='django', slug='django')
24         self.product = Product.objects.create(
25             category=self.category,
26             name='martillo de bronce',
27             description='martillo hecho completamente de bronce',
28             price='20.00',
29             image='django',
30             slug='hammerHead',
31             is_active=True,
32             stock=20
33         )
```

#### 4.1.3 Pruebas Unitarias para las Vistas index, product\_detail, y category\_list

**Objetivo:** Verificar que las vistas principales de la aplicación se comporten como se espera, mostrando la información correcta y respondiendo adecuadamente a las solicitudes.

- **Prueba:** Vista index:

**Descripción:** Esta prueba verifica que la vista index muestra correctamente los productos disponibles en la página principal.

**Verificación:** Se asegura que la vista index devuelve un estado HTTP 200 y que la lista de productos está presente en el contexto.

**Importancia:** Es la página de entrada principal para los usuarios, por lo que es crucial que funcione correctamente y muestre los productos disponibles.

- **Prueba:** Vista product\_detail:

**Descripción:** Esta prueba verifica que la vista product\_detail muestra correctamente los detalles de un producto específico cuando se accede a su URL.

**Verificación:** Se asegura que la vista product\_detail devuelve un estado HTTP 200 y que la información del producto está presente en el contexto.

**Importancia:** Proporciona a los usuarios la información detallada que necesitan sobre un producto antes de realizar una compra.

- **Prueba:** Vista category\_list:

**Descripción:** Esta prueba verifica que la vista category\_list muestra correctamente los productos que pertenecen a una categoría específica.

**Verificación:** Se asegura que la vista category\_list devuelve un estado HTTP 200 y que la lista de productos para la categoría está presente en el contexto.

**Importancia:** Facilita a los usuarios la navegación por categorías, mejorando la experiencia de compra al permitirles encontrar productos específicos más fácilmente.

```
34
35
36 def test_index_view(self):
37     response = self.c.get(reverse('index'))
38     self.assertEqual(response.status_code, 200)
39     self.assertTemplateUsed(response, 'store/index.html')
40     self.assertContains(response, self.product.name)
41
42 def test_product_detail_view(self):
43     response = self.c.get(reverse('product', args=[self.product.slug]))
44     self.assertEqual(response.status_code, 200)
45     self.assertTemplateUsed(response, 'store/detail.html')
46     self.assertContains(response, self.product.name)
47
48 def test_category_list_view(self):
49     response = self.c.get(reverse('category', args=[self.category.slug]))
50     self.assertEqual(response.status_code, 200)
51     self.assertTemplateUsed(response, 'store/category.html')
52     self.assertContains(response, self.category.name)
53
```

#### 4.1.4 Pruebas unitarias para las urls

**Objetivo:** Verificar que las urls estén correctamente mapeadas a sus respectivas vistas y que respondan adecuadamente.

- **Prueba:** url / (Vista index):

**Descripción:** Esta prueba verifica que la URL / accede correctamente a la vista index.

**Verificación:** Se asegura que la URL / devuelve un estado HTTP 200 y que la vista index se carga correctamente.

**Importancia:** Es fundamental que la URL principal del sitio funcione correctamente, ya que es la primera página que ven los usuarios.

- **Prueba:** URL /product/<slug>/ (Vista product\_detail):

**Descripción:** Esta prueba verifica que la URL /product/<slug>/ accede correctamente a la vista product\_detail para un producto específico.

**Verificación:** Se asegura que la URL /product/<slug>/ devuelve un estado HTTP 200 y que la vista product\_detail se carga con los datos correctos del producto.

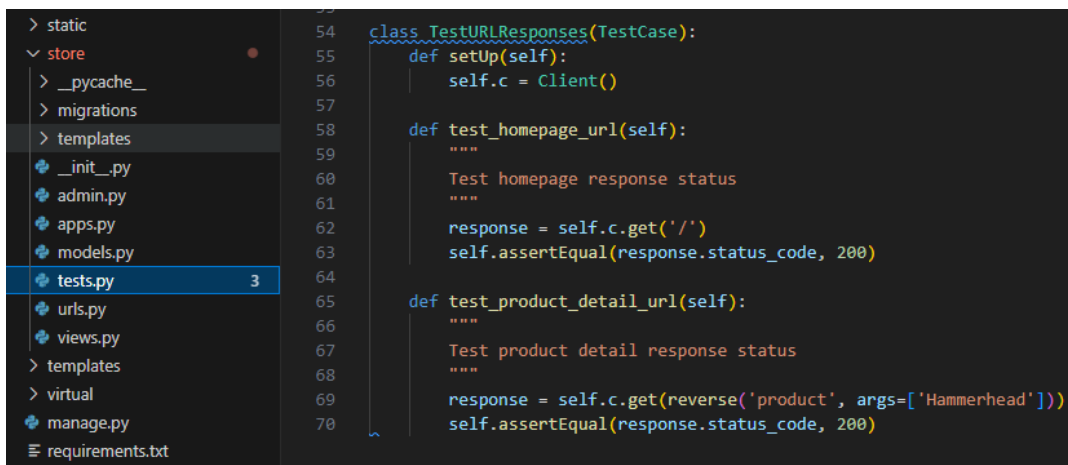
**Importancia:** Asegura que los usuarios pueden acceder a los detalles de productos específicos, lo cual es clave para la experiencia de compra.

- **Prueba:** URL /category/<slug>/ (Vista category\_list):

**Descripción:** Esta prueba verifica que la URL /category/<slug>/ accede correctamente a la vista category\_list para una categoría específica.

**Verificación:** Se asegura que la URL /category/<slug>/ devuelve un estado HTTP 200 y que la vista category\_list se carga con los productos correctos de la categoría.

**Importancia:** Es esencial para la navegación por categorías, permitiendo a los usuarios encontrar productos agrupados de manera lógica.



```
54 class TestURLResponses(TestCase):
55     def setUp(self):
56         self.c = Client()
57
58     def test_homepage_url(self):
59         """
60         Test homepage response status
61         """
62         response = self.c.get('/')
63         self.assertEqual(response.status_code, 200)
64
65     def test_product_detail_url(self):
66         """
67         Test product detail response status
68         """
69         response = self.c.get(reverse('product', args=['Hammerhead']))
70         self.assertEqual(response.status_code, 200)
```

#### 4.1.5 Matriz de Datos para Pruebas unitarias

La matriz de datos para pruebas documenta cada prueba realizada, identificando los artefactos y los resultados esperados.

ID de Prueba	Descripción de la Prueba	Datos de Entrada	Resultado Esperado	Resultado Obtenido	Estado
<b>TST01</b>	Verificar que una categoría se crea correctamente en el modelo Category	Modelo Category	La categoría se crea correctamente y puede ser recuperada desde la base de datos	La categoría se creó y recuperó correctamente	Aprobado
<b>TST02</b>	Probar que la vista index muestra correctamente los productos	vista "index"	La vista index debe devolver un estado 200 y mostrar los productos disponibles	La vista devolvió un estado 200 y mostró los productos disponibles	Aprobado
<b>TST03</b>	Probar que la vista product_detail muestra los detalles de un producto	"product_detail"	La vista product_detail debe devolver un estado 200 y mostrar los detalles del producto seleccionado	La vista devolvió un estado 200 y mostró los detalles del producto	Aprobado
<b>TST04</b>	Probar que la vista category_list muestra los productos de una categoría	"category_list"	La vista category_list debe devolver un estado 200 y mostrar los productos correspondientes a la categoría	La vista devolvió un estado 200 y mostró los productos de la categoría	Aprobado
<b>TST05</b>	Verificar que la URL / accede correctamente a la vista index	"/"	La URL / debe dirigir a la vista index y devolver un estado 200	La URL / dirigió correctamente a la vista index y devolvió un estado 200	Aprobado
<b>TST06</b>	Verificar que la URL /product/<slug>/ accede a la vista product_detail	"/product/<slug>"	La URL /product/<slug>/ debe dirigir a la vista product_detail y devolver un estado 200	La URL /product/<slug>/ dirigió correctamente a la vista product_detail y devolvió un estado 200	Aprobado
<b>TST07</b>	Verificar que la URL /category/<slug>/ accede a la vista category_list	"/category/<slug>/"	La URL /category/<slug>/ debe dirigir a la vista category_list y devolver un estado 200	La URL /category/<slug>/ dirigió correctamente a la vista category_list y devolvió un estado 200	Aprobado



#### 4.1.6 Cómo Ejecutar las Pruebas

##### Configurar el Entorno:

- Asegurarse de tener Django configurado correctamente en el entorno de desarrollo.
- Verificar que la base de datos esté conectada y configurada.

##### Ejecutar Pruebas Unitarias:

- Usar el comando de Django para ejecutar las pruebas unitarias: **python manage.py test**

```
(virtual) PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria> python manage.py test
Found 6 test(s).
Creating test database for alias 'default'...
```

- Las pruebas se ejecutarán automáticamente y los resultados se mostrarán en la consola.

```
-----
Ran 6 tests in 17.518s

FAILED (failures=1)
Destroying test database for alias 'default'...
(virtual) PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria>
```

##### Verificar los Resultados:

- Comparar los resultados obtenidos con los resultados esperados.
- Marcar las pruebas como "Aprobadas" o "Fallidas" en la matriz de datos.

#### 4.2 Pruebas de aceptación con Selenium IDE

##### Objetivo

Asegurar que el inicio de sesión funcione correctamente tanto para credenciales válidas como inválidas, y validar la funcionalidad de la cesta de la compra, incluyendo la adición y eliminación de artículos, el cálculo correcto del precio total y la persistencia de la cesta.

## Categorización

- **Pruebas de aceptación:** Se verifica que el sistema cumpla con los requisitos y comportamientos esperados desde el punto de vista del usuario.
- **Pruebas funcionales:** Se enfoca en validar que las funcionalidades específicas del sistema operen según lo previsto.

## Recursos utilizados

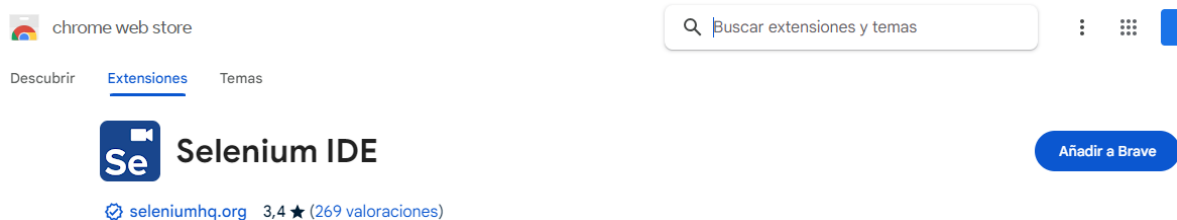
- **Selenium IDE:** Para la automatización de las pruebas de aceptación en el navegador.

## Entornos de Ejecución

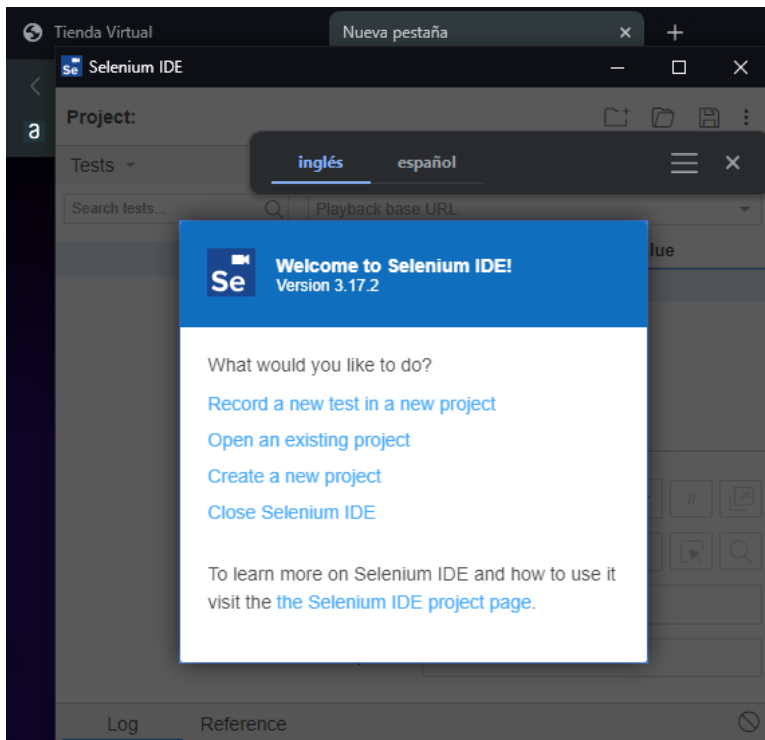
- **Ejecución local:** Las pruebas se ejecutarán en un entorno local, utilizando el navegador configurado con Selenium IDE para simular y verificar las acciones de usuario en la aplicación.

### 4.2.1 Configuración la aplicación

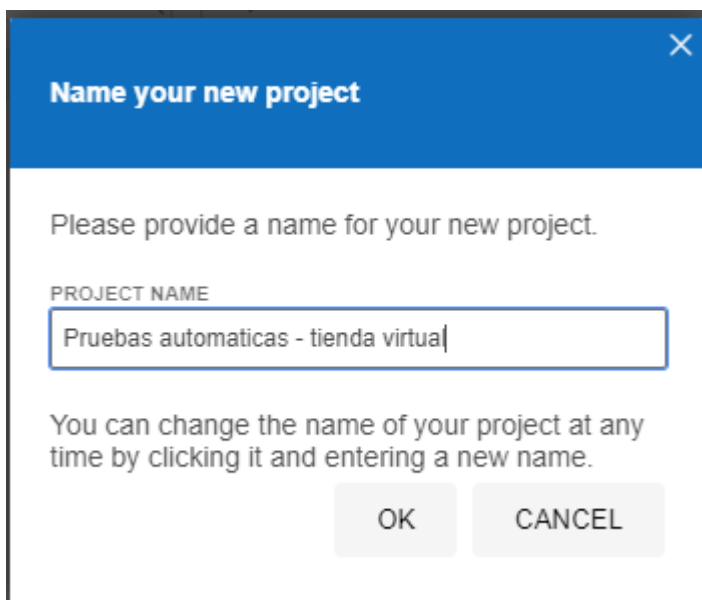
Ir a la tienda de aplicaciones de nuestro navegador e instalar la extensión Selenium IDE.



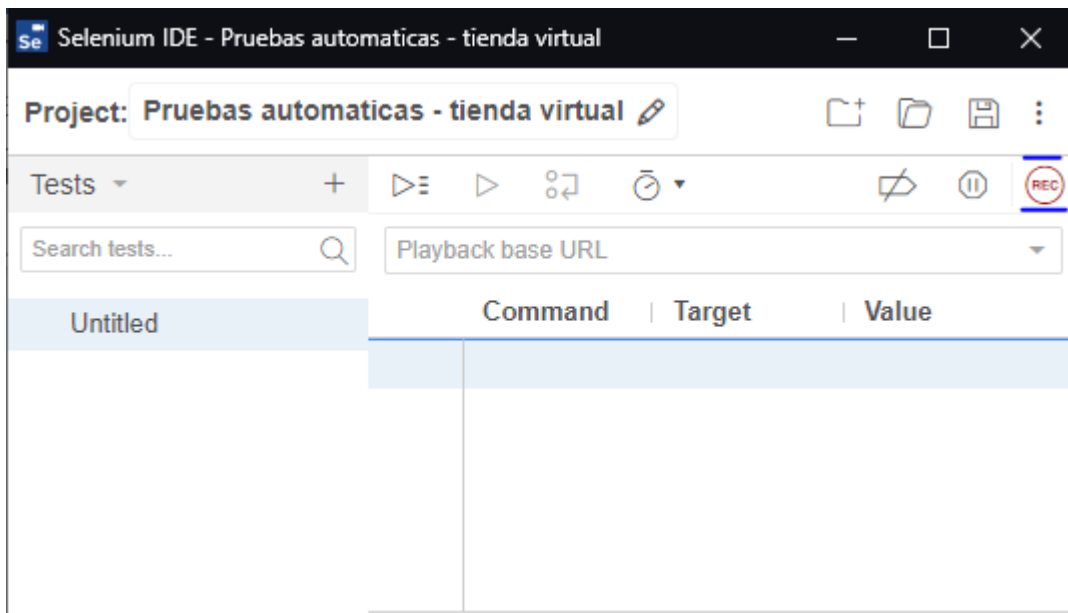
Para ejecutar la extensión damos clic en el icono de Selenium IDE que se agregó a nuestro navegador.



Damos clic en la opción 3 Create a new Project. Ya que vamos a probar la tienda virtual nombramos nuestro proyecto de la siguiente forma:



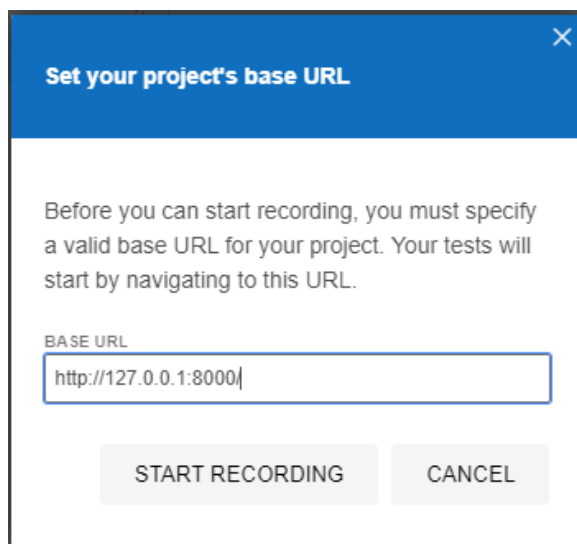
En la interfaz de Selenium, ubicamos la función "Record and Play". Esta función nos permite grabar las acciones que realizamos manualmente en la página web y luego reproducirlas automáticamente. Así, podemos ejecutar casos de prueba manuales y luego automatizarlos.



#### 4.2.2 Ejecución de la prueba

##### Pasos

1. Presionamos el botón "REC" para iniciar la grabación de las acciones. Luego, escribimos la URL de la página que queremos probar con Selenium. El objetivo de esta etapa es simular el proceso de inicio de sesión de un usuario que ya se encuentra registrado.



2. Agregamos manualmente los datos de inicio de sesión de este usuario.

# Inicia sesión

Usuario

Marquitos

Contraseña

.....

Ingresar

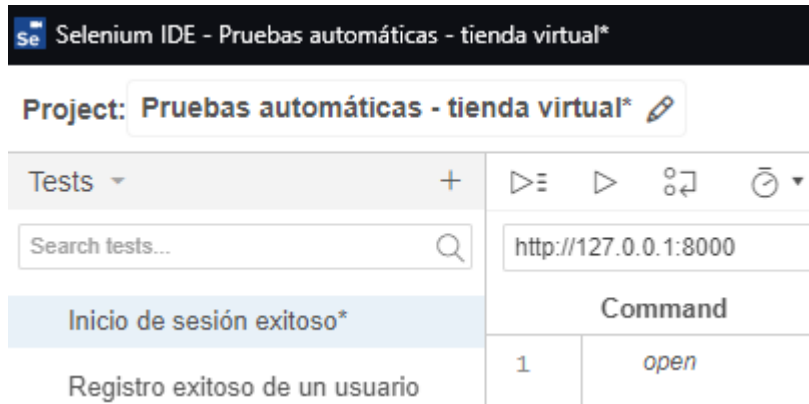
Se

Selenium IDE is recording...

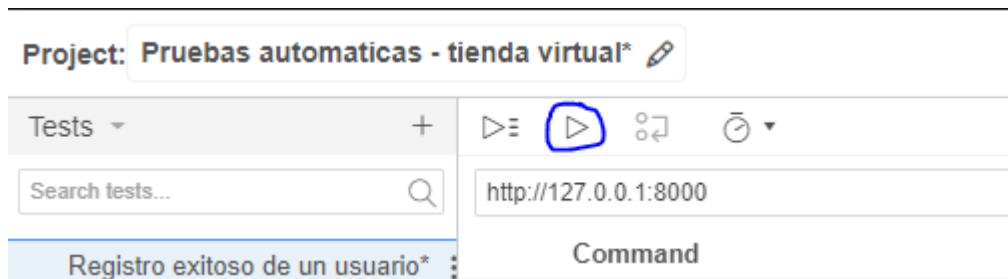
3. En esta sección se muestra las acciones que ejecutamos sobre la página.

	Command	Target	Value
1	open	/	
2	set window size	516x728	
3	click	css=.navbar-toggler-icon	
4	click	linkText=Ingresar	
5	click	name=username	
6	type	name=username	usertest10
7	click	name=password	
8	type	name=password	prueba111
9	click	css=.btn-primary	

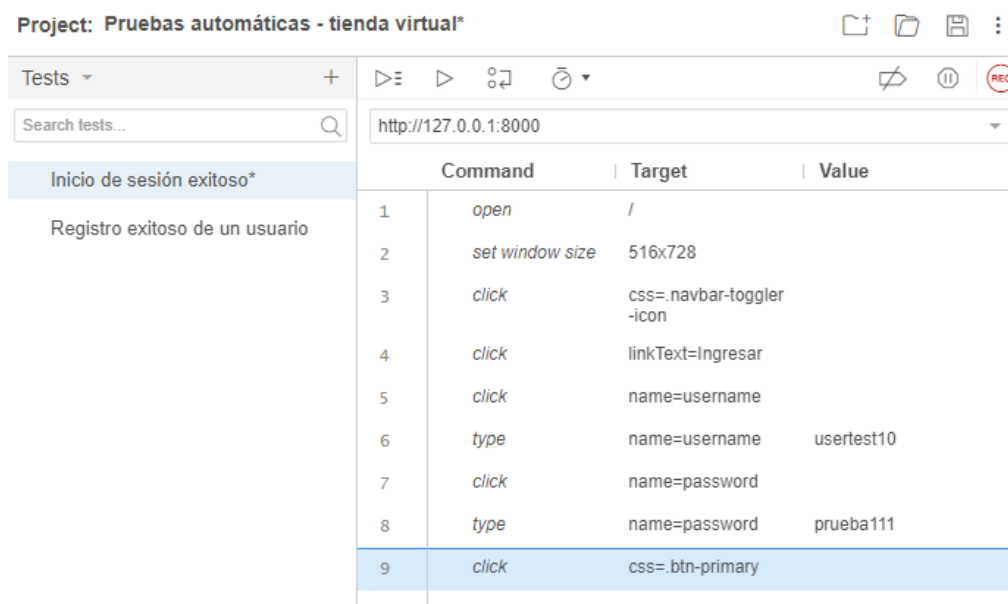
Detenemos la grabación pulsando el botón rec y elegimos un nombre para nuestra prueba.



4. Gracias a la función de grabación de Selenium, la prueba que acabamos de realizar puede ser ejecutada automáticamente. Para ejecutar el script generado, seleccionamos la prueba grabada y hacemos clic en el botón "Play Current Test".



Durante la reproducción, Selenium repetirá los pasos grabados, incluyendo el ingreso de la misma información del usuario.



## 4.3 Pruebas api

### Objetivo

Comprobar la integridad de la información enviada por la API y validar que los endpoints respondan correctamente según las especificaciones definidas. Esto incluye verificar que los datos recibidos sean precisos, completos y que los endpoints funcionen de acuerdo con los contratos de la API.

### Categorización:

- **Pruebas de integridad de datos:** Asegurarse de que los datos enviados y recibidos por la API sean correctos y consistentes.
- **Pruebas de funcionalidad de endpoints:** Verificar que cada endpoint de la API funcione según lo especificado en la documentación, incluyendo la correcta gestión de parámetros, respuestas esperadas y códigos de estado HTTP.

### Recursos utilizados:

- **Postman:** Herramienta principal utilizada para enviar solicitudes a la API, verificar las respuestas, validar la integridad de los datos y documentar los resultados de las pruebas.

### Entornos de Ejecución:

- **Ejecución local:** Las pruebas se ejecutarán en un entorno local, utilizando Postman enviar solicitudes a la API.

## 4.1 Ejecución de las pruebas

### Pasos:

1. Cree una nueva solicitud en Postman para el EndPoint de la API de productos.
2. Envíe una solicitud para obtener los datos del producto.
3. Utilizar los snippets de postman para analizar los datos de respuesta.
4. Verifique la integridad de los datos.

### API ENDPOINT

- api/products

## Lista de productos almacenados en la base de datos

GET

http://127.0.0.1:8000/api/products

Send

ParamsAuthorizationHeaders (7)BodyPre-request ScriptTestsSettingsCookies


BodyCookiesHeaders (10)Test Results (1/1)

Status: 200 OKTime: 68 msSize: 1.05 KBSave as example

PrettyRawPreviewVisualizeJSON

```
1  [
2    {
3      "id": 7,
4      "name": "Flexometro",
5      "price": 31000,
6      "description": "Instrumento de medición formado por una delgada cinta metálica flexible y auto enrollable en una carcasa, que puede ser tanto metálica como de plástico, y equipada con un sistema de freno."
7    },
8    {
9      "id": 8,
10     "name": "Escalera Multifuncional De 12 Pasos En Aluminio",
11     "price": 450000,
12     "description": "Escalera fabricada en aluminio que puede soportar un peso máximo de 150 kg"
13   },
14   {
15     "id": 9,
16     "name": "Mazo De Goma 20 Oz Stanley",
17     "price": 31000,
18     "description": "Mazo con cabeza fabricada en neopreno y mango de madera"
19   },
20   {
21     "id": 10,
22     "name": "Martillo de goma",
23     "price": 10000,
24     "description": "Herramienta manual utilizada para clavar, calzar partes o romper objetos. Fácil manejo gracias a su mango en fibra de vidrio y caucho."
25   }
26 ]
```

## Productos en la tienda




Instrumento de medida

**Flexometro**

\$31000

Ver producto




Escaleras

**Escalera Multifuncional De 12 Pasos En Aluminio**

\$450000

Ver producto




Martillos y Mazos

**Mazo De Goma 20 Oz Stanley**

\$31000

Ver producto



Martillos y Mazos

**Martillo de goma**

\$10000

Ver producto



## Mensaje de error cuando se busca un id que no existe

The screenshot shows a Postman interface with a GET request to `http://127.0.0.1:8000/api/products/70`. The status is **404 Not Found**. The response body is a JSON object: `{ "detail": "No Product matches the given query." }`.

Key	Value	Description

## Prueba: Comprobación del código de estado de la respuesta

The screenshot shows a Postman interface with a GET request to `http://127.0.0.1:8000/api/products`. The **Tests** tab is active, showing a script:

```
1 pm.test("El código de estado es 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

The status bar at the bottom indicates "Sending request..."

## Resultado de la prueba

The screenshot shows the same Postman interface, but now the **Test Results** tab is active. The test passed, showing a green **PASS** status and the message "El código de estado es 200".

All	Passed	Skipped	Failed

**Resultados esperados:**

- La API debería devolver los datos de los productos solicitados.
- La API devuelve un código de estado 200.

**Resultados obtenidos:**

- LA API products recibe la solicitud y envía de vuelta una respuesta.
- En respuesta a la petición realizada (GET), la API envía la información solicitada junto a un código de respuesta 200.