17-8-2023

Proyecto: Django Web App – Comunicación Ciudadana

Creando una aplicación desde el framework de python

Contenido

[Introducción 2](#_Toc143720548)

[Problemática 3](#_Toc143720549)

[Solución: “Seguimiento Ciudadano” 3](#_Toc143720550)

[Descripción del proyecto 3](#_Toc143720551)

[Características 3](#_Toc143720552)

[Módulos 4](#_Toc143720553)

[Pantallas del sistema 4](#_Toc143720554)

[Diagrama de casos de uso 6](#_Toc143720555)

[Plan de pruebas Django Python con Selenium 7](#_Toc143720556)

[Objetivo del plan de pruebas 7](#_Toc143720557)

[Alcance del plan de pruebas 7](#_Toc143720558)

[Estrategia de pruebas 7](#_Toc143720559)

[Casos de prueba 8](#_Toc143720560)

[Caso de prueba 1: Inicio de sesión 8](#_Toc143720561)

[Caso de prueba 2: Registro de solicitud 8](#_Toc143720562)

[Caso de prueba 3: Integración API REST 12](#_Toc143720563)

[Caso de prueba 4: Pruebas unitarias 15](#_Toc143720564)

[Conclusiones 17](#_Toc143720565)

# Introducción

Django es un framework de desarrollo web de alto nivel y de código abierto basado en Python. Ofrece una estructura robusta y completa para la creación de aplicaciones web, permitiendo a los desarrolladores ahorrar tiempo y esfuerzo al proporcionar una serie de características y herramientas integradas.

Además, Django proporciona una administración automática del sitio web, que permite gestionar fácilmente el contenido y los usuarios sin necesidad de escribir código adicional. También incluye un sistema de autenticación y autorización integrado para la gestión de usuarios y sesiones.

La flexibilidad de Django permite la creación de aplicaciones web de cualquier tamaño y complejidad, desde pequeños sitios hasta grandes proyectos empresariales. Su enfoque en la seguridad, el rendimiento y la escalabilidad lo convierte en una opción popular entre los desarrolladores de Python.

En resumen: Django es una poderosa herramienta de desarrollo web que simplifica el proceso de creación de aplicaciones web, proporcionando una estructura sólida y características integradas para agilizar el desarrollo. Con su comunidad activa y su amplia documentación, Django se ha convertido en una opción confiable para aquellos que buscan construir aplicaciones web rápidas, seguras y escalables en **Python**.

# Problemática

Debido a la enorme cantidad de solicitudes que llegan a las organizaciones gubernamentales, se enfrentan a un desafío considerable en términos de eficiencia y efectividad al momento de gestionar, dar seguimiento y responder a cada una de estas solicitudes. Esta situación se presenta debido a la falta de un sistema adecuado que permita manejar de manera óptima la gran carga de trabajo que implica atender todas las solicitudes recibidas.

# Solución: “Seguimiento Ciudadano”

Descripción del proyecto

“Seguimiento ciudadano” es una aplicación basada en el estándar open311 que permita a la ciudadanía informar, generar solicitudes, compartir comentarios y hacer preguntas acerca de problemas como:

1. Necesidades de reparación de calles y aceras
2. Recolección de basura y residuos
3. Mantenimiento de parques y alumbrado público
4. Gestión de emergencias a desastres naturales

Además, esta aplicación permite a los gobiernos locales y servidores públicos:

* Impulsar flujos de trabajo eficientes
* Manejo adecuado de dichas solicitudes
* Rastreo e histórico de solicitudes

## Características

1. Autenticación Web segura (usuario y/o Token JWT)
2. Manejo de solicitudes
3. Seguimiento de solicitudes
4. Endpoints API Implementados
5. Pruebas Funcionales, Unitarias e Integradoras incluidas

## Módulos

El sistema cuenta con un manejo de usuarios, permisos por usuario, y autenticación mediante tokens de seguridad, que permiten a los usuarios crear solicitudes de servicios públicos o denuncia de problemas con la infraestructura del estado.

## Pantallas del sistema



Ilustración Pantalla de inicio

|  |  |
| --- | --- |
| Ilustración Pantalla de registro | Ilustración Pantalla de Inicio de sesión |



Ilustración Pantalla de lista de solicitudes

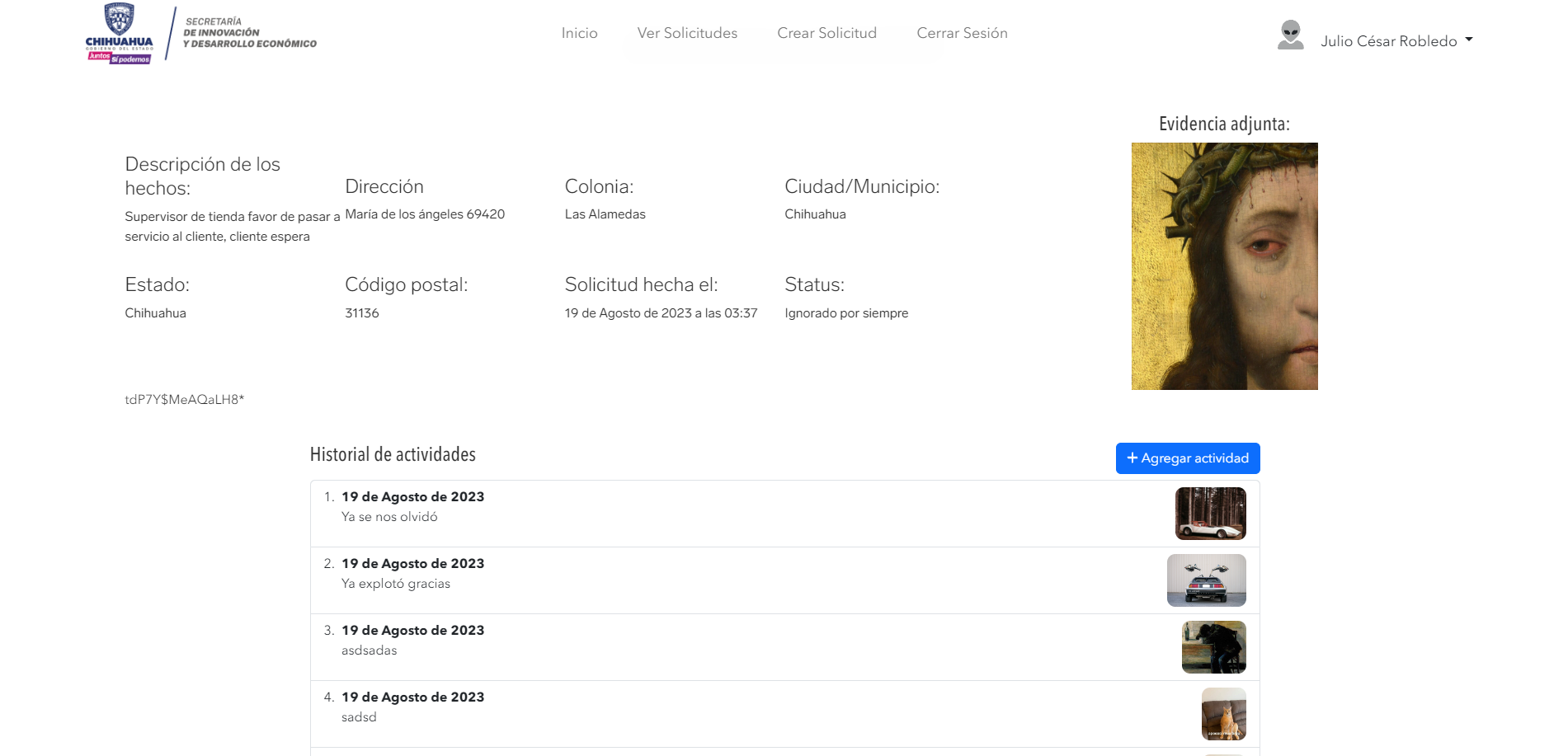
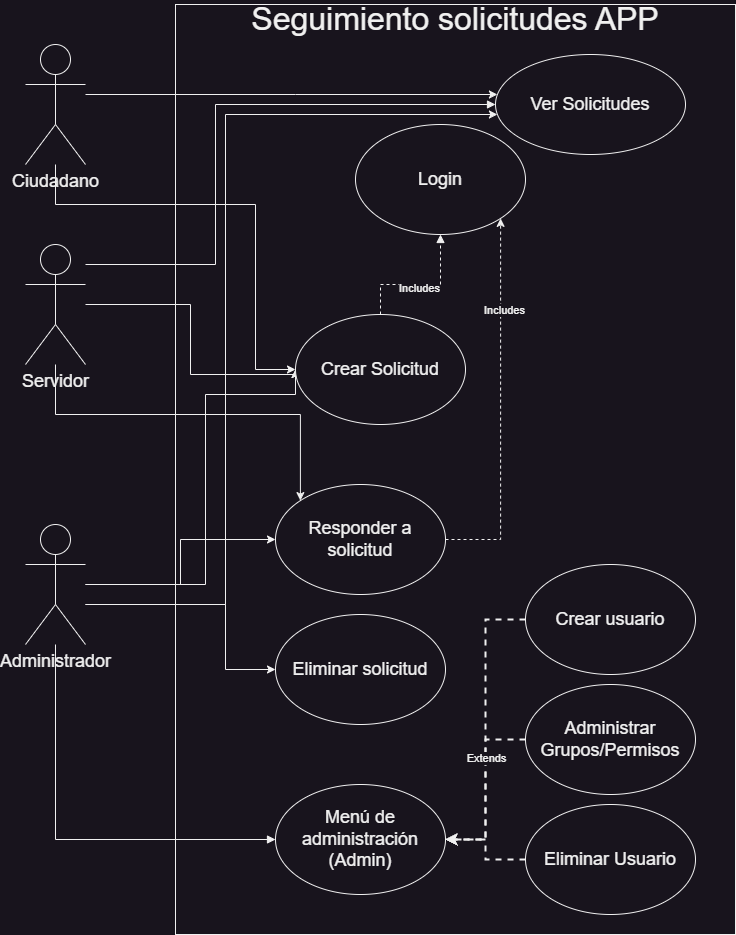


Ilustración Pantalla seguimiento solicitud

## Diagrama de casos de uso



# Plan de pruebas Django Python con Selenium

## Objetivo del plan de pruebas

El objetivo de este plan de pruebas es verificar el correcto funcionamiento de las funcionalidades implementadas en una aplicación web desarrollada con Django utilizando el framework de pruebas Selenium. Se verificará que todas las páginas y elementos de la interfaz de usuario funcionen correctamente y cumplan con los requisitos establecidos.

## Alcance del plan de pruebas

Este plan de pruebas cubrirá las siguientes áreas de la aplicación web:

* Inicio de sesión
* Registro de usuarios
* Funcionalidades principales de la aplicación

## Estrategia de pruebas

Se utilizará Selenium para realizar pruebas automatizadas de la interfaz de usuario. Se crearán casos de prueba que simularán interacciones del usuario con la aplicación web, verificando que los resultados obtenidos sean los esperados.

## Casos de prueba

### Caso de prueba 1: Inicio de sesión

* Descripción: Verificar que un usuario pueda iniciar sesión correctamente.
* Pasos:
  1. Abrir el navegador y acceder a la página de inicio de sesión.
  2. Ingresar las credenciales de un usuario válido.
  3. Hacer clic en el botón de inicio de sesión.
* Resultado esperado: El usuario debe ser redirigido a la página principal y se debe mostrar un mensaje de bienvenida.
* Diseño de caso de prueba: Tanto este caso de prueba como el siguiente, son parte del mismo ecosistema, por lo que los resultados de ambos se muestran en el siguiente caso de prueba.

### Caso de prueba 2: Registro de solicitud

* Descripción: Verificar que un usuario pueda registrarse correctamente.
* Pasos:
  1. Abrir el navegador y acceder a la página de inicio de sesión.
  2. Ingresar las credenciales de un usuario válido.
  3. Hacer clic en el botón de inicio de sesión.
  4. Ir a la página de creación de solicitudes
  5. Llenar los campos
  6. Guardar una nueva solicitud
  7. Verificar en lista de solicitudes
* Resultado esperado: El usuario debe ser redirigido a la página de inicio de sesión y se debe mostrar un mensaje de éxito.
* Desarrollo de prueba:

class test\_funcional\_selenium(StaticLiveServerTestCase):

    def setUp(self):

        self.options = Options()

        self.options.page\_load\_strategy = 'normal'

        self.browser = webdriver.Chrome()

        self.browser.implicitly\_wait(2)

        self.browser.maximize\_window

        self.wait = WebDriverWait(self.browser, timeout=3)

        self.usuario=User.objects.create\_user('jrobledotest', password='Jr1811De')

        self.usuario.is\_superuser=True

        self.usuario.is\_staff=True

        self.usuario.first\_name = 'Test'

        self.usuario.last\_name = 'prueba'

        self.usuario.save()

        self.zipcode = PostalCode.objects.create(d\_codigo='01140', d\_asenta='José Maria Pino Suárez',D\_mnpio='Álvaro Obregón', d\_ciudad='Ciudad de México', c\_estado= '09',c\_oficina= '01131',c\_tipo\_asenta= '09',c\_mnpio= '010',id\_asenta\_cpcons= '0060',d\_zona= 'Urbano',c\_cve\_ciudad= '01',d\_CP= '01131')

        self.zipcode.save()

        self.tiposoli = tiposolicitud.objects.create(nombreTipoSolicitud= "Tipo Solicitud de prueba")

        tiposolicitud.objects.create(nombreTipoSolicitud= "Tipo Solicitud de prueba 2")

        tiposolicitud.objects.create(nombreTipoSolicitud= "Tipo Solicitud de prueba 3")

        self.solicitud = Solicitudes.objects.create(tipo\_solicitud=self.tiposoli,descripcion = "Solicitud de prueba creado por tests ",street\_address = "Che Guevara",bld\_number = "814",city= "Chihuahua",state= "Chihuahua",country="México",zip\_code=31126,colonia="Tierra y Libertad",status="Prueba")

        self.url\_nueva\_solicitud = reverse('seguimiento\_ciudadano:nueva\_solicitud')

        self.url\_solicitudes = reverse('seguimiento\_ciudadano:Lista\_solicitudes')

        self.url\_index = reverse('seguimiento\_ciudadano:index')

        self.descripcion = "Solicitud de prueba"

    def test\_1\_selenium\_login(self):

        self.browser.maximize\_window()

        print(self.live\_server\_url)

        self.browser.get(self.live\_server\_url)

        time.sleep(1)

        print(self.browser.find\_element(By.XPATH, "//a[contains(@href,'login')]//parent::button"))

        self.browser.find\_element(By.XPATH, "//a[contains(@href,'login')]//parent::button").click()

        self.browser.find\_element(By.ID, "username").send\_keys('jrobledotest')

        self.browser.find\_element(By.NAME, "password").send\_keys('Jr1811De')

        time.sleep(1)

        self.browser.find\_element(By.XPATH, "//button[@type='submit']").click()

        self.wait.until(lambda d : self.browser.find\_element(By.XPATH,"//div[@class='header-nav']//child::a[last()]//div").is\_displayed)

        print(self.browser.find\_element(By.XPATH,"//div[@class='header-nav']//child::a[last()]//div").text)

        self.assertEqual(self.browser.find\_element(By.XPATH,"//div[@class='header-nav']//child::a[last()]//div").text, 'Cerrar Sesión')

        print(self.browser.find\_element(By.ID,"btn-usuario").text)

        self.assertIn(self.browser.find\_element(By.ID,"btn-usuario").text, 'Test Prueba')

        print(f'{self.id()} pass')

    def test\_2\_selenium\_crear\_solicitud(self):

        self.browser.maximize\_window()

        print(self.live\_server\_url)

        self.browser.get(self.live\_server\_url)

        time.sleep(1)

        print(self.browser.find\_element(By.XPATH, "//a[contains(@href,'login')]//parent::button"))

        self.browser.find\_element(By.XPATH, "//a[contains(@href,'login')]//parent::button").click()

        self.browser.find\_element(By.ID, "username").send\_keys('jrobledotest')

        self.browser.find\_element(By.NAME, "password").send\_keys('Jr1811De')

        time.sleep(1)

        self.browser.find\_element(By.XPATH, "//button[@type='submit']").click()

        self.wait.until(lambda d : self.browser.find\_element(By.XPATH,"//div[@class='header-nav']//child::a[last()]//div").is\_displayed)

        print(self.browser.find\_element(By.XPATH,"//div[@class='header-nav']//child::a[last()]//div").text)

        self.assertEqual(self.browser.find\_element(By.XPATH,"//div[@class='header-nav']//child::a[last()]//div").text, 'Cerrar Sesión')

        print(self.browser.find\_element(By.ID,"btn-usuario").text)

        self.assertIn(self.browser.find\_element(By.ID,"btn-usuario").text, 'Test Prueba')

        # Seleccionamos la opción para crear una nueva solicitud

        self.browser.find\_element(By.XPATH, "//div[@class='header-content']//descendant::div[text()='Crear Solicitud']//parent::a").click()

        tipo = Select(self.browser.find\_element(By.ID,'id\_tipo\_solicitud'))

        time.sleep(2)

        tipo.select\_by\_index(2)

        self.browser.find\_element(By.ID,'id\_descripcion').send\_keys(self.descripcion)

        time.sleep(2)

        self.browser.find\_element(By.ID,'id\_street\_address').send\_keys('Ningún lugar')

        self.browser.find\_element(By.ID,'id\_bld\_number').send\_keys('13')

        unfocushelper = self.browser.find\_element(By.ID,'id\_apt\_number')

        unfocushelper.send\_keys('000')

        self.browser.find\_element(By.ID,'id\_zip\_code').send\_keys('01140')

        unfocushelper.click()

        unfocushelper.send\_keys(Keys.SHIFT)

        self.browser.find\_element(By.ID,'id\_descripcion').send\_keys(Keys.TAB)

        self.browser.find\_element(By.ID,'id\_descripcion').send\_keys(Keys.TAB)

        time.sleep(5) #esperamos a que el javascript cargue las colonias del zip code

        colonia = Select(self.browser.find\_element(By.ID,'id\_colonia'))

        for option in colonia.options:

            print(option)

            print(option.id)

            print(option.text)

        time.sleep(5) #esperamos a que el javascript cargue las colonias del zip code

        colonia.select\_by\_value(self.zipcode.d\_asenta)

        print(self.browser.find\_element(By.ID,'id\_state').text)

        self.assertEqual(self.browser.find\_element(By.NAME,'state').get\_attribute('value'), self.zipcode.D\_mnpio)

        self.browser.find\_element(By.XPATH,"//button[@type='submit']").click()

        time.sleep(2)

        print(self.browser.find\_element(By.CLASS\_NAME,"toast-body"))

        self.assertTrue(self.browser.find\_element(By.CLASS\_NAME,"toast-body").is\_displayed)

        self.assertEquals(self.browser.find\_element(By.CLASS\_NAME,"toast-body").text,"Se guardo correctamente")

        self.browser.find\_element(By.XPATH,"//div[text()='Ver Solicitudes']//parent::a").click()

        time.sleep(2)# esperamos 2 segundos por si hay alguna otra petición en proceso

        self.assertIn(self.descripcion, str(self.browser.find\_element(By.XPATH, "//table").text))

        #end test

        print(f'{self.id()} pass')

* Resultados obtenidos

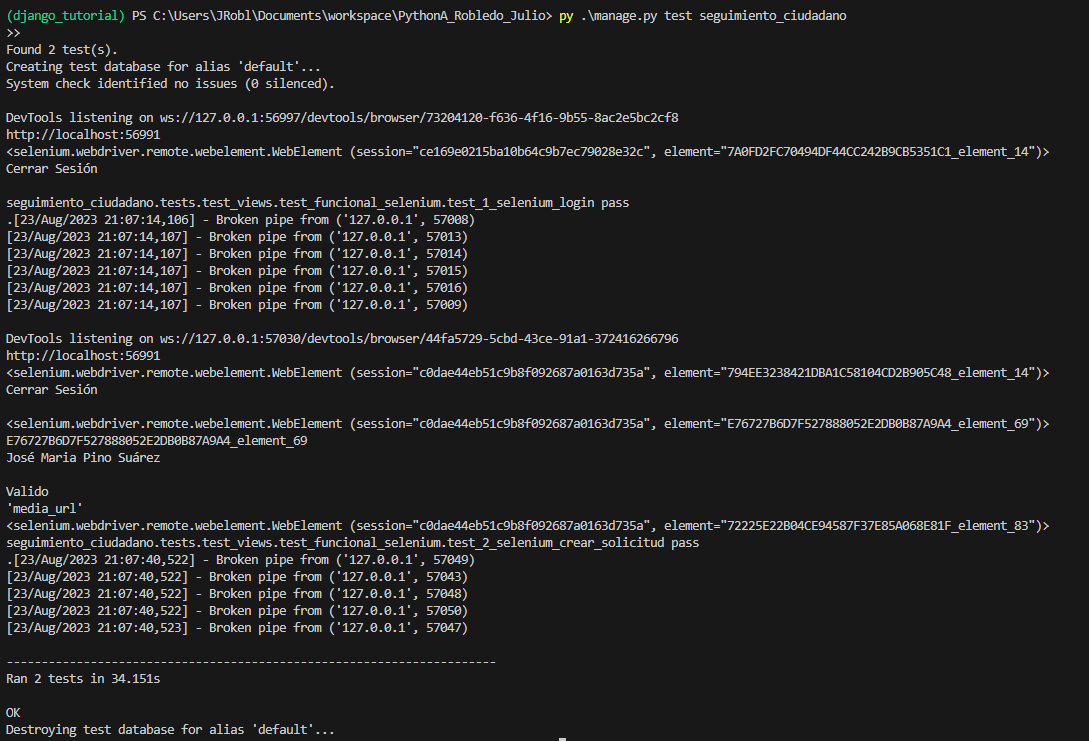


Figura Todas las pruebas fueron satisfactorias

### Caso de prueba 3: Integración API REST

* Descripción: Que la API REST del proyecto funcione correctamente
* Pasos:

1. Consultar lista protegida por token (401 no autorizado)
2. Generar un token para el usuario
3. Consultar lista protegida con token (200)
4. Guardar un nuevo registro

* Resultado esperado: La vista debe estar correctamente protegida ante consultas sin autorización, y que guarde correctamente un nuevo registro
* Desarrollo de prueba:

class TestIntegraciónAPI(TestCase):

    def setUp(self) -> None:

        self.usuario=User.objects.create\_user('jrobledotest', password="Jr1811De")

        self.usuario.is\_superuser=True

        self.usuario.is\_staff=True

        self.usuario.first\_name = 'Test'

        self.usuario.last\_name = 'prueba'

        self.usuario.save()

        self.Autor = catModels.Author.objects.create(first\_name="Howard Phillips", last\_name="Lovecraft" , date\_of\_birth="1890-08-20", date\_of\_death="1937-03-15")

        self.Autor = catModels.Author.objects.create(first\_name="Stephen", last\_name="King" , date\_of\_birth="1947-09-21")

        self.client = Client()

        self.url\_Authors = reverse('catalog:Autores')

        self.url\_api\_token = reverse('token\_obtain\_pair')

        self.access=""

        self.refresh=""

        return super().setUp()

    def test\_get\_authors\_and\_authorization(self):

        response = self.client.get(self.url\_Authors)

        print(response.content)

        print(response.status\_code)

        self.assertEquals(response.status\_code , 401) #No autorizado, necesita recibir el token e iniciar sesión

        ### Obtener Token

        authjson = {

            "username":self.usuario.username,

            "password":"Jr1811De",

        }

        authRequest = self.client.post(self.url\_api\_token, authjson, format="json")

        time.sleep(2)

        print(authRequest)

        print(authRequest.status\_code)

        print(authRequest.content)

        self.assertEqual(authRequest.status\_code,200)

        accessJson = json.loads(authRequest.content)

        self.access = str("Bearer "+str(accessJson["access"]))

        self.refresh = str(accessJson["refresh"])

        print(self.access)

        self.assertIn(accessJson["access"],self.access )

        ###

        ### lista de autores

        print("Getting author list...")

        time.sleep(1)

        clientWithToken = Client(HTTP\_AUTHORIZATION=self.access) #necesaria una instancia nueva para agregar al header el token de acceso, las otras formas que encontré (client.credentials, \*\*header o \*\*extra) no jalaron

        response = clientWithToken.get(self.url\_Authors)

        print(response.content)

        print(response.status\_code)

        self.assertEquals(response.status\_code , 200)

        ###

        ###agregar un autor

        jsonAuthor = {

            "first\_name": "Juana de",

            "last\_name": "Arco",

            "date\_of\_birth": "1968-08-17",

            "date\_of\_death": "2014-12-24"

        }

        POSTauthorRequest = clientWithToken.post(self.url\_Authors, data=jsonAuthor)

        print(POSTauthorRequest.content)

        print(POSTauthorRequest.status\_code)

        self.assertEquals(POSTauthorRequest.status\_code, 201)

        self.assertEqual(jsonAuthor["last\_name"],json.loads(POSTauthorRequest.content)["last\_name"])

        ### Prueba de objeto con response

        created\_author = catModels.Author.objects.get(first\_name="Juana de")

        self.assertEqual(created\_author.last\_name, jsonAuthor["last\_name"])

        print(f'{self.id()} pass')

* Resultados obtenidos:

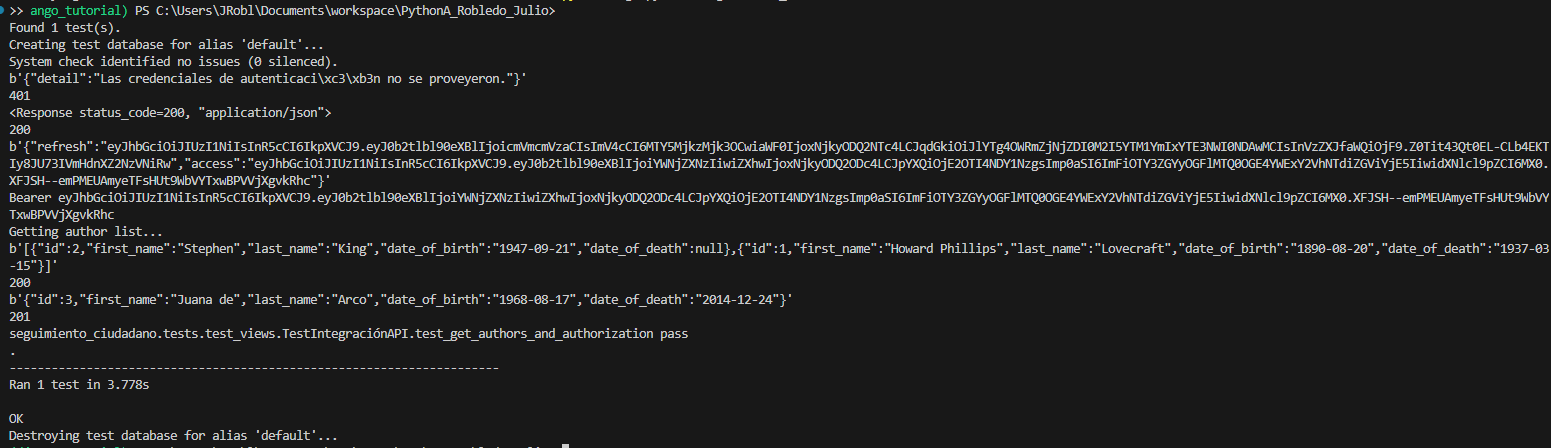


Figura Todas las pruebas fueron superadas

### Caso de prueba 4: Pruebas unitarias

* Descripción: Verificar que los componentes individuales de las vistas funcionen correctamente
* Pasos:
  1. Verificar los tipos de vista obtenidos
  2. Verificar vistas de clase con argumentos
  3. Verificar vistas protegidas con autenticación
  4. Verificar listas de datos desde DB
* Resultado esperado: El elemento debe ser editado correctamente y se debe mostrar un mensaje de éxito.

class unitTesting(TestCase):

    def setUp(self) -> None:

        self.usuario=User.objects.create\_user('jrobledotest', password='Jr1811De')

        self.usuario.is\_superuser=True

        self.usuario.is\_staff=True

        self.usuario.first\_name = 'Test'

        self.usuario.last\_name = 'prueba'

        self.usuario.save()

        self.tiposoli = tiposolicitud.objects.create(nombreTipoSolicitud= "Tipo Solicitud de prueba")

        self.solicitud = Solicitudes.objects.create(

            tipo\_solicitud=self.tiposoli,

            descripcion = "Solicitud de prueba creado por tests ",

            street\_address = "Che Guevara",

            bld\_number = "814",

            city= "Chihuahua",

            state= "Chihuahua",

            country="México",

            zip\_code=31126,

            colonia="Tierra y Libertad",

            status="Prueba",

            )

        self.url\_nueva\_solicitud = reverse('seguimiento\_ciudadano:nueva\_solicitud')

        self.url\_solicitudes = reverse('seguimiento\_ciudadano:Lista\_solicitudes')

        self.url\_index = reverse('seguimiento\_ciudadano:index')

        self.client = Client()

    def test\_view\_class(self):

        url = reverse('seguimiento\_ciudadano:Lista\_solicitudes\_api')

        # print(resolve(url))

        self.assertEqual(resolve(url).func.view\_class, lista\_solicitudes\_api)

        print(f'{self.id()} pass')

    def test\_view\_function(self):

        url = reverse('seguimiento\_ciudadano:Registro Usuario')

        # print(resolve(url))

        self.assertEqual(resolve(url).func.\_\_name\_\_, 'signup')

        self.assertEqual(resolve(url).func, signup)

        print(f'{self.id()} pass')

    def test\_view\_class\_with\_args(self):

        url = reverse('seguimiento\_ciudadano:Seguimiento', kwargs={'request\_id':2})

        # print(resolve(url))

        self.assertEqual(resolve(url).func.view\_class, seguimiento\_solicitud)

        print(f'{self.id()} pass')

    def test\_solicitudes\_get\_crear\_solicitud\_form(self):

        response = self.client.get(self.url\_nueva\_solicitud)

        # Verificamos que el estado sea 200 (OK), es decir todo esta bien

        self.assertEquals(response.status\_code , 401)# la vista nueva\_solicitud sólo la pueden ver usuarios autenticados, arroja 401 Unauthorized

        # Verificar que en la lista de Solicitudes esté el objet Solicitud creado en setUp

        print(f'{self.id()} pass')

    def test\_solicitudes\_get\_all\_solicitudes(self):

        response = self.client.get(self.url\_solicitudes)

        self.assertEquals(response.status\_code , 200)# la vista nueva\_solicitud sólo la pueden ver usuarios autenticados, arroja 401 Unauthorized

        self.assertEqual(response.context['solicitudes'].first(), self.solicitud)

        self.assertIn(self.solicitud, response.context['solicitudes'])

        print(f'{self.id()} pass')

* Resultados obtenidos:

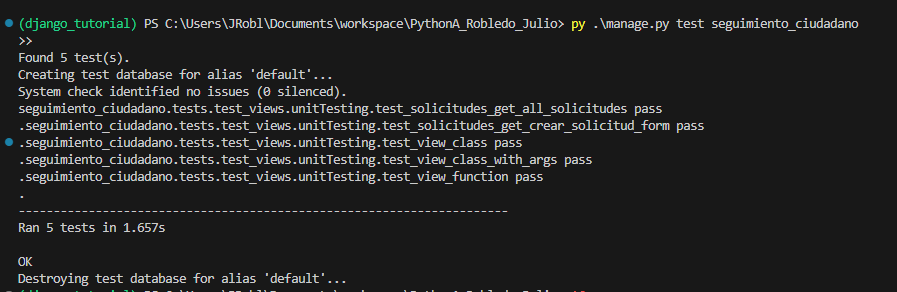


Figura Todas las pruebas fueron satisfactorias. No se encontraron errores

## Conclusiones

”Seguimiento Ciudadano” puede ser una solución altamente efectiva para abordar los desafíos de gestión y respuesta a un gran número de solicitudes. Django, como un marco de desarrollo web flexible, proporciona a los desarrolladores todas las herramientas necesarias para construir aplicaciones robustas y escalables.

Al diseñar un plan de pruebas para un proyecto Django Python gubernamental, es importante considerar los siguientes aspectos clave:

1. **Identificación y priorización de los requisitos**: Antes de comenzar con las pruebas, es esencial comprender claramente los requisitos del proyecto. Esto ayudará a establecer los criterios de aceptación y a definir las funcionalidades críticas que deben probarse exhaustivamente.
2. **Diseño de casos de prueba**: Basándose en los requisitos identificados, se deben diseñar casos de prueba efectivos y exhaustivos. Esto implica identificar diferentes escenarios de prueba, incluyendo casos positivos y negativos, para garantizar que todas las funcionalidades del sistema se evalúen adecuadamente.
3. **Configuración del entorno de prueba**: Es importante configurar un entorno de prueba adecuado que refleje el entorno de producción lo más fielmente posible. Esto incluye la configuración de una base de datos de prueba y la instalación de todas las dependencias necesarias.
4. **Ejecución de pruebas**: Una vez que los casos de prueba estén diseñados y el entorno de prueba esté configurado, se pueden ejecutar las pruebas. Esto implica verificar el funcionamiento correcto de todas las funcionalidades y comprobar si se cumplen los criterios de aceptación establecidos.
5. **Informes de prueba y seguimiento de problemas**: Durante las pruebas, es importante mantener un registro detallado de los resultados de las pruebas y cualquier problema identificado. Esto permitirá un seguimiento efectivo de los problemas y su posterior resolución.

En resumen, el proyecto Seguimiento Ciudadano puede proporcionar una solución eficiente y escalable para gestionar y responder a un gran número de solicitudes. Al diseñar un plan de pruebas, es esencial identificar y priorizar los requisitos, diseñar casos de prueba efectivos, configurar un entorno de prueba adecuado, ejecutar pruebas exhaustivas y realizar un seguimiento de los problemas identificados. Esto garantizará un sistema funcional y confiable que cumpla con las necesidades del gobierno y sus usuarios. Además, las capacidades de almacenamiento en caché de Django hacen que las aplicaciones desarrolladas con Django sean altamente eficientes, lo que beneficia tanto a los desarrolladores como a los usuarios finales.

El plan de pruebas propuesto permite verificar las funcionalidades principales de la aplicación web desarrollada con Django utilizando Selenium. Se recomienda ejecutar estos casos de prueba de manera regular para asegurar que cualquier cambio o actualización en la aplicación no afecte su correcto funcionamiento.