

# Acoplamiento de Cucumber con C# y Automatización de Pruebas con Cucumber + Python

Mario Zambrano<sup>1</sup>

<sup>1</sup> Software - Cuarto Semestre Paralelo B, Facultad de Ciencias de la Computación y Diseño Digital,  
Universidad Técnica Estatal de Quevedo. Ecuador

## 1. Introducción

En el panorama actual del desarrollo de software, uno de los desafíos más persistentes es la brecha de comunicación entre los equipos técnicos y los stakeholders del negocio. Frecuentemente, los requisitos funcionales se malinterpretan o se definen de manera ambigua, lo que conduce a un software que no satisface plenamente las expectativas del usuario final y genera costosos ciclos de retrabajo [1], [2]. Esta desconexión fundamental ha impulsado la evolución de metodologías que priorizan la colaboración y un entendimiento compartido de los objetivos del proyecto. En este contexto, el Desarrollo Guiado por Comportamiento (BDD, por sus siglas en inglés) emerge como una solución robusta y un refinamiento del Desarrollo Guiado por Pruebas (TDD) [3], [4]. BDD no se centra únicamente en la validación técnica del código, sino que pone el foco en el comportamiento del software desde la perspectiva del negocio, asegurando que el producto final se alinee directamente con las necesidades del mercado [5].

La metodología BDD, introducida por Dan North, se fundamenta en la creación de una conversación continua entre desarrolladores, analistas de calidad y expertos de dominio [3]. El objetivo es construir un lenguaje generalizado que describa con precisión cómo debe comportarse el sistema en respuesta a estímulos específicos [6]. Para materializar este diálogo, BDD utiliza un lenguaje de especificación estructurado y en formato de lenguaje natural denominado Gherkin. A través de la sintaxis “Given”, “When”, “Then” (“Dado”, “Cuando”, “Entonces”), Gherkin permite redactar escenarios que son, simultáneamente, requisitos comprensibles para los humanos y pruebas ejecutables para las máquinas [7], [8]. Esta dualidad transforma la documentación de requisitos de un artefacto estático, propenso a la desactualización, en una “documentación viva” que se valida continuamente contra el código base del proyecto, garantizando su precisión y relevancia a lo largo del ciclo de vida del software [9]. La calidad de estas especificaciones es, de hecho, un factor crítico que impacta directamente en la calidad del producto final [10].

Para implementar BDD en la práctica, se requiere un conjunto de herramientas que puedan interpretar los archivos de Gherkin y ejecutar el código de prueba asociado. La herramienta por excelencia en este ámbito es Cucumber, un software que automatiza las pruebas de aceptación escritas

en formato de comportamiento [8], [11]. Cucumber actúa como un puente que conecta las especificaciones en lenguaje natural con las definiciones de pasos (step definitions), que son los fragmentos de código que realmente interactúan con la aplicación y verifican sus resultados [7]. La versatilidad de Cucumber le permite integrarse con una multitud de lenguajes de programación, lo que lo ha convertido en un estándar de la industria para la adopción de BDD en diversos ecosistemas tecnológicos [12].

Este informe se centra en la aplicación práctica de BDD mediante el acoplamiento de Cucumber en dos de las plataformas de desarrollo más relevantes en la actualidad: C# (en el ecosistema .NET) y Python. Para C#, se explorará el uso de SpecFlow, un potente framework de código abierto que integra Gherkin de manera nativa en el entorno de .NET, permitiendo a los desarrolladores definir, gestionar y ejecutar pruebas de aceptación automatizadas [13], [14], [15]. Para Python, se demostrará la automatización de pruebas utilizando Behave, un framework inspirado en Cucumber que aprovecha la simplicidad y legibilidad del lenguaje Python para implementar los principios de BDD [16], [17]. El objetivo de este documento es proporcionar una guía detallada y referenciada sobre el proceso de instalación, configuración, escritura de escenarios y validación de pruebas en ambas plataformas, demostrando cómo BDD puede ser implementado para mejorar la calidad del software y la eficiencia del proceso de desarrollo.

A continuación, se explorará cómo acoplar Cucumber con C# y Python para aprovechar los beneficios de BDD en el ciclo de vida del desarrollo de software.

## **2. Acoplamiento De Cucumber (Lenguaje Gherkin) Con La Plataforma De C#**

Para integrar Cucumber en un entorno .NET, se utiliza una herramienta llamada SpecFlow, que actúa como un puente entre los archivos de características de Gherkin y el código de prueba en C# [13], [18]. SpecFlow es un proyecto de código abierto que permite a los equipos de desarrollo de .NET definir, gestionar y ejecutar pruebas de aceptación automatizadas [14], [15]. La combinación de SpecFlow y Gherkin permite a los equipos de C# adoptar plenamente las prácticas de BDD, asegurando que el software desarrollado cumpla con los requisitos del negocio de manera precisa [19], [20].

## 2.1. Instalación

- a. Primero, se debe asegurar de tener instalado el SDK de .NET (no solo el Runtime). En caso de no tenerlo instalado se lo puede descargar desde el sitio oficial de Microsoft. Para verificar si lo tienes, abre una terminal y escribe “dotnet --version”.

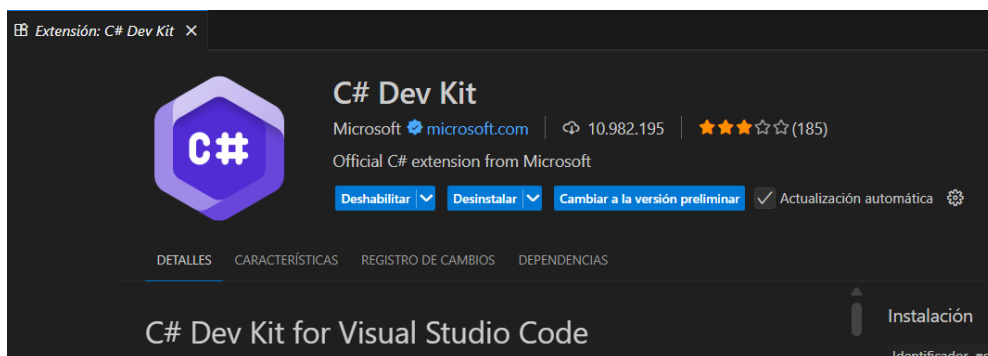
```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\1EHP1895\Downloads\PruebaCucumber> dotnet --version
9.0.301
PS C:\Users\1EHP1895\Downloads\PruebaCucumber> █
```

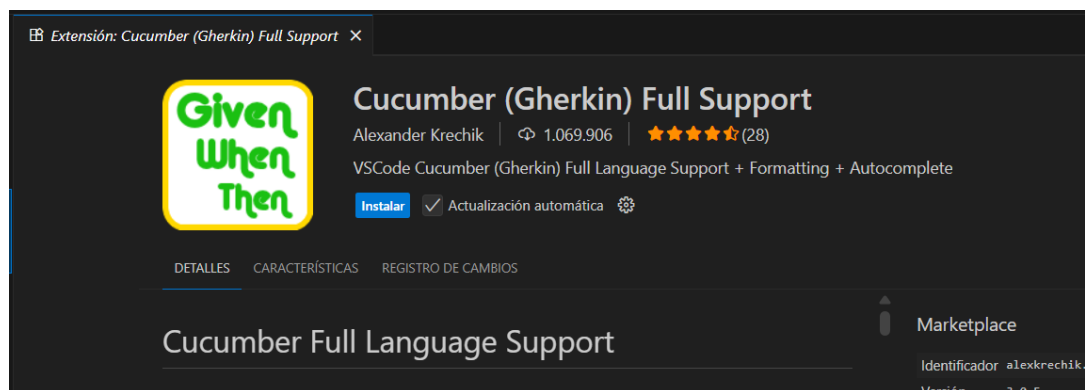
Si nos refleja una versión numérica como muestra la captura (9.0.301) significa que ya lo tenemos instalado y estamos listos para comenzar con la configuración.

- b. Para una buena experiencia, instala estas extensiones desde el panel de Extensiones:

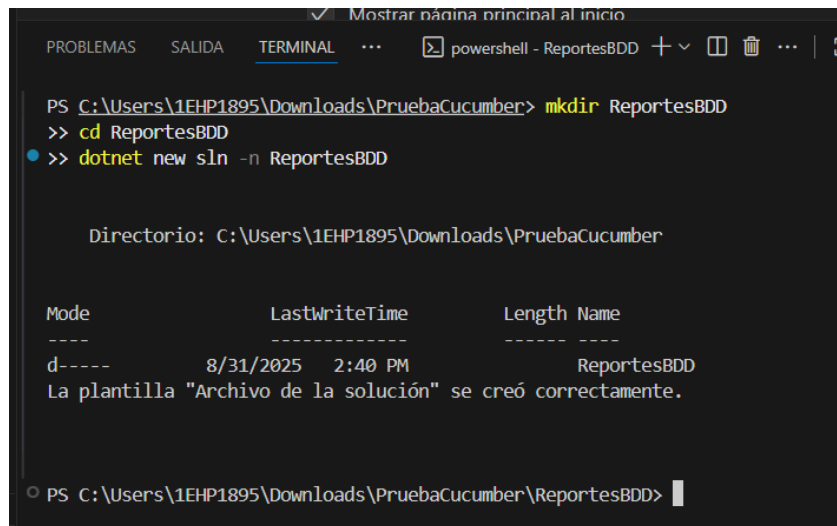
- **C# Dev Kit:** El paquete oficial de Microsoft para el desarrollo en C#.



- **Cucumber (Gherkin) Full Support:** Para que la sintaxis de tus archivos .feature se vea con colores y sea más legible.



- c. Luego, se crea una carpeta para tu proyecto (ej. PruebaCucumber) y se abre (en VS Code).
- d. Después se abre la terminal de VS Code (Ctrl+Ñ) y se ejecuta el siguiente comando para crear una carpeta y solución:
- `mkdir ReportesBDD`
  - `cd ReportesBDD`
  - `dotnet new sln -n ReportesBDD`



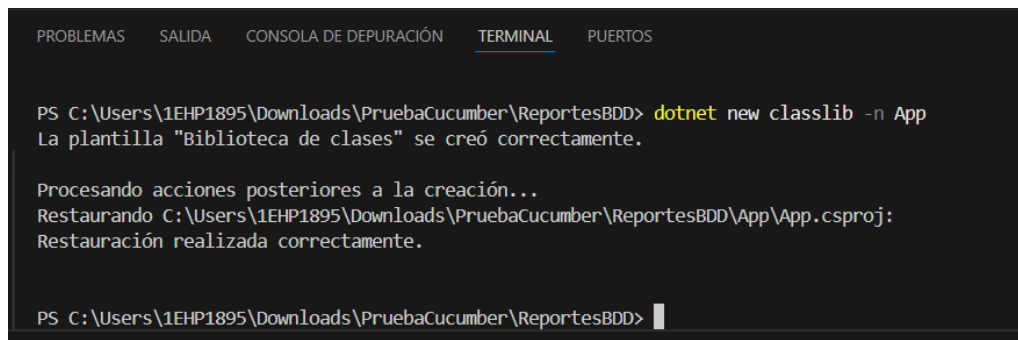
```
PS C:\Users\1EHP1895\Downloads\PruebaCucumber> mkdir ReportesBDD
>> cd ReportesBDD
>> dotnet new sln -n ReportesBDD

Directorio: C:\Users\1EHP1895\Downloads\PruebaCucumber

Mode                LastWriteTime         Length Name
----                -
d-----          8/31/2025   2:40 PM              ReportesBDD
La plantilla "Archivo de la solución" se creó correctamente.

PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD>
```

- e. Después en la terminal mismo creamos un proyecto de aplicación (SUT: Sistema Bajo Prueba) con el comando: `dotnet new classlib -n App`.



```
PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> dotnet new classlib -n App
La plantilla "Biblioteca de clases" se creó correctamente.

Procesando acciones posteriores a la creación...
Restaurando C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD\App\App.csproj:
Restauración realizada correctamente.

PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD>
```

- f. Luego creamos un proyecto de pruebas con xUnit (SpecFlow se añadirá aquí) con el comando: `dotnet new xunit -n App.Specs`.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

● PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> dotnet new xunit -n App.Specs
La plantilla "Proyecto de prueba de xUnit" se creó correctamente.

Procesando acciones posteriores a la creación...
Restaurando C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj:
Restauración realizada correctamente.

○ PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> █
```

- g. Procedemos ahora a agregar los proyectos a la solución con los siguientes comandos: `dotnet sln add App/App.csproj`, `dotnet sln add App.Specs/App.Specs.csproj`.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

● PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> dotnet sln add App/App.csproj
>> dotnet sln add App.Specs/App.Specs.csproj
Se ha agregado el proyecto "App\App.csproj" a la solución.
Se ha agregado el proyecto "App.Specs\App.Specs.csproj" a la solución.
○ PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> █
```

- h. Para finalizar la creación de soluciones y proyectos referenciamos la app desde las pruebas, esto se hace con los siguientes comandos:

```
cd ReportesBDD
```

```
cd App.Specs
```

```
dotnet add reference ../App/App.csproj.
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\1EHP1895\Downloads\PruebaCucumber> cd ReportesBDD
PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> cd App.Specs
PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs> dotnet add reference ../App/App.csproj
Se ha agregado la referencia "..\App\App.csproj" al proyecto.
PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs> █
```

- i. Ahora procedemos a instalar los paquetes NuGet necesarios en el proyecto de pruebas:

- `dotnet add package SpecFlow`
- `dotnet add package SpecFlow.xUnit`
- `dotnet add package SpecFlow.Tools.MsBuild.Generation`

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
powershell - App.Specs

PS C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs> dotnet add package SpecFlow
>> dotnet add package SpecFlow.xUnit
>> dotnet add package SpecFlow.Tools.MsBuild.Generation
>>

Compilación realizado correctamente en 0,9s
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la firma de código.
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la marca de tiempo.
info : Agregando PackageReference para el paquete "SpecFlow" al proyecto "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : GET https://api.nuget.org/v3/registration5-gz-semver2/specflow/index.json
info : OK https://api.nuget.org/v3/registration5-gz-semver2/specflow/index.json 627 ms
info : Restaurando paquetes para C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj...
info : CACHE https://api.nuget.org/v3/vulnerabilities/index.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.08.29.23.26.55/vulnerability.base.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.08.29.23.26.55/vulnerability.update.json
info : El paquete "SpecFlow" es compatible con todos los marcos de trabajo especificados del proyecto "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : Se agregó PackageReference para la versión "3.9.74" del paquete "SpecFlow" al archivo "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : Generación de archivo MSBuild C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\App.Specs.csproj.nuget.g.props.
info : Generación de archivo MSBuild C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\App.Specs.csproj.nuget.g.targets.
info : Escribiendo el archivo de recursos en el disco. Ruta de acceso: C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\project.assets.json
log : Se ha restaurado C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj (en 769 ms).

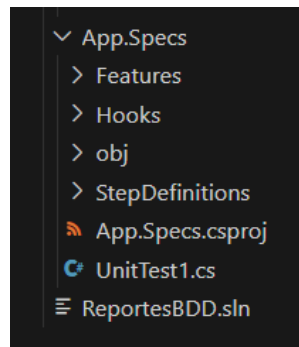
Compilación realizado correctamente en 0,9s
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la firma de código.
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la marca de tiempo.
info : Agregando PackageReference para el paquete "SpecFlow.xUnit" al proyecto "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : GET https://api.nuget.org/v3/registration5-gz-semver2/specflow.xunit/index.json
info : OK https://api.nuget.org/v3/registration5-gz-semver2/specflow.xunit/index.json 1497 ms
info : Restaurando paquetes para C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj...
info : CACHE https://api.nuget.org/v3/vulnerabilities/index.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.08.29.23.26.55/vulnerability.base.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.08.29.23.26.55/vulnerability.update.json
info : El paquete "SpecFlow.xUnit" es compatible con todos los marcos de trabajo especificados del proyecto "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : Se agregó PackageReference para la versión "3.9.74" del paquete "SpecFlow.xUnit" al archivo "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : Generación de archivo MSBuild C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\App.Specs.csproj.nuget.g.props.
info : Generación de archivo MSBuild C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\App.Specs.csproj.nuget.g.targets.
info : Escribiendo el archivo de recursos en el disco. Ruta de acceso: C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\project.assets.json
log : Se ha restaurado C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj (en 581 ms).

Compilación realizado correctamente en 0,9s
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la firma de código.
info : GET https://api.nuget.org/v3/registration5-gz-semver2/specflow.tools.msbuild.generation/index.json
info : OK https://api.nuget.org/v3/registration5-gz-semver2/specflow.tools.msbuild.generation/index.json 3276 ms
info : Restaurando paquetes para C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj...
info : CACHE https://api.nuget.org/v3/vulnerabilities/index.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.08.29.23.26.55/vulnerability.base.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.08.29.23.26.55/vulnerability.update.json
info : El paquete "SpecFlow.Tools.MsBuild.Generation" es compatible con todos los marcos de trabajo especificados del proyecto "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : Se agregó PackageReference para la versión "3.9.74" del paquete "SpecFlow.Tools.MsBuild.Generation" al archivo "C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj".
info : Escribiendo el archivo de recursos en el disco. Ruta de acceso: C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\obj\project.assets.json
log : Se ha restaurado C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\App.Specs\App.Specs.csproj (en 517 ms).
```

La instalación de estos paquetes NuGet es un paso fundamental en la configuración inicial de un marco de pruebas BDD robusto [12]. Paquetes como SpecFlow.xUnit son cruciales para la integración entre el motor de SpecFlow y el corredor de pruebas (en este caso, xUnit), permitiendo que los escenarios sean ejecutables y sus resultados verificables [18], [20]. Adicionalmente, SpecFlow.Tools.MsBuild.Generation desempeña el rol de traductor automático, convirtiendo las especificaciones Gherkin en código C# que el compilador puede entender, un mecanismo central en la arquitectura de SpecFlow [14], [19].

- j. Una vez hecho esto, nuestra estructura del proyecto de pruebas debería de tener necesariamente estas carpetas y archivos antes de proceder con la escritura de escenarios.

```
App.Specs/
├─ Features/      # Escenarios Gherkin (*.feature)
├─ StepDefinitions/ # Implementación de Given/When/Then
├─ Hooks/         # Código adicional (before/after)
└─ App.Specs.csproj
```



Una vez hecho esto, se prosigue con la escritura de los escenarios.

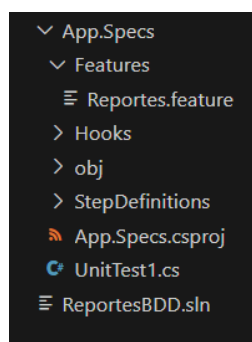
## 2.2. Escritura de Escenarios

La escritura de escenarios en Gherkin es el núcleo de BDD. Cada escenario describe un comportamiento específico del sistema a través de una serie de pasos que siguen la estructura Given-When-Then [7], [11]. Given establece el contexto o las precondiciones, When describe la acción que realiza el usuario o el sistema, y Then especifica el resultado esperado [3], [8]. Esta estructura no solo hace que los escenarios sean fáciles de leer y entender para cualquier miembro del equipo, sino que también impone una disciplina que ayuda a definir claramente los requisitos funcionales [21], [22].

Por ejemplo, un escenario para una funcionalidad del proyecto de aula (Sistema de Gestión de Tutorías Académicas) podría ser el RF07 - Generación de reportes institucionales.

### Proceso paso a paso:

- a. Dentro de la carpeta Features, creamos un nuevo archivo llamado Reportes.feature.



- b. En ese archivo se pega el contenido del requisito RF07 que usaremos:

Feature: Generación de reportes institucionales

In order to facilitar el monitoreo y análisis de tutorías

As Coordinación académica

I want to generar reportes filtrables y exportarlos en PDF

Scenario: Generación y exportación exitosa de un reporte de tutorías en PDF

Given que el coordinador académico ha iniciado sesión en el sistema

And se encuentra en la página de "Generación de Reportes"

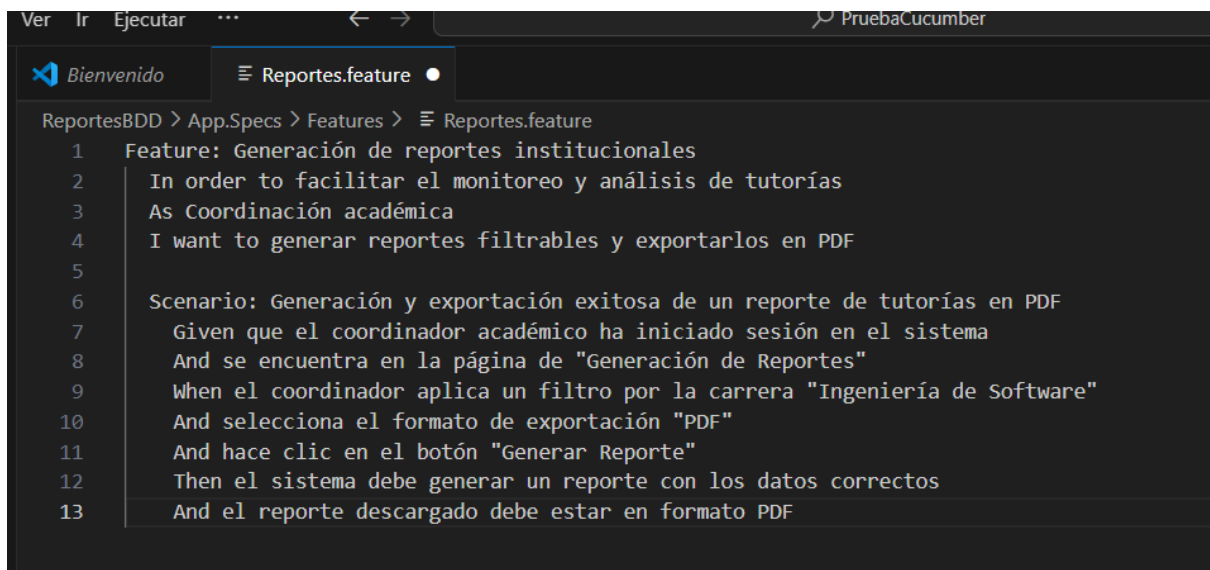
When el coordinador aplica un filtro por la carrera "Ingeniería de Software"

And selecciona el formato de exportación "PDF"

And hace clic en el botón "Generar Reporte"

Then el sistema debe generar un reporte con los datos correctos

And el reporte descargado debe estar en formato PDF



The screenshot shows a code editor with a dark theme. The top bar includes a search icon and the text 'PruebaCucumber'. Below the top bar, there's a tab labeled 'Reportes.feature'. The main editor area shows the following Gherkin code:

```
ReportesBDD > App.Specs > Features > Reportes.feature
1 Feature: Generación de reportes institucionales
2   In order to facilitar el monitoreo y análisis de tutorías
3   As Coordinación académica
4   I want to generar reportes filtrables y exportarlos en PDF
5
6   Scenario: Generación y exportación exitosa de un reporte de tutorías en PDF
7     Given que el coordinador académico ha iniciado sesión en el sistema
8     And se encuentra en la página de "Generación de Reportes"
9     When el coordinador aplica un filtro por la carrera "Ingeniería de Software"
10    And selecciona el formato de exportación "PDF"
11    And hace clic en el botón "Generar Reporte"
12    Then el sistema debe generar un reporte con los datos correctos
13    And el reporte descargado debe estar en formato PDF
```

Este escenario asegura trazabilidad con el RF07, validando que el sistema permita generar y exportar un reporte en formato PDF.

- c. También se crea una clase Reporte.cs en el proyecto de aplicación (App/Reporte.cs), encargada de encapsular la lógica de generación de reportes. Esta clase permite alinear los escenarios de prueba con el requisito RF07.

```
namespace App
{
    public class Reporte
    {
        public string Carrera { get; private set; }
        public string Formato { get; private set; }
        public bool Generado { get; private set; }
    }
}
```



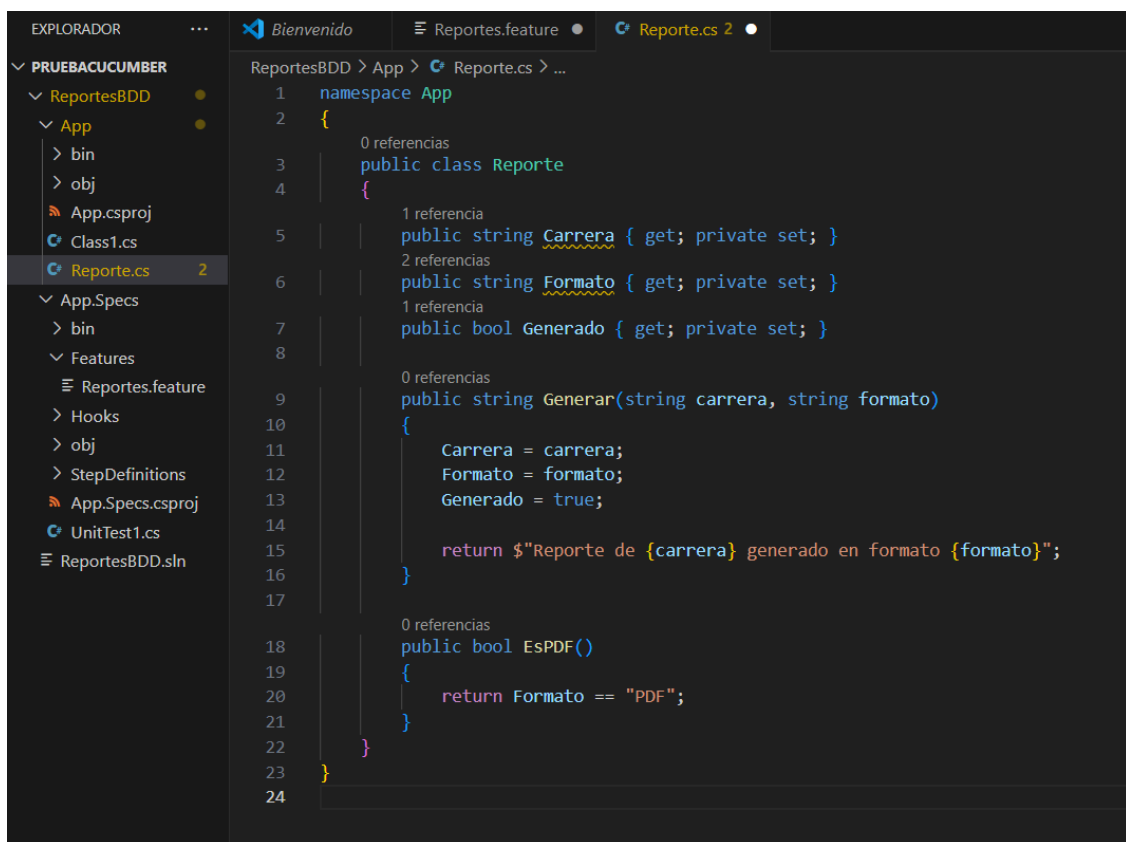
```

public string Generar(string carrera, string formato)
{
    Carrera = carrera;
    Formato = formato;
    Generado = true;

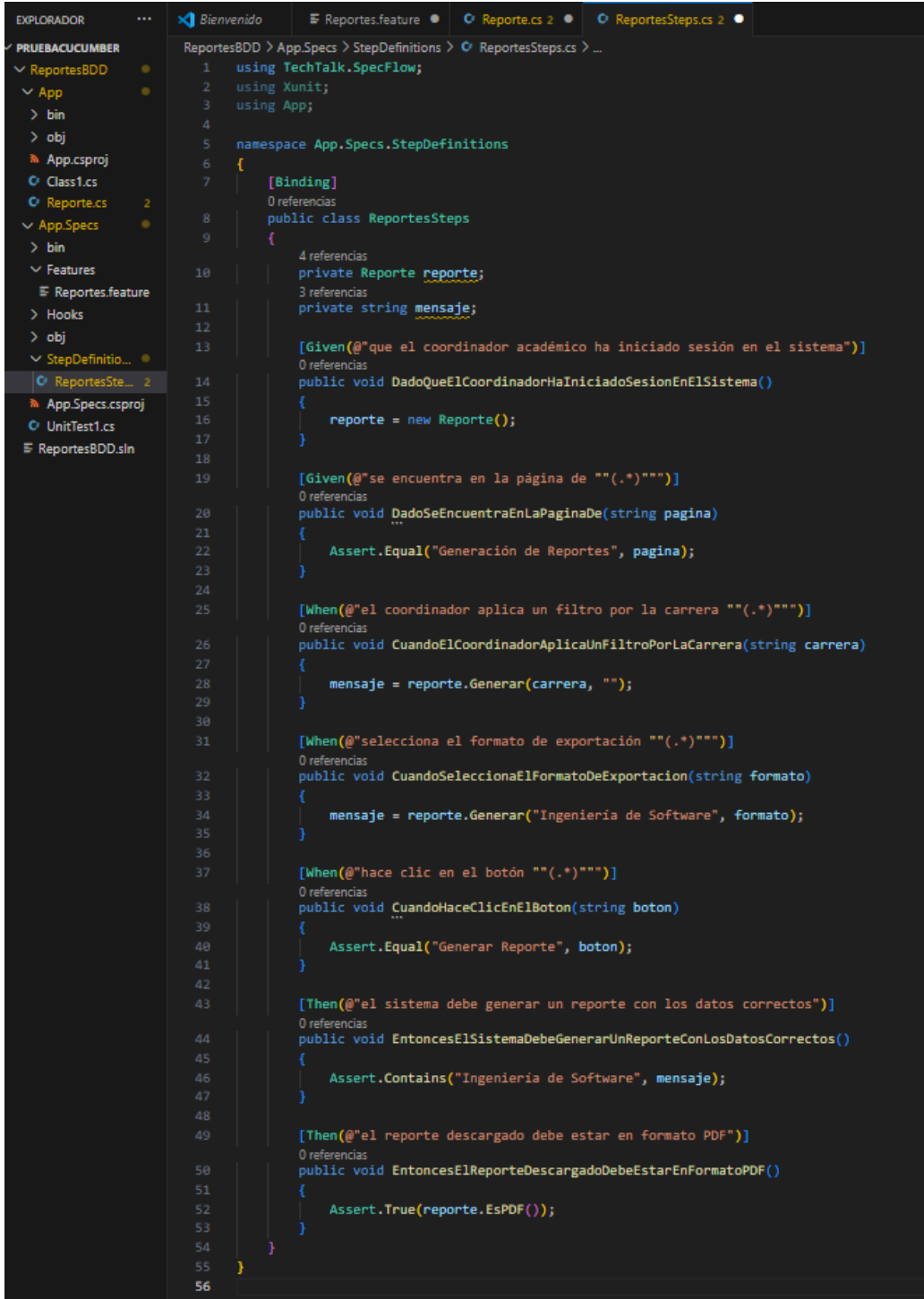
    return $"Reporte de {carrera} generado en formato {formato}";
}

public bool EsPDF()
{
    return Formato == "PDF";
}
}
}

```



- d. Luego se definen los pasos (Step Definitions), cada paso del escenario en Gherkin se debe enlazar con código ejecutable en C#. Se crea el archivo ReportesSteps.cs en el proyecto de pruebas.



```
1 using TechTalk.SpecFlow;
2 using Xunit;
3 using App;
4
5 namespace App.Specs.StepDefinitions
6 {
7     [Binding]
8     0 referencias
9     public class ReportesSteps
10    {
11        4 referencias
12        private Reporte reporte;
13        3 referencias
14        private string mensaje;
15
16        [Given(@"que el coordinador académico ha iniciado sesión en el sistema")]
17        0 referencias
18        public void DadoQueElCoordinadorHaIniciadoSesionEnElSistema()
19        {
20            reporte = new Reporte();
21        }
22
23        [Given(@"se encuentra en la página de ""(.*)""")]
24        0 referencias
25        public void DadoSeEncuentraEnLaPaginaDe(string pagina)
26        {
27            Assert.Equal("Generación de Reportes", pagina);
28        }
29
30        [When(@"el coordinador aplica un filtro por la carrera ""(.*)""")]
31        0 referencias
32        public void CuandoElCoordinadorAplicaUnFiltroPorLaCarrera(string carrera)
33        {
34            mensaje = reporte.Generar(carrera, "");
35        }
36
37        [When(@"selecciona el formato de exportación ""(.*)""")]
38        0 referencias
39        public void CuandoSeleccionaElFormatoDeExportacion(string formato)
40        {
41            mensaje = reporte.Generar("Ingeniería de Software", formato);
42        }
43
44        [When(@"hace clic en el botón ""(.*)""")]
45        0 referencias
46        public void CuandoHaceClicEnElBoton(string boton)
47        {
48            Assert.Equal("Generar Reporte", boton);
49        }
50
51        [Then(@"el sistema debe generar un reporte con los datos correctos")]
52        0 referencias
53        public void EntoncesElSistemaDebeGenerarUnReporteConLosDatosCorrectos()
54        {
55            Assert.Contains("Ingeniería de Software", mensaje);
56        }
57
58        [Then(@"el reporte descargado debe estar en formato PDF")]
59        0 referencias
60        public void EntoncesElReporteDescargadoDebeEstarEnFormatoPDF()
61        {
62            Assert.True(reporte.EsPDF());
63        }
64    }
65 }
```

Teniendo hasta aquí con los escenarios definidos y los pasos procederemos con la prueba y validación del escenario.

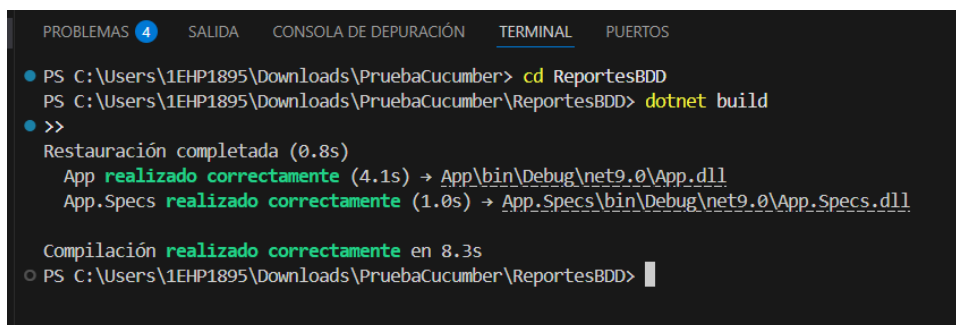
## 2.3. Prueba y Validación

Una vez que se ha escrito un escenario en un archivo .feature, el siguiente paso es implementar las definiciones de pasos (step definitions) en C#. Las definiciones de pasos son métodos de C# que están vinculados a los pasos de Gherkin a través de atributos como “Given”, “When”, y “Then” [7], [19]. SpecFlow se encarga de hacer coincidir cada paso en el escenario con el método correspondiente en el código de C#. Dentro de estos métodos, los desarrolladores escriben el código de automatización que interactúa con la aplicación y verifica su comportamiento [18], [20].

La validación se realiza mediante aserciones dentro de los métodos “Then”. Estas aserciones comprueban si el estado de la aplicación después de la acción “When” coincide con el resultado esperado [12], [16]. Si todas las aserciones en un escenario pasan, la prueba se considera exitosa. El Test Explorer de Visual Studio se utiliza para ejecutar estas pruebas y ver los resultados. Este ciclo de escritura de escenarios, implementación de definiciones de pasos y ejecución de pruebas permite una validación continua del comportamiento del software a lo largo del ciclo de desarrollo [21], [23].

### Proceso paso a paso:

- a. Se compila la solución con el siguiente comando en el terminal: `dotnet build`. No olvides estar en la ruta del proyecto (ej. `C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD>`).



```
PROBLEMAS 4 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
PS C:\Users\1EHP1895\Downloads\PruebaCucumber> cd ReportesBDD
PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD> dotnet build
>>
Restauración completada (0.8s)
App realizado correctamente (4.1s) -> App\bin\Debug\net9.0\App.dll
App.Specs realizado correctamente (1.0s) -> App.Specs\bin\Debug\net9.0\App.Specs.dll

Compilación realizado correctamente en 8.3s
PS C:\Users\1EHP1895\Downloads\PruebaCucumber\ReportesBDD>
```

- b. Finalmente, después de la compilación solo falta ejecutar las pruebas y ver el resultado esperado con “`dotnet test`”. El runner de xUnit junto con SpecFlow interpreta los archivos .feature, ejecuta los Step Definitions y valida que los resultados coincidan con lo esperado. En este caso, el resultado esperado será que el reporte se genere con los datos de la carrera Ingeniería de Software y que el reporte esté exportado en formato PDF.

```
PROBLEMAS 4 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS powershell - ReportesBDD +v | ... | [?] x
PS C:\Users\IEHP1895\Downloads\PruebaCucumber> cd ReportesBDD
PS C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD> dotnet test
Restauración completada (0.9s)
App realizado correctamente (0.3s) -> App\bin\Debug\net9.0\App.dll
App.Specs correcto con 2 advertencias (0.3s) -> App.Specs\bin\Debug\net9.0\App.Specs.dll
C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\AppData\Local\Temp\ReportesSteps.cs(10,25): warning CS8618: El elemento campo "reporte" que no acepta valores NU
LL debe contener un valor distinto de NULL al salir del constructor. Considere la posibilidad de agregar el modificador "required" o declarar el campo como un valor que acepta va
lores NULL.
C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\AppData\Local\Temp\ReportesSteps.cs(11,24): warning CS8618: El elemento campo "mensaje" que no acepta valores NU
LL debe contener un valor distinto de NULL al salir del constructor. Considere la posibilidad de agregar el modificador "required" o declarar el campo como un valor que acepta va
lores NULL.
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.2+699d445a1a (64-bit .NET 9.0.6)
[xUnit.net 00:00:00.12] Discovering: App.Specs
[xUnit.net 00:00:00.18] Discovered: App.Specs
[xUnit.net 00:00:00.18] Starting: App.Specs
-> Loading plugin C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\AppData\Local\Temp\TechTalk.SpecFlow.xUnit.SpecFlowPlugin.dll
-> Loading plugin C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD\AppData\Local\Temp\ReportesSteps.cs
-> Using default config
Reporte de Ingeniería de Software generado en formato PDF
[xUnit.net 00:00:00.59] Finished: App.Specs
App.Specs prueba realizado correctamente (1.7s)

Resumen de pruebas: total: 2; con errores: 0; correcto: 2; omitido: 0; duración: 1.7 s
Compilación correcto con 2 advertencias en 3.7s
PS C:\Users\IEHP1895\Downloads\PruebaCucumber\ReportesBDD>
```

Con esto se logra una trazabilidad completa entre el RF07, el escenario en Gherkin y las pruebas automatizadas en C#.

### 3. Demostrar La Automatización De Pruebas En Cucumber + Python (Instalación, Configuración Visual Code, Escribir Escenarios, Ejecutar Escenario)

Cucumber también se puede utilizar con Python para la automatización de pruebas, aprovechando el amplio ecosistema de bibliotecas de Python para el desarrollo y las pruebas de software. El framework más común para implementar BDD con Python es Behave, que está inspirado en Cucumber y utiliza el mismo lenguaje Gherkin para definir los escenarios de prueba [16], [17]. La combinación de Behave y Python ofrece una solución potente y flexible para la automatización de pruebas en una variedad de aplicaciones, desde servicios web hasta aplicaciones de escritorio [24].

#### 3.1. Instalación y Configuración en Visual Studio Code

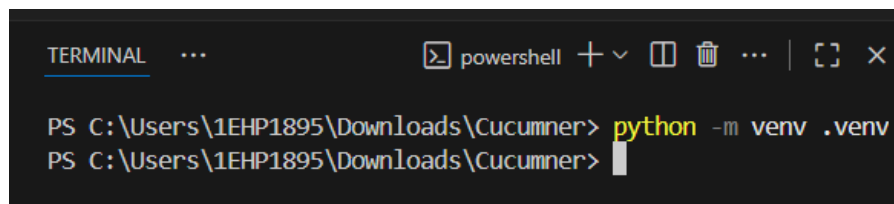
Para comenzar a trabajar con Behave y Python, primero es necesario instalar Python y el gestor de paquetes pip [24]. Luego, Behave se puede instalar fácilmente a través de la línea de comandos con el comando `pip install behave` [17]. Es una buena práctica gestionar estas dependencias dentro de un entorno virtual para evitar conflictos entre proyectos.

En Visual Studio Code, se recomienda instalar la extensión oficial de Python de Microsoft, que proporciona soporte para la depuración, el linting y la finalización de código. También se puede instalar una extensión como "Cucumber (Gherkin) Full Support" para obtener resaltado de sintaxis en los archivos `.feature` y otras funcionalidades útiles que mejoran la experiencia de desarrollo [17], [24]. La configuración del proyecto implica la creación de una estructura de directorios específica: una carpeta `features/` que contendrá los archivos `.feature` y una subcarpeta `features/steps/` para los archivos de Python con las definiciones de los pasos [16].

### Serie de pasos a seguir para este punto:

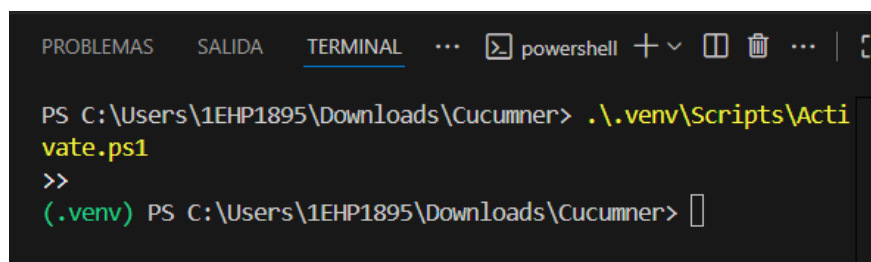
#### 1. Primero, creamos un espacio de trabajo limpio y a instalar las herramientas necesarias.

- a. Crea una Carpeta para tu Proyecto: En tu explorador de archivos, crea una nueva carpeta. Por poner un ejemplo, Cucumner.
- b. Abre la Carpeta en VS Code: En VS Code, ve a Archivo > Abrir carpeta... y selecciona la que acabas de crear.
- c. Abre la Terminal: Usa el atajo Ctrl+Ñ (o View > Terminal) para abrir la terminal integrada.
- d. Crea un Entorno Virtual: Es una buena práctica aislar las dependencias de tu proyecto. Ejecuta el siguiente comando en la terminal: “python -m venv .venv”



```
TERMINAL ... powershell + - 
PS C:\Users\1EHP1895\Downloads\Cucumner> python -m venv .venv
PS C:\Users\1EHP1895\Downloads\Cucumner>
```

- e. Activa el Entorno Virtual: Para que VS Code use este entorno, ejecutas: “.venv\Scripts\Activate”



```
PROBLEMAS SALIDA TERMINAL ... powershell + - 
PS C:\Users\1EHP1895\Downloads\Cucumner> .venv\Scripts\Activate.ps1
>>
(.venv) PS C:\Users\1EHP1895\Downloads\Cucumner>
```

#### 2. Instalación de Behave y Dependencias

Con el entorno virtual activo, instalamos Behave, que es el framework que interpreta Gherkin en Python [16], [17].

Ejecuta el siguiente comando en tu terminal: “pip install behave”

```

PROBLEMAS          SALIDA          CONSOLA DE DEPURACIÓN          PUERTOS
TERMINAL
X powershell + v [ ] [ ] ... [ ] [ ] X

PS C:\Users\1EHP1895\Downloads\Cucumber> .\venv\Scripts\Activate.ps1
>>
(.venv) PS C:\Users\1EHP1895\Downloads\Cucumber> pip install behave
collecting behave
  Using cached behave-1.3.2-py2.py3-none-any.whl.metadata (10 kB)
collecting cucumber-tag-expressions>=4.1.0 (from behave)
  Using cached cucumber_tag_expressions-6.2.0-py2.py3-none-any.whl.metadata (6.8 kB)
collecting cucumber-expressions>=17.1.0 (from behave)
  Using cached cucumber_expressions-18.0.1-py3-none-any.whl.metadata (2.5 kB)
collecting parse>=1.18.0 (from behave)
17 kB)
Using cached behave-1.3.2-py2.py3-none-any.whl (223 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Using cached cucumber_expressions-18.0.1-py3-none-any.whl (20 kB)
Using cached cucumber_tag_expressions-6.2.0-py2.py3-none-any.whl (9.3 kB)
Using cached parse-1.20.2-py2.py3-none-any.whl (20 kB)
Using cached parse_type-0.6.6-py2.py3-none-any.whl (27 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: parse, six, cucumber-tag-expressions, cucumber-expressions, colorama, parse-type, behave
Successfully installed behave-1.3.2 colorama-0.4.6 cucumber-expressions-18.0.1 cucumber-tag-expressions-6.2.0 parse-1.20.2 parse-type-0.6.6 six-1.17.0

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\1EHP1895\Downloads\Cucumber>

```

Esto instalará el paquete behave y sus dependencias necesarias dentro de tu entorno .venv.

### 3. Creación de la Estructura de Carpetas

Behave requiere una estructura de carpetas específica para encontrar tus archivos automáticamente [17].

- En el explorador de archivos de VS Code, crea una nueva carpeta llamada `features`.
- Dentro de la carpeta `features`, crea otra carpeta llamada `steps`.

Tu proyecto debería verse así:



### 3.2. Escritura y Ejecución de Escenarios

Al igual que con C#, los escenarios se escriben en archivos .feature utilizando la sintaxis de Gherkin. La estructura y las palabras clave son las mismas, lo que permite una transición fluida para los equipos que trabajan con múltiples lenguajes de programación [8], [11]. Un escenario de ejemplo en Gherkin para un proyecto de Python sería idéntico al mostrado en la sección anterior para C#. La diferencia radica en la implementación de las definiciones de pasos.

En Python, las definiciones de pasos se escriben en archivos .py dentro de la carpeta features/steps/. Cada función de Python que implementa un paso está decorada con @given, @when, o @then del módulo behave [16], [17].

Para ejecutar los escenarios, se utiliza el comando behave en la terminal, desde el directorio raíz del proyecto, la ejecución del comando descubre automáticamente los archivos .feature y muestra los resultados en la consola [17]. Además, se encarga de hacer coincidir las cadenas de texto de los decoradores con los pasos en los archivos .feature [24]. La salida indicará qué escenarios pasaron, fallaron o fueron omitidos, proporcionando una retroalimentación rápida sobre el estado del comportamiento de la aplicación [16].

### **Serie de pasos a seguir para este punto:**

#### **1. Escritura del Escenario Gherkin (.feature)**

- a. Ahora, traducimos el requisito RF07 a Gherkin, lo guardamos dentro de la carpeta features.
- b. Dentro de la carpeta features, se crea un nuevo archivo llamado generacion\_reportes.feature.
- c. Se pega el siguiente contenido. Es el mismo escenario, ya que Gherkin es un lenguaje universal para BDD [8], [11].

Feature: RF07 - Generación de reportes institucionales

Como Coordinación académica

Quiero generar reportes filtrables de las tutorías

Para facilitar el monitoreo y análisis a nivel institucional

Scenario: Generación y exportación exitosa de un reporte de tutorías en PDF

Given que el coordinador académico ha iniciado sesión en el sistema

And se encuentra en la página de "Generación de Reportes"

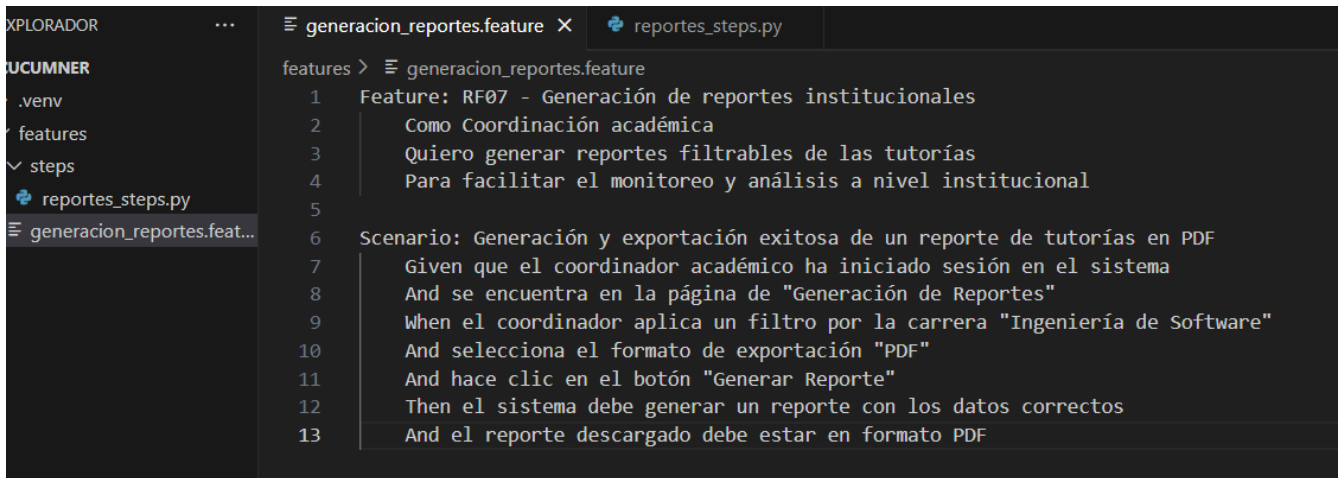
When el coordinador aplica un filtro por la carrera "Ingeniería de Software"

And selecciona el formato de exportación "PDF"

And hace clic en el botón "Generar Reporte"

Then el sistema debe generar un reporte con los datos correctos

And el reporte descargado debe estar en formato PDF



The screenshot shows a code editor with two tabs: 'generacion\_reportes.feature' and 'reportes\_steps.py'. The left sidebar shows a file explorer with a tree structure: 'UCUMNER' > '.venv' > 'features' > 'steps' > 'reportes\_steps.py'. The main editor area shows the content of 'generacion\_reportes.feature' with line numbers 1 through 13. The feature is titled 'Feature: RF07 - Generación de reportes institucionales' and describes the goal of generating filterable reports for institutional monitoring. It includes a scenario for successful PDF generation and export, with steps for logging in, applying a filter, selecting the PDF format, clicking the 'Generate Report' button, and verifying the report is in PDF format.

```
features > generacion_reportes.feature
1 Feature: RF07 - Generación de reportes institucionales
2   Como Coordinación académica
3   Quiero generar reportes filtrables de las tutorías
4   Para facilitar el monitoreo y análisis a nivel institucional
5
6 Scenario: Generación y exportación exitosa de un reporte de tutorías en PDF
7   Given que el coordinador académico ha iniciado sesión en el sistema
8   And se encuentra en la página de "Generación de Reportes"
9   When el coordinador aplica un filtro por la carrera "Ingeniería de Software"
10  And selecciona el formato de exportación "PDF"
11  And hace clic en el botón "Generar Reporte"
12  Then el sistema debe generar un reporte con los datos correctos
13  And el reporte descargado debe estar en formato PDF
```

## 2. Implementación de los Pasos en Python

- Aquí es donde conectamos el Gherkin con código Python ejecutable.
- Dentro de la carpeta features/steps, se crea un nuevo archivo llamado reportes\_steps.py.
- Se pega el siguiente código. Observa cómo los decoradores @given, @when, y @then de Behave vinculan cada función con un paso del .feature [16], [17].

```
from behave import given, when, then

# El 'context' es un objeto que Behave usa para pasar datos entre los pasos.

@given('que el coordinador académico ha iniciado sesión en el sistema')

def step_impl(context):

    # Aquí iría la lógica real de login.

    print("Paso 1: El coordinador ha iniciado sesión.")

    context.logged_in = True

@given('se encuentra en la página de "Generación de Reportes"')

def step_impl(context):

    # Lógica para navegar a la página.

    print("Paso 2: Navegando a la página de reportes.")

    pass
```



```
@when('el coordinador aplica un filtro por la carrera "{carrera}")

def step_impl(context, carrera):

    print(f"Paso 3: Aplicando filtro para la carrera: {carrera}")

    pass

@when('selecciona el formato de exportación "{formato}")

def step_impl(context, formato):

    print(f"Paso 4: Seleccionando el formato: {formato}")

    context.formato_seleccionado = formato

@when('hace clic en el botón "Generar Reporte"')

def step_impl(context):

    print("Paso 5: Haciendo clic en generar reporte.")

    # Simulamos que la generación fue exitosa.

    context.reporte_generado = True

@then('el sistema debe generar un reporte con los datos correctos')

def step_impl(context):

    print("Paso 6: Verificando que el reporte se generó.")

    # La aserción de Python verifica si una condición es verdadera.

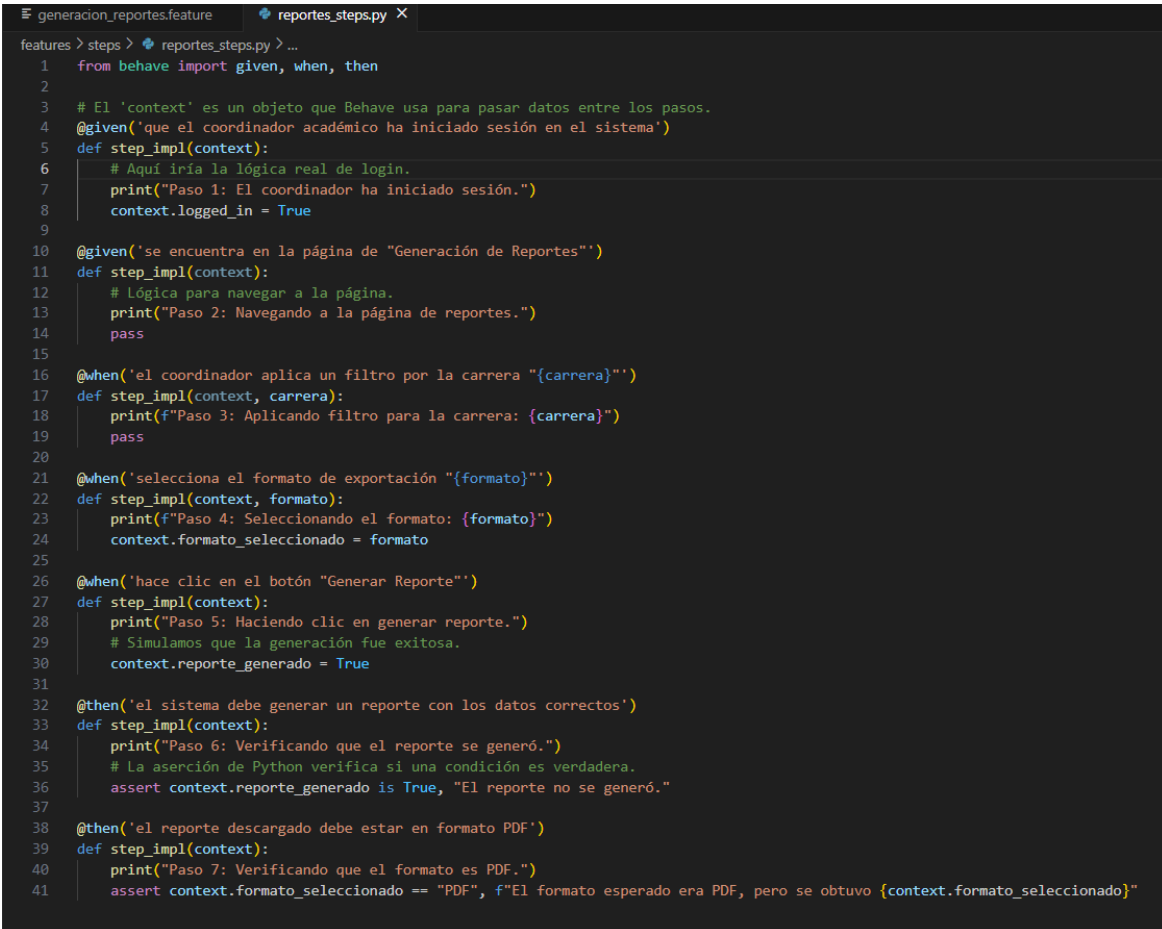
    assert context.reporte_generado is True, "El reporte no se generó."

@then('el reporte descargado debe estar en formato PDF')

def step_impl(context):

    print("Paso 7: Verificando que el formato es PDF.")
```

```
assert context.formato_seleccionado == "PDF", f"El formato esperado era PDF, pero se obtuvo {context.formato_seleccionado}"
```



```
features > steps > reportes_steps.py > ...
1  from behave import given, when, then
2
3  # El 'context' es un objeto que Behave usa para pasar datos entre los pasos.
4  @given('que el coordinador académico ha iniciado sesión en el sistema')
5  def step_impl(context):
6      # Aquí iría la lógica real de login.
7      print("Paso 1: El coordinador ha iniciado sesión.")
8      context.logged_in = True
9
10 @given('se encuentra en la página de "Generación de Reportes"')
11 def step_impl(context):
12     # Lógica para navegar a la página.
13     print("Paso 2: Navegando a la página de reportes.")
14     pass
15
16 @when('el coordinador aplica un filtro por la carrera "{carrera}"')
17 def step_impl(context, carrera):
18     print(f"Paso 3: Aplicando filtro para la carrera: {carrera}")
19     pass
20
21 @when('selecciona el formato de exportación "{formato}"')
22 def step_impl(context, formato):
23     print(f"Paso 4: Seleccionando el formato: {formato}")
24     context.formato_seleccionado = formato
25
26 @when('hace clic en el botón "Generar Reporte"')
27 def step_impl(context):
28     print("Paso 5: Haciendo clic en generar reporte.")
29     # Simulamos que la generación fue exitosa.
30     context.reporte_generado = True
31
32 @then('el sistema debe generar un reporte con los datos correctos')
33 def step_impl(context):
34     print("Paso 6: Verificando que el reporte se generó.")
35     # La aserción de Python verifica si una condición es verdadera.
36     assert context.reporte_generado is True, "El reporte no se generó."
37
38 @then('el reporte descargado debe estar en formato PDF')
39 def step_impl(context):
40     print("Paso 7: Verificando que el formato es PDF.")
41     assert context.formato_seleccionado == "PDF", f"El formato esperado era PDF, pero se obtuvo {context.formato_seleccionado}"
```

### 3. Ejecución del Escenario

Con todo en su lugar, ejecutar la prueba es muy sencillo.

Asegúrate de que tu terminal siga en la carpeta raíz (ProyectoPythonBDD) y que el entorno virtual esté activo. Simplemente ejecuta el comando: “behave”.

Behave encontrará y ejecutará automáticamente tu archivo .feature. La salida en la terminal te mostrará cada paso ejecutándose y, al final, un resumen que indica que el escenario se completó con éxito.

```

(.venv) PS C:\Users\1EHP1895\Downloads\Cucumner> behave
USING RUNNER: behave.runner:Runner
Feature: RF07 - Generación de reportes institucionales # features/generacion_reportes.feature:1
  atures/steps/reportes_steps.py:10 0.000s
  atures/steps/reportes_steps.py:10 0.000s
    When el coordinador aplica un filtro por la carrera "Ingeniería de Software" # features/steps/reportes_steps.py:16 0.000s
    And selecciona el formato de exportación "PDF" # features/steps/reportes_steps.py:21 0.001s
    And hace clic en el botón "Generar Reporte" # features/steps/reportes_steps.py:26 0.001s
    Then el sistema debe generar un reporte con los datos correctos # features/steps/reportes_steps.py:32 0.000s
    And el reporte descargado debe estar en formato PDF # features/steps/reportes_steps.py:38 0.001s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
7 steps passed, 0 failed, 0 skipped
Took 0min 0.002s
(.venv) PS C:\Users\1EHP1895\Downloads\Cucumner>

```

La ejecución de las pruebas con Behave validó el requisito RF07 Generación de reportes institucionales. El reporte indicó 1 feature, 1 escenario y 7 pasos, todos exitosos y sin errores, con un tiempo de 0.002 segundos. Estos resultados evidencian que la automatización en Python refleja correctamente el comportamiento definido en los escenarios Gherkin.

#### 4. Conclusión

Este informe ha demostrado la aplicación práctica y efectiva del Desarrollo Guiado por Comportamiento (BDD) en los entornos de C# y Python. A través de SpecFlow y Behave, se validó cómo los requisitos de negocio, escritos en el lenguaje natural de Gherkin, se transforman exitosamente en pruebas automatizadas y ejecutables.

Se concluye que, si bien cada plataforma presenta particularidades en su configuración SpecFlow dependiendo del ecosistema .NET y NuGet, y Behave de la estructura de carpetas de Python, ambas implementaciones cumplen el objetivo central de BDD: crear una "documentación viva". Este enfoque garantiza una alineación constante entre la especificación y el código, mejorando la calidad del software y fomentando una colaboración más clara entre los roles técnicos y no técnicos del proyecto.

#### 5. Referencias

- [1] T. Olasehinde, "Behavior-Driven Development: Bridging the Gap Between Developers and Stakeholders," 2023. [Online]. Available: <https://www.researchgate.net/publication/386135623>
- [2] L. P. Binamungu and S. Maro, "Behaviour Driven Development: A Systematic Mapping Study," *Journal of Systems and Software*, vol. 203, May 2023, doi: 10.1016/j.jss.2023.111749.

- [3] Dan North, "Introducing BDD," Associates Limited. Accessed: Aug. 29, 2025. [Online]. Available: <https://dannorth.net/blog/introducing-bdd/>
- [4] Taynara Jacon, "Mastering Behavior-Driven Development (BDD) | by Taynara Jacon | Medium." Accessed: Aug. 29, 2025. [Online]. Available: <https://medium.com/@taynarajacon/mastering-behavior-driven-development-bdd-c69eb3acf3b6>
- [5] A. E. Soraluz Soraluz, M. Á. Valles Coral, D. Lévano Rodríguez, A. E. Soraluz Soraluz, M. Á. Valles Coral, and D. Lévano Rodríguez, "Desarrollo guiado por comportamiento: buenas prácticas para la calidad de software," *Ingeniería y Desarrollo*, vol. 39, no. 1, pp. 190–204, Apr. 2021, doi: 10.14482/INDE.39.1.005.3.
- [6] J. Ferguson. Smart, Jan. Molak, and Daniel. Terhorst-North, "BDD in action : behavior-driven development for the whole software lifecycle," p. 488, 2023, Accessed: Aug. 29, 2025. [Online]. Available: [https://books.google.com/books/about/BDD\\_in\\_Action\\_Second\\_Edition.html?hl=es&id=hIK3EAAAQBAJ](https://books.google.com/books/about/BDD_in_Action_Second_Edition.html?hl=es&id=hIK3EAAAQBAJ)
- [7] Lakshay Sharma, "What is Step Definition And How to write Step Definitions in Cucumber." Accessed: Aug. 29, 2025. [Online]. Available: <https://toolsqa.com/cucumber/step-definition/>
- [8] Matt. Wynne, Aslak. Hellesøy, and Steve. Tooke, *The cucumber book : behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017.
- [9] C. Solís and X. Wang, "A study of the characteristics of behaviour driven development," *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, pp. 383–387, 2011, doi: 10.1109/SEAA.2011.76.
- [10] L. P. Binamungu, S. M. Embury, and N. Konstantinou, "Characterising the Quality of Behaviour Driven Development Specifications," *Lecture Notes in Business Information Processing*, vol. 383 LNBIP, pp. 87–102, 2020, doi: 10.1007/978-3-030-49392-9\_6/TABLES/3.
- [11] Matt. Wynne, Aslak. Hellesøy, and Steve. Tooke, "The cucumber book : behaviour-driven development for testers and developers," 2017, Accessed: Aug. 29, 2025. [Online]. Available: <https://pragprog.com/titles/hwcuc2/the-cucumber-book-second-edition>
- [12] S. Srinivas and E. L. Goel, "Designing and Implementing Robust Test Automation Frameworks using Cucumber BDD and Java," May 2025, doi: 10.36676/URR.V12.
- [13] S. Holder, "Why specflow is needed in cucumber even its supporting BDD Tools - Stack Overflow." Accessed: Aug. 29, 2025. [Online]. Available: <https://stackoverflow.com/questions/32680848/why-specflow-is-needed-in-cucumber-even-its-supporting-bdd-tools?utm>
- [14] Mahesh Patidar, "SpecFlow: a Free and Open Source BDD Framework for .NET | by Mahesh Patidar | Globant | Medium." Accessed: Aug. 29, 2025. [Online]. Available: <https://medium.com/globant/specflow-a-free-and-open-source-bdd-framework-for-net-55cc579a3a2b>
- [15] "Does SpecFlow use Cucumber? - LambdaTest Community." Accessed: Aug. 29, 2025. [Online]. Available: <https://community.lambdatest.com/t/does-specflow-use-cucumber/18714?utm>

- [16] T. Storer and R. Bob, "Behave nicely! automatic generation of code for behaviour driven development test suites," *Proceedings - 19th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2019*, pp. 228–237, Sep. 2019, doi: 10.1109/SCAM.2019.00033.
- [17] "behave 1.4.0.dev0 documentation." Accessed: Aug. 29, 2025. [Online]. Available: <https://behave.readthedocs.io/en/latest/?utm>
- [18] BrowserStack, "SpecFlow Tutorial for Automation Testing | BrowserStack." Accessed: Aug. 29, 2025. [Online]. Available: <https://www.browserstack.com/guide/specflow-automated-testing-tutorial?utm>
- [19] Daniel Delimata, "SpecFlow — Cucumber in C#. In this post we will go through setting... | by Daniel Delimata | Medium." Accessed: Aug. 29, 2025. [Online]. Available: <https://daniel-delimata.medium.com/specflow-cucumber-in-c-e642c63469b2>
- [20] "Puntos de datos: Diseño controlado por comportamiento con SpecFlow | Microsoft Learn." Accessed: Aug. 29, 2025. [Online]. Available: <https://learn.microsoft.com/es-es/archive/msdn-magazine/2013/july/data-points-behavior-driven-design-with-specflow>
- [21] S. Santos, T. Pimentel, F. G. Rocha, and M. S. Soares, "Using Behavior-Driven Development (BDD) for Non-Functional Requirements," *Software 2024, Vol. 3, Pages 271-283*, vol. 3, no. 3, pp. 271–283, Jul. 2024, doi: 10.3390/SOFTWARE3030014.
- [22] M. Z. Anjum, S. Togneri, M. Mahon, and F. McCaffery, "Development of Health Software using Behaviour Driven Development-BDD", doi: 10.5220/0008984201490157.
- [23] "IEEE Recommended Practice for Software Requirements Specifications," Jun. 1998, doi: 10.1109/IEEESTD.1998.88286.
- [24] "3.13.7 Documentation." Accessed: Aug. 29, 2025. [Online]. Available: <https://docs.python.org/3/?utm>