|  |
| --- |
| Workshop de ALM - Laboratorio 03 |
| Testing |
|  |

|  |
| --- |
|  |

# Pre requisitos

Para este laboratorio se requiere que tenga un Visual Studio Online con permiso de Administrador. Si no la tiene puede crear una subscripción Gratuita hasta 5 desarrolladores desde la URL <https://www.visualstudio.com/products/what-is-visual-studio-online-vs.aspx>

## Laboratorio 1 Pruebas Unitarias

Una prueba unitaria es una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

La idea es escribir casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto.

Cuando desarrollamos para SharePoint y en una menor medida para Office 365 se tiene la creencia generalizada de que NO se puede realizar pruebas unitarias. Hay varias opciones una el utilizar herramientas “comerciales” como TypeMock o JustMock de Telerik que nos permiten crear simular el comportamiento de clases cerradas o privadas como es la API de SharePoint. Este tipo de pruebas nos permiten incrementar el porcentaje de cobertura de código pero en mi opinión no garantiza que la lógica de negocio se cumpla.

**Manos a la obra**

Partiendo del caso anterior, en un primer lugar vamos a implementar una interfaz IPersonService, esta interfaz tendrá dos métodos tal y como sigue:

public interface IPersonService

{

IEnumerable<Person> GetAll();

Person GetById(int id);

}

Sobre esta interfaz vamos a implementarla por un lado una interfaz de modo Mock y otra interfaz que acceda los datos de SharePoint. La definición de la función de Mock es la siguiente:

public class PersonMockService:IPersonService

{

Person[] personRepository= new Person[]

{

new Person{ Id=1,LastName = "Diaz",Name="Adrian"},

new Person {Id=2,LastName = "Canales",Name="Marco"}

};

public IEnumerable<Person> GetAll()

{

return personRepository;

}

public Person GetById(int id)

{

return personRepository.FirstOrDefault((p) => p.Id == id);;

}

}

La implementación que accede los datos a SharePoint seria la siguiente:

public class PersonService:IPersonService

{

public IEnumerable<Person> GetAll()

{

var spContext = SharePointContext.GetClientContext();

var spList = spContext.Web.Lists.GetByTitle("Person");

spContext.Load(spList);

spContext.ExecuteQuery();

if (spList == null || spList.ItemCount <= 0) return null;

var camlQuery = new CamlQuery

{

ViewXml = @"<View>

<RowLimit>20</RowLimit>

</View>",

DatesInUtc = true

};

var listItems = spList.GetItems(camlQuery);

spContext.Load(listItems);

spContext.ExecuteQuery();

var resultItems = new List<Person>();

foreach (var item in listItems)

{

var personItem = new Person

{

Id = item.Id,

Name = item["Name"].ToString(),

LastName = item["LastName"].ToString()

};

resultItems.Add(personItem);

}

return resultItems;

}

public Person GetById(int id)

{

var spContext = SharePointContext.GetClientContext();

var spList = spContext.Web.Lists.GetByTitle("Person");

spContext.Load(spList);

spContext.ExecuteQuery();

if (spList == null || spList.ItemCount <= 0) return null;

var camlQuery = new CamlQuery

{

ViewXml = @"<View>

<RowLimit>20</RowLimit>

</View>",

DatesInUtc = true

};

var listItems = spList.GetItemById(id);

var personItem = new Person

{

Id = listItems.Id,

Name = listItems["Name"].ToString(),

LastName = listItems["LastName"].ToString()

};

return personItem;

}

}

Ahora lo que queremos es mostrar estos datos en nuestra aplicación para ello dentro de nuestro HomeController agregamos este método:

public HomeController(IPersonService person)

{

personService = person;

}

public ActionResult Person(string code)

{

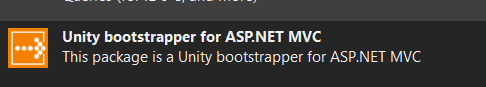
var list = personService.GetAll();

//Show the contacts

return View(list);

}

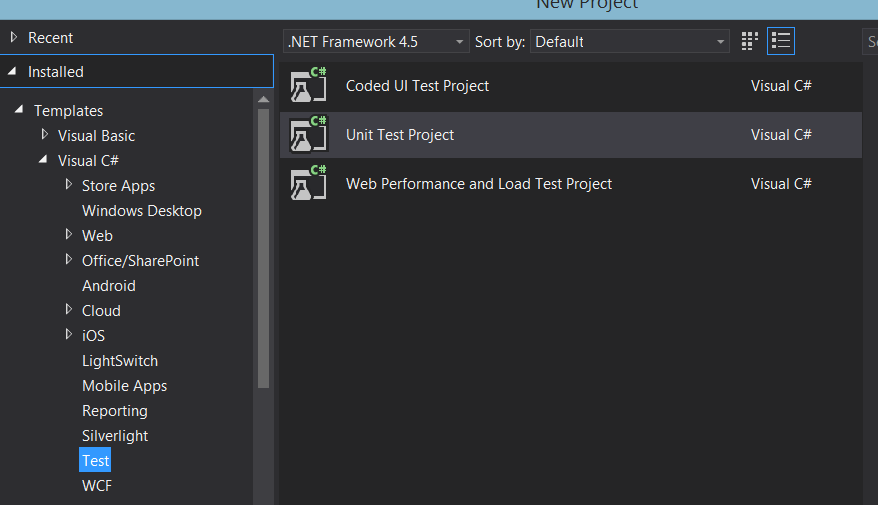
En este controlador vamos a inyectar la interfaz IPersonService mediante la técnica de inyección de dependencias. En este ejemplo hemos utilizado el FrameWork Unity (aunque la elección de la herramienta dependerá de otros factores que no vienen al caso), Para agregar Unity en nuestra solución lo que vamos a realizar es agregar el siguiente paquete de Nuget:



Esto nos agregará una seria de archivos a nuestra solución, abrimos el fichero de configuración de Unity y agregamos esta instrucción:

container.RegisterType<IPersonService, PersonService>();

Ahora creamos un proyecto de Visual Studio-> C# -> Test -> Unit Test Project



Modificamos el nombre de la Clase por defecto por UTPersonService y agregaremos las siguientes pruebas:

public class UTPersonService

{

private IPersonService personService;

[TestInitialize]

public void Init()

{

personService=new PersonMockService();

}

[TestMethod]

public void GetAll()

{

var result = personService.GetAll();

Assert.AreEqual(result.Any(),true);

}

[TestMethod]

public void GetByIdFail()

{

var result = personService.GetById(99);

Assert.AreEqual(result==null, true);

}

[TestMethod]

public void GetById()

{

var result = personService.GetById(1);

Assert.AreEqual(result.Id==1, true);

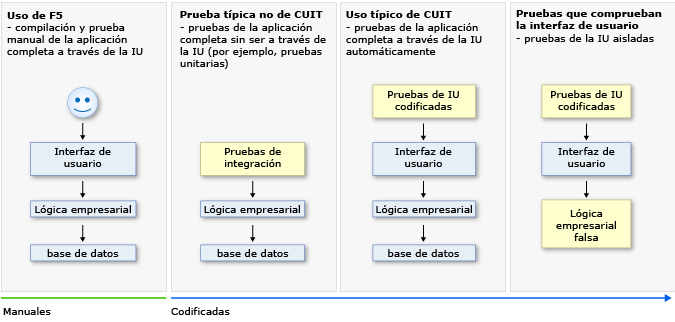
}

}

## Laboratorio 2 Pruebas UI

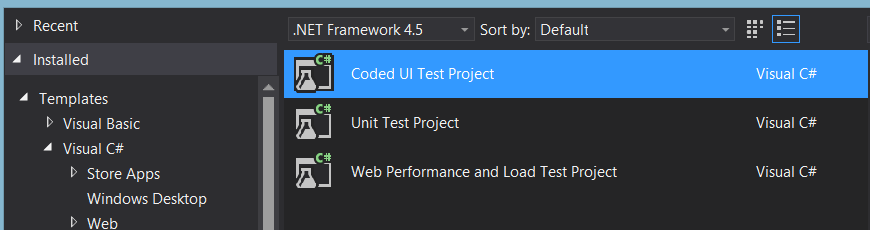
Estas pruebas incluyen una comprobación funcional de los controles de la interfaz de usuario. Permiten comprobar si toda la aplicación, incluida la interfaz de usuario, funciona correctamente. Las pruebas de IU codificadas son especialmente útiles donde haya una validación o cualquier otra lógica en la interfaz de usuario, por ejemplo, en una página web. También se suelen usar para automatizar una prueba manual existente.

Como se muestra en la ilustración siguiente, una experiencia típica de desarrollo podría ser aquella donde, inicialmente, el usuario se limite a compilar la aplicación (F5) y a hacer clic en los controles de la interfaz de usuario a fin de comprobar que todo funciona correctamente. Después, puede decidir crear una prueba codificada de forma que no sea necesario seguir probando la aplicación manualmente. Dependiendo de la funcionalidad concreta que se prueba en la aplicación, puede escribir código para una prueba funcional o bien una prueba de integración que puede que incluya o no la realización de pruebas en el nivel de interfaz de usuario. Si simplemente desea tener acceso directamente a alguna lógica de negocios, puede codificar una prueba unitaria. Sin embargo, en algunas circunstancias, puede ser beneficioso incluir pruebas de los diversos controles de IU en la aplicación. Una prueba de IU codificada puede automatizar el escenario inicial (F5), comprobando que la renovación de código no afecte a la funcionalidad de la aplicación.

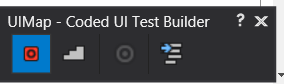


**Manos a la obra**

1. Creamos un proyecto de Visual Studio-> C# -> Test -> Coded UI



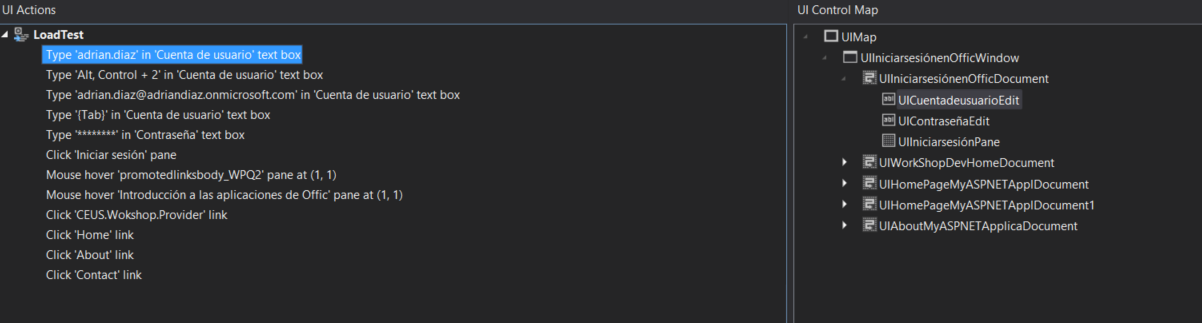
1. Grabamos una prueba de UI



Una vez finalizada pulsamos sobre el botón que se encargará de generar el código de la prueba.

1. Ver el código generado
2. Edite los detalles de las acciones de prueba y las aserciones.

Abra UIMap.uitest. Este archivo se abre en el editor de pruebas de IU codificadas, donde puede editar cualquier secuencia de acciones que haya grabado, así como las aserciones.



1. Ejecute la prueba.

Use el explorador de pruebas o abra el menú contextual del método de prueba y, a continuación, elija Ejecutar pruebas.

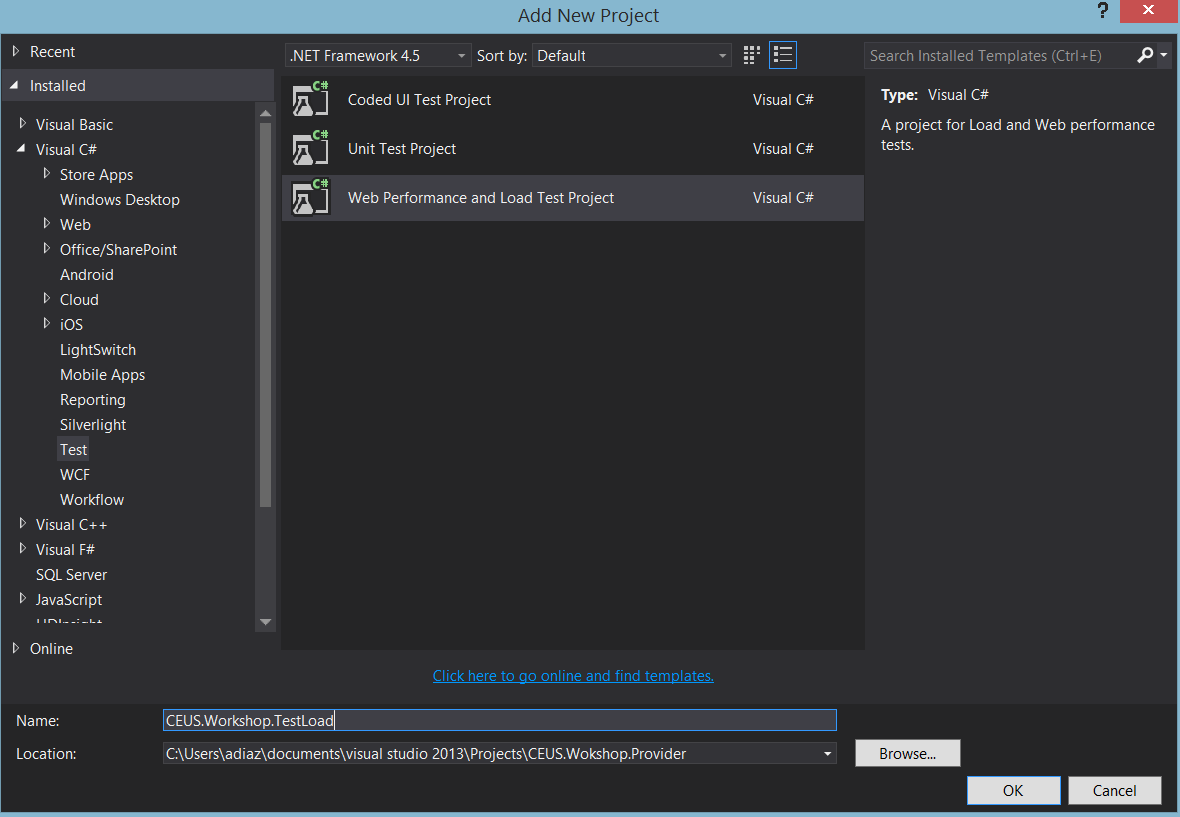
## Laboratorio 3 Pruebas de Carga

Un test de carga está compuesto por un conjunto de juegos de prueba que pueden ser de todo tipo. Pero en el caso de querer probar una aplicación web, lo más recomendable es generar ese juego de pruebas que vamos a ejecutar en formato de Web Performance Tests.

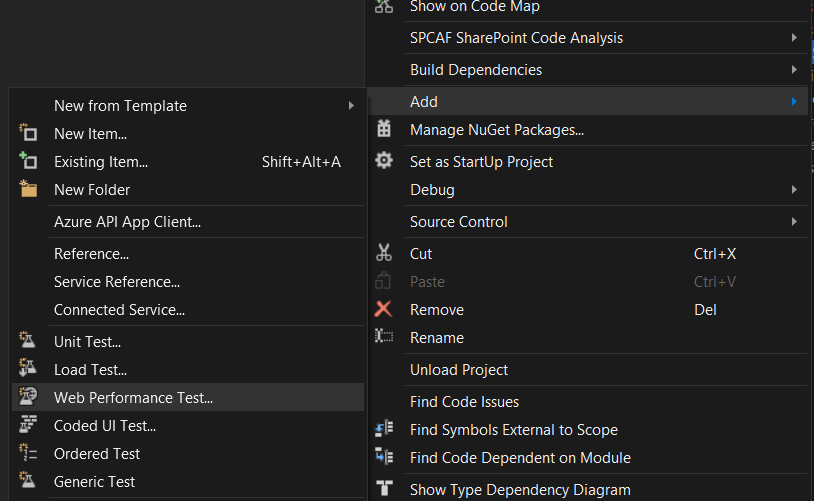
Un Web Performance Test está compuesto por una serie peticiones web, dentro de estas peticiones podemos especificar varias propiedades. Además podemos crear unas reglas para extraer datos que vienen en la respuesta y unos validadores que nos ayudarán a saber si el resultado de esa petición es el esperado.

**Manos a la obra**

1. Creamos un proyecto de Visual Studio-> C#-> Test-> Web And Performance Test



1. Grabar una Prueba de Carga
   * Cree una prueba de rendimiento web.

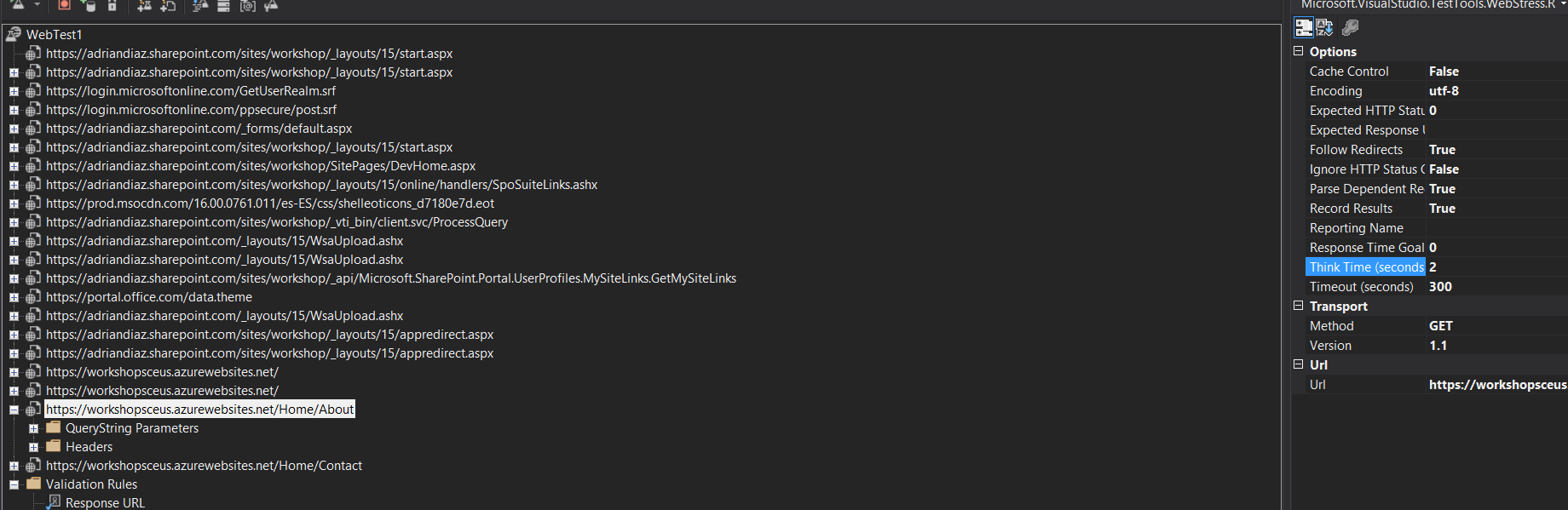


1. Especifique la dirección URL del sitio web que desea probar.
   * Cuando termine, detenga la grabación
2. Utilice la aplicación como espera que lo hagan los clientes
3. La grabadora capturará las solicitudes HTTP y las respuestas.
4. Cuando termine, detenga la grabación.

Ahora, Visual Studio busca los parámetros dinámicos de las respuestas HTTP en cada una de las solicitudes HTTP. Mientras tanto, se muestra una barra de progreso.

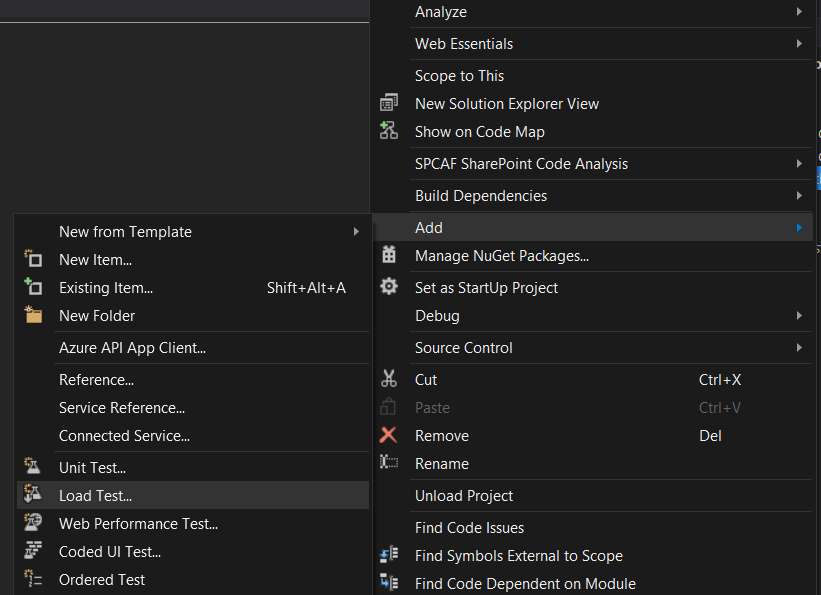
Si se encuentran parámetros dinámicos, aparecerá una tabla. Esta tabla le permitirá [asignar valores constantes a cada uno de los parámetros dinámicos](https://msdn.microsoft.com/es-es/library/ff356199.aspx).

1. Edite las propiedades de la prueba para especificar objetivos de rendimiento. Por ejemplo, puede establecer el objetivo de tiempo de respuesta de la página en 1 segundo.



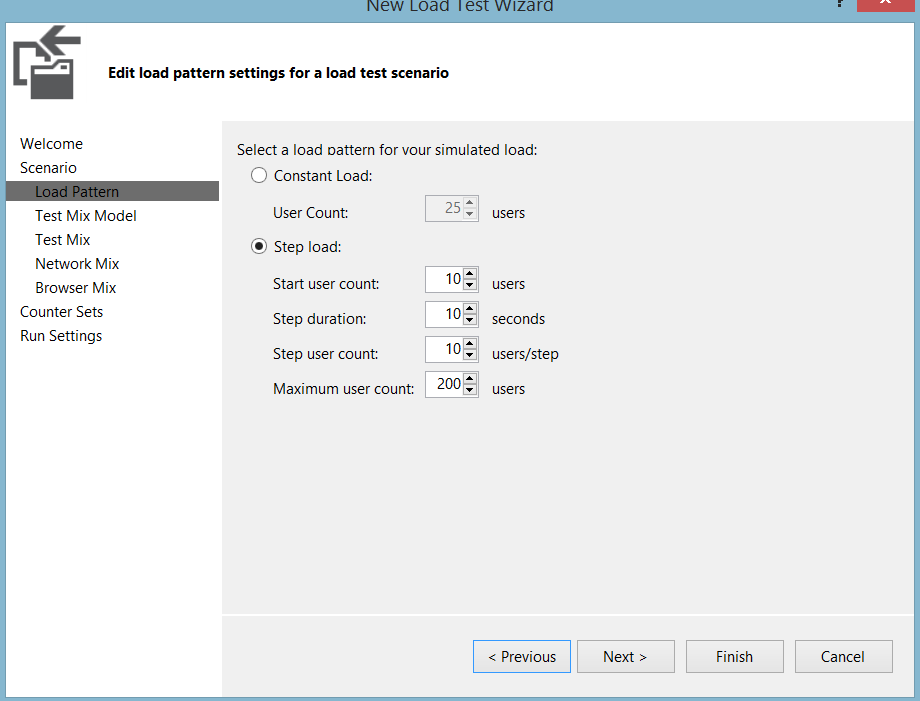
## [Crear una prueba de carga](javascript:void(0))

1. Cree una nueva prueba de carga en el proyecto de rendimiento web y prueba de carga.

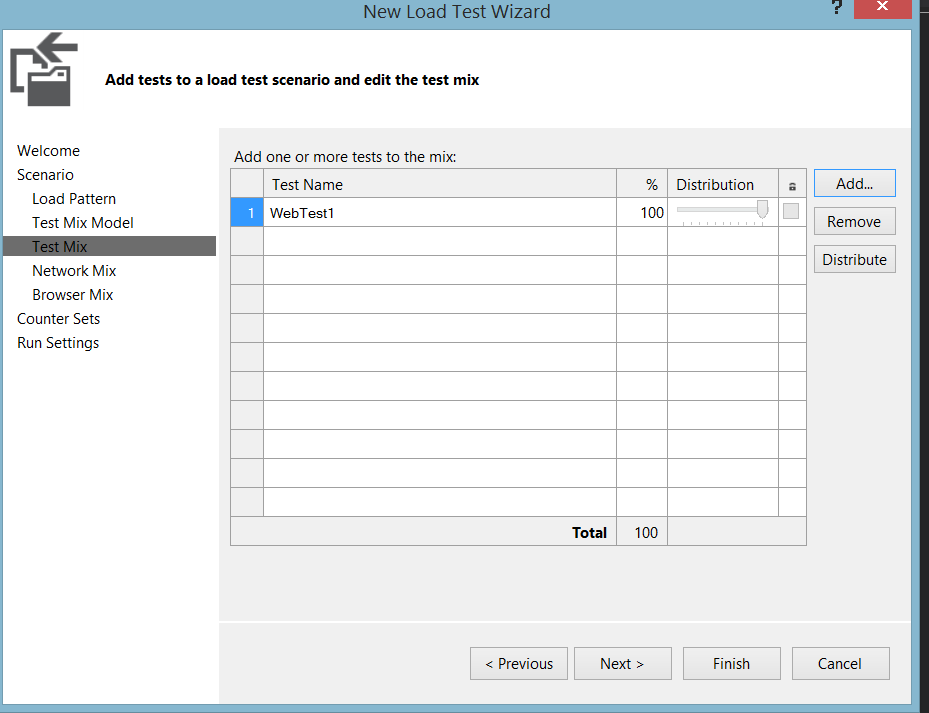


1. Cuando aparezca el Asistente para pruebas de carga, elija el paso Patrón de carga.

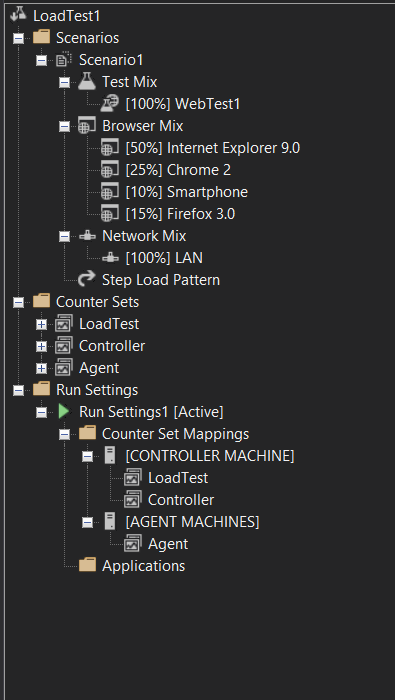
Cambie el patrón de carga a carga por pasos. De este modo, los usuarios se irán agregando gradualmente con el transcurso del tiempo.



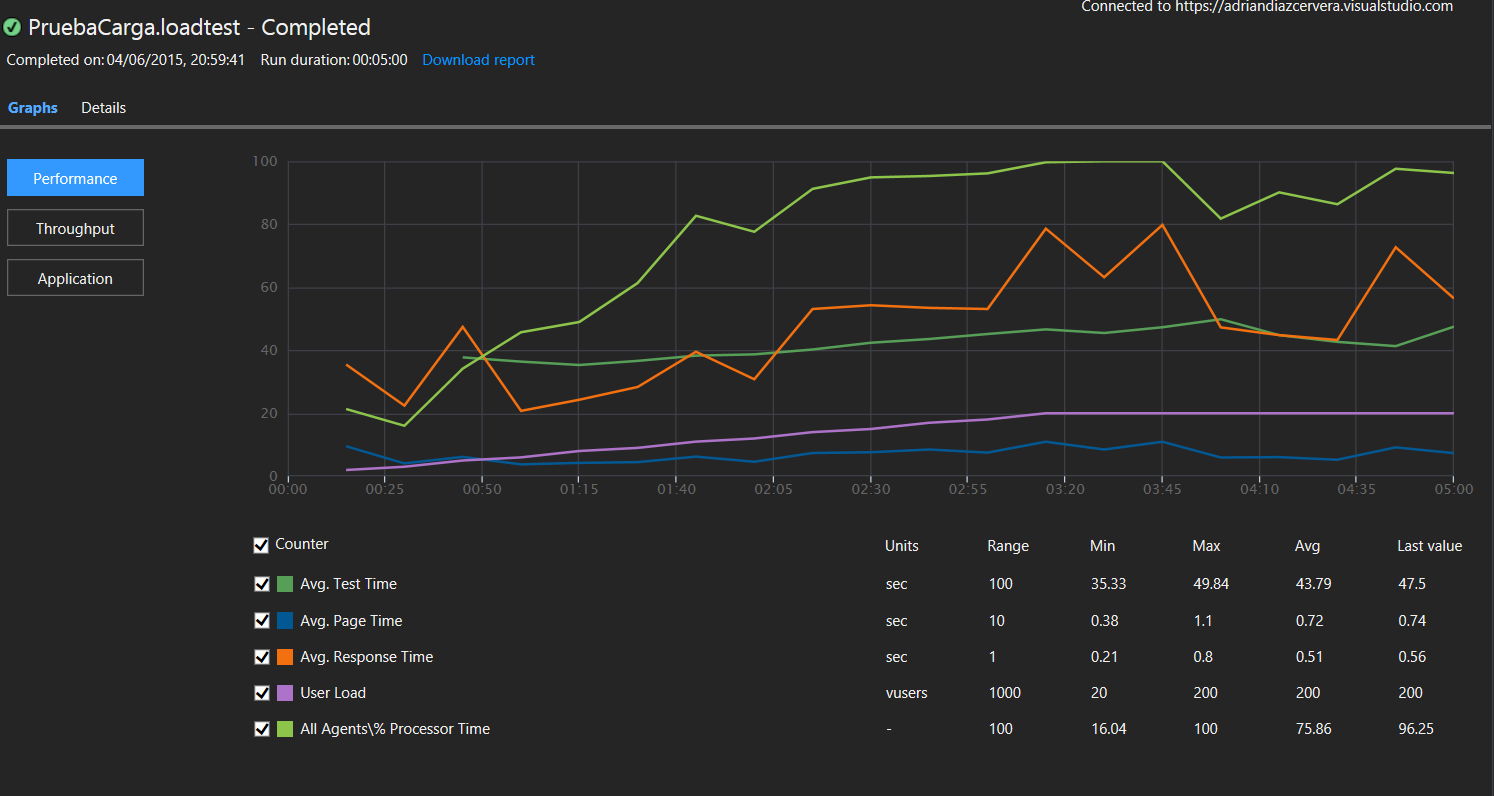
1. Elija el Patrón de Pruebas Agregue la prueba de rendimiento web que creamos.



1. Cuando finalice el asistente, la prueba de rendimiento web se agregará a la prueba de carga y aparecerá en el Editor de prueba de carga

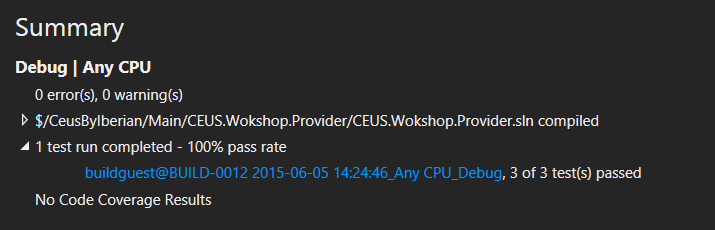


1. Ejecutamos el test y podemos observar una gráfica similar a la siguiente:



# Finalizar

Ahora volvemos a Lanzar la BUILD y veremos como ya están integradas las pruebas dentro de nuestro proceso.



# Resumen del laboratorio

En el anterior laboratorio fue posible ejercitarse en:

* Crear Pruebas Unitarias
* Crear Pruebas de Interfaz Grafica
* Crear Pruebas de Carga.
* Grabar una Prueba de Carga
* Configurar el Test de Prueba de Carga