Documentación Test C#

En primer lugar, se ha creado un proyecto con Xamarin para la elaboración del test. En el he incluido el proyecto de Android, de iOS y un proyecto de test.

En el proyecto común, se han añadido los siguientes paquetes NuGets:

- Fody
- PropertyChanged.Fody
- Newtonsoft.Json
- Xamarin.CommunityToolkit

En el proyecto de test, se han añadido los siguientes paquetes NuGets:

- FluentAssertions
- Moq

Además en el proyecto de test se ha añadido la referencia al proyecto común para poder instanciar objetos del mismo.

En cuanto a la API, he tenido que realizar algunas modificaciones para corregir errores existentes.

```
[HttpPost]
public IHttpActionResult Create([FromBody] TodoItem item)
{
	if (item == null)
	{
		return BadRequest();
	}

	TodoItems.Add(item);
	//return CreatedAtRoute("GetTodo", new { id = item.Key }, item);
	return Ok();
}

[HttpPut]
//[Route("{id}")]
public IHttpActionResult Update([FromBody] TodoItem item)
{
	if (item == null)
	{
		return BadRequest();
	}

	var todo = TodoItems.Find(item.Key);
	if (todo == null)
	{
		return NotFound();
	}

	item.Key = todo.Key;

	TodoItems.Update(item);
	return Ok();
}
```

```
[HttpDelete]
//[Route("{id}")]
public IHttpActionResult Delete(string id)
{
        var todo = TodoItems.Find(id);
        if (todo == null)
        {
                return NotFound();
        }

        TodoItems.Remove(id);
        return Ok();
}
```

He corregido, estos métodos y las pruebas las he realizado con Postman para comprobar su funcionamiento.

A continuación se detallan las historias:

- Primera historia de Usuario:

Se crea un **BaseVM**, que contiene toda la lógica de navegación entre vistas de la aplicación, propiedades que vamos a utilizar en las diferentes ViewModel que hereden de este Base, y el DependecyService para poder resolver lo servicios como el HttpClientService.

Se añade al proyecto un control con un ActivityIndicador, para mostrar por pantalla que se está ejecutando una operación. Además de mostrar el indicador de actividad se podrá visualizar también un mensaje mediante una BindableProperty. Los textos de la aplicación se manejan mediante un fichero de recursos (.resx). Para poder utilizar estos textos en las vistas se añade también una extensión de marcado, para poder bindear los textos a las propiedades de Text.

Se crean varios ficheros de estilos, uno de colores y otro de estilos del control Label.

En la carpeta de **Services**, se ha añadido el HttpClientService, con el cual se van a realizar todas las operaciones CRUD a la API.

En la carpeta de **Model**, se ha añadido el modelo del objeto con el cual se va a trabajar. Este modelo tiene tres propiedades. Además se han incluido los métodos que puede contener en este modelo. De esta manera tenemos la lógica encapsulada en el modelo y se puede instanciar desde cualquier sitio.

- Segunda Historia de Usuario:

Se ha creado una nueva vista con su ViewModel para añadir la funcionalidad de añadir un nuevo item.

En el modelo se ha añadido un nuevo método para realizar una petición POST para guardar el nuevo item.

Además se ha añadido el servicio de diálogos. Con el que se puede informar al usuario que el campo es obligatorio.

A modo de diseño se ha creado un render para eliminar el borde del entry.

- Tercera Historia de Usuario:

Se ha añadido el control “SwipeView” a los items de la lista. Cuando el usuario desplaza hacia la izquierda se muestra la opción de eliminar el item.

En el modelo se ha añadido un método para realizar una petición DELETE a la API. Si la petición es correcta se muestra un mensaje informando al usuario que la operación se ha realizado correctamente.

- Cuarta Historia de Usuario:

Se ha añadido un gesto al DataTemplate que ejecuta el comando que cambia el estado del item. Esta modificación se realiza solo a nivel local no se comunica con la API. Si se quisiera añadir esta funcionalidad simplemente habría que mandar este objeto a API con la propiedad actualizada.

- Quinta Historia de Usuario:

Se ha añadido el control “RefreshView” con su comando y la propiedad IsRefreshing. Cada vez que se ejecuta el comando llama a la API y se actualiza la lista con los datos.

En esta prueba se han incluido algunos test aunque en un proyecto real se deberían de incluir más para terminar de probar todas las funcionalidades.

(*) El test solo ha sido probado en iOS por no conseguir conectar el simulador de Android con la API localhost lanzada desde Mac.