Universidad ORT Uruguay
Facultad de Ingeniería

Diseño de Aplicaciones 2

Primer Obligatorio

Cristian Palma 208443

Federico Alonso 182999

Evidencia de Clean Code y de la aplicación de TDD

Grupo N6A

Repositorio: https://github.com/ORT-DA2/182999_208443

Índice

1. Aplicación de TDD

1.1 Descripción de la estrategia de TDD seguida

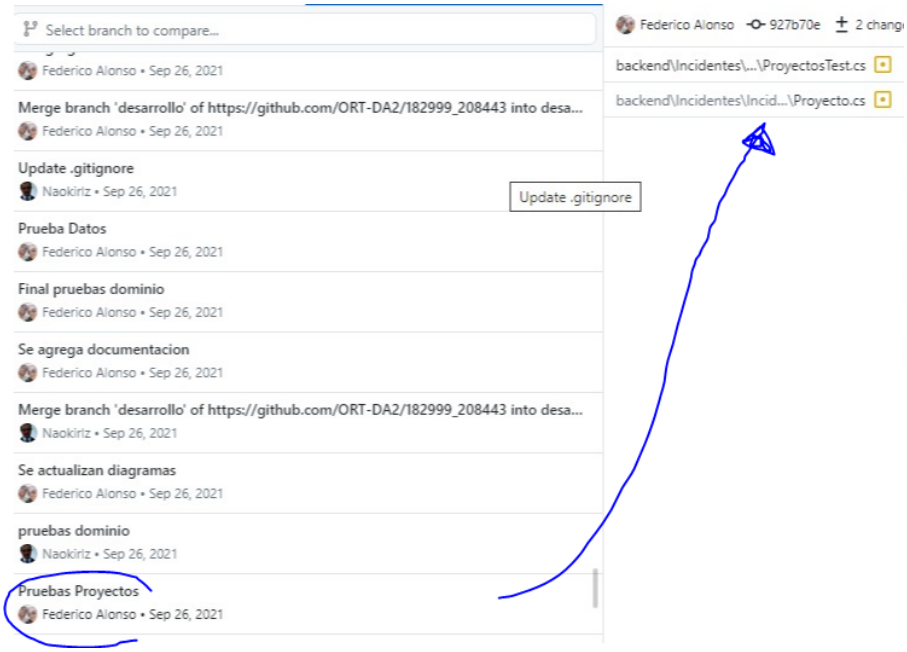
Para el desarrollo del obligatorio se hizo utilizando TDD (Test-Driven Development) siguiendo los siguientes pasos:

1- Escribir las pruebas.
2- Hacer el código mínimo para que la prueba sea fallida.
3- Hacer la implementación mínima para que las pruebas pasen.
4- Refactor para mejorar el código de las etapas anteriores.

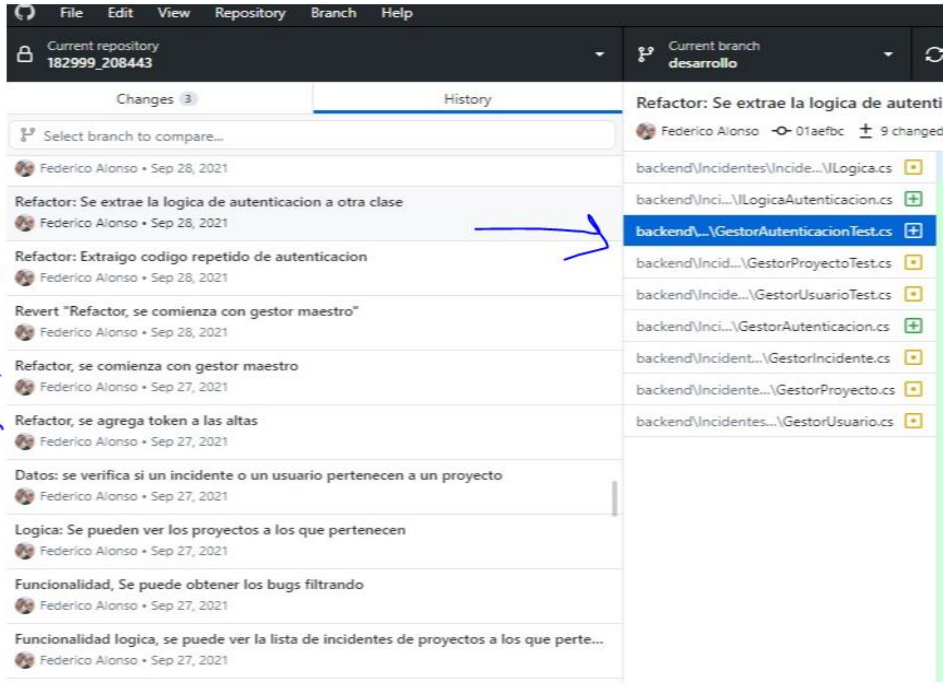
Las pruebas antes mencionadas cumplen con el principio FIRST:

- FAST. Las pruebas deben ser rápidas de ejecutarse (aislarlas de factores que las puedan enlentecer como llamadas de red)
- INDEPENDENT. Las pruebas no deben depender una de la otra. Se debe poder correr las pruebas en cualquier orden y estas deberían pasar.
- REPEATABLE. Se deben poder repetir en cualquier ambiente.
- SELF-VALIDATING. Deben tener output booleano, las propias pruebas nos deben decir si pasan o no.
- TIMELY. Las pruebas se deben escribir antes que el código que las haga pasar.

Cabe destacar que esta práctica la podemos ver reflejada en los commits realizados en GitHub, como se muestra el siguiente ejemplo a continuación:



(*) Cuando se comienza con una funcionalidad, esta se va desarrollando con sus respectivas pruebas.



.

(*) Además de las etapas de desarrollo (que llamamos funcionalidad), pasamos luego por una etapa de refactor.

1.2 Informe de cobertura para todas las pruebas desarrolladas

1.2.1 Cobertura Paquete de Datos

		% Cubiertos		
Incidentes.Datos	68	17,66 %	317	82,34 %
Contexto	5	8,06 %	57	91,94 %
RepositorioBase<T>	0	0,00 %	29	100,00 %
RepositorioGestores	0	0,00 %	19	100,00 %
RepositorioIncidente...	0	0,00 %	2	100,00 %
RepositorioProyecto...	62	46,62 %	71	53,38 %
Modificar(Incide...	62	100,00 %	0	0,00 %
ObtenerProyecto...	0	0,00 %	28	100,00 %
ObtenerProyecto...	0	0,00 %	15	100,00 %
RepositorioProye...	0	0,00 %	2	100,00 %
VerificarIncidente...	0	0,00 %	13	100,00 %
VerificarUsuarioP...	0	0,00 %	13	100,00 %
RepositorioUsuariosE...	1	0,71 %	139	99,29 %

(*) El método funciona utilizando la base de datos real, pero cuando aplicamos en las pruebas Inmemory falla. La descripción se encuentra en errores conocidos.

1.2.2 Cobertura Paquete Dominio

		% Cubiertos		
Incidentes.Dominio	2	4,00 %	48	96,00 %
Incidente	0	0,00 %	18	100,00 %
Proyecto	0	0,00 %	14	100,00 %
Usuario	2	11,11 %	16	88,89 %

(*) El 11% no cubierto corresponde a la lista virtual de proyectos que se usa para generar la relación N a N en EF.

1.2.3 Cobertura Paquete Lógica

		% Cubiertos		
Incidentes.Logica	7	0,83 %	836	99,17 %
FabricalFuente	0	0,00 %	7	100,00 %
FuenteTXT	0	0,00 %	64	100,00 %
FuenteXML	0	0,00 %	65	100,00 %
GestorAutorizacion	2	4,17 %	46	95,83 %
GestorIncidente	1	0,52 %	192	99,48 %
GestorProyecto	0	0,00 %	182	100,00 %
GestorProyecto.<>c_...	0	0,00 %	2	100,00 %
GestorUsuario	4	1,63 %	241	98,37 %
GestorUsuario.<>c__...	0	0,00 %	4	100,00 %
Validaciones	0	0,00 %	33	100,00 %

1.2.4 Controladores de la WebAPI

		% Cubiertos		
Incidentes.WebApi.Controllers	8	2,43 %	321	97,57 %
AsociacionesController	8	7,02 %	106	92,98 %
EstadosController	0	0,00 %	15	100,00 %
ImportacionesController	0	0,00 %	9	100,00 %
IncidentesController	0	0,00 %	24	100,00 %
LoginController	0	0,00 %	15	100,00 %
ProyectosController	0	0,00 %	94	100,00 %
ReportesController	0	0,00 %	44	100,00 %
UsuariosController	0	0,00 %	14	100,00 %

2. Evidencia de Clean Code

2.1 Nombres según Clean Code

- Se utilizan nombres que revelen intención y nombres pronunciables.
- Elegir sustantivos y no verbos para los nombres de las Clases.
- Para los métodos elegir verbos que indiquen lo que el método hace

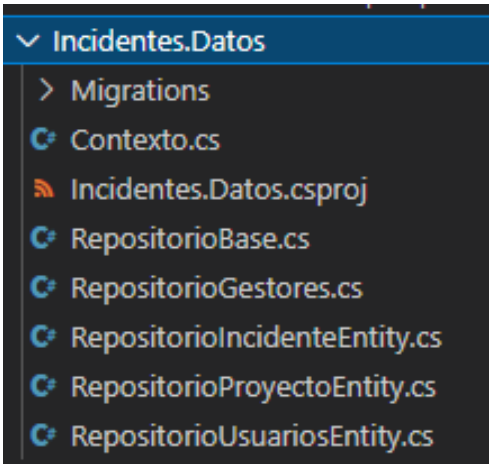
2.1.1 Variables y Properties

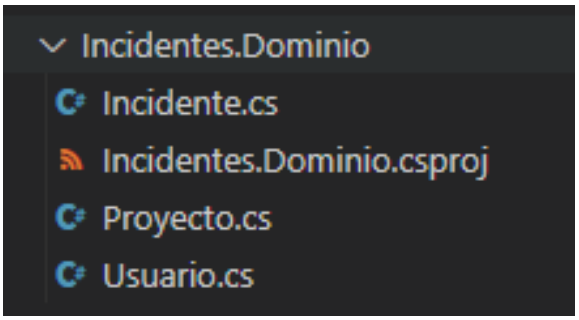
```
public class Incidente
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public int ProyectoId { get; set; }
    public string Descripcion { get; set; }
    public string Version { get; set; }
    public Estado EstadoIncidente { get; set; }
    public int DesarrolladorId { get; set; }
    public int UsuarioId { get; set; }
```

2.1.2 Métodos

```
private Usuario ObtenerPorNombreUsuario(string nombreUsuario) {
    Usuario buscado = this._repositorioGestor.RepositorioUsuario
    .ObtenerPorCondicion(c => c.NombreUsuario == nombreUsuario, false)
    .FirstOrDefault();
    return buscado;
}
```

2.1.3 Clases





2.2 Formato según Clean Code

Para el formato se tuvo en cuenta las siguientes recomendaciones:

- Leer de arriba hacia abajo tanto en importancia como en la lógica de las llamadas.
- Funciones que no tengan ningún argumento, pero en el caso de haber lo “normal” sería tolerar hasta 2.
- Evitar comentarios.
- La Metáfora del diario
- Dejar que el código respire
- Indentar consistentemente y agrupar correctamente

```
public class GestorProyecto : ILogicaProyecto
{
    IRepositorioGestores _repositorioGestor;
    private const string argumento_nulo = "El argumento no puede ser nulo";
    private const string elemento_no_existe = "El elemento no existe";
    private const string acceso_no_autorizado = "No tiene permisos para realizar dicha acción";
    private const string elemento_ya_existe = "Un elemento con similares atributos ya existe";
    private const int largo_maximo_nombre = 25;
    private const int largo_minimo_nombre = 5;

    public GestorProyecto(IRepositorioGestores repositorioGestores)
    {
        _repositorioGestor = repositorioGestores;
    }

    public void AgregarDesarrolladorAProyecto(List<int> idsUsuarios, int idProyecto)
    {
        bool existeProyecto =_repositorioGestor.RepositorioProyecto.Existe(p => p.Id == idProyecto);
        if (!existeProyecto)
            throw new ExcepcionElementoNoExiste(elemento_no_existe);
        List<Usuario> asignados = new List<Usuario>();
        foreach(int idUsu in idsUsuarios)
        {
            Usuario aAgregar = _repositorioGestor.RepositorioUsuario.ObtenerPorCondicion(d => d.Id == idUsu, false).FirstOrDefault();

            if (aAgregar != null)
            {
                asignados.Add(aAgregar);
            }
        }
        Proyecto aModificar = _repositorioGestor.RepositorioProyecto.ObtenerProyectoPorIdCompleto(idProyecto);
        aModificar.Asignados = asignados;

        _repositorioGestor.RepositorioProyecto.Modificar(aModificar);
    }

    public Proyecto Alta(Proyecto entity)
    {
        if (entity == null) throw new ExcepcionArgumentoNoValido(argumento_nulo);
        bool existe = _repositorioGestor.RepositorioProyecto.Existe(c => c.Nombre == entity.Nombre);
        if (existe) throw new ExcepcionArgumentoNoValido(elemento_ya_existe);

        Validaciones.ValidarLargoTexto(entity.Nombre, largo_maximo_nombre, largo_minimo_nombre, "Nombre Proyecto");

        _repositorioGestor.RepositorioProyecto.Alta(entity);
        _repositorioGestor.Save();

        return entity;
    }
```

2.3 Manejo de errores

Como se puede apreciar en la imagen anterior para el manejo de errores se favorecer el uso de excepciones en lugar de retornar códigos de error, como enteros o enumerados.