

Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Diseño de Aplicaciones 2

Obligatorio 1

3-TDD y Clean Code

Santiago Castro – 168347

<https://github.com/ORT-DA2/168347>

Grupo M6B

Tabla de contenido

1. TDD.....	1
2. Clean Code	8

1. TDD

En una primera instancia se implementaron las funcionalidades que no requerían hacerlas con TDD como pueden ser el manejo de usuarios y sesiones, para esto, se crearon los controladores de la WebApi UsersController y SessionsController, luego se crearon las interfaces de la lógica de negocio IUserLogic y IUserSessions con sus respectivas clases que las implementan. Adicionalmente se creó el paquete Repository el cual es el responsable de la persistencia de estos usuarios y sesiones.

Luego se identificaron otras funcionalidades que no requerían hacerlas con TDD como la de consultar el estado de una solicitud, para implementarla se tuvo que introducir ciertas clases del dominio como Area, Topic, TopicType, AdditionalField, Applicant y Request con sus respectivas lógicas y manejadores del repositorio. Aquí se introdujo la Inyección de dependencias.

En este punto se tuvo una API capaz de manejar usuarios y sesiones, además de consultar solicitudes agregadas a mano en la base de datos. Luego se decidió comenzar a implementar la primera funcionalidad con TDD que es la de ingreso de solicitudes por parte de usuarios.

Funcionalidad de creación de solicitudes:

Se creó el paquete de testing Tarea.BusinessLogic.Tests con la clase RequestTests la cual contiene el primer test de creación de solicitudes:

Etapas Red: Este test debe probar el método Create de la clase RequestLogic el cual dada una solicitud pasada por parámetro la debe insertar en la base de datos, para eso se debe utilizar los métodos Add y Save de una clase que implemente IRepository. Algunas clases del repositorio ya están implementadas ya que son necesarias para la funcionalidad de mostrar el estado de una solicitud. Esto facilitó al momento de escribir el test.

<https://github.com/ORT-DA2/168347/commit/b2d26abf1f6ca06ca5b1e8f60a09c4ed4d91da7a>

Etapas Green: En esta etapa se implementó el método Create de la clase RequestLogic el cual ya estaba creado, pero retornaba null ya que la clase implementa la interfaz ILogic. Se llama a los métodos Add y Save del repositorio.

<https://github.com/ORT-DA2/168347/commit/1221c596c416fbd85b455831b72876a1743bd37b>

Etapas Refactor: no aplicada.

Etapas Red: Se creó un nuevo test el cual prueba que si al momento de crear una solicitud con solicitante vacío debe retornar una excepción.

<https://github.com/ORT-DA2/168347/commit/d5e4a0c9ea28171d50c1a5f9aa779c89f2c0d325>

Etapas Green: Se creó un método Validate el cual verifica si los datos del solicitante están presentes en la solicitud a crear.

<https://github.com/ORT-DA2/168347/commit/45d9c1410b976379dd86c2ec5c7ae2c8c68f4d90>

Etapas Refactor: no aplicada.

De esta forma se fueron creando pruebas y luego modificando la lógica para que las pruebas sean satisfactorias. Cada ciclo se TDD se probó algo en concreto como por ejemplo si el campo detalles tiene largo mayor a 2000, si el campo Applicant.email es un email válido, etc

Green: Request AdditionalFieldData data validation between range Santiago Castro committed 23 days ago	ccd167e	<>
Red: Request AdditionalFieldData data out of range test9 Santiago Castro committed 23 days ago	9fe1496	<>
Green: Data attribute validation Santiago Castro committed 23 days ago	a71b325	<>
Red: AdditionalFieldData data null or empty test Santiago Castro committed 23 days ago	3b177d9	<>
Green: AdditionalField null validation Santiago Castro committed 23 days ago	5550742	<>
Red: AdditionalFiledData test created, null AdditionalField Santiago Castro committed 23 days ago	c88af9d	<>
Green: Moved Applicant validation to BusinessLogic, now validates email Santiago Castro committed 23 days ago	a63ef30	<>
Red: test invalid Applicant email address Santiago Castro committed 23 days ago	28207e9	<>
Refactor: deleted isAdmin attribute in User class, removed redundant ... Santiago Castro committed 23 days ago	8bb4d3f	<>
Commits on Apr 16, 2020		
Green + Refactor: added AdditionalFieldData validation Santiago Castro committed 26 days ago	7035674	<>
Red: CreateInvalidRequestTest6 additional field data invalid test Santiago Castro committed 26 days ago	8f14e48	<>
Green: Added additionalField validation Santiago Castro committed 26 days ago	e4169f8	<>
Red: added TareaExpectedException tocheck exception msg Santiago Castro committed 26 days ago	a8b06a4	<>
Red: CreateInvalidRequestTest5 Santiago Castro committed 26 days ago	beee221	<>
Green: check if request has additional fields Santiago Castro committed 26 days ago	0a08312	<>
Red: added VerifyAll to all mocks on all tests Santiago Castro committed 26 days ago	030d6c8	<>
Green: Create AdditionalFieldData when creating request Santiago Castro committed 26 days ago	10f85ea	<>
Red: Request creation test with correct additional field Santiago Castro committed 26 days ago	31b4843	<>
Green: Changed Domain architecture, Request class has TopicType and A... Santiago Castro committed 27 days ago	9aed2ed	<>
Commits on Apr 15, 2020		
Red: Area not found Santiago Castro committed 27 days ago	9f32cd7	<>
Green: Added Area validation creating request. Fixed first test Santiago Castro committed 27 days ago	2d1e02f	<>
Red: Request missing area data test Santiago Castro committed 27 days ago	8002b48	<>
Green: Added Request details validation Santiago Castro committed 27 days ago	f4d74d1	<>
Red: Request invalid test, empty information Santiago Castro committed 27 days ago	1543949	<>
Green: added Request validation Santiago Castro committed 27 days ago	45d9c14	<>
Red: new invalid test, missing applicant info Santiago Castro committed 27 days ago	d5e4a0c	<>
Green: Created Create method in RequestLogic class Santiago Castro committed 27 days ago	1221c59	<>
Red: Created Request first test: create Santiago Castro committed 27 days ago	b2d26ab	<>

Luego se pasó a implementar la parte del Backend para esta funcionalidad, para esto se creó el paquete de prueba Tarea.WebApi.Tests con la clase RequestsControllerTest y su primer test.

Etapas Red: Se creó el test CreateVaildRequestTest el cual prueba el funcionamiento del método Post de la clase RequestController pasando una instancia de ReqequestModel. El test es correcto si se ejecuta el método Create de RequestLogic y se retorna el status 200.

<https://github.com/ORT-DA2/168347/commit/b264a60a063f4d631cd83da78416bdd4226db261>

Etapas Green: Se creó el método Post de la clase RequestController el cual llama a la lógica y devuelve el status 200.

<https://github.com/ORT-DA2/168347/commit/755125afb7f97fcca625eef4eb8858a0f47dde9d>

Refactor: deleted custom exception, now it uses common System.Excepti... Santiago Castro committed 20 days ago	71cec7c	<>
Added Swagger support Santiago Castro committed 20 days ago	a06b728	<>
Refactor: moved AdditionalFieldData validation from Domain to Busines... Santiago Castro committed 20 days ago	de87330	<>
Commits on Apr 20, 2020		
Refactor: request creation logic, added Validate(T) method to ILogic Santiago Castro committed 22 days ago	4323d77	<>
Preparations for Repository, created entities repositories Santiago Castro committed 22 days ago	35fb74a	<>
Refactor: Create method now creates a new entity Santiago Castro committed 23 days ago	49c7021	<>
Commits on Apr 19, 2020		
Green: Created RequestsController class with Models Santiago Castro committed 23 days ago	755125a	<>
Red: RequestsControllerTest file created. Test valid POST Santiago Castro committed 23 days ago	b264a60	<>

Funcionalidad creación de tipos:

Etapas Red:

Se creó la clase TopicTypeTest con un test que prueba la creación de tipos. Para esto se debe llamar al repositorio para agregar el tipo a la base de datos y luego salvar el contexto. Además, se agregó la prueba en caso de que el argumento Topic es null, debe retornar una excepción.

<https://github.com/ORT-DA2/168347/commit/f81f443893ac9943907b1ae2500d1942b807c1ec>

Etapas Green: Se creó el método Create de la clase TopicTypeLogic el cual valida el parámetro de entrada y luego llama al repositorio para agregar el tipo a la base de datos.







































<https://github.com/ORT-DA2/168347/commit/fa84c4dc86522d10c783e5daf6540538ec48064b>

Etapas Red: Se prueba si el parámetro tema del tipo es null, debe retornar una excepción.

<https://github.com/ORT-DA2/168347/commit/ad88822b0225ddb89e335dedfc9b2ceee340e110>

Etapas Green: Implementada la validación al momento de crear tipos.

<https://github.com/ORT-DA2/168347/commit/187e5ab4028ef70f45bee8ae1e3c2b2b18521cf0>

Green: Added FieldType content validation Santiago Castro committed 19 days ago	 aec085e	
Red: AdditionalField FieldType invalid Santiago Castro committed 19 days ago	 9ae96a6	
Green: null or empty check for FieldType attribute Santiago Castro committed 19 days ago	 e23d72b	
Red: AdditonalField attribute FieldType invalid. Santiago Castro committed 19 days ago	 678d096	
Green: Added ValidateAdditionalFields method Santiago Castro committed 19 days ago	 df1e67b	
Red: AdditionalField attribute Name null or empty test (CreateInvalid... Santiago Castro committed 19 days ago	 4d2efbf	
Refactor: fixed minor test CreateValidTopicTypeTest1 error Santiago Castro committed 19 days ago	 46ddb31	
Green: Added DuplicateAdditionalFieldCheck Method Santiago Castro committed 19 days ago	 1b060b5	
Red: Multiple AdditonalFields with the same attribute Name. test (Cre... Santiago Castro committed 19 days ago	 43e130e	
Refactor: Topic class new attribute: List<TopicType> TopicTypes used ... Santiago Castro committed 19 days ago	 e452605	
Green: Added duplicateCheck method to TopicTypeLogic Santiago Castro committed 19 days ago	 1d24288	
Red: Duplicate TopicType name for the same Topic test Santiago Castro committed 19 days ago	 342c68f	
Refactor: moved ValidateTopic method Santiago Castro committed 19 days ago	 92e59f5	
Green: Added TopicType.Name argument validation Santiago Castro committed 19 days ago	 6ac98ce	
Red: TopicType name argument null test Santiago Castro committed 19 days ago	 bc5a407	
Green: use TopicLogic.Get() creaing TopicType Santiago Castro committed 19 days ago	 187e5ab	
Red: Topic not found Santiago Castro committed 19 days ago	 ad88822	
Green: Created Topic validation if null Santiago Castro committed 19 days ago	 fa84c4d	
Red: TopicType creation test, valid and Invalid (topic null) Santiago Castro committed 19 days ago	 f81f443	

En cada etapa de TDD se fueron creando pruebas de casos en que falla la creación de tipos como por ejemplo nombre duplicado para un tema, nombre nulo o vacío, nombres de campos adicionales duplicados, etc. Adicionalmente se fue haciendo refactor cuando se vio necesario mejorar la calidad de código extrayendo métodos o mejorando la lógica.

Etapas Red: Se creó la clase TopicTypesControllerTests con el primer caso de prueba de creación de tipos el cual prueba el método Post de la clase TopicTypesController. Este método debe llamar a la lógica y devolver el código status 201.

<https://github.com/ORT-DA2/168347/commit/6aa33956bbfe2b0141eb8f62d31b0fb03589d71a>

Etapas Green: Fue necesario crear las clases TopicTypeModel, AdditionalFieldModel, TopicModel y TopicTypeController. Luego se implementó el método Post el cual crea una instancia de TopicType y llama al método Create de TopicTypeLogic, finalmente devuelve el status 201.

<https://github.com/ORT-DA2/168347/commit/7c0141867298730c72efba3469ccdbd8aad91b13>

Funcionalidad Listar solicitudes existentes

Etapas Red: Se creó un nuevo test en RequestControllerTest donde llama al método Get de RequestController el cual debe devolver la lista de solicitudes del sistema, para eso se debe ejecutar el método GetAll de la clase RequestLogic.

<https://github.com/ORT-DA2/168347/commit/538d706482bbdbbba6b6645dc1fbfc6943da12ed>

Etapas Green: Se implementó el método Get de clase RequestsController el cual obtiene la lista de solicitudes del sistema por medio del método RequestLogic.GetAll() y devuelve la lista de solicitudes junto al código status 200.

<https://github.com/ORT-DA2/168347/commit/6130d198db3a0b40b16285b233c5cbd87f7dc540>

Funcionalidad Editar status de una solicitud

Etapas Red: Se creó el test UpdateRequestStatusTest1 en la clase RequestsControllerTest, el cual prueba el método Put de la clase RequestsController.

<https://github.com/ORT-DA2/168347/commit/b423fb28b4c1d2c20d30f167b08bbba8c91b39e4>

Etapas Green: Implementada la lógica que permita cambiar el estado de una solicitud sin a ver ninguna validación, además se implementó el método Put de la clase RequestsController el cual llama a la lógica mencionada.

<https://github.com/ORT-DA2/168347/commit/ac3181d5977ecd1d733395d9440e6d5c64e5d1dd>

Etapas Refactor: El estado de una solicitud ahora es una string en vez de un enumerado. Se pasó una parte de la lógica de validación de estado de solicitudes del Backend hacia BusinessLogic.

<https://github.com/ORT-DA2/168347/commit/13bf758b81c9863bac38cd49001d8ef651ec04c4>

Resultado de la ejecución de las pruebas y cobertura

Tarea.Repository.Tests – Se hicieron pruebas unitarias para todas las clases del paquete, La única cobertura que no fue realizada fue la ejecución de los métodos set; de los DbSet en la clase TareaContext. Sin contar esto se alcanzó un 100% de cobertura

By namespace, Level: 3							
Collapse all Expand all		Grouping:		Filter: <input type="text"/>			
Name	Covered	Uncovered	Coverable	Total	Line coverage	Branch coverage	
+ Tarea.Domain	18	20	38	117	47.3%		
- Tarea.Repository	133	4057	4190	3125	3.1%		
- Tarea.Repository	133	9	142	328	93.6%		
Tarea.Repository.AdditionalFieldDataRepository	11	0	11	27	100%		
Tarea.Repository.AdditionalFieldRepository	11	0	11	27	100%		
Tarea.Repository.ApplicantRepository	11	0	11	27	100%		
Tarea.Repository.AreaRepository	11	0	11	27	100%		
Tarea.Repository.BaseRepository' 1	13	0	13	36	100%		
Tarea.Repository.RequestRepository	16	0	16	32	100%		
Tarea.Repository.SessionRepository	11	0	11	27	100%		
Tarea.Repository.TareaContext	15	9	24	43	62.5%		
Tarea.Repository.TopicRepository	11	0	11	27	100%		
Tarea.Repository.TopicTypeRepository	12	0	12	28	100%		
Tarea.Repository.UserRepository	11	0	11	27	100%		
+ Tarea.Repository.Migrations	0	4048	4048	2797	0%		

Tarea.BusinessLogic.Tests – Se hicieron pruebas unitarias para todas las clases de la lógica alcanzando un 100% de cobertura.

By assembly							
Collapse all Expand all		Grouping:		Filter: <input type="text"/>			
Name	Covered	Uncovered	Coverable	Total	Line coverage	Branch coverage	
- Tarea.BusinessLogic	529	0	529	854	100%		
Tarea.BusinessLogic.AdditionalFieldDataLogic	80	0	80	121	100%		
Tarea.BusinessLogic.AdditionalFieldLogic	63	0	63	100	100%		
Tarea.BusinessLogic.ApplicantLogic	35	0	35	65	100%		
Tarea.BusinessLogic.AreaLogic	23	0	23	48	100%		
Tarea.BusinessLogic.RequestLogic	129	0	129	184	100%		
Tarea.BusinessLogic.SessionLogic	41	0	41	72	100%		
Tarea.BusinessLogic.TopicLogic	23	0	23	48	100%		
Tarea.BusinessLogic.TopicTypeLogic	75	0	75	119	100%		
Tarea.BusinessLogic.UserLogic	60	0	60	97	100%		
- Tarea.Domain	28	10	38	117	73.6%		
Tarea.Domain.AdditionalField	4	0	4	12	100%		
Tarea.Domain.AdditionalFieldData	2	1	3	13	66.6%		
Tarea.Domain.Applicant	3	1	4	12	75%		
Tarea.Domain.Area	0	3	3	12	0%		
Tarea.Domain.Request	7	0	7	16	100%		
Tarea.Domain.Topic	2	2	4	13	50%		
Tarea.Domain.TopicType	4	1	5	14	80%		
Tarea.Domain.User	4	0	4	13	100%		
Tarea.Domain.UserSession	2	2	4	12	50%		
+ Tarea.Repository	0	4190	4190	3125	0%		
- Tarea.Tools	26	7	33	72	78.7%		
Tarea.Tools.EmailValidation	8	0	8	18	100%		
Tarea.Tools.TareaException	5	2	7	19	71.4%		
Tarea.Tools.TareaExpectedException	13	5	18	35	72.2%		

Tarea.WebApi.Tests – Se hicieron pruebas unitarias para todas las clases del paquete Controllers alcanzando un 100% de cobertura. Debido a la naturaleza de los filtros, no es posible probarlos de forma fácil, se consultó con el docente y se decidió no hacer pruebas unitarias, el funcionamiento de estos filtros se puede probar a través de pruebas de integración detalladas en el siguiente capítulo. De forma similar, no se hicieron pruebas unitarias a la clase ErrorModel del paquete Tarea.WebApi.Model por ser una clase que depende mucho de la entrada y sería tedioso probarla, lo ideal es probar su funcionamiento con pruebas de integración.

Coverage								
By namespace, Level: 3								
Collapse all Expand all								
Filter:								
Name	Covered	Uncovered	Coverable	Total	Line coverage	Branch coverage		
+ Tarea.BusinessLogic	0	529	529	854	0%	0%		
+ Tarea.Domain	30	8	38	117	78.9%			
+ Tarea.Repository	0	4190	4190	3125	0%			
+ Tarea.Tools	18	15	33	72	54.5%	100%		
- Tarea.WebApi	295	204	499	1046	59.1%	10.4%		
- Tarea.WebApi	0	63	63	123	0%	0%		
Tarea.WebApi.Program	0	8	8	29	0%			
Tarea.WebApi.Startup	0	55	55	94	0%	0%		
- Tarea.WebApi.Controllers	117	0	117	362	100%	100%		
Tarea.WebApi.Controllers.AreasController	16	0	16	51	100%	100%		
Tarea.WebApi.Controllers.RequestsController	29	0	29	78	100%	100%		
Tarea.WebApi.Controllers.SessionsController	12	0	12	46	100%			
Tarea.WebApi.Controllers.TopicTypesController	20	0	20	62	100%	100%		
Tarea.WebApi.Controllers.TopicsController	11	0	11	40	100%	100%		
Tarea.WebApi.Controllers.UsersController	29	0	29	85	100%	100%		
- Tarea.WebApi.Filters	0	37	37	82	0%	0%		
Tarea.WebApi.Filters.AuthFilter	0	30	30	60	0%	0%		
Tarea.WebApi.Filters.ExceptionFilter	0	7	7	22	0%			
- Tarea.WebApi.Models	178	104	282	479	63.1%	6.1%		
Tarea.WebApi.Models.AdditionalFieldDataModel	13	0	13	26	100%			
Tarea.WebApi.Models.AdditionalFieldModel	22	0	22	36	100%	100%		
Tarea.WebApi.Models.ApplicantModel	16	0	16	28	100%			
Tarea.WebApi.Models.AreaModel	13	0	13	25	100%	100%		
Tarea.WebApi.Models.ErrorModel	0	104	104	169	0%	0%		
Tarea.WebApi.Models.LoginModel	2	0	2	12	100%			
Tarea.WebApi.Models.RequestModel	36	0	36	54	100%	100%		
Tarea.WebApi.Models.RequestStatusModel	13	0	13	29	100%			
Tarea.WebApi.Models.TopicModel	13	0	13	25	100%	100%		
Tarea.WebApi.Models.TopicTypeModel	28	0	28	41	100%	100%		
Tarea.WebApi.Models.UserModel	22	0	22	34	100%			

2. Clean Code

Este trabajo fue realizado teniendo en cuenta los criterios de Clean Code para que el código sea fácil de leer, entender y modificar. Para esto se aplicaron los siguientes puntos:

Nombres significativos (métodos y variables):

- Revelan la intención: el nombre de los métodos explica qué hace. Nombre de variables explica para qué va a ser usada esa variable.
- Pronunciables
- No incluye su tipo: variables no incluyen su tipo (nombreString)
- Convenciones: Interfaces comienzan con la letra I (ILogic, IRequestLogic, IUserLogic)

```
public Request Create(Request request)
{
    GenerateRequestData(request);
    Validate(request);
}
```

```
11 references
private IRepository<Request> requestRepository;
3 references
private ILogic<Applicant> ApplicantLogic;
2 references
private ILogic<TopicType> TopicTypeLogic;
3 references
private ILogic<AdditionalFieldData> AdditionalFieldDataLogic;
```

```
C# IAreaLogic.cs
C# ILogic.cs
C# IRequestLogic.cs
C# ISession.cs
C# ITopicLogic.cs
C# ITopicTypeLogic.cs
C# IUserLogic.cs
C# IUserSession.cs
```

Funciones:

- Chicas
- Pocos bloques
- Hacen una sola cosa y la hacen bien
- La menor cantidad de parámetros posible
- Excepciones en vez de códigos de error

Comentarios:

- No se usaron, el código se debe explicar por si solo

Estilo:

- Si una función llama a otra entonces tienen que estar cerca
- Cada nivel sigue su sangría

```
1 reference
private void GenerateRequestData(Request entity)
{
    ValidateTopicType(entity.TopicType);
    entity.Id = Guid.NewGuid();
    entity.Status = "Created";
    entity.StatusDescription = "";
    entity.TopicType = TopicTypeLogic.Get(entity.TopicType.Id);
}

1 reference
public void Validate(Request request)
{
    if (String.IsNullOrEmpty(request.Details))
    {
        throw new TareaException("ERR_REQUEST_DETAILS_NULL_EMPTY");
    }
    if (request.Details.Length > 2000)
    {
        throw new TareaException("ERR_REQUEST_DETAILS_LONG");
    }
    ApplicantLogic.Validate(request.Applicant);
    ValidateTopicType(request.TopicType);
    ValidateAdditionalFields(request.AdditionalFields, request.TopicType);
}

1 reference
private List<AdditionalFieldData> GenerateFields(List<AdditionalFieldData> additionalFields)
{
    var fields = new List<AdditionalFieldData>();
    if (additionalFields != null)
    {
        additionalFields.ForEach(af => fields.Add(AdditionalFieldDataLogic.Create(af)));
    }
    return fields;
}

2 references
private void ValidateTopicType(TopicType topicType)
{
    if (topicType == null)
    {
        throw new TareaException("ERR_REQUEST_TOPIC_TYPE_NULL_NOTFOUND");
    }
}
```

Objetos y estructuras de datos:

- Ocultar implementación de métodos a través de interfaces: ILogic, IRequestLogic, etc.
- Uso de polimorfismo para definir distinto comportamiento de una función para distintos tipos de objetos que implementen algo en común: Clase abstracta BaseRepository.

Manejo de errores:

- Independiente de la lógica: Uso de ExceptionFilter el cual su funcionamiento ya fue explicado.

Clases:

- Chicas y con pocas responsabilidades.