



FORMATO PRUEBA DE SOFTWARE

- **Denominación del Programa de Formación:** Tecnología en Análisis y Desarrollo de software
- **Nombre del Proyecto:** GESCOMPH
- **Fase del Proyecto:** Ejecución
- **Actividad de Proyecto:** Construir la capa de datos, lógica de negocios y presentación del sistema de información aplicando estándares de calidad y buenas prácticas de ergonomía.
- **Competencia:** Construir el sistema que cumpla con los requisitos de la solución informática.
- **Resultados de aprendizaje a alcanzar:** Ejecutar y documentar las pruebas del software, aplicando técnicas de ensayo-error, de acuerdo con el plan diseñado y los procedimientos establecidos por la empresa

INTRODUCCIÓN

Este documento consolida el proceso de verificación y validación del sistema **GESCOMPH**, integrando pruebas unitarias orientadas a garantizar la calidad técnica del software durante su construcción. El objetivo principal es asegurar que los componentes definidos en la arquitectura multicapa —**Entidad, Datos, Negocio y Presentación**— funcionen de manera individual y en conjunto, cumpliendo los requisitos funcionales y no funcionales establecidos.

Las **pruebas unitarias**, como parte central del proceso, permiten evaluar de manera aislada métodos críticos, repositorios, servicios y controladores. Su ejecución permite identificar defectos tempranamente, mejorar la robustez del código y fortalecer el proceso de integración continua.

Dentro del alcance del proyecto se evalúan tanto:

Requerimientos funcionales, orientados a:

- Validar el comportamiento esperado de los módulos.
- Verificar cálculos de obligaciones, filtros de contratos, roles, parámetros del sistema y lógica comercial.
- Confirmar la correcta persistencia, consulta y eliminación de datos en la capa de infraestructura.

Requerimientos no funcionales, relacionados con:

- Rendimiento en operaciones de consulta.
- Seguridad en procesos de autenticación y validación de acceso.
- Fiabilidad en cálculos financieros basados en UVT.



- Usabilidad en las respuestas entregadas por la API.

Para garantizar una verificación completa, se consideran los cuatro tipos principales de pruebas:

1. Pruebas Unitarias

Evaluán métodos o componentes específicos, por ejemplo:

- Repositorios (ContractRepository, UserRepository, EstablishmentsRepository...)
- Servicios de negocio (ContractBusiness, ObligationMonthBusiness...)
- Controladores (ContractController)



1. Contenido

1.	Contenido	3
1.	CAPA DATA.....	5
1.1.	GetByPersonAsync.....	5
1.2.	ObligationMonthRepository	8
1.3.	GetByContractQueryableOrdersDescAndFiltersDeleted.....	10
1.4.	GetTotalObligationsPaidByDayAsyncReturnsSumForGivenDate.....	12
1.5.	GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth	14
1.6.	GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth	16
2.	SystemParameterRepository	18
3.	PersonRepository.....	21
4.	EstablishmentsRepository.....	23
4.1.	GetBasicsByIdsAsyncProjectsValues	25
4.2.	GetInactiveIdsAsyncReturnsOnlyInactive	28
5.	UserRepository	30
5.1.	GetByEmailVariants.....	33
5.2.	GetByPersonIdReturnsUser	35
5.3.	GetByIdReturnsUserWithIncludes	37
6.	RolRepository.....	39
6.1.	GetAllExcludesDeletedAndOrdersByCreatedAtDescThenIdDesc	41
6.2.	UpdatePreservesIdAndCreatedAt	43
6.3.	DeleteRemovesEntity	45
7.	Capa Bussines	47
7.1.	Usuario NO admin sin persona asociada	49
7.2.	Usuario NO admin con persona asociada obtiene contratos filtrados.....	51
7.3.	Id inválido en GetObligationsAsync.....	53
7.4.	Delegación al servicio de obligaciones	55
8.	ObligationMonthBusiness	57
8.1.	Cálculo del total pagado por mes	59
9.	SystemParameterBussines.....	61
10.	EstablishmentsBusiness	63
10.1.	UpdateAsync retorna null cuando no existe.....	67
10.2.	UpdateAsync correcto: recalcula valor UVT.....	69
11.	Capa controller	71
11.1.	DownloadContractPdf retorna 404 cuando el contrato no existe	73



11.2. POST Create retorna 200 OK (no 201)	75
11.3. ChangeActiveStatus retorna 204 No Content.....	77
11.4. Delete retorna 204 No Content.....	78
11.5. GetObligations retorna 404 cuando el contrato no existe	81
11.6. GetObligations retorna 200 OK cuando el contrato existe	83



1. CAPA DATA

1.1. GetByPersonAsync

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetByPersonAsync

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Obtener todos los contratos asociados a una persona, cargando además las relaciones con Persona y User.
Objetivo	Verificar que GetByPersonAsync(personId) filtra por persona correctamente y carga la relación Person → User.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	1
Procedimiento de Prueba	<ul style="list-style-type: none">• Crear ApplicationDbContext InMemory.• Insertar personas, usuarios y contratos (solo uno pertenece a PersonId = 1).• Ejecutar SaveChangesAsync().• Llamar a repo.GetByPersonAsync(1).• Validar: Solo retorna 1 contrato. Relación cargada: Person.User.Email = "p1@mail".
Datos de salida - Resultado Esperado	Un único contrato con PersonId = 1. Relación Person → User cargada



	correctamente.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()



```
1
2 [Fact]
3 public async Task GetByPersonAsyncFiltersByPersonAndLoadsRelations()
4 {
5     await using var ctx = CreateContext();
6     var repo = new ContractRepository(ctx);
7
8     var p1 = CreatePerson(1);
9     var p2 = CreatePerson(2);
10
11    var u1 = new User { Id = 1, Email = "p1@mail", Password = "x", Person = p1 };
12    var u2 = new User { Id = 2, Email = "p2@mail", Password = "x", Person = p2 };
13
14    p1.User = u1;
15    p2.User = u2;
16
17    ctx.Persons.AddRange(p1, p2);
18    ctx.Users.AddRange(u1, u2);
19
20    var now = DateTime.UtcNow;
21
22    ctx.Contracts.AddRange(
23        new Contract
24        {
25            Id = 1,
26            PersonId = 1,
27            Person = p1,
28            StartDate = now.AddMonths(-2),
29            EndDate = now.AddMonths(1),
30            Active = true,
31            IsDeleted = false,
32            CreatedAt = now.AddDays(-1)
33        },
34        new Contract
35        {
36            Id = 2,
37            PersonId = 2,
38            Person = p2,
39            StartDate = now.AddMonths(-1),
40            EndDate = now.AddMonths(2),
41            Active = true,
42            IsDeleted = false,
43            CreatedAt = now
44        }
45    );
46
47    await ctx.SaveChangesAsync();
48
49    var result = (await repo.GetByPersonAsync(1)).ToList();
50
51    result.Should().ContainSingle();
52    result.Single().Person!.User!.Email.Should().Be("p1@mail");
53 }
```



1.2. ObligationMonthRepository

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetByContractYearMonthReturnsExpected

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Obtener una obligación según contrato-año-mes.
Objetivo	Validar que el método devuelve la obligación correcta o null cuando no coincide.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	ContractId = 5, Year = 2024, Month = 2
Procedimiento de Prueba	<ul style="list-style-type: none">• Crear contexto InMemory.• Insertar dos obligaciones (mes 1 y 2).• Ejecutar GetByContractYearMonthAsyn c(5, 2024, 2).• Verificar que: Existe la obligación (hit). La búsqueda para año/mes inexistente devuelve null (miss).
Datos de salida - Resultado Esperado	Primer resultado: no null. Segundo resultado: null.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetByContractYearMonthReturnsExpected()
3 {
4     await using var ctx = CreateContext();
5     var repo = new ObligationMonthRepository(ctx);
6
7     ctx.ObligationMonths.AddRange(
8         new ObligationMonth
9         {
10             Id = 1,
11             ContractId = 5,
12             Year = 2024,
13             Month = 1,
14             DueDate = DateTime.Today,
15             Status = Status.Pendiente
16         },
17         new ObligationMonth
18         {
19             Id = 2,
20             ContractId = 5,
21             Year = 2024,
22             Month = 2,
23             DueDate = DateTime.Today,
24             Status = Status.Pendiente
25         }
26     );
27
28     await ctx.SaveChangesAsync();
29
30     var hit = await repo.GetByContractYearMonthAsync(5, 2024, 2);
31     hit.Should().NotBeNull();
32
33     var miss = await repo.GetByContractYearMonthAsync(5, 2023, 12);
34     miss.Should().BeNull();
35 }
36 }
```



1.3. GetByContractQueryableOrdersDescAndFiltersDeleted

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetByContractQueryableOrdersDescAndFiltersDeleted

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Listar obligaciones de un contrato ordenadas DESC y sin eliminadas.
Objetivo	Validar el orden y el filtrado de registros eliminados (IsDeleted = true).
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	ContractId = 7 3 obligaciones (una eliminada).
Procedimiento de Prueba	<ul style="list-style-type: none">Insertar obligaciones: meses 12/2023, 1/2024 (eliminada), 2/2024.Ejecutar GetByContractQueryable(7).ToList().Validar: Solo 2 registros (filtrado eliminada). Primer elemento tiene Month = 2 (orden descendente).
Datos de salida - Resultado Esperado	Conteo = 2 Primer registro = mes 2
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1  public async Task GetByContractQueryableOrdersDescAndFiltersDeleted()
2  {
3      await using var ctx = CreateContext();
4      var repo = new ObligationMonthRepository(ctx);
5
6      ctx.ObligationMonths.AddRange(
7          new ObligationMonth
8          {
9              Id = 1,
10             ContractId = 7,
11             Year = 2023,
12             Month = 12,
13             DueDate = DateTime.Today,
14             Status = Status.Pendiente
15         },
16         new ObligationMonth
17         {
18             Id = 2,
19             ContractId = 7,
20             Year = 2024,
21             Month = 1,
22             DueDate = DateTime.Today,
23             Status = Status.Pendiente,
24             IsDeleted = true
25         },
26         new ObligationMonth
27         {
28             Id = 3,
29             ContractId = 7,
30             Year = 2024,
31             Month = 2,
32             DueDate = DateTime.Today,
33             Status = Status.Pendiente
34         }
35     );
36
37     await ctx.SaveChangesAsync();
38
39     var list = repo.GetByContractQueryable(7).ToList();
40
41     list.Should().HaveCount(2);
42     list.First().Month.Should().Be(2);
43 }
44 }
```



1.4. GetTotalObligationsPaidByDayAsyncReturnsSumForGivenDate

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetTotalObligationsPaidByDayAsyncReturnsSumForGivenDate

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Calcular total pagado en un día específico.		
Objetivo	Validar suma sobre obligaciones aprobadas con PaymentDate = today.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	<ul style="list-style-type: none">Obligaciones aprobadas: 1000 + 500Obligaciones ignoradas: pendiente y aprobada pero de otro día.		
Procedimiento de Prueba	<ul style="list-style-type: none">Insertar 4 obligaciones: 2 aprobadas hoy, 1 pendiente hoy, 1 aprobada ayer.Ejecutar GetTotalObligationsPaidByDayAsync(today).Verificar total = 1500.		
Datos de salida - Resultado Esperado	Total = 1500		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



```
1 [Fact]
2 public async Task GetTotalObligationsPaidByDayAsyncReturnsSumForGivenDate()
3 {
4     await using var ctx = CreateContext();
5     var repo = new ObligationMonthRepository(ctx);
6
7     var today = DateTime.Today;
8
9     ctx.ObligationMonths.AddRange(
10        new ObligationMonth { Id = 1, Status = Status.Aprobada, PaymentDate = today, TotalAmount = 1000 },
11        new ObligationMonth { Id = 2, Status = Status.Aprobada, PaymentDate = today, TotalAmount = 500 },
12        new ObligationMonth { Id = 3, Status = Status.Pendiente, PaymentDate = today, TotalAmount = 999 },
13        new ObligationMonth { Id = 4, Status = Status.Aprobada, PaymentDate = today.AddDays(-1), TotalAmount = 300 }
14    );
15
16    await ctx.SaveChangesAsync();
17
18    var total = await repo.GetTotalObligationsPaidByDayAsync(today);
19
20    total.Should().Be(1500);
21 }
22 }
```



1.5. GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Sumar obligaciones aprobadas de un mes y año específico.
Objetivo	Validar acumulado mensual de montos aprobados.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">Año = 2025Mes = 10Aprobadas del mes: 200 + 800
Procedimiento de Prueba	<ul style="list-style-type: none">Insertar 4 obligaciones (dos aprobadas en 10/2025, una pendiente, una aprobada fuera del mes).Ejecutar GetTotalObligationsPaidByMonthAsync(2025, 10).Verificar total = 1000.
Datos de salida - Resultado Esperado	Total = 1500
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth()
3 {
4     await using var ctx = CreateContext();
5     var repo = new ObligationMonthRepository(ctx);
6
7     ctx.ObligationMonths.AddRange(
8         new ObligationMonth
9         {
10             Id = 1,
11             Status = Status.Aprobada,
12             PaymentDate = new DateTime(2025, 10, 1),
13             TotalAmount = 200
14         },
15         new ObligationMonth
16         {
17             Id = 2,
18             Status = Status.Aprobada,
19             PaymentDate = new DateTime(2025, 10, 15),
20             TotalAmount = 800
21         },
22         new ObligationMonth
23         {
24             Id = 3,
25             Status = Status.Pendiente,
26             PaymentDate = new DateTime(2025, 10, 20),
27             TotalAmount = 500
28         },
29         new ObligationMonth
30         {
31             Id = 4,
32             Status = Status.Aprobada,
33             PaymentDate = new DateTime(2025, 9, 30),
34             TotalAmount = 999
35         }
36     );
37
38     await ctx.SaveChangesAsync();
39
40     var total = await repo.GetTotalObligationsPaidByMonthAsync(2025, 10);
41
42     total.Should().Be(1000);
43 }
44 }
```



1.6. GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Sumar obligaciones aprobadas de un mes y año específico.
Objetivo	Validar acumulado mensual de montos aprobados.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">Año = 2025Mes = 10Aprobadas del mes: 200 + 800
Procedimiento de Prueba	<ul style="list-style-type: none">Insertar 4 obligaciones (dos aprobadas en 10/2025, una pendiente, una aprobada fuera del mes).Ejecutar GetTotalObligationsPaidByMonthAsync(2025, 10).Verificar total = 1000.
Datos de salida - Resultado Esperado	Total = 1500
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetTotalObligationsPaidByMonthAsyncReturnsSumForGivenMonth()
3 {
4     await using var ctx = CreateContext();
5     var repo = new ObligationMonthRepository(ctx);
6
7     ctx.ObligationMonths.AddRange(
8         new ObligationMonth
9         {
10            Id = 1,
11            Status = Status.Aprobada,
12            PaymentDate = new DateTime(2025, 10, 1),
13            TotalAmount = 200
14        },
15        new ObligationMonth
16        {
17            Id = 2,
18            Status = Status.Aprobada,
19            PaymentDate = new DateTime(2025, 10, 15),
20            TotalAmount = 800
21        },
22        new ObligationMonth
23        {
24            Id = 3,
25            Status = Status.Pendiente,
26            PaymentDate = new DateTime(2025, 10, 20),
27            TotalAmount = 500
28        },
29        new ObligationMonth
30        {
31            Id = 4,
32            Status = Status.Aprobada,
33            PaymentDate = new DateTime(2025, 9, 30),
34            TotalAmount = 999
35        }
36    );
37
38    await ctx.SaveChangesAsync();
39
40    var total = await repo.GetTotalObligationsPaidByMonthAsync(2025, 10);
41
42    total.Should().Be(1000);
43}
44
```



2. SystemParameterRepository

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	AddGetAllUpdateDeleteLogicWorks

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar el ciclo completo CRUD lógico para SystemParameter.
Objetivo	Comprobar que Add, GetAll, Update y DeleteLogic funcionan según lo esperado.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">Parámetro inicial:Key: "UVT"Value: "49798.75"
Procedimiento de Prueba	<ul style="list-style-type: none">Crear contexto InMemory.Ejecutar AddAsync() para insertar el parámetro.Verificar con GetAllAsync() que existe exactamente un elemento.Modificar Value a "50000" y actualizar con UpdateAsync().Obtener nuevamente con GetByIdAsync() y confirmar el valor actualizado.Ejecutar DeleteLogicAsync(p.Id) y verificar retorno true.Confirmar que GetByIdAsync() devuelve null (eliminación lógica efectiva).
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">Creación exitosa.GetAllAsync() retorna 1 elemento.Actualización modifica el valor correctamente.Eliminación lógica retorna true.El registro deja de aparecer en consultas (null).



Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()
------------------------------------------	-----------------------------	-----------------	---------------



```
1 [Fact]
2 public async Task AddGetAllUpdateDeleteLogicWorks()
3 {
4     var db = Guid.NewGuid().ToString();
5     await using var ctx = CreateContext(db);
6     var repo = new DataGeneric<SystemParameter>(ctx);
7
8     var p = await repo.AddAsync(new SystemParameter
9     {
10         Key = "UVT",
11         Value = "49798.75",
12         EffectiveFrom = DateTime.UtcNow
13     });
14
15     (await repo.GetAllAsync()).Should().ContainSingle(x => x.Id == p.Id);
16
17     p.Value = "50000";
18     await repo.UpdateAsync(p);
19
20     var again = await repo.GetByIdAsync(p.Id);
21     again!.Value.Should().Be("50000");
22
23     (await repo.DeleteLogicAsync(p.Id)).Should().BeTrue();
24
25     (await repo.GetByIdAsync(p.Id)).Should().BeNull();
26 }
```



3. PersonRepository

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	AddGetDeleteWorks

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar operaciones básicas Add, Get y Delete en Person.
Objetivo	Confirmar que un registro Person se inserta, se obtiene y se elimina físicamente correctamente.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">Datos de la persona (FirstName, LastName, ...)
Procedimiento de Prueba	<ul style="list-style-type: none">Crear contexto InMemory.Ejecutar AddAsync() para insertar el Person.Confirmar con GetByIdAsync() que el registro existe.Ejecutar DeleteAsync(p.Id) y verificar retorno true.Validar que GetByIdAsync(p.Id) devuelve null (eliminación definitiva exitosa).
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">Inserción exitosa.Recuperación correcta mediante GetByIdAsync().Eliminación física retorna true.El registro deja de existir (null).
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task AddGetDeleteWorks()
3 {
4     var db = Guid.NewGuid().ToString();
5     await using var ctx = CreateContext(db);
6     var repo = new DataGeneric<Person>(ctx);
7
8     var p = await repo.AddAsync(new Person { FirstName = "A", LastName = "B" });
9
10    (await repo.GetByIdAsync(p.Id)).Should().NotBeNull();
11
12    (await repo.DeleteAsync(p.Id)).Should().BeTrue();
13
14    (await repo.GetByIdAsync(p.Id)).Should().BeNull();
15 }
```



4. EstablishmentsRepository

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetAllAsyncFiltersByActiveOnly

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que GetAllAsync filtre correctamente los establecimientos según el parámetro ActivityFilter (Any y ActiveOnly).
Objetivo	Confirmar que: Con Any se obtienen todos los establecimientos. Con ActiveOnly solo los establecimientos con Active = true.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">• Plaza:• Id = 1, Active = true• Establecimientos:• Id = 1, Active = true, Plazald = 1• Id = 2, Active = false, Plazald = 1• Filtros probados:• ActivityFilter.Any• ActivityFilter.ActiveOnly
Procedimiento de Prueba	<ul style="list-style-type: none">• Insertar plaza y 2 establecimientos (uno activo, uno inactivo).• Ejecutar GetAllAsync(ActivityFilter.Any) y contar resultados.• GetAllAsync(ActivityFilter.ActiveOnly) y contar resultados.• Verificar que el único registro activo tenga Id = 1.
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Con Any: 2 registros.• Con ActiveOnly: 1 registro con



	Id = 1 y Active = true.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()

```
1
2 [Fact]
3 public async Task GetAllAsyncFiltersByActiveOnly()
4 {
5     await using var ctx = CreateContext();
6     var repo = new EstablishmentsRepository(ctx);
7
8     var plaza = new Plaza { Id = 1, Name = "P", Description = "", Active = true, Location = "L" };
9     ctx.Plazas.Add(plaza);
10
11    ctx.Establishments.AddRange(
12        new Establishment { Id = 1, Name = "A", Description = "", Active = true, PlazaId = 1, Plaza = plaza },
13        new Establishment { Id = 2, Name = "B", Description = "", Active = false, PlazaId = 1, Plaza = plaza }
14    );
15
16    await ctx.SaveChangesAsync();
17
18    var all = await repo.GetAllAsync(Entity.Enum.ActivityFilter.Any);
19    all.Should().HaveCount(2);
20
21    var active = await repo.GetAllAsync(Entity.Enum.ActivityFilter.ActiveOnly);
22    active.Should().HaveCount(1);
23    active.First().Id.Should().Be(1);
24 }
```



4.1. GetBasicsByIdsAsyncProjectsValues

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetBasicsByIdsAsyncProjectsValues

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que GetBasicsByIdsAsync proyecte correctamente los valores básicos del establecimiento (Id, RentValueBase, UvtQty) solo para los activos.
Objetivo	Confirmar que, al consultar varios Ids, solo se retorna el establecimiento activo y con los valores proyectados correctos.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">• Plaza inicial: Id = 1• Establecimientos:• Id = 10, Active = true, RentValueBase = 100, UvtQty = 2• Id = 11, Active = false, RentValueBase = 200, UvtQty = 3
Procedimiento de Prueba	<ul style="list-style-type: none">• Crear contexto InMemory.• Insertar plaza y los 2 establecimientos.• Ejecutar GetBasicsByIdsAsync([10, 11]).• Verificar que el resultado contenga un solo elemento con:• Id = 10• RentValueBase = 100• UvtQty = 2
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Lista con un solo registro:• Id = 10, RentValueBase = 100, UvtQty = 2.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()





```
● ● ●  
1 Description = "", Active = true, Location = "L" };  
2 ctx.Plazas.Add(plaza);  
3  
4 ctx.Establishments.AddRange(  
5     new Establishment { Id = 10, Name = "A", Description = "", Active = true, PlazaId = 1, Plaza = plaza, RentValueBase = 100, UvtQty = 2 },  
6     new Establishment { Id = 11, Name = "B", Description = "", Active = false, PlazaId = 1, Plaza = plaza, RentValueBase = 200, UvtQty = 3 }  
7 );  
8  
9 await ctx.SaveChangesAsync();  
10  
11 var basics = await repo.GetBasicsByIdsAsync([10, 11]);  
12  
13 basics.Should().ContainSingle(x =>  
14     x.Id == 10 &&  
15     x.RentValueBase == 100 &&  
16     x.UvtQty == 2  
17 );  
18 }
```



4.2. GetInactiveIdsAsyncReturnsOnlyInactive

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetInactiveIdsAsyncReturnsOnlyInactive

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que GetInactiveIdsAsync devuelva únicamente los Ids de establecimientos inactivos dentro del conjunto consultado.
Objetivo	Confirmar que el repositorio identifica correctamente qué Ids corresponden a establecimientos con Active = false.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">• Plaza:• Id = 1, Active = true• Establecimientos:• Id = 20, Active = true, Plazald = 1• Id = 21, Active = false, Plazald = 1
Procedimiento de Prueba	<ul style="list-style-type: none">• Insertar plaza y establecimientos 20 y 21.• Ejecutar GetInactiveIdsAsync([20, 21]).• Verificar que el resultado contenga únicamente el Id 21.
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Lista con un solo registro:• Id = 21.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetInactiveIdsAsyncReturnsOnlyInactive()
3 {
4     await using var ctx = CreateContext();
5     var repo = new EstablishmentsRepository(ctx);
6
7     var plaza = new Plaza { Id = 1, Name = "P", Description = "", Active = true, Location = "L" };
8     ctx.Plazas.Add(plaza);
9
10    ctx.Establishments.AddRange(
11        new Establishment { Id = 20, Name = "A", Description = "", Active = true, PlazaId = 1, Plaza = plaza },
12        new Establishment { Id = 21, Name = "B", Description = "", Active = false, PlazaId = 1, Plaza = plaza }
13    );
14
15    await ctx.SaveChangesAsync();
16
17    var inactive = await repo.GetInactiveIdsAsync([20, 21]);
18
19    inactive.Should().ContainSingle(i => i == 21);
20 }
```



5. UserRepository

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	ExistsByEmailVariants

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que el repositorio detecte correctamente si un email existe y que soporte exclusión por Id al verificar duplicados.
Objetivo	Confirmar que ExistsByEmailAsync detecta emails existentes. El parámetro excludId evita falsos positivos. GetByIdByEmailAsync retorna el Id correcto.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">• Usuarios:<ul style="list-style-type: none">• (Id=1, Email="a@mail")• (Id=2, Email="b@mail")
Procedimiento de Prueba	<ul style="list-style-type: none">• Registrar Department, City, Persons y dos Users.• Ejecutar:<ul style="list-style-type: none">• ExistsByEmailAsync("a@mail") → true• ExistsByEmailAsync("a@mail", excludId:1) → false• ExistsByEmailAsync("a@mail", excludId:2) → true• GetByIdByEmailAsync("b@mail") → 2
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Email encontrado correctamente.• Exclusión por Id funciona.• Retorno de Id correcto.
Resultado Obtenido Prueba Exitosa	¿Prueba <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



	Exitosa?		
--	-----------------	--	--



```
1 [Fact]
2 public async Task ExistsByEmailVariants()
3 {
4     await using var ctx = CreateContext();
5     var repo = new UserRepository(ctx);
6
7     ctx.Set<Department>().Add(new Department { Id = 1, Name = "Dept" });
8     ctx.Set<City>().Add(new City { Id = 1, Name = "Neiva", DepartmentId = 1 });
9
10    ctx.Persons.Add(new Entity.Domain.Models.Implements.Persons.Person
11    {
12        Id = 1,
13        FirstName = "A",
14        LastName = "B",
15        CityId = 1
16    });
17
18    ctx.Persons.Add(new Entity.Domain.Models.Implements.Persons.Person
19    {
20        Id = 2,
21        FirstName = "C",
22        LastName = "D",
23        CityId = 1
24    });
25
26    ctx.Users.AddRange(NewUser(1, "a@mail"), NewUser(2, "b@mail"));
27    await ctx.SaveChangesAsync();
28
29    (await repo.ExistsByEmailAsync("a@mail")).Should().BeTrue();
30
31    (await repo.ExistsByEmailAsync("a@mail", excludeId: 1)).Should().BeFalse();
32    (await repo.ExistsByEmailAsync("a@mail", excludeId: 2)).Should().BeTrue();
33
34    (await repo.GetIdByEmailAsync("b@mail")).Should().Be(2);
35 }
```



5.1. GetByEmailVariants

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetByEmailVariants

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que el repositorio obtenga correctamente un usuario por email y que la variante para autenticación incluya el password.
Objetivo	Confirmar que GetByEmailAsync retorna usuario con relación Person. GetAuthUserByEmailAsync retorna usuario con credenciales (Password).
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">• Usuario:• Id = 1• Email = "a@mail"• Password = "hash"• Person asociada:• Id = 1
Procedimiento de Prueba	<ul style="list-style-type: none">• Registrar Department, City, Person y User.• Ejecutar:• GetByEmailAsync("a@mail") → valida PersonId• GetAuthUserByEmailAsync("a@mail") → valida Password
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Usuario encontrado.• PersonId = 1• Password = hash
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetByEmailVariants()
3 {
4     await using var ctx = CreateContext();
5     var repo = new UserRepository(ctx);
6
7     ctx.Set<Department>().Add(new Department { Id = 1, Name = "Dept" });
8     ctx.Set<City>().Add(new City { Id = 1, Name = "Neiva", DepartmentId = 1 });
9
10    ctx.Persons.Add(new Entity.Domain.Models.Implements.Persons.Person
11    {
12        Id = 1,
13        FirstName = "A",
14        LastName = "B",
15        CityId = 1
16    });
17
18    ctx.Users.Add(NewUser(1, "a@mail"));
19    await ctx.SaveChangesAsync();
20
21    var full = await repo.GetByEmailAsync("a@mail");
22    full.Should().NotBeNull();
23    full!.PersonId.Should().Be(1);
24
25    var auth = await repo.GetAuthUserByEmailAsync("a@mail");
26    auth.Should().NotBeNull();
27    auth!.Password.Should().Be("hash");
28 }
29
```



5.2. GetByPersonIdReturnsUser

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetByPersonIdReturnsUser

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que el repositorio retorne el usuario asociado a un PersonId.
Objetivo	Confirmar que GetByPersonIdAsync devuelva el usuario correcto.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">• Person Id = 1• User:<ul style="list-style-type: none">◦ Id = 1◦ Email = "a@mail"◦ PersonId = 1
Procedimiento de Prueba	<ul style="list-style-type: none">• Registrar Department, City, Person y User.• Ejecutar: GetByPersonIdAsync(1)• Validar que el usuario retornado tenga Id = 1.
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Usuario encontrado.• Id = 1
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetByPersonIdReturnsUser()
3 {
4     await using var ctx = CreateContext();
5     var repo = new UserRepository(ctx);
6
7     ctx.Set<Department>().Add(new Department { Id = 1, Name = "Dept" });
8     ctx.Set<City>().Add(new City { Id = 1, Name = "Neiva", DepartmentId = 1 });
9
10    ctx.Persons.Add(new Entity.Domain.Models.Implements.Persons.Person
11    {
12        Id = 1,
13        FirstName = "A",
14        LastName = "B",
15        CityId = 1
16    });
17
18    ctx.Users.Add(NewUser(1, "a@mail"));
19    await ctx.SaveChangesAsync();
20
21    var user = await repo.GetByPersonIdAsync(1);
22
23    user.Should().NotBeNull();
24    user!.Id.Should().Be(1);
25 }
```



5.3. GetByIdReturnsUserWithIncludes

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	.GetByIdReturnsUserWithIncludes

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que el repositorio cargue todas las relaciones del usuario: Person y City.		
Objetivo	Confirmar que GetByIdAsync incluya la navegación completa:		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	<ul style="list-style-type: none">• User:<ul style="list-style-type: none">○ Id = 1○ Email = "a@mail"• Person asociada:<ul style="list-style-type: none">○ Id = 1• City:<ul style="list-style-type: none">○ Id = 1		
Procedimiento de Prueba	<ul style="list-style-type: none">• Registrar Department, City, Person y User.• Ejecutar: GetByIdAsync(1)• Validar:• user != null• user.Person != null• user.Person.City != null		
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Usuario encontrado con todas las relaciones cargadas.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



```
1 [Fact]
2     public async Task GetByIdReturnsUserWithIncludes()
3     {
4         await using var ctx = CreateContext();
5         var repo = new UserRepository(ctx);
6
7         ctx.Set<Department>().Add(new Department { Id = 1, Name = "Dept" });
8         ctx.Set<City>().Add(new City { Id = 1, Name = "Neiva", DepartmentId = 1 });
9
10        ctx.Persons.Add(new Entity.Domain.Models.Implements.Persons.Person
11        {
12            Id = 1,
13            FirstName = "A",
14            LastName = "B",
15            CityId = 1
16        });
17
18        ctx.Users.Add(NewUser(1, "a@mail"));
19        await ctx.SaveChangesAsync();
20
21        var user = await repo.GetByIdAsync(1);
22
23        user.Should().NotBeNull();
24        user!.Person.Should().NotBeNull();
25        user.Person.City.Should().NotBeNull();
26    }
```



6. RolRepository

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	AddAndGetByIdWorksAndRespectsSoftDelete

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que la entidad Rol pueda insertarse, recuperarse y que el borrado lógico impida su posterior consulta.
Objetivo	Confirmar que AddAsync crea correctamente el Rol. GetByIdAsync lo devuelve antes del borrado lógico. DeleteLogicAsync marca el registro como eliminado. GetByIdAsync ya no lo devuelve.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">Rol: Name = "Admin", Description = "desc"
Procedimiento de Prueba	<ul style="list-style-type: none">Insertar registro.Recuperar por Id.Aplicar DeleteLogicAsync.Verificar que GetByIdAsync retorna null.
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">Id asignado.Consulta previa al borrado válida.Consulta posterior al borrado nula.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task AddAndGetByIdWorksAndRespectsSoftDelete()
3 {
4     var dbName = Guid.NewGuid().ToString();
5     await using var ctx = CreateContext(dbName);
6     var repo = new DataGeneric<Rol>(ctx);
7
8     var created = await repo.AddAsync(new Rol { Name = "Admin", Description = "desc" });
9     created.Id.Should().BeGreaterThan(0);
10
11    var fetched = await repo.GetByIdAsync(created.Id);
12    fetched.Should().NotBeNull();
13
14    await repo.DeleteLogicAsync(created.Id);
15    var afterSoftDelete = await repo.GetByIdAsync(created.Id);
16    afterSoftDelete.Should().BeNull();
17 }
```



6.1. GetAllExcludesDeletedAndOrdersByCreatedAtDescThenIdDesc

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetAllExcludesDeletedAndOrdersByCreatedAtDescThenIdDesc

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que GetAllAsync: Excluya registros eliminados lógicamente. Ordene por CreatedAt DESC y luego Id DESC.		
Objetivo	Confirmar que AddAsync crea correctamente el Rol. GetByIdAsync lo devuelve antes del borrado lógico. DeleteLogicAsync marca el registro como eliminado. GetByIdAsync ya no lo devuelve.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	<p>Garantizar que la colección retornada:</p> <ul style="list-style-type: none">• Solo incluya roles no eliminados.• Ordene correctamente: más reciente primero.		
Procedimiento de Prueba	<ul style="list-style-type: none">• Rol A: CreatedAt = hoy - 1 día• Rol B: CreatedAt = hoy• Rol C: CreatedAt = hoy• Rol A → borrado lógico		
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Insertar A, B, C.• Marcar A como eliminado.• Ejecutar GetAllAsync().• Validar cantidad = 2.• Validar orden: C → B.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()



```
1 [Fact]
2 public async Task GetAllExcludesDeletedAndOrdersByCreatedAtDescThenIdDesc()
3 {
4     var dbName = Guid.NewGuid().ToString();
5     await using var ctx = CreateContext(dbName);
6     var repo = new DataGeneric<Rol>(ctx);
7
8     var older = await repo.AddAsync(new Rol
9     {
10         Name = "A",
11         Description = "",
12         CreatedAt = DateTime.UtcNow.AddDays(-1)
13     });
14
15     var newer1 = await repo.AddAsync(new Rol
16     {
17         Name = "B",
18         Description = "",
19         CreatedAt = DateTime.UtcNow
20     });
21
22     var newer2 = await repo.AddAsync(new Rol
23     {
24         Name = "C",
25         Description = "",
26         CreatedAt = DateTime.UtcNow
27     });
28
29     await repo.DeleteLogicAsync(older.Id);
30
31     var all = (await repo.GetAllAsync()).ToList();
32     all.Should().HaveCount(2);
33
34     all.Select(x => x.Name).Should().Equal(new[]
35     {
36         newer2.Name,
37         newer1.Name
38     });
39 }
```



6.2. UpdatePreservesIdAndCreatedAt

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	UpdatePreservesIdAndCreatedAt

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Verificar que UpdateAsync: No modifique el Id. No modifique el CreatedAt original. Actualice el resto de campos.
Objetivo	Confirmar comportamiento correcto de actualización parcial sin alterar metadatos de creación.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<p>Garantizar que la colección retornada:</p> <ul style="list-style-type: none">• Solo incluya roles no eliminados.• Ordene correctamente: más reciente primero.
Procedimiento de Prueba	<ul style="list-style-type: none">• Insertar rol.• Guardar Id y CreatedAt original.• Enviar objeto actualizado.• Validar:<ul style="list-style-type: none">◦ Id igual al original.◦ CreatedAt intacto.◦ Name y Description actualizados.
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">• Id y CreatedAt preservados.• Campos actualizados correctamente.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task UpdatePreservesIdAndCreatedAt()
3 {
4     var dbName = Guid.NewGuid().ToString();
5     await using var ctx = CreateContext(dbName);
6     var repo = new DataGeneric<Rol>(ctx);
7
8     var created = await repo.AddAsync(new Rol { Name = "Old", Description = "D" });
9     var originalId = created.Id;
10    var originalCreatedAt = created.CreatedAt;
11
12    var updated = await repo.UpdateAsync(new Rol
13    {
14        Id = originalId,
15        Name = "New",
16        Description = "ND",
17        CreatedAt = DateTime.UtcNow.AddYears(-10)
18    });
19
20    updated.Id.Should().Be(originalId);
21    updated.CreatedAt.Should().Be(originalCreatedAt);
22    updated.Name.Should().Be("New");
23    updated.Description.Should().Be("ND");
24 }
```



6.3. DeleteRemovesEntity

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	DeleteRemovesEntity

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que DeleteAsync elimine de manera permanente la entidad.
Objetivo	Confirmar que: Primera eliminación retorne true. Segunda eliminación retorne false.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	<ul style="list-style-type: none">Rol: Name = "X", Description = "Y"
Procedimiento de Prueba	<ul style="list-style-type: none">Insertar rol.Ejecutar DeleteAsync(id) dos veces.Validar: true → false.
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">Eliminación efectiva la primera vez.Sin eliminación la segunda.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task DeleteRemovesEntity()
3 {
4     var dbName = Guid.NewGuid().ToString();
5     await using var ctx = CreateContext(dbName);
6     var repo = new DataGeneric<Rol>(ctx);
7
8     var created = await repo.AddAsync(new Rol { Name = "X", Description = "Y" });
9
10    (await repo.DeleteAsync(created.Id)).Should().BeTrue();
11    (await repo.DeleteAsync(created.Id)).Should().BeFalse();
12 }
```



7. Capa Bussines

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Validación de obtención de contratos para administrador

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que un usuario administrador pueda obtener todos los contratos existentes.
Objetivo	Confirmar que el servicio GetMineAsync() retorne el listado completo de contratos cuando el usuario autenticado posee rol de administrador.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Usuario actual con EsAdministrador = true.
Procedimiento de Prueba	Configurar _currentUser como administrador. Simular que _contracts.GetAllAsync() devuelve la lista esperada. Configurar el mapeo hacia ContractSelectDto. Ejecutar _service.GetMineAsync(). Validar contenido de la respuesta.
Datos de salida - Resultado Esperado	Se obtiene una lista con un contrato.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
1 [Fact]
2 public async Task GetMineAdminReturnsMappedContracts()
3 {
4     _currentUser.SetupGet(u => u.EsAdministrador).Returns(true);
5
6     var entities = new List<Contract>
7     {
8         new Contract { Id = 1, PersonId = 10 }
9     };
10
11    _contracts.Setup(r => r.GetAllAsync())
12        .ReturnsAsync(entities);
13
14    _mapper.Setup(m => m.Map<IEnumerable<ContractSelectDto>>(entities))
15        .Returns(new List<ContractSelectDto>
16        {
17            new ContractSelectDto { Id = 1, PersonId = 10 }
18        });
19
20    var result = await _service.GetMineAsync();
21
22    Assert.Single(result);
23    Assert.Equal(10, result.First().PersonId);
24 }
```



7.1. Usuario NO admin sin persona asociada

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Usuario NO admin sin persona asociada

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que se bloquee la obtención de contratos cuando el usuario no posee rol administrador ni persona asociada.		
Objetivo	Confirmar que GetMineAsync() arroje BusinessException cuando PersonId es nulo para un usuario no administrador.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	EsAdministrador = false , PersonId = null		
Procedimiento de Prueba	Configurar usuario sin permisos y sin persona asignada. Ejecutar _service.GetMineAsync(). Verificar que se lanza BusinessException.		
Datos de salida - Resultado Esperado	Excepción con mensaje indicando ausencia de persona asociada.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()



```
1 [Fact]
2 public async Task GetMineNonAdminWithoutPersonThrows()
3 {
4     _currentUser.SetupGet(u => u.EsAdministrador).Returns(false);
5     _currentUser.SetupGet(u => u.PersonId).Returns((int?)null);
6
7     var ex = await Assert.ThrowsAsync<BusinessException>(() => _service.GetMineAsync());
8
9     Assert.Contains("persona asociada", ex.Message, StringComparison.OrdinalIgnoreCase);
10 }
```



7.2. Usuario NO admin con persona asociada obtiene contratos filtrados

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Usuario NO admin con persona asociada obtiene contratos filtrados

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que un usuario estándar obtenga solo sus contratos.		
Objetivo	Comprobar que el servicio filtre por PersonId y retorne únicamente los contratos asociados.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	EsAdministrador = false, PersonId = 99		
Procedimiento de Prueba	<p>Configurar usuario estándar con PersonId = 99.</p> <p>Simular GetByPersonAsync(99).</p> <p>Configurar mapeo a DTO.</p> <p>Ejecutar GetMineAsync().</p>		
Datos de salida - Resultado Esperado	Lista con un contrato.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



```
1 [Fact]
2 public async Task GetMineNonAdminWithPersonReturnsMappedContracts()
3 {
4     _currentUser.SetupGet(u => u.EsAdministrador).Returns(false);
5     _currentUser.SetupGet(u => u.PersonId).Returns(99);
6
7     var entities = new List<Contract>
8     {
9         new Contract { Id = 2, PersonId = 99 }
10    };
11
12    _contracts.Setup(r => r.GetByPersonAsync(99))
13        .ReturnsAsync(entities);
14
15    _mapper.Setup(m => m.Map<IEnumerable<ContractSelectDto>>(entities))
16        .Returns(new List<ContractSelectDto>
17        {
18            new ContractSelectDto { Id = 2, PersonId = 99 }
19        });
20
21    var result = await _service.GetMineAsync();
22
23    Assert.Single(result);
24    Assert.Equal(99, result.First().PersonId);
25 }
```



7.3. Id inválido en GetObligationsAsync

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Id inválido en GetObligationsAsync

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que se rechacen solicitudes de obtención de obligaciones con identificadores inválidos.
Objetivo	Confirmar que un id de contrato igual a 0 dispare BusinessException.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Id = 0
Procedimiento de Prueba	Ejecutar GetObligationsAsync(0). Evaluar excepción generada.
Datos de salida - Resultado Esperado	Excepción con mensaje indicando que el id es inválido.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
1 [Fact]
2 public async Task GetObligationsInvalidIdThrows()
3 {
4     var ex = await Assert.ThrowsAsync<BusinessException>(() =>
5         _service.GetObligationsAsync(0));
6
7     Assert.Contains("inválido", ex.Message, StringComparison.OrdinalIgnoreCase);
8 }
9
```



7.4. Delegación al servicio de obligaciones

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Delegación al servicio de obligaciones

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar la invocación correcta del servicio de obligaciones.
Objetivo	Confirmar que GetObligationsAsync() invoque exactamente una vez al servicio _obligationSvc.GetByContractAsync(id).
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Id = 5
Procedimiento de Prueba	Configurar mock del servicio de obligaciones. Ejecutar GetObligationsAsync(5). Verificar la delegación correcta.
Datos de salida - Resultado Esperado	La lista devuelta tiene un elemento. _obligationSvc.GetByContractAsync(5) se invoca exactamente una vez.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
1 [Fact]
2 public async Task GetObligationsDelegatesToService()
3 {
4     _obligationSvc.Setup(s => s.GetByContractAsync(5))
5         .ReturnsAsync(new List<ObligationMonthSelectDto> { new() });
6
7     var result = await _service.GetObligationsAsync(5);
8
9     Assert.Single(result);
10    _obligationSvc.Verify(s => s.GetByContractAsync(5), Times.Once);
11 }
```



8. ObligationMonthBusiness

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Cálculo del total pagado por día

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que el sistema devuelva el total de obligaciones pagadas en una fecha específica.
Objetivo	Confirmar que el método <code>GetTotalObligationsPaidByDayAsync()</code> consulte correctamente al repositorio y retorne el valor esperado.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Fecha: 01/10/2024
Procedimiento de Prueba	<p>Configurar mock del repositorio para retornar 1500m al consultar por la fecha dada.</p> <p>Ejecutar <code>_service.GetTotalObligationsPaidByDayAsync(date)</code>.</p> <p>Validar el resultado.</p> <p>Verificar que la llamada al repositorio se ejecute una sola vez.</p>
Datos de salida - Resultado Esperado	Resultado decimal: 1500m. Repositorio invocado exactamente una vez.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetTotalObligationsPaidByDayAsyncReturnsExpected()
3 {
4     var date = new DateTime(2024, 10, 1);
5
6     _obligationRepo.Setup(r => r.GetTotalObligationsPaidByDayAsync(date))
7         .ReturnsAsync(1500m);
8
9     var result = await _service.GetTotalObligationsPaidByDayAsync(date);
10
11    Assert.Equal(1500m, result);
12    _obligationRepo.Verify(r => r.GetTotalObligationsPaidByDayAsync(date), Times.Once);
13 }
14
```



8.1. Cálculo del total pagado por mes

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Cálculo del total pagado por mes

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar el cálculo del total de obligaciones pagadas en un mes específico.
Objetivo	Confirmar que el método <code>GetTotalObligationsPaidByMonthAsync()</code> utilice correctamente el repositorio y devuelva el total esperado.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Año: 2024, Mes: 10
Procedimiento de Prueba	<p>Configurar mock del repositorio para retornar 5000m para el año y mes especificados.</p> <p>Ejecutar <code>_service.GetTotalObligationsPaidByMonthAsync(year, month)</code>.</p> <p>Validar el resultado.</p> <p>Verificar que la llamada se ejecute exactamente una vez.</p>
Datos de salida - Resultado Esperado	Resultado decimal: 5000m.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task GetTotalObligationsPaidByMonthAsyncReturnsExpected()
3 {
4     int year = 2024, month = 10;
5
6     _obligationRepo.Setup(r => r.GetTotalObligationsPaidByMonthAsync(year, month))
7         .ReturnsAsync(5000m);
8
9     var result = await _service.GetTotalObligationsPaidByMonthAsync(year, month);
10
11    Assert.Equal(5000m, result);
12    _obligationRepo.Verify(r => r.GetTotalObligationsPaidByMonthAsync(year, month), Times.Once);
13 }
```



9. SystemParameterBussines

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Validación de recuperación y eliminación de parámetros del sistema

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que el servicio rechace solicitudes de obtención de parámetros con identificadores inválidos.
Objetivo	Confirmar que GetByIdAsync(0) genere una BusinessException indicando que el identificador es incorrecto.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Id = 0
Procedimiento de Prueba	Ejecutar <code>_service.GetByIdAsync(0)</code> . Evaluar si se lanza la excepción esperada.
Datos de salida - Resultado Esperado	Se lanza BusinessException.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
1 [Fact]
2 public async Task DeleteWrapsDbUpdateException()
3 {
4     _repo.Setup(r => r.GetByIdAsync(1))
5         .ReturnsAsync(new SystemParameter
6     {
7         Id = 1,
8         Key = "K",
9         Value = "V",
10        Active = false
11    });
12
13    _repo.Setup(r => r.DeleteAsync(1))
14        .ThrowsAsync(new Microsoft.EntityFrameworkCore.DbUpdateException("fk"));
15
16    var ex = await Assert.ThrowsAsync<BusinessException>(() => _service.DeleteAsync(1));
17
18    Assert.Contains("restricciones de datos", ex.Message);
19 }
```



10. EstablishmentsBusiness

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Validación de creación, actualización y cálculo de valores UVT en establecimientos

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que no se permita crear un establecimiento cuando los valores mínimos requeridos son inválidos.
Objetivo	Confirmar que CreateAsync() genere BusinessException cuando AreaM2, UvtQty o Plazald sean cero.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	DTO inválido: AreaM2 = 0, UvtQty = 0, Plazald = 0
Procedimiento de Prueba	Crear DTO con valores cero. Ejecutar _service.CreateAsync(dto). Validar que se lance BusinessException.
Datos de salida - Resultado Esperado	Se lanza BusinessException.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
1 [Fact]
2 public async Task CreateThrowsWhenInvalidValues()
3 {
4     var dto = new EstablishmentCreateDto
5     {
6         AreaM2 = 0,
7         UvtQty = 0,
8         PlazaId = 0
9     };
10
11    await Assert.ThrowsAsync<BusinessException>(() => _service.CreateAsync(dto));
12 }
```



DOCUMENTOS DE REFERENCIA

Versión:	Título:
1.0	CreateAsync correcto: cálculo de UVT, persistencia y retorno

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que la creación de un establecimiento realice correctamente el cálculo del valor base de renta (UVT), persista la entidad y retorne el DTO completo.
Objetivo	Confirmar que: Se obtenga el UVT desde parámetros del sistema. Se calcule RentValueBase. Se persista correctamente. Se devuelva la versión final usando GetByIdAnyAsync.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	DTO válido: AreaM2 = 1, UvtQty = 2, Plazald = 1
Procedimiento de Prueba	Configurar el contexto con parámetro UVT = 10. Simular que AddAsync asigna Id = 7. Simular que GetByIdAnyAsync retorna el establecimiento final. Ejecutar service.CreateAsync(dto).
Datos de salida - Resultado Esperado	Id = 7. RentValueBase = UvtQty * UVT → 2 * 10 = 20.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task CreateSucceedsReturnsSelect()
3 {
4     var dto = new EstablishmentCreateDto
5     {
6         Name = "N",
7         Description = "D",
8         AreaM2 = 1,
9         UvtQty = 2,
10        PlazaId = 1
11    };
12
13    _repo.Setup(r => r.AddAsync(It.IsAny<Establishment>()))
14        .ReturnsAsync((Establishment e) =>
15    {
16        e.Id = 7;
17        return e;
18    });
19
20    _repo.Setup(r => r.GetByIdAnyAsync(7))
21        .ReturnsAsync(new EstablishmentSelectDto
22    {
23        Id = 7,
24        Name = "N",
25        Description = "D",
26        AreaM2 = 1,
27        UvtQty = 2,
28        RentValueBase = 20,
29        PlazaId = 1
30    });
31
32    var result = await _service.CreateAsync(dto);
33
34    Assert.Equal(7, result.Id);
35    Assert.Equal(20, result.RentValueBase);
36 }
```



10.1. UpdateAsync retorna null cuando no existe

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	UpdateAsync retorna null cuando no existe

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que la actualización retorne null cuando el establecimiento buscado no existe.		
Objetivo	Confirmar que UpdateAsync() detecte entidades inexistentes.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	DTO de actualización: Id = 9, Establecimiento inexistente (mock retorna null).		
Procedimiento de Prueba	Configurar repositorio para retornar null. Ejecutar _service.UpdateAsync(dto). Validar el resultado.		
Datos de salida - Resultado Esperado	Resultado = null.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()



```
1 [Fact]
2 public async Task UpdateReturnsNullWhenNotFound()
3 {
4     var dto = new EstablishmentUpdateDto
5     {
6         Id = 9,
7         Name = "X",
8         Description = "D",
9         AreaM2 = 1,
10        RentValueBase = 10,
11        UvtQty = 1,
12        PlazaId = 1
13    };
14
15    _repo.Setup(r => r.GetByIdAsync(9))
16        .ReturnsAsync((Establishment?)null);
17
18    var result = await _service.UpdateAsync(dto);
19
20    Assert.Null(result);
21 }
```



10.2. UpdateAsync correcto: recalcula valor UVT

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	UpdateAsync correcto: recalcula valor UVT

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que la actualización: Obtenga UVT actual. Recalcule el valor base de renta. Retorne el DTO final.
Objetivo	Confirmar que UpdateAsync() recalcule RentValueBase usando el parámetro UVT vigente y que los datos persistan correctamente.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	DTO: Id = 5, AreaM2 = 10, UvtQty = 3
Procedimiento de Prueba	Configurar repositorio para retornar un establecimiento existente. Configurar repositorio para aceptar actualización. Configurar retorno final por GetByIdAnyAsync(5). Ejecutar service.UpdateAsync(dto).
Datos de salida - Resultado Esperado	RentValueBase calculado: $3 * 10 = 30$. DTO final contiene: Id = 5 RentValueBase = 30
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task UpdateSucceedsRecalculatesRentValue()
3 {
4     var dto = new EstablishmentUpdateDto
5     {
6         Id = 5,
7         Name = "Updated",
8         Description = "Desc",
9         AreaM2 = 10,
10        UvtQty = 3,
11        PlazaId = 2
12    };
13
14    var existing = new Establishment
15    {
16        Id = 5,
17        Name = "Old",
18        Description = "OldDesc",
19        AreaM2 = 5,
20        UvtQty = 1,
21        PlazaId = 2,
22        RentValueBase = 10
23    };
24
25    _repo.Setup(r => r.GetByIdAsync(5))
26        .ReturnsAsync(existing);
27
28    _repo.Setup(r => r.UpdateAsync(It.IsAny<Establishment>()))
29        .ReturnsAsync((Establishment e) => e);
30
31    _repo.Setup(r => r.GetByIdAnyAsync(5))
32        .ReturnsAsync(new EstablishmentSelectDto
33        {
34            Id = 5,
35            Name = "Updated",
36            Description = "Desc",
37            AreaM2 = 10,
38            UvtQty = 3,
39            PlazaId = 2,
40            RentValueBase = 30
41        });
42
43    var result = await _service.UpdateAsync(dto);
44
45    Assert.NotNull(result);
46    Assert.Equal(5, result!.Id);
47    Assert.Equal(30, result.RentValueBase);
48 }
```



11. Capa controller

Contract

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	Validación de endpoints del controlador de contratos (ContractController)

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que el controlador retorne correctamente el listado de contratos propios del usuario autenticado.		
Objetivo	Confirmar que GetMine() devuelva OkObjectResult cuando el servicio retorna una colección válida.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	Servicio retorna: List<ContractSelectDto> vacío.		
Procedimiento de Prueba	<p>Configurar GetMineAsync() para devolver una lista vacía.</p> <p>Invocar GetMine().</p> <p>Validar que el resultado sea OkObjectResult.</p>		
Datos de salida - Resultado Esperado	Código 200 OK.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()



```
1 [Fact]
2 public async Task PostReturnsOkWithResult()
3 {
4     var dto = new ContractCreateDto
5     {
6         FirstName = "A",
7         LastName = "B",
8         Document = "1",
9         Phone = "P",
10        Address = "Addr",
11        CityId = 1,
12        EstablishmentIds = new() { 1 }
13    };
14
15    var expected = new ContractSelectDto { PersonId = 7 };
16
17    _service.Setup(s => s.CreateAsync(dto)).ReturnsAsync(expected);
18
19    var res = await Create().Post(dto);
20
21    var ok = Assert.IsType<OkObjectResult>(res.Result);
22
23    Assert.Equal(expected, ok.Value);
24}
25
```



11.1. DownloadContractPdf retorna 404 cuando el contrato no existe

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	DownloadContractPdf retorna 404 cuando el contrato no existe

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que no se permita descargar el PDF cuando el contrato no existe.
Objetivo	Confirmar que DownloadContractPdf(id) retorne NotFoundObjectResult ante un contrato inexistente.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	GetByIdAsync(77) → null.
Procedimiento de Prueba	Configurar servicio para devolver null. Invocar DownloadContractPdf(77). Validar el tipo de respuesta.
Datos de salida - Resultado Esperado	Código 404 Not Found.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



● ● ●

```
1 [Fact]
2 public async Task ChangeActiveStatusNoContent()
3 {
4     _service.Setup(s => s.UpdateActiveStatusAsync(5, true))
5         .Returns(Task.CompletedTask);
6
7     _service.Setup(s => s.GetByIdAsync(5))
8         .ReturnsAsync(new ContractSelectDto { PersonId = 7 });
9
10    _notify.Setup(n => n.NotifyContractStatusChanged(5, true, 7))
11        .Returns(Task.CompletedTask);
12
13    var res = await Create().ChangeActiveStatus(5, new WebGESCOMPH.Contracts.Requests.ChangeActiveStatusRequest { Active = true });
14    Assert.IsType<NoContentResult>(res);
15 }
```



11.2. POST Create retorna 200 OK (no 201)

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	POST Create retorna 200 OK (no 201)

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que la creación de contratos retorne 200 con el objeto creado.
Objetivo	Confirmar que Post() retorne OkObjectResult con el contrato creado.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	DTO con datos válidos.
Procedimiento de Prueba	Configurar servicio CreateAsync(dto) para retornar el contrato creado. Invocar Post(dto). Validar respuesta y objeto retornado.
Datos de salida - Resultado Esperado	Código 200 OK.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? <input checked="" type="checkbox"/> Si (x) <input type="checkbox"/> No ()



```
1 [Fact]
2 public async Task DeleteNoContent()
3 {
4     _service.Setup(s => s.GetByIdAsync(3))
5         .ReturnsAsync(new ContractSelectDto { PersonId = 9 });
6
7     _service.Setup(s => s.DeleteAsync(3)).ReturnsAsync(true);
8
9     _notify.Setup(n => n.NotifyContractDeleted(3, 9))
10        .Returns(Task.CompletedTask);
11
12    var res = await Create().Delete(3);
13    Assert.IsType<NoContentResult>(res);
14 }
```



11.3. ChangeActiveStatus retorna 204 No Content

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	ChangeActiveStatus retorna 204 No Content

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar el cambio de estado de un contrato.
Objetivo	Confirmar que ChangeActiveStatus retorne NoContentResult y realice la notificación respectiva.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	Contrato existente con PersonId = 7.
Procedimiento de Prueba	<p>Configurar UpdateActiveStatusAsync(5, true) para completar exitosamente.</p> <p>Simular GetByIdAsync(5) devolviendo contrato válido.</p> <p>Simular notificación exitosa.</p> <p>Invocar método.</p>
Datos de salida - Resultado Esperado	Código 204 No Content.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
 1 [Fact]
 2 public async Task ChangeActiveStatusNoContent()
 3 {
 4     _service.Setup(s => s.UpdateActiveStatusAsync(5, true))
 5         .Returns(Task.CompletedTask);
 6
 7     _service.Setup(s => s.GetByIdAsync(5))
 8         .ReturnsAsync(new ContractSelectDto { PersonId = 7 });
 9
10    _notify.Setup(n => n.NotifyContractStatusChanged(5, true, 7))
11        .Returns(Task.CompletedTask);
12
13    var res = await Create().ChangeActiveStatus(5, new WebGESCOMPH.Contracts.Requests.ChangeActiveStatusRequest { Active = true });
14    Assert.IsType<NoContentResult>(res);
15 }
```

11.4. Delete retorna 204 No Content



DOCUMENTOS DE REFERENCIA

Versión:	Título:
1.0	Delete retorna 204 No Content

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que la consulta de obligaciones no proceda si el contrato no existe.		
Objetivo	Confirmar que GetObligations() retorne NotFoundResult cuando el contrato no se encuentre.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	GetByIdAsync(9) → null.		
Procedimiento de Prueba	Configurar servicio para retornar null. Invocar método. Verificar respuesta.		
Datos de salida - Resultado Esperado	Código 204 No Content.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



```
1 [Fact]
2 public async Task DeleteNoContent()
3 {
4     _service.Setup(s => s.GetByIdAsync(3))
5         .ReturnsAsync(new ContractSelectDto { PersonId = 9 });
6
7     _service.Setup(s => s.DeleteAsync(3)).ReturnsAsync(true);
8
9     _notify.Setup(n => n.NotifyContractDeleted(3, 9))
10        .Returns(Task.CompletedTask);
11
12    var res = await Create().Delete(3);
13    Assert.IsType<NoContentResult>(res);
14 }
15
```



11.5. GetObligations retorna 404 cuando el contrato no existe

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetObligations retorna 404 cuando el contrato no existe

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms
Requerimientos Funcional de la prueba	Validar que la consulta de obligaciones no proceda si el contrato no existe.
Objetivo	Confirmar que GetObligations() retorne NotFoundResult cuando el contrato no se encuentre.
Tipo de Prueba	Unitaria
Datos de entrada de la prueba	GetByIdAsync(9) → null.
Procedimiento de Prueba	Configurar servicio para retornar null. Invocar método. Verificar respuesta.
Datos de salida - Resultado Esperado	Código 404 Not Found.
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa? Si (x) No ()



```
1  -
2  [Fact]
3  public async Task GetObligationsOkWhenExists()
4  {
5      _service.Setup(s => s.GetByIdAsync(1)).ReturnsAsync(new ContractSelectDto());
6
7      _service.Setup(s => s.GetObligationsAsync(1))
8          .ReturnsAsync(new List<Entity.DTOs.Implements.Business.ObligationMonth.ObligationMonthSelectDto> { new() });
9
10     var res = await Create().GetObligations(1);
11     Assert.IsType<OkObjectResult>(res);
12 }
13
```



11.6. GetObligations retorna 200 OK cuando el contrato existe

DOCUMENTOS DE REFERENCIA	
Versión:	Título:
1.0	GetObligations retorna 200 OK cuando el contrato existe

Detalle de la prueba

Fecha de realización: 08/09/2025	Duración de la prueba: 3 ms		
Requerimientos Funcional de la prueba	Validar que se devuelvan las obligaciones asociadas cuando el contrato existe.		
Objetivo	Confirmar que GetObligations() retorne NotFoundResult cuando el contrato no se encuentre.		
Tipo de Prueba	Unitaria		
Datos de entrada de la prueba	Contrato existente. Servicio retorna una lista con obligaciones.		
Procedimiento de Prueba	Configurar GetByIdAsync(1) para retornar contrato válido. Configurar GetObligationsAsync(1) para retornar lista con datos. Ejecutar método.		
Datos de salida - Resultado Esperado	Lista de obligaciones.		
Resultado Obtenido Prueba Exitosa	¿Prueba Exitosa?	Si (x)	No ()



```
1 [Fact]
2 public async Task GetObligationsOkWhenExists()
3 {
4     _service.Setup(s => s.GetByIdAsync(1)).ReturnsAsync(new ContractSelectDto());
5
6     _service.Setup(s => s.GetObligationsAsync(1))
7         .ReturnsAsync(new List<Entity.DTOs.Implements.Business.ObligationMonth.ObligationMonthSelectDto> { new() });
8
9     var res = await Create().GetObligations(1);
10    Assert.IsType<OkObjectResult>(res);
11 }
12 }
```