UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

Laboratorio Nº 01 – Revisión de Código

Curso: *Construcción de Software II*

Docente: *Mag. Ricardo Eduardo Valcárcel Alvarado*

Integrantes:

- *Arenas Paz Soldan, Miguel Jesus* *(2017059282)*
- *Céspedes Medina, Christian Alexander* *(2010036257).*

Tacna – Perú

2025

Índice

Introducción

Las especificaciones funcionales son documentos que detallan minuciosamente un requisito de desarrollo, incluyendo una descripción exhaustiva de la lógica prevista, los datos y los criterios de prueba, con el objetivo de que el desarrollador tenga la información necesaria para elaborar una especificación técnica, desarrollar el desarrollo y llevar a cabo las pruebas técnicas requeridas para garantizar la calidad del desarrollo proporcionado.

A pesar de que es necesario un análisis adecuado de las necesidades empresariales para asegurar la calidad de la especificación funcional, se requieren otras consideraciones de diseño en función del desarrollo necesario. En realidad, un mismo requisito puede incluir el desarrollo de varios componentes, cada uno de una categoría distinta, por lo que la especificación debe especificar los componentes para cada uno y explicar cómo pueden interactuar; desde ese punto de vista, la guía de laboratorio se enfocará en el perfeccionamiento de las especificaciones funcionales aplicadas en los proyectos escogidos en el curso de Construcción de Software I para materializarlo en la entrega final del producto con una calidad aceptable.

Guía de Laboratorio N.º 01 – REVISIÓN DE CÓDIGO

1. Información sobre el evento práctico

1.1. Objetivos

- Identificar el punto de partida del proyecto existente
- Realizar la revisión de código contemplando buenas prácticas y estándares de construcción de software.

1.2. Equipos, materiales, programas y recursos

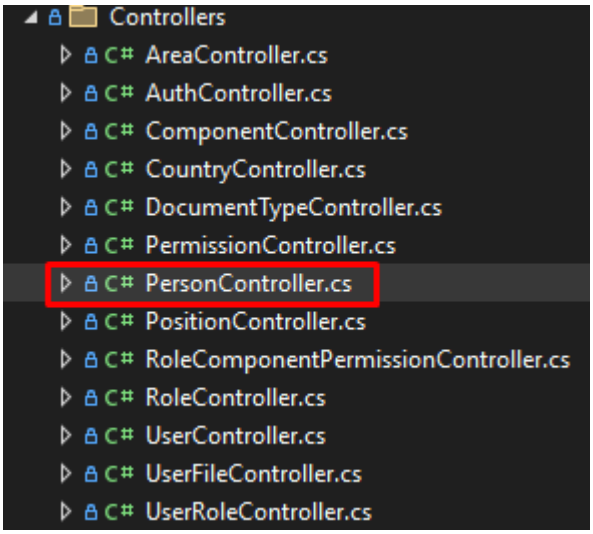
- Software Ofimático
- Software requerido para su aplicación
- Aplicación diseñada e implementada en el Curso de Construcción de Software I

1.3. Seguridad

- Protocolo de Seguridad para laboratorios y talleres
- Protocolo ante la presencialidad en Laboratorios

2. Desarrollo de la Práctica

- El requerimiento RF-01 Gestionar Persona abarca los procesos de registrar, listar, actualizar y eliminar personas de la base de datos.
- Procedimiento de cómo se conecta el Frontend con el Backend en PersonController.cs

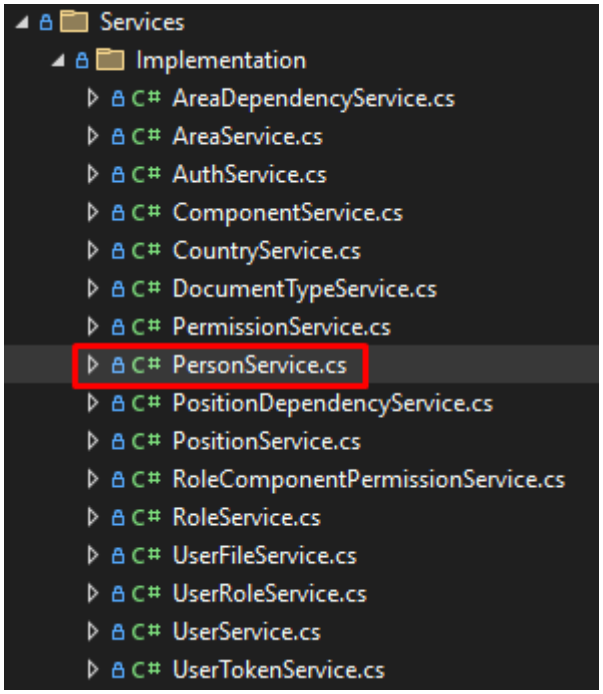



```
using SGTD_WebApi.Services;

namespace SGTD_WebApi.Controllers;

[Route("[controller]")]
[ApiController]
1 referencia | Christian Cespedes, Hace 148 días | 1 autor, 5 cambios
public class PersonController : Controller
{
    private readonly IPersonService _personService;

    0 referencias | Christian Cespedes, Hace 176 días | 1 autor, 1 cambio
    public PersonController(IPersonService personService)
    {
        _personService = personService;
    }

    [Route("")]
    [HttpPost]
    0 referencias | Christian Cespedes, Hace 148 días | 1 autor, 4 cambios
    public async Task<ActionResult> CreateAsync(PersonRequestParams requestParams)
    {
        try
        {
            await _personService.CreateAsync(requestParams);
            return Ok();
        }
        catch (ValidationException ex)
        {
            return BadRequest(ex.Message);
        }
        catch (Exception ex)
        {
            return StatusCode(500, ex.Message);
        }
```

```
35          }
36      }
37
38      [Route("")]
39      [HttpPut]
        0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
40      public async Task<ActionResult> UpdateAsync(PersonRequestParams requestParams)
41      {
42          try
43          {
44              await _personService.UpdateAsync(requestParams);
45              return Ok();
46          }
47          catch (ArgumentException ex)
48          {
49              return BadRequest(ex.Message);
50          }
51          catch (Exception ex)
52          {
53              return BadRequest(ex.Message);
54          }
55      }
56
57      [Route("")]
58      [HttpGet]
        0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
59      public async Task<ActionResult> GetAllAsync()
60      {
61          try
62          {
63              var response = await _personService.GetAllAsync();
64              return Ok(response);
65          }
66          catch (Exception ex)
```

- Descripción: PersonController.cs funciona como el intermediario crucial entre el frontend y backend de tu aplicación web, implementando un controlador API RESTful en C# que gestiona operaciones CRUD para entidades "Person". El código utiliza inyección de dependencias para recibir un IPersonService, define endpoints para crear, actualizar y obtener registros de personas, y emplea un manejo estructurado de excepciones que captura errores de validación, argumentos inválidos y excepciones generales, transformándolos en respuestas HTTP apropiadas. Esta estructura permite que las solicitudes del frontend se procesen de manera ordenada, se validen adecuadamente y se comuniquen con la lógica de negocio subyacente, todo mientras mantiene una separación clara de responsabilidades siguiendo buenas prácticas de desarrollo.

- En la parte de implementación de los servicios estaría la lógica de guardar, modificar y eliminar



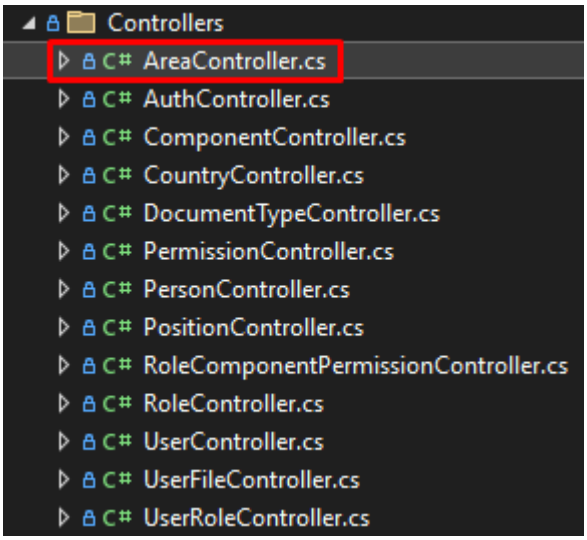

```
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;
using SGTD_WebApi.DbModel.Context;
using SGTD_WebApi.DbModel.Entities;
using SGTD_WebApi.Models.Person;

namespace SGTD_WebApi.Services.Implementation;

2 referencias | Christian Cespedes, Hace 148 días | 1 autor, 3 cambios
public class PersonService : IPersonService
{
    private readonly DatabaseContext _context;

    0 referencias | Christian Cespedes, Hace 176 días | 1 autor, 1 cambio
    public PersonService(DatabaseContext context)
    {
        _context = context;
    }

    2 referencias | Christian Cespedes, Hace 148 días | 1 autor, 2 cambios
    public async Task CreateAsync(PersonRequestParams requestParams)
    {
        var personExists = await _context.People
            .Where(q => q.Phone.Equals(requestParams.Phone) || q.DocumentNumber.Equals(requestParams.DocumentNumber))
            .AnyAsync();

        if (personExists)
        {
            throw new ValidationException("Ya existe una persona con el mismo número de teléfono o DNI");
        }

        var person = new Person
        {
            FirstName = requestParams.FirstName,
```

En primer lugar, tendríamos los requerimientos funcionales, siendo un total de 16. Luego de su corrección se estarán trabajando los requerimientos 1, 2, 3, 4, 5 y 6 recordando que los requerimientos fueron tratados en el Laboratorio 01: Revisión de Código.

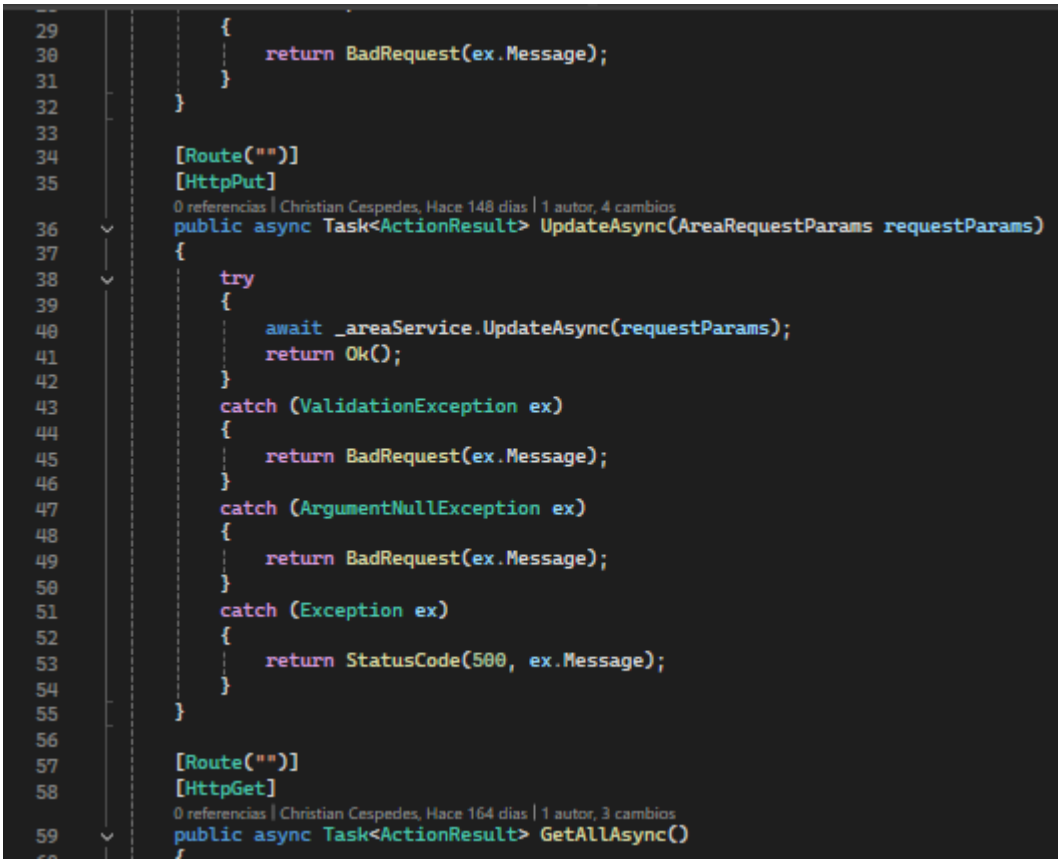
```
        {
            FirstName = requestParams.FirstName,
            LastName = requestParams.LastName,
            Phone = requestParams.Phone,
            NationalityCode = requestParams.NationalityCode,
            DocumentNumber = requestParams.DocumentNumber,
            Gender = requestParams.Gender,
            CreatedAt = DateTime.Now
        };
        _context.People.Add(person);
        await _context.SaveChangesAsync();
    }

    2 referencias | Christian Cespedes, Hace 176 dias | 1 autor, 2 cambios
    public async Task UpdateAsync(PersonRequestParams requestParams)
    {
        if (requestParams.Id == 0)
            throw new ArgumentException("Person Id is required for update.");

        var person = await _context.People.FirstOrDefaultAsync(p => p.Id == requestParams.Id);
        if (person == null)
            throw new KeyNotFoundException("Person not found.");

        person.FirstName = requestParams.FirstName;
        person.LastName = requestParams.LastName;
        person.Phone = requestParams.Phone;
        person.NationalityCode = requestParams.NationalityCode;
        person.DocumentNumber = requestParams.DocumentNumber;
        person.Gender = requestParams.Gender;
        person.UpdatedAt = DateTime.Now;

        await _context.SaveChangesAsync();
    }
```

- Descripción: PersonService.cs implementa la lógica que interactúa con la base de datos para gestionar datos de personas. Utiliza Entity Framework Core para verificar duplicados antes de crear registros, generar nuevas entidades Person con la información recibida, actualizar registros existentes validando su existencia, y persistir todos estos cambios en la base de datos, sirviendo como capa intermedia entre el controlador API y el almacenamiento de datos.

- El requerimiento RF-02 Gestionar Área abarca los procesos de registrar, listar, actualizar y eliminar personas de la base de datos.
- Procedimiento de cómo se conecta el Frontend con el Backend en AreaController.cs



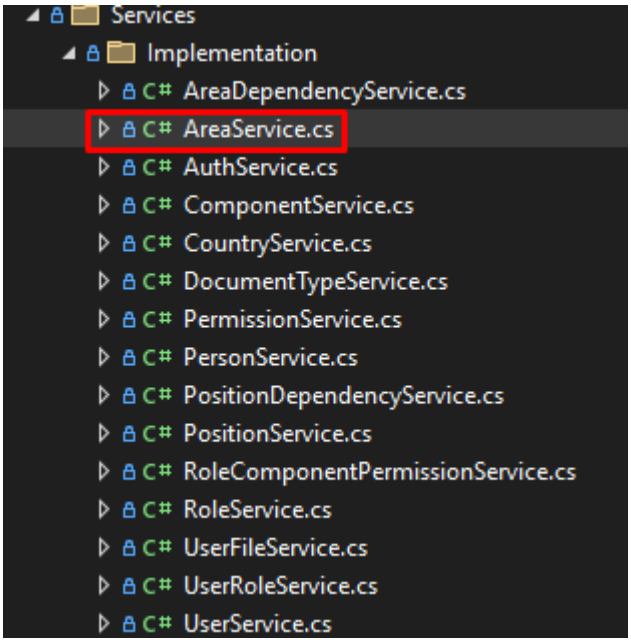

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc;
using SGTD_WebApi.Models.Area;
using SGTD_WebApi.Services;

namespace SGTD_WebApi.Controllers;

[Route("[controller]")]
[ApiController]
1 referencia | Christian Cespedes, Hace 148 días | 1 autor, 4 cambios
public class AreaController : Controller
{
    private readonly IAreaService _areaService;

    0 referencias | Christian Cespedes, Hace 176 días | 1 autor, 1 cambio
    public AreaController(IAreaService areaService)
    {
        _areaService = areaService;
    }

    [Route("")]
    [HttpPost]
    0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
    public async Task<ActionResult> CreateAsync(AreaRequestParams requestParams)
    {
        try
        {
            await _areaService.CreateAsync(requestParams);
            return Ok();
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
```



```
29          {
30              return BadRequest(ex.Message);
31          }
32      }
33
34      [Route("")]
35      [HttpPut]
        0 referencias | Christian Cespedes, Hace 148 dias | 1 autor, 4 cambios
36      public async Task<ActionResult> UpdateAsync(AreaRequestParams requestParams)
37      {
38          try
39          {
40              await _areaService.UpdateAsync(requestParams);
41              return Ok();
42          }
43          catch (ValidationException ex)
44          {
45              return BadRequest(ex.Message);
46          }
47          catch (ArgumentNullException ex)
48          {
49              return BadRequest(ex.Message);
50          }
51          catch (Exception ex)
52          {
53              return StatusCode(500, ex.Message);
54          }
55      }
56
57      [Route("")]
58      [HttpGet]
        0 referencias | Christian Cespedes, Hace 164 dias | 1 autor, 3 cambios
59      public async Task<ActionResult> GetAllAsync()
```

- Descripción: AreaController.cs actúa como punto de entrada API para gestionar áreas en la aplicación, recibiendo solicitudes HTTP del frontend y delegando el procesamiento al servicio IAreaService. Implementa endpoints para crear nuevas áreas (POST), actualizar áreas existentes (PUT) y obtener todas las áreas (GET), manejando diferentes tipos de excepciones y devolviendo respuestas HTTP adecuadas según el resultado de cada operación, lo que permite al frontend interactuar de manera estructurada con los datos de áreas almacenados en el backend.

- En la parte de implementación de los servicios AreaService.cs estaría la lógica de guardar, modificar y eliminar

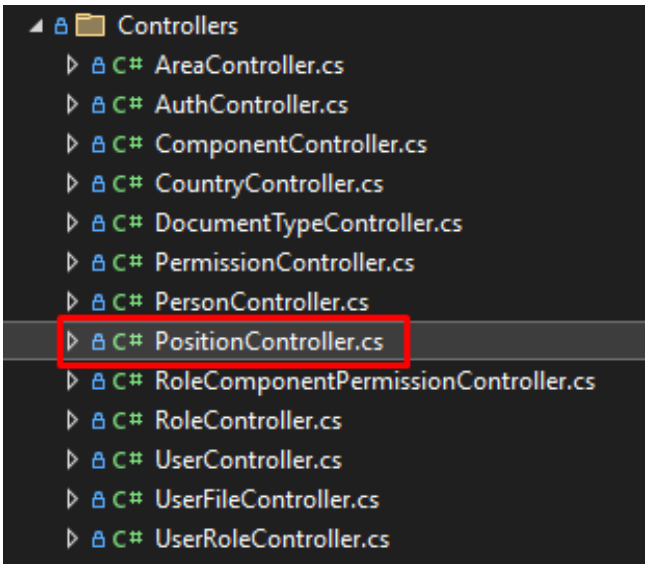



```
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;
using SGTD_WebApi.DbModel.Context;
using SGTD_WebApi.DbModel.Entities;
using SGTD_WebApi.Models.Area;
using SGTD_WebApi.Models.AreaDependency;

namespace SGTD_WebApi.Services.Implementation;

6 referencias | Christian Cespedes, Hace 148 días | 1 autor, 3 cambios
public class AreaService : IAreaService
{
    private readonly DatabaseContext _context;
    private readonly IAreaDependencyService _areaDependencyService;

    2 referencias | Christian Cespedes, Hace 168 días | 1 autor, 1 cambio
    public AreaService(DatabaseContext context, IAreaDependencyService areaDependencyService)
    {
        _context = context;
        _areaDependencyService = areaDependencyService;
    }

    15 referencias | Christian Cespedes, Hace 168 días | 1 autor, 1 cambio
    public async Task CreateAsync(AreaRequestParams requestParams)
    {
        if (await IsAreaNameUniqueAsync(requestParams.Name))
        {
            var area = new Area
            {
                Name = requestParams.Name,
                Description = requestParams.Description,
                Status = requestParams.Status
            };
```

```
            _context.Areas.Add(area);
            await _context.SaveChangesAsync();

            if (requestParams.ParentAreaId.HasValue)
            {
                var dependencyRequest = new AreaDependencyRequestParams
                {
                    ParentAreaId = requestParams.ParentAreaId.Value,
                    ChildAreaId = area.Id
                };

                await _areaDependencyService.CreateAsync(dependencyRequest);
            }
        }
        else
        {
            throw new InvalidOperationException("Area name already exists.");
        }
    }

    10 referencias | Christian Cespedes, Hace 148 días | 1 autor, 3 cambios
    public async Task UpdateAsync(AreaRequestParams requestParams)
    {
        if (requestParams.Id == null)
            throw new ValidationException("Area Id is required for update.");

        var area = await _context.Areas.FirstOrDefaultAsync(q => q.Id == requestParams.Id);
        if (area == null)
            throw new KeyNotFoundException("Area not found.");

        if (!await IsAreaNameUniqueAsync(requestParams.Name, requestParams.Id))
            throw new InvalidOperationException("Area name already exists.");

        if (requestParams.ParentAreaId.HasValue)
```

- Descripción: AreaService.cs implementa la lógica de negocio para gestionar áreas en la aplicación, utilizando Entity Framework Core para interactuar con la base de datos. La clase recibe por inyección un DatabaseContext y un IAreaDependencyService para manejar las relaciones jerárquicas entre áreas. En el método CreateAsync verifica la unicidad del nombre antes de crear una nueva área y, si se especifica un área padre, establece esta relación mediante el servicio de dependencias. El método UpdateAsync valida que se proporcione un ID, verifica la existencia del área, comprueba que el nuevo nombre sea único y actualiza los datos, manteniendo también las relaciones jerárquicas cuando corresponde. Este servicio garantiza la integridad de los datos al implementar validaciones antes de realizar operaciones en la base de datos.

- El requerimiento RF-03 Gestionar Cargo abarca los procesos de registrar, listar, actualizar y eliminar personas de la base de datos.
- Procedimiento de cómo se conecta el Frontend con el Backend en PositionController.cs

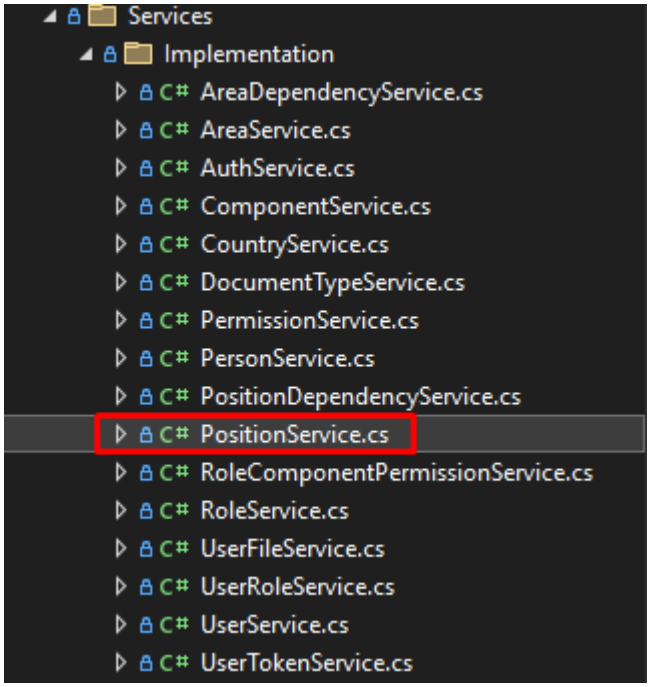



```
using Microsoft.AspNetCore.Mvc;
using SGTD_WebApi.Models.Position;
using SGTD_WebApi.Services;

namespace SGTD_WebApi.Controllers;

[Route("[controller]")]
[ApiController]
1 referencia | Christian Cespedes, Hace 164 días | 1 autor, 5 cambios
public class PositionController : Controller
{
    private readonly IPositionService _positionService;

    0 referencias | Christian Cespedes, Hace 176 días | 1 autor, 1 cambio
    public PositionController(IPositionService positionService)
    {
        _positionService = positionService;
    }

    [Route("")]
    [HttpPost]
    0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
    public async Task<ActionResult> CreateAsync(PositionRequestParams requestParams)
    {
        try
        {
            await _positionService.CreateAsync(requestParams);
            return Ok();
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
```

```
32
33      [Route("")]
34      [HttpPut]
        0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
35      public async Task<ActionResult> UpdateAsync(PositionRequestParams requestParams)
36      {
37          try
38          {
39              await _positionService.UpdateAsync(requestParams);
40              return Ok();
41          }
42          catch (Exception ex)
43          {
44              return BadRequest(ex.Message);
45          }
46      }
47
48      [Route("")]
49      [HttpGet]
        0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
50      public async Task<ActionResult> GetAllAsync()
51      {
52          try
53          {
54              var response = await _positionService.GetAllAsync();
55              return Ok(response);
56          }
57          catch (Exception ex)
58          {
59              return BadRequest(ex.Message);
60          }
61      }
```

- Descripción: PositionController.cs funciona como intermediario entre el frontend y backend para gestionar cargos o posiciones en la aplicación. Este controlador API implementa tres endpoints principales: un método POST para crear nuevos cargos, un método PUT para actualizar cargos existentes, y un método GET para recuperar todos los cargos disponibles. Utiliza inyección de dependencias para recibir un IPositionService que maneja la lógica de negocio, y cada método captura excepciones devolviendo respuestas HTTP apropiadas con mensajes de error cuando es necesario. Este diseño permite que la interfaz de usuario interactúe de forma estructurada con la gestión de cargos mientras mantiene la separación de responsabilidades en la arquitectura de la aplicación.

- En la parte de implementación de los servicios PositionService.cs estaría la lógica de guardar, modificar y eliminar

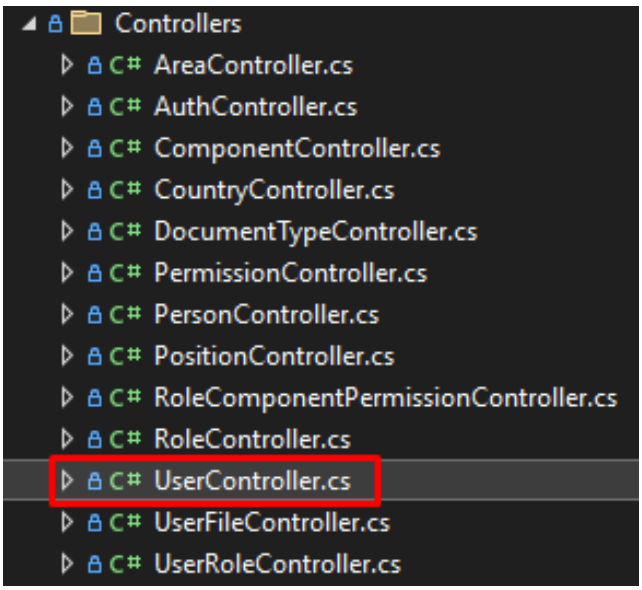



```
using Microsoft.EntityFrameworkCore;
using SGTD_WebApi.DbModel.Context;
using SGTD_WebApi.DbModel.Entities;
using SGTD_WebApi.Models.Position;
using SGTD_WebApi.Models.PositionDependency;

namespace SGTD_WebApi.Services.Implementation;

2 referencias | Christian Cespedes, Hace 156 dias | 1 autor, 5 cambios
public class PositionService : IPositionService
{
    private readonly DatabaseContext _context;
    private readonly IPositionDependencyService _positionDependencyService;
    private readonly IUserRoleService _userRoleService;

    0 referencias | Christian Cespedes, Hace 156 dias | 1 autor, 1 cambio
    public PositionService(
        DatabaseContext context,
        IPositionDependencyService positionDependencyService,
        IUserRoleService userRoleService)
    {
        _context = context;
        _positionDependencyService = positionDependencyService;
        _userRoleService = userRoleService;
    }

    2 referencias | Christian Cespedes, Hace 168 dias | 1 autor, 3 cambios
    public async Task CreateAsync(PositionRequestParams requestParams)
    {
        if (await IsPositionNameUniqueAsync(requestParams.Name))
        {
            var position = new Position
```

```
            var position = new Position
            {
                Name = requestParams.Name,
                Description = requestParams.Description,
                AreaId = requestParams.AreaId
            };

            _context.Positions.Add(position);
            await _context.SaveChangesAsync();

            if (requestParams.ParentPositionId.HasValue)
            {
                var dependencyRequest = new PositionDependencyRequestParams
                {
                    ParentPositionId = requestParams.ParentPositionId.Value,
                    ChildPositionId = position.Id
                };
                await _positionDependencyService.CreateAsync(dependencyRequest);
            }
        }
        else
        {
            throw new InvalidOperationException("Position name already exists.");
        }
    }
```

- Descripción: PositionService.cs implementa la lógica de negocio para la gestión de cargos utilizando Entity Framework Core. Esta clase recibe tres dependencias mediante inyección: DatabaseContext para acceso a datos, IPositionDependencyService para manejar relaciones jerárquicas entre cargos y IUserRoleService para gestionar roles de usuario asociados. El método CreateAsync verifica la unicidad del nombre del cargo antes de crearlo, lo asocia con un área específica y establece relaciones jerárquicas si se proporciona un cargo padre. La implementación garantiza la integridad de datos validando información antes de realizar operaciones en la base de datos y maneja adecuadamente los casos donde ya existe un cargo con el mismo nombre, lanzando excepciones apropiadas para informar al controlador.

- El requerimiento RF-04 Gestionar Usuario abarca los procesos de registrar, listar, actualizar y eliminar personas de la base de datos.
- Procedimiento de cómo se conecta el Frontend con el Backend en UserController.cs

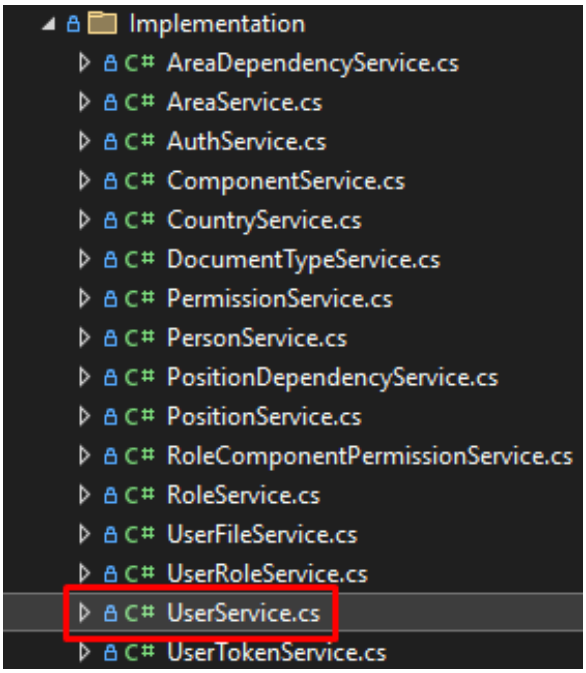



```
using Microsoft.AspNetCore.Mvc;
using SGTD_WebApi.Models.User;
using SGTD_WebApi.Services;
using System;

namespace SGTD_WebApi.Controllers;

[Route("[controller]")]
[ApiController]
1 referencia | Christian Cespedes, Hace 164 días | 1 autor, 4 cambios
public class UserController : Controller
{
    private readonly IUserService _userService;

    0 referencias | Christian Cespedes, Hace 176 días | 1 autor, 1 cambio
    public UserController(IUserService userService)
    {
        _userService = userService;
    }

    [Route("")]
    [HttpPost]
    0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
    public async Task<ActionResult> CreateAsync(UserRequestParams requestParams)
    {
        try
        {
            await _userService.CreateAsync(requestParams);
            return Ok();
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
```

```
32          }
33
34          [Route("")]
35          [HttpPut]
            0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
36          public async Task<ActionResult> UpdateAsync(UserRequestParams requestParams)
37          {
38              try
39              {
40                  await _userService.UpdateAsync(requestParams);
41                  return Ok();
42              }
43              catch (Exception ex)
44              {
45                  return BadRequest(ex.Message);
46              }
47          }
48
49          [Route("")]
50          [HttpGet]
            0 referencias | Christian Cespedes, Hace 164 días | 1 autor, 3 cambios
51          public async Task<ActionResult> GetAllAsync()
52          {
53              try
54              {
55                  var response = await _userService.GetAllAsync();
56                  return Ok(response);
57              }
58              catch (Exception ex)
59              {
60                  return BadRequest(ex.Message);
61              }
62          }
```

- Descripción: UserController.cs actúa como interfaz API para gestionar usuarios en la aplicación, conectando el frontend con la lógica del backend. Este controlador recibe el servicio IUserService mediante inyección de dependencias y expone tres endpoints principales: un método POST para crear usuarios nuevos, un método PUT para actualizar información de usuarios existentes, y un método GET para obtener la lista completa de usuarios. Cada endpoint delega el procesamiento al servicio correspondiente y maneja las excepciones devolviendo respuestas HTTP adecuadas con mensajes de error cuando es necesario. Esta estructura permite que la interfaz de usuario interactúe de manera organizada con la capa de servicios para realizar operaciones CRUD sobre los datos de usuarios en el sistema.

- En la parte de implementación de los servicios UserService.cs estaría la lógica de guardar, modificar y eliminar

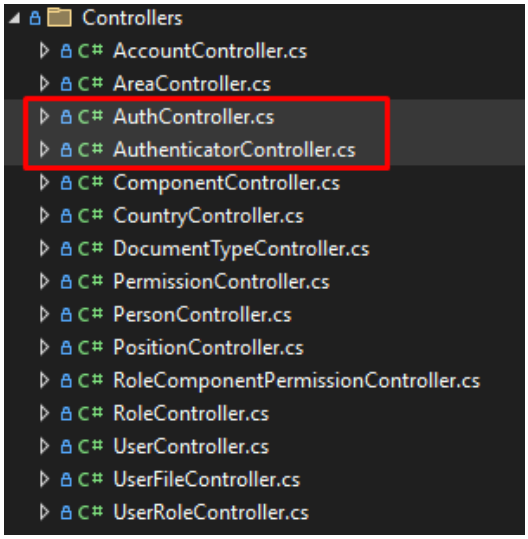



```
using Microsoft.EntityFrameworkCore;
using SGTD_WebApi.DbModel.Context;
using SGTD_WebApi.DbModel.Entities;
using SGTD_WebApi.Helpers;
using SGTD_WebApi.Models.User;

namespace SGTD_WebApi.Services.Implementation;

2 referencias | Christian Cespedes, Hace 131 días | 1 autor, 5 cambios
public class UserService : IUserService
{
    private readonly DatabaseContext _context;
    private readonly EncryptHelper _encryptHelper;

    0 referencias | Christian Cespedes, Hace 148 días | 1 autor, 1 cambio
    public UserService(DatabaseContext context, IConfiguration configuration)
    {
        _context = context;
        _encryptHelper = new EncryptHelper(configuration);
    }

    2 referencias | Christian Cespedes, Hace 131 días | 1 autor, 5 cambios
    public async Task CreateAsync(UserRequestParams requestParams)
    {
        var user = new User
        {
            PersonId = requestParams.PersonId,
            Email = requestParams.Email,
            Password = _encryptHelper.PasswordEncrypt(requestParams.Password),
            StorageSize = requestParams.StorageSize,
            Status = requestParams.Status,
            CreatedAt = DateTime.Now,
            UserGuid = Guid.NewGuid(),
            FolderPath = GenerateFolderPath(),
            PositionId = requestParams.PositionId
```

```
            PositionId = requestParams.PositionId
        };
        _context.Users.Add(user);
        await _context.SaveChangesAsync();
    }

    2 referencias | Christian Cespedes, Hace 131 dias | 1 autor, 5 cambios
    public async Task UpdateAsync(UserRequestParams requestParams)
    {
        if (requestParams.UserGuid == null)
            throw new ArgumentNullException(nameof(requestParams.UserGuid), "User Id is required for update.");

        var user = await _context.Users.FirstOrDefaultAsync(u => u.UserGuid == requestParams.UserGuid);
        if (user == null)
            throw new KeyNotFoundException("User not found.");

        user.PersonId = requestParams.PersonId;
        user.Email = requestParams.Email;

        if (!string.IsNullOrEmpty(requestParams.Password))
        {
            user.Password = BCrypt.Net.BCrypt.HashPassword(requestParams.Password, workFactor: 12);
        }

        user.StorageSize = requestParams.StorageSize;
        user.Status = requestParams.Status;
        user.PositionId = requestParams.PositionId;

        await _context.SaveChangesAsync();
    }
```

- Descripción: UserService.cs implementa la lógica de negocio para gestionar usuarios en la aplicación, sirviendo como puente entre el controlador y la base de datos. Utilizando Entity Framework Core, este servicio realiza operaciones de creación y actualización de usuarios, aplicando encriptación a las contraseñas y validando los datos recibidos. Al crear usuarios, genera identificadores únicos y rutas de carpetas personalizadas, mientras que al actualizarlos verifica su existencia previa y sólo modifica los campos necesarios. Esta clase representa la capa de servicio que procesa las solicitudes del controlador antes de persistir los cambios en la base de datos.

- El requerimiento RF-05 Autentificación abarca los procesos de iniciar sesión en el proyecto web en los cuales intervienen los Controllers AuthController.cs y AuthenticatorController.cs



- Primero se verifica que el usuario exista, solo se verifica el correo en el método LoginAsync del servicio _authService en el controlador AuthController.cs

```
22      [AllowAnonymous]
23      [HttpPost("login")]
        0 referencias | kichitano, Hace 28 días | 2 autores, 4 cambios
24      public async Task<ActionResult> LoginAsync(AuthRequestParams requestParams)
25      {
26          try
27          {
28              var response = await _authService.LoginAsync(requestParams);
29              return Ok(response);
30
31          }
32          catch (ValidationException ex)
33          {
34              return BadRequest(ex.Message);
35          }
36          catch (Exception ex)
37          {
38              return BadRequest(ex.Message);
39          }
40      }
41
```

```
public async Task<bool> LoginAsync(AuthRequestParams requestParams)
{
    var user = await _context.Users // DbSet<User>
        .FirstOrDefaultAsync(u :User => u.Email == requestParams.Email); // Task<User?>

    return user != null;
}
```

- Dicho método devuelve un valor verdadero o falso, si el valor es verdadero se procede a verificar el código OTP en el método VerifyAuthenticatorOTPAsync del servicio _authenticatorService en el controlador VerifyAuthenticatorOTPAsync

```
[AllowAnonymous]
[HttpPost(template: "verify-authenticator-otp")]
0 referencias
public async Task<ActionResult> VerifyAuthenticatorOtpAsync([FromBody] AuthenticatorOtpRequestParams requestParams
{
    try
    {
        bool isValid;
        if (requestParams.Email.Equals("test@test.com"))
        {
            isValid = true;
        }
        else
        {
            isValid = await _authenticatorService.VerifyAuthenticatorOtpAsync(requestParams);
        }

        if (isValid)
        {
            var response:AuthDto = await _authService.LoginOtpAsync(requestParams);
            if (response.Success)
            {
                var cookieOptions = _authService.SetRefreshTokenCookie(response.RefreshToken);
                Response.Cookies.Append("refresh_token", response.RefreshToken, cookieOptions);
                return Ok(response);
            }

            return BadRequest(error: new { message = "Credenciales inválidas." });
        }

        return BadRequest(error: new { message = "Código OTP inválido." });
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

```
2 referencias
public async Task<bool> VerifyAuthenticatorOtpAsync(AuthenticatorOtpRequestParams requestParams)
{
    var userGuid= await _context.Users // DbSet<User>
        .Where(q:User => q.Email == requestParams.Email) // IQueryable<User>
        .Select(q:User => q.UserGuid) // IQueryable<Guid>
        .FirstOrDefaultAsync(); // Task<Guid>

    if (userGuid == Guid.Empty)
        throw new InvalidOperationException("Invalid User.");

    var secretKey:string? = await _context.Authenticators // DbSet<Authenticator>
        .Where(q:Authenticator =>
            q.UserGuid == userGuid
            && q.IsActive) // IQueryable<Authenticator>
            //&& q.ExpiresAt > DateTime.UtcNow)
        .OrderByDescending(q:Authenticator => q.Id) // IOrderedQueryable<Authenticator>
        .Select(q:Authenticator => q.SecretKey) // IQueryable<string>
        .FirstOrDefaultAsync(); // Task<string?>

    if (secretKey == null)
        throw new InvalidOperationException("Invalid Secret Key.");

    var bytes:byte[]? = Base32Encoding.ToBytes(secretKey);
    var totp = new Totp(bytes);

    var isValid = totp.VerifyTotp(
        requestParams.OtpCode,
        out _,
        new VerificationWindow(previous: 1, future: 1)
    );

    return isValid;
}
```

- Una vez verificado y validado el código OTP se procede a validar las credenciales y el método LoginOtpAsync del servicio AuthService en el controlador VerifyOtpAsync

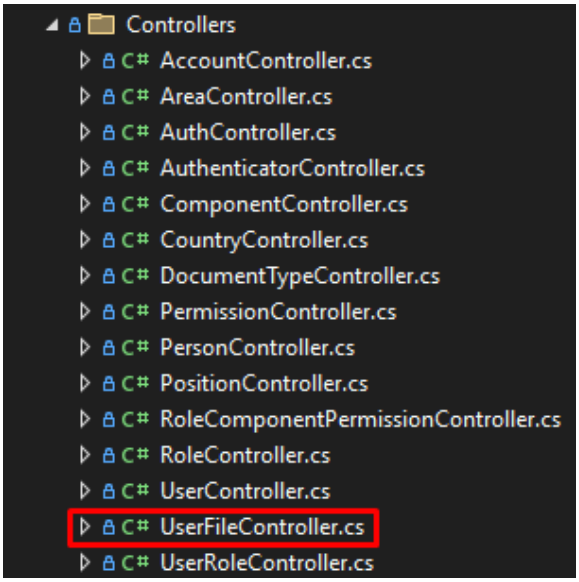
```
2 referencias
public async Task<AuthDto> LoginOtpAsync(AuthenticatorOtpRequestParams requestParams)
{
    var user = await _context.Users // DbSet<User>
        .FirstOrDefaultAsync(u :User => u.Email == requestParams.Email); // Task<User?>

    if (user == null)
    {
        return new AuthDto { Success = false };
    }

    bool isValidPassword = BCrypt.Net.BCrypt.Verify( text: requestParams.Password, hash: user.Password);
    if (!isValidPassword)
    {
        throw new ValidationException("Credenciales incorrectas");
    }

    var token :string = await _userTokenService.GenerateTokenAsync(user.UserGuid);
    var refreshToken :string? = await _context.UserTokens // DbSet<UserToken>
        .Where(q :UserToken => q.Token.Equals(token)) // IQueryable<UserToken>
        .Select(q :UserToken => q.RefreshToken) // IQueryable<string>
        .FirstOrDefaultAsync(); // Task<string?>

    return new AuthDto
    {
        Success = true,
        Token = token,
        RefreshToken = refreshToken ?? string.Empty
    };
}
```

- Una vez validadas las credenciales del usuario se procede a crear el token de acceso y el token de actualización en el método SetRefreshTokenCookie del servicio AuthService en el controlador VerifyOtpAsync

```
3 referencias
public CookieOptions SetRefreshTokenCookie(string refreshToken)
{
    var cookieOptions = new CookieOptions
    {
        HttpOnly = true,
        Secure = true,
        SameSite = SameSiteMode.None,
        Expires = DateTime.UtcNow.AddDays(7)
    };
    return cookieOptions;
}
```

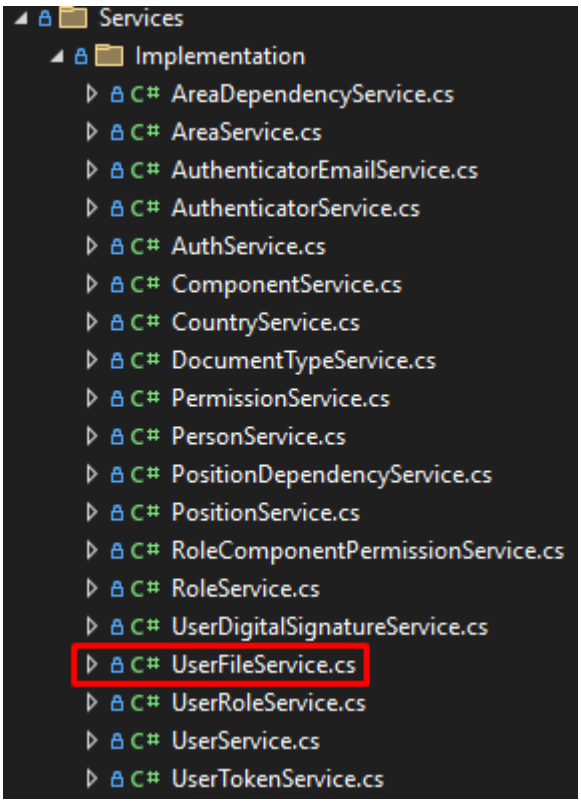
- Creados los tokens se procede a enviar al Frontend una cookie para el acceso correspondiente y el usuario puede hacer uso del sistema

- El requerimiento RF-06 Gestionar Archivos donde se podrá guardar y compartir archivos dentro del sistema
- Procedimiento de cómo se conecta el Frontend con el Backend en UserFileController.cs




```
using Microsoft.AspNetCore.Mvc;
using SGTD_WebApi.Services;

namespace SGTD_WebApi.Controllers;

[Route("[controller]")]
[ApiController]
1 referencia | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
public class UserFileController : ControllerBase
{
    private readonly IUserFileService _fileService;

    0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public UserFileController(IUserFileService fileService)
    {
        _fileService = fileService;
    }

    [Route("{userGuid}")]
    [HttpGet]
    0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task<ActionResult> GetByUserGuIdAsync(Guid userGuid)
    {
        try
        {
            var response = await _fileService.GetByUserGuIdAsync(userGuid);
            return Ok(response);
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
```

- En la parte de implementación de los servicios UserFileService.cs estaría la lógica de obtener por el ID del usuario subir archivos, descargar archivo y descargar múltiples archivos

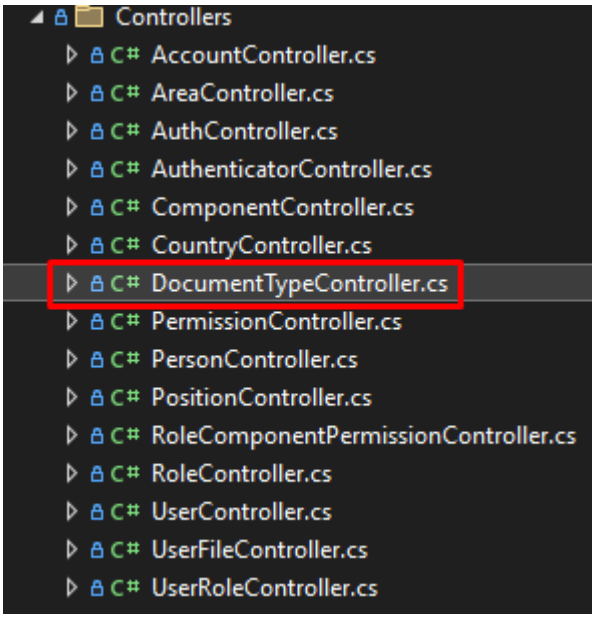



```
2 referencias | kichitano, Hace 4 días | 2 autores, 3 cambios
public class UserFileService : IUserFileService
{
    private readonly DatabaseContext _context;
    private readonly EncryptHelper _encryptHelper;
    private readonly string _basePath;

    0 referencias | Christian Cespedes, Hace 188 días | 1 autor, 1 cambio
    public UserFileService(DatabaseContext context, IConfiguration configuration)
    {
        _context = context;
        _encryptHelper = new EncryptHelper(configuration);
        _basePath = configuration["FilesPath:BasePath"] ?? string.Empty;
    }

    2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task<List<UserFileDto>> GetByUserGuIdAsync(Guid userGuid)...

    2 referencias | kichitano, Hace 4 días | 2 autores, 3 cambios
    public async Task UploadFilesAsync(List<IFormFile> userFiles, Guid userGuid)...

    3 referencias | Christian Cespedes, Hace 177 días | 1 autor, 2 cambios
    public async Task<UserFileByteDto> DownloadFileAsync(int id)...

    2 referencias | Christian Cespedes, Hace 188 días | 1 autor, 1 cambio
    public async Task<byte[]> DownloadMultipleFilesAsync(List<int> ids)...
}
```

- En la implementación del método UploadFilesAsync del servicio UserFileService se encuentra la lógica de almacenar archivos donde si el usuario sube uno o varios archivos, estos son encriptados o almacenados en la ruta compuesta por la carpeta base y la carpeta segura generada al crear el usuario

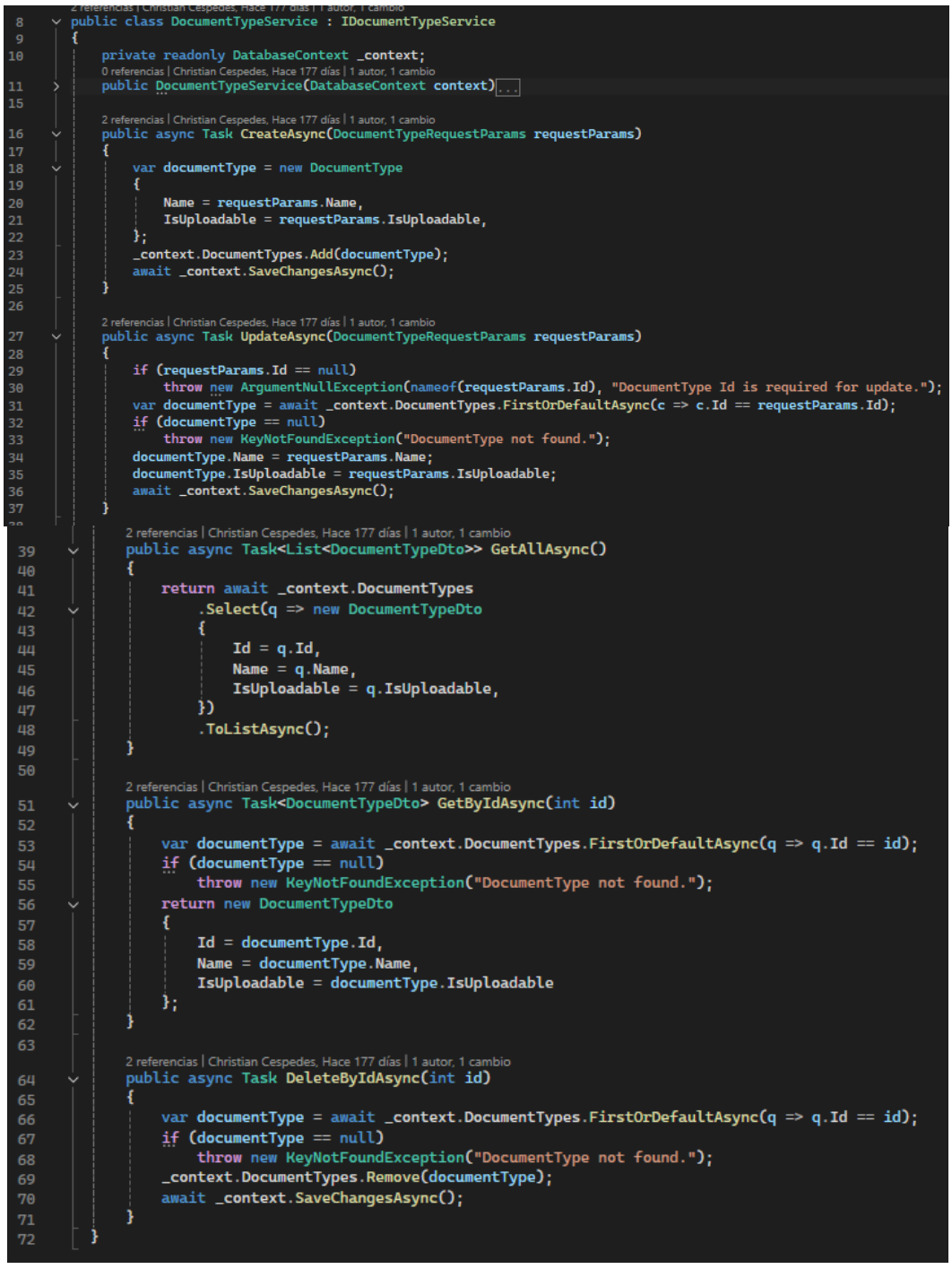
```
2 referencias | kichitaro, Hace 4 días | 2 autores, 3 cambios
public async Task UploadFilesAsync(List<IFormFile> userFiles, Guid userGuid)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.UserGuid == userGuid);

    if (user == null)
    {
        throw new ValidationException("Usuario no encontrado");
    }

    var folderPath = Path.Combine(_basePath, user.FolderPath);
    if (!Directory.Exists(folderPath))
    {
        Directory.CreateDirectory(folderPath);
    }

    foreach (var userFile in userFiles)
    {
        var fileName = userFile.FileName;
        var filePath = Path.Combine(folderPath, fileName);

        await using (var stream = new FileStream(filePath, FileMode.Create))
        {
            var encryptedContent = _encryptHelper.FileEncrypt(userFile.OpenReadStream());
            await stream.WriteAsync(encryptedContent, 0, encryptedContent.Length);
        }

        if (Path.Exists(filePath))
        {
            var file = new UserFile
            {
                UserId = user.Id,
                FileName = userFile.FileName,
                FileSize = userFile.Length,
                ContentType = Path.GetExtension(fileName).TrimStart('.'),
                CreatedAt = DateTime.UtcNow
            };
            await _context.UserFiles.AddAsync(file);
        }
        else
        {
            throw new FileNotFoundException("No se pudo guardar el archivo correctamente.");
        }
    }
    await _context.SaveChangesAsync();
}
```

- El requerimiento RF-07 Gestionar Documento abarca los procesos de registrar, listar, actualizar y eliminar tipos de documento




```
9   public class DocumentTypeController : Controller
10  {
11      private readonly IDocumentTypeService _documentTypeService;
12
        0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
13      public DocumentTypeController(IDocumentTypeService documentTypeService)
14      {
15          _documentTypeService = documentTypeService;
16      }
17
18      [Route("")]
19      [HttpPost]
        0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
20      public async Task<ActionResult> CreateAsync(DocumentTypeRequestParams requestParams)
21      {
22          try
23          {
24              await _documentTypeService.CreateAsync(requestParams);
25              return Ok();
26          }
27          catch (Exception ex)
28          {
29              return BadRequest(ex.Message);
30          }
31      }
32
33      [Route("")]
34      [HttpPut]
        0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
35      public async Task<ActionResult> UpdateAsync(DocumentTypeRequestParams requestParams)...
47
48      [Route("")]
49      [HttpGet]
        0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
50      public async Task<ActionResult> GetAllAsync()...
62
63      [Route("{id}")]
64      [HttpGet]
        0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
65      public async Task<ActionResult> GetByIdAsync(int id)...
77
78      [HttpDelete("{id}")]
        0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
79      public async Task<ActionResult> DeleteByIdAsync(int id)...
91  }
```

- En la parte de implementación de los servicios estaría la lógica de guardar, modificar y eliminar



```
2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
public class DocumentTypeService : IDocumentTypeService
{
    private readonly DatabaseContext _context;
    0 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public DocumentTypeService(DatabaseContext context)...

    2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task CreateAsync(DocumentTypeRequestParams requestParams)
    {
        var documentType = new DocumentType
        {
            Name = requestParams.Name,
            IsUploadable = requestParams.IsUploadable,
        };
        _context.DocumentTypes.Add(documentType);
        await _context.SaveChangesAsync();
    }

    2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task UpdateAsync(DocumentTypeRequestParams requestParams)
    {
        if (requestParams.Id == null)
            throw new ArgumentNullException(nameof(requestParams.Id), "DocumentType Id is required for update.");
        var documentType = await _context.DocumentTypes.FirstOrDefaultAsync(c => c.Id == requestParams.Id);
        if (documentType == null)
            throw new KeyNotFoundException("DocumentType not found.");
        documentType.Name = requestParams.Name;
        documentType.IsUploadable = requestParams.IsUploadable;
        await _context.SaveChangesAsync();
    }
    2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task<List<DocumentTypeDto>> GetAllAsync()
    {
        return await _context.DocumentTypes
            .Select(q => new DocumentTypeDto
            {
                Id = q.Id,
                Name = q.Name,
                IsUploadable = q.IsUploadable,
            })
            .ToListAsync();
    }

    2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task<DocumentTypeDto> GetByIdAsync(int id)
    {
        var documentType = await _context.DocumentTypes.FirstOrDefaultAsync(q => q.Id == id);
        if (documentType == null)
            throw new KeyNotFoundException("DocumentType not found.");
        return new DocumentTypeDto
        {
            Id = documentType.Id,
            Name = documentType.Name,
            IsUploadable = documentType.IsUploadable
        };
    }

    2 referencias | Christian Cespedes, Hace 177 días | 1 autor, 1 cambio
    public async Task DeleteByIdAsync(int id)
    {
        var documentType = await _context.DocumentTypes.FirstOrDefaultAsync(q => q.Id == id);
        if (documentType == null)
            throw new KeyNotFoundException("DocumentType not found.");
        _context.DocumentTypes.Remove(documentType);
        await _context.SaveChangesAsync();
    }
}
```