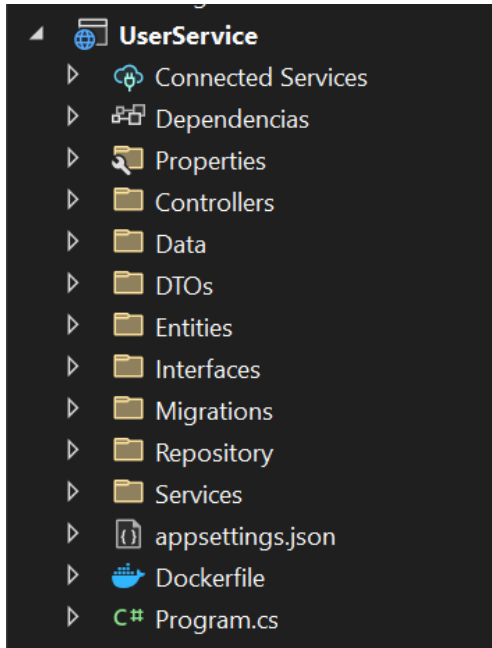


Prueba Backend.

*Nota: Por términos del tiempo, solo pude acabar el primer reactivo de la prueba

El proyecto esta segmentado en 2 Microservicios, UserService y MailService

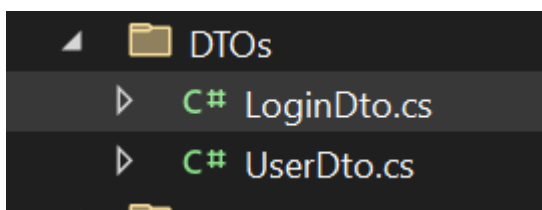


UserService esta conformado de las siguientes carpetas:

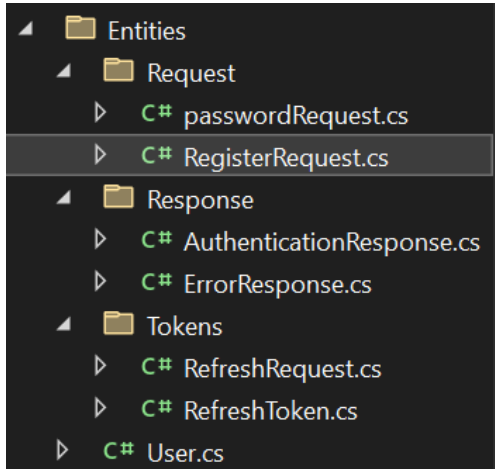
Data: Contiene la implementación del contexto de la base de datos en EF, contiene las tablas Users y RefreshTokens

```
9 referencias
public class DataContext : DbContext
{
    0 referencias
    public DataContext(DbContextOptions<DataContext> options) : base(options)
    {
    }
    5 referencias
    public DbSet<User> Users => Set<User>();
    4 referencias
    public DbSet<RefreshToken> RefreshTokens => Set<RefreshToken>();
}
```

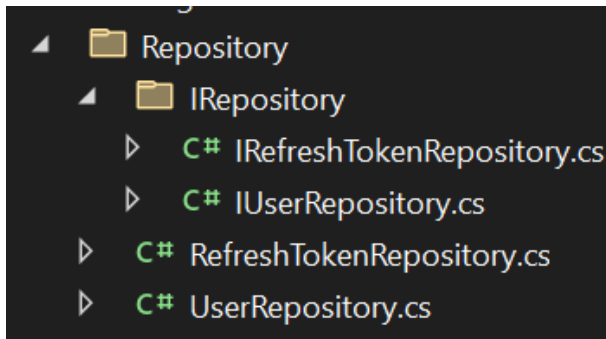
DTOs: Contiene el modelo que se utiliza para hacer login en la aplicación y el modelo del usuario que se devuelve al hacer la consulta.



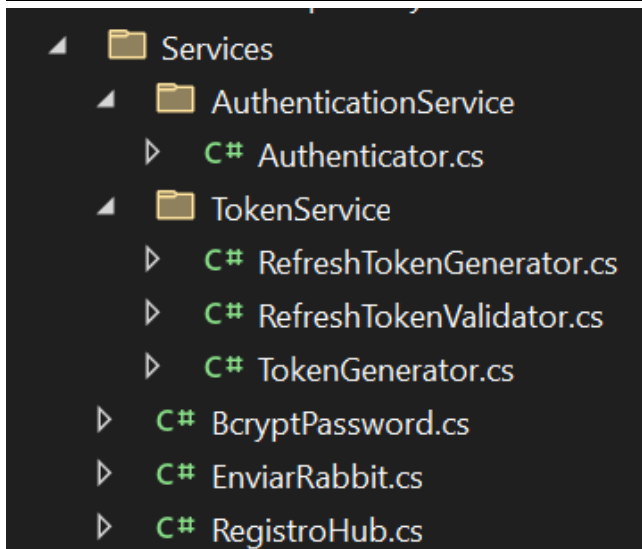
Entities: Contiene los modelos a utilizar para hacer request a los endpoints de refresh tokens, recuperación y registro, los modelos de respuesta de tokens y errores y el modelo del usuario en el que se basa la tabla en BD.



Repository: Contiene la declaración e implementación de los repositorios que utiliza el servicio



Services: Contiene los servicios de autenticación y generación de tokens, así como el servicio de encriptación de contraseña y los servicios de SignalR para la activación de notificaciones como el servicio que hace la comunicación con RabbitMQ al servicio de Mail



La declaración de los servicios correspondientes a UserService.

```
//Implementación de la autenticación con JWT
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(o =>
{
    o.TokenValidationParameters = new TokenValidationParameters
    {
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        ValidAudience = builder.Configuration["Jwt:Audience"],
        IssuerSigningKey = new SymmetricSecurityKey
            (Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"])),
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = false,
        ValidateIssuerSigningKey = true,
    };
});

//Declaración de los esquemas de servicio
builder.Services.AddSingleton<EnviarRabbit>();
builder.Services.AddSingleton<BcryptPassword>();
builder.Services.AddSingleton<TokenGenerator>();
builder.Services.AddSingleton<RefreshTokenGenerator>();
builder.Services.AddSingleton<RefreshTokenValidator>();
builder.Services.AddScoped<RegistroHub>();
builder.Services.AddScoped<Authenticator>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IRefreshTokenRepository, RefreshTokenRepository>();
```

El endpoint de registro con la llamada al servicio RegistroHub.

```
[AllowAnonymous]
[HttpPost("register")]
0 referencias
public async Task<IActionResult> Register([FromBody] RegisterRequest userTemp)
{
    if (!ModelState.IsValid)
    {
        return BadRequestModelState();
    }

    var existingUser = await _UserRepository.GetByCorreo(userTemp.correo);

    if (existingUser != null)
    {
        return Conflict(new ErrorResponse("Ya hay una cuenta con ese correo"));
    }

    if (userTemp.password != userTemp.confirmPassword)
        return BadRequest(new ErrorResponse("Las contraseñas no coinciden"));

    LoginDto login = new LoginDto()
    {
        correo = userTemp.correo,
        password = userTemp.password
    };

    userTemp.password = _passwordHasher.Hash(userTemp.password);

    await _UserRepository.Create(userTemp);
    await _hub.NotificarRegistro(userTemp.nombre);
    return await Login(login);
}
```

La implementación de RegistroHub, se le envía el nombre de usuario, hace una llamada al repositorio de Usuarios y procede a enviar las llamadas al servicio de correos para mandar la notificación.

```
4 referencias
public class RegistroHub : Hub
{
    private readonly EnviarRabbit _enviar;
    private readonly IUserRepository _repository;
    0 referencias
    public RegistroHub(EnviarRabbit enviar, IUserRepository repository)
    {
        _enviar = enviar;
        _repository = repository;
    }
    1 referencia
    public async Task NotificarRegistro(string nombreUsuario)
    {
        // Notificar a todos los clientes conectados
        await Clients.All.SendAsync("UsuarioRegistrado", nombreUsuario);

        await EnvioMultiple(nombreUsuario);
    }
    1 referencia
    private async Task EnvioMultiple(string nombreUsuario)
    {
        string contenido = $"Nuevo usuario registrado {nombreUsuario}";
        var users = await _repository.GetAll();

        foreach (var user in users)
        {
            _enviar.enviar(user.correo, contenido, "notificacion");
        }
    }
}
```

Tanto el endpoint de registro como el de recuperar hacen una llamada al servicio de correos.

```
[AllowAnonymous]
[HttpPost("recuperar")]
0 referencias
public async Task<IActionResult> Recuperar([FromBody] string correo)
{
    var existingUser = await _userRepository.GetByCorreo(correo);

    if (existingUser == null)
    {
        return Unauthorized();
    }
    var token = await _authenticator.Authentication(existingUser);
    //Consumo de api
    _enviar.enviar(correo, token.token, "recuperar");

    return Ok("Se ha enviado la confirmación al correo");
}
```

La llamada al servicio esta declarada en la clase EnviarRabbit, en la cual se envía el correo en cuestión, el contenido y el protocolo de la llamada.

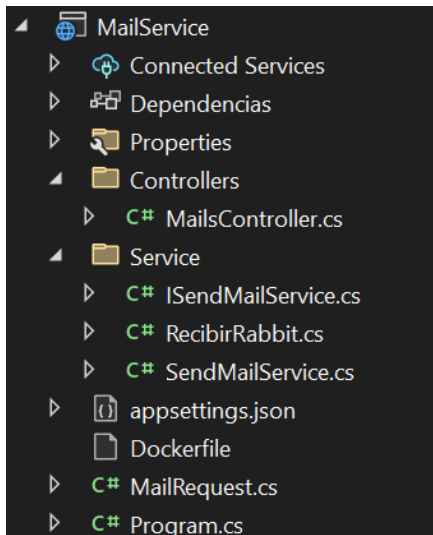
```
5 referencias
public class EnviarRabbit
{
    3 referencias
    public void enviar(string correo, string contenido, string protocolo)
    {
        // Conexión a RabbitMQ
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.QueueDeclare(queue: "cola",
                                durable: false,
                                exclusive: false,
                                autoDelete: false,
                                arguments: null);

            string message = $"data\\: {\"mail\\\": \"+correo+\", \"contenido\\\": \"+contenido+ \"\", \"protocolo\\\": \"+protocolo+\"}\"";
            var body = Encoding.UTF8.GetBytes(message);

            // Publicar el mensaje en la cola
            channel.BasicPublish(exchange: "",
                                routingKey: "cola",
                                basicProperties: null,
                                body: body);

            Console.WriteLine("Mensaje enviado: {0}", message);
        }
    }
}
```

Del lado del servicio de correos(MailService), la distribución es la siguiente:



Los servicios en cuestión que utiliza son SendMailService donde se usa SMTP para el envío del

```
namespace MailService.Service
{
    2 referencias
    public class SendMailService : ISendMailService
    {
        private readonly SmtClient _smtpClient;
        private readonly IConfiguration _config;
        0 referencias
        public SendMailService(IConfiguration config)
        {
            _config = config;
            _smtpClient = new SmtClient("smtp.gmail.com")
            {
                Port = 587,
                Credentials = new NetworkCredential(_config.GetValue<string>("Email:mail"), _config.GetValue<string>("Email:pass")),
                EnableSsl = true
            };
        }

        6 referencias
        public void SendEmail(string from, string to, string subject, string body)
        {
            var mailMessage = new MailMessage(from, to, subject, body)
            {
                IsBodyHtml = true
            };
            _smtpClient.Send(mailMessage);
        }
    }
}
```

Y RecibirRabbit en donde se obtiene y se descripta el mensaje/petición.

```
1 0 referencias
2 public void Recibir()
3 {
4     var factory = new ConnectionFactory() { HostName = "localhost" };
5     using (var connection = factory.CreateConnection())
6     using (var channel = connection.CreateModel())
7     {
8         channel.QueueDeclare(queue: "cola",
9             durable: false,
10            exclusive: false,
11            autoDelete: false,
12            arguments: null);
13
14         var consumer = new EventingBasicConsumer(channel);
15         consumer.Received += (model, ea) =>
16         {
17             var body = ea.Body.ToArray();
18             var message = Encoding.UTF8.GetString(body);
19             Console.WriteLine("Mensaje recibido: {0}", message);
20
21             // Aqui analizas el mensaje para determinar qué método o endpoint llamar y cualquier otro dato necesario
22
23             // Llamar al método o endpoint correspondiente en el microservicio receptor
24             // Método ficticio
25             ProcesarMensaje(message);
26         };
27         channel.BasicConsume(queue: "mi_cola",
28             autoAck: true,
29             consumer: consumer);
30
31         Console.WriteLine("Presione cualquier tecla para salir.");
32         Console.ReadLine();
33     }
34 }
```

El proceso de acción esta dividido por protocolos:

Test: donde se envía un correo de prueba para comprobar que el servicio este en línea.

Recuperar: Donde se envía un token y la ruta para la recuperación de la contraseña y actualización de la misma.

Notificación: Donde se envía una notificación al cliente en cuestión.

```
1 MailService
2 1 referencia
3 public async Task ProcesarMensaje(string message)
4 {
5     // Convertir el mensaje JSON en un objeto
6     var mensajeObjeto = JsonSerializer.Deserialize<Mensaje>(message);
7
8     // Verificar que el mensaje tenga todos los campos necesarios
9     if (mensajeObjeto != null && !string.IsNullOrEmpty(mensajeObjeto.correo) && !string.IsNullOrEmpty(mensajeObjeto.contenido) && !string.IsNu
10 {
11     switch (mensajeObjeto.protocolo)
12     {
13         case "recuperar":
14             BackgroundJob.Enqueue(() => _service.SendEmail("Servicio", mensajeObjeto.correo, "Recuperacion", "Accede a la ruta Users/restablecer/
15             break;
16         case "test":
17             BackgroundJob.Enqueue(() => _service.SendEmail("Servicio", mensajeObjeto.correo, "Test", "Este es un test de envio de correo:" + mensa
18             break;
19         case "notificacion":
20             BackgroundJob.Enqueue(() => _service.SendEmail("Servicio", mensajeObjeto.correo, "Nuevo", "Notificacion:" + mensajeObjeto.cont
21             break;
22     }
23 }
24 else
25 {
26     Console.WriteLine("El mensaje no contiene todos los campos necesarios.");
27 }
28 }
```



Todas las rutas usan Hangfire para la administración de las colas de tareas.

Aquí esta la implementación de los servicios a utilizar en MailService.

```
23
24
25 var connectionString = builder.Configuration.GetConnectionString("MyDb");
26 builder.Services.AddHangfire(configuration => configuration
27     .SetDataCompatibilityLevel(CompatibilityLevel.Version_170)
28     .UseSimpleAssemblyNameTypeSerializer()
29     .UseRecommendedSerializerSettings()
30     .UseSqlServerStorage(connectionString, new SqlServerStorageOptions
31     {
32         CommandBatchMaxTimeout = TimeSpan.FromMinutes(5),
33         SlidingInvisibilityTimeout = TimeSpan.FromMinutes(5),
34         QueuePollInterval = TimeSpan.Zero,
35         UseRecommendedIsolationLevel = true,
36         DisableGlobalLocks = true
37     });
38
39 // Agregar Hangfire Server para procesar las colas en segundo plano
40 builder.Services.AddHangfireServer();
41
42 builder.Services.AddSingleton<ISendMailService, SendMailService>();
43 builder.Services.AddSingleton<RecibirRabbit>();
```

Como base para este proyecto, utilice un proyecto anterior en donde utilizaba el patron DDD, para hacer un servicio de autenticación:

[PruebasTécnicas](#) / PruebaTecnica / 

 AxelAZSA First Kick 

Name	Last commit message
 ..	
 Prueba.Application	First Kick
 Prueba.Domain	First Kick
 Prueba.Infrastructure	First Kick

Aparte de usar la documentación correspondiente del middleware a utilizar.