



ESCOLA  
SUPERIOR  
DE TECNOLOGIA  
E GESTÃO

# **Relatório do Projeto**

**Versão 2.0**

**Elaborado por Grupo 13**

**Laboratório de Desenvolvimento de Software**

**12/01/2021**

# Índice

<b>1. Introdução .....</b>	<b>1</b>
<b>2. Explicação Geral do funcionamento da aplicação .....</b>	<b>2</b>
2.1 Visão Geral .....	2
2.2 Estrutura da aplicação .....	2
2.3 Modelo de dados .....	2
2.4 Autenticação JWT (Json Web Token) .....	3
2.5 Implementação do chat em SignalR (Backend) .....	7
2.6 Implementação do chat em SignalR (Frontend) .....	11
2.7 Ferramentas e Metodologias utilizadas .....	13
<b>3. Arquitetura do Projeto .....</b>	<b>17</b>
<b>4. Diagrama de Classes .....</b>	<b>18</b>
<b>5. A aplicação do ponto de vista do Utilizador .....</b>	<b>19</b>
<b>6. A aplicação do ponto de vista do Admin .....</b>	<b>28</b>
<b>7. Conclusão e limitações .....</b>	<b>31</b>
<b>8. Bibliografia .....</b>	<b>32</b>

# Índice de figuras

Figura 1 - Autenticação via JWT .....	3
Figura 2 - Autenticação via JWT .....	3
Figura 3 - Classe GenerateToken.....	4
Figura 4 - Ficheiro Startup.cs.....	5
Figura 5 - Autenticação via JWT .....	5
Figura 6 - Authorization Bearer.....	6
Figura 7 - Adicionar o SignalR.....	7
Figura 8 - Atualização do método Configure .....	7
Figura 9 - Model que representa as mensagens .....	8
Figura 10 - Método GetIdFromClaimIdentity .....	8
Figura 11 - Método onDisconnectedAsync .....	9
Figura 12 - Método GetChats do controller Messages .....	9
Figura 13 - Método GetMessages da classe MessageRepo.cs .....	10
Figura 14 - Model Message (Frontend) .....	11
Figura 15 - Criação da HubConnection .....	11
Figura 16 - hubConnection.StopAsync.....	11
Figura 17 - Chat Frontend SignalR.....	12
Figura 18 - JetBrains Rider (IDE).....	13
Figura 19 - ASP.NET Core.....	13
Figura 20 - GitLab ESTG .....	14
Figura 21 - Documentação Swagger.....	15
Figura 22 - Xamarin logo.....	16
Figura 23 - Arquitetura do Projeto.....	17
Figura 24 - Diagrama de Classes .....	18
Figura 25 - LoginPage reBOOK.....	19
Figura 26 - RegistrationPage reBOOK.....	19
Figura 27 - Email com credenciais de acesso .....	20
Figura 28 - Recuperar password .....	20
Figura 29 - Adicionar Livro.....	21
Figura 30 - Lista de livros.....	21
Figura 31 - Página principal.....	22
Figura 32 - PopUp Report .....	23
Figura 33 - Dashboard da app.....	23
Figura 34 - Contactar Suporte .....	24
Figura 35 - Lista de Favoritos.....	24
Figura 36 - Procurar utilizador.....	25
Figura 37 - Pesquisar utilizador .....	25
Figura 38 - Localização .....	26
Figura 39 - Editar perfil .....	26
Figura 40 - Lista de Matches .....	27
Figura 41 - Sistema de troca de mensagens .....	27
Figura 42 - Dashboard Admin .....	28
Figura 43 - Lista de todos os reports.....	28
Figura 44 - Procurar Reports por Id.....	29
Figura 45 - Ações Admin.....	29
Figura 46 - SwipeCardView Admin.....	30
Figura 47 - Alerta por email.....	30

## Historico de revisões

Nome	Data	Razão da alteração	Versão
João Leocádio	17/12/2020	Arquitetura do Projeto	0.1
João Leocádio	22/12/2020	Diagrama de Classes	0.2
João Leocádio	27/12/2020	Autenticação JWT	0.3
João Leocádio	07/01/2021	Ferramentas e Metodologias	0.4
João Leocádio	08/01/2021	Índice de imagens	0.5
João Leocádio	12/01/2021	Xamarin e Bibliografia	0.6
João Leocádio	12/01/2021	A aplicação do ponto de vista do utilizador	0.7
João Leocádio	12/01/2021	A aplicação do ponto de vista do Admin	0.8
Paulo Brito	12/01/2021	Formatação e revisão da ortografia	0.9
João Leocádio	18/01/2021	Chat e SignalR	1.0
Paulo Brito	19/01/2021	SignalR Frontend	1.1
Paulo Brito	19/01/2021	Formatação e revisão da ortografia	1.2

# 1. Introdução

No âmbito do 3º ano do curso de Engenharia Informática, da Escola Superior de Tecnologia e Gestão, foi proposto a realização de um trabalho prático que foi realizado em grupos de dois elementos, de modo que, o grupo é constituído por: Paulo Brito e João Leocádio.

Este projeto, foi idealizado pelo grupo 13 e aprovado pelos respetivos docentes da unidade curricular, e o objetivo principal passava por aplicar os conceitos aprendidos durante o curso. como por exemplo, metodologias de testes, metodologias de desenvolvimento ágeis, Continuous Integration (CI) e planos de Software Configuration Management (SCM).

Assim e resumindo, foi elaborado uma aplicação mobile que funcionará na plataforma Android e permitirá a troca de livros entre vários utilizadores, ou seja, o utilizador consegue ver vários livros e poderá dar “like” e assim adicionar livros aos favoritos.

Deste modo e através do desenvolvimento desta aplicação, pode-se afirmar que expandimos as nossas respetivas competências de análise de um problema, identificação de requisitos, planear e implementar um plano de testes. Utilizar metodologias de desenvolvimento ágil como o SCRUM no desenvolvimento de software, usar ferramentas de suporte (ASP NET core, Swagger, Xunit, Gitlab) e perceber a importância de trabalhar em equipa assim como manter um ambiente saudável entre todos.

## **2. Explicação Geral do funcionamento da aplicação**

### **2.1 Visão Geral**

A Rebook é uma aplicação focada na troca de livros usados. Após registo, um utilizador pode adicionar à sua lista os livros que tem para troca, cada livro deve ter nome, autor e uma avaliação do seu estado (1 para muito mau estado, 5 para novo), assim como uma ou mais fotos.

Por outro lado, um utilizador pode ver os livros de outros utilizadores no seu range através da Swipe View, se gostar de um livro este é adicionado à sua lista de favoritos, se o dono desse livro tiver um dos livros do utilizador na sua lista de favoritos ocorre um match abrindo a funcionalidade de mensagens entre utilizadores de maneira a que estes possam combinar uma troca de livros.

Se o utilizador não gostar do livro este não volta a aparecer na Swipe View, pode no entanto encontrá-lo recorrendo à funcionalidade de "Search Book".

Outra funcionalidade inerente à aplicação passa por poder avaliar outro utilizador. Isto cria um sistema de reputação através do qual podemos definir se um utilizador é digno de confiança e se é seguro trocar livros com ele.

Um utilizador pode também submeter um report a outro utilizador, assim como bloqueá-lo.

### **2.2 Estrutura da aplicação**

O back end da aplicação foi construído em Asp.Net core, usando como base de dados MySql ao qual se liga através de EF Core. Através das funções presentes em Data regulamos o acesso à base de dados, e através dos controllers regulamos a comunicação com o front end.

O front end foi construído em Xamarin.Forms. De momento apenas está a ser considerado o deployment da aplicação em android, estando em aberta a possibilidade de deployment para iOS no futuro.

### **2.3 Modelo de dados**

#### **As nossas entidades base são:**

- - Books: Para guardar livros
- - Users: Para guardar dados de utilizador
- - Reports: Para guardar reports
- - Messages: Para guardar mensagens.

#### **Temos entidades cujo propósito é relacionar estas entidades:**

- - BooksLiked: Associa utilizador a livros que gostou
- - UsersRated: Associa utilizador a outro que este avaliou
- - Matches: Associa dois utilizadores e verifica se há match

## 2.4 Autenticação JWT (Json Web Token)

A segurança é um dos aspectos mais importantes nas aplicações. Por essa razão, utilizou-se o método JWT (Json Web Token) que permite a autenticação entre duas partes por meio de um token que valida uma requisição web. Esse token, não é mais que um simples código em Base64 que permite armazenar objetos JSON juntamente com os dados que permitem a autenticação da requisição.

Na figura seguinte, é possível ver um cliente a enviar no corpo do request HTTP dados como endereço de e-mail e senha ao endpoint de autenticação da API.

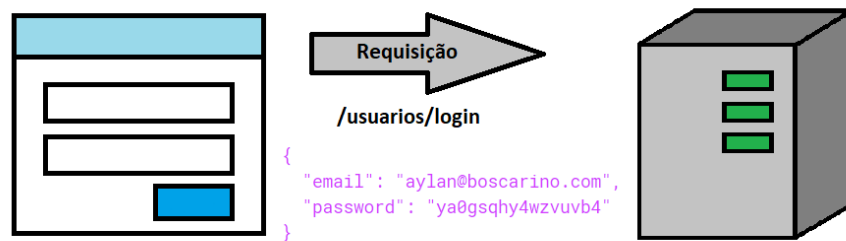


Figura 1 - Autenticação via JWT

Depois dos dados enviados pelo cliente serem devidamente autenticados pelo servidor, é criado um token JWT assinado com um segredo interno da API e enviará este token de volta ao cliente, como podemos ver na imagem seguinte.

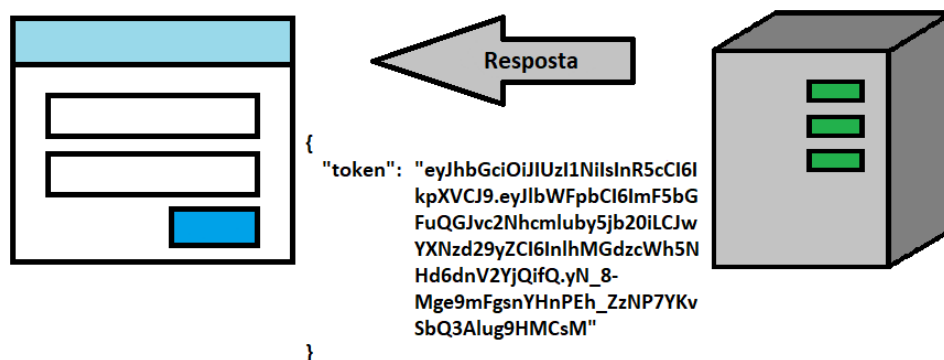


Figura 2 - Autenticação via JWT

Depois de estar munido com o token autenticado, o cliente detém acesso aos endpoints da aplicação que antes não tinha. Para ocorrer esse acesso, é necessário informar esse token no header, e por convenção, após a palavra Bearer.

### 2.4.1 Implementação Backend JWT

Após um estudo cuidadoso sobre JWT, partiu-se para a implementação do mesmo. Como podemos verificar na próxima imagem. Esta classe é responsável por gerar um token para o utilizador.

```
namespace REBOOK.Services
{
    [1 usage] [1] JOAO EMANUEL CARVALHO LEOCADIO
    public static class TokenService
    {
        [1 usage] [1] JOAO EMANUEL CARVALHO LEOCADIO
        public static string GenerateToken(User user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key :byte[] = Encoding.ASCII.GetBytes(Settings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new Claim( type: ClaimTypes.Name, value: user.Name.ToString()),
                    new Claim( type: ClaimTypes.Gender, value: user.Password.ToString())
                }),
                Expires = DateTime.UtcNow.AddHours(2),
                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), algorithm: SecurityAlgorithms.HmacSha256Signature)
            };
            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
```

*Figura 3 - Classe GenerateToken*



## *Relatório do Projeto*

Várias configurações foram feitas no ficheiro *Startup.cs*, para conseguirmos utilizar a autenticação por Json Web Token, como é demonstrado na próxima imagem.

```
// This method gets called by the runtime. Use this method to add services to the container.
using IOAO;
using IOAO.Emanuel.Carvalho.Leocadio;

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

    services.AddSession();

    services.AddSession(options => {
        options.IdleTimeout = TimeSpan.FromMinutes(30);
    });

    services.AddCors();
    services.AddControllers()
        .AddNewtonsoftJson(opt => {
            opt.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
        });
    services.AddSwaggerGen(c => {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "REBOOK", Version = "v1" });
    });

    var key = Encoding.ASCII.GetBytes(Settings.Secret);
    services.AddAuthentication(x => {
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(x => {
        x.RequireHttpsMetadata = false;
        x.SaveToken = true;
        x.TokenValidationParameters = new TokenValidationParameters() {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
            ValidateIssuer = false,
            ValidateAudience = false
        };
    });
}
```

*Figura 4 - Ficheiro Startup.cs*

Feitas todas as implementações, partiu-se para o teste no Postman. A imagem seguinte mostra-nos a autenticação via JWT.

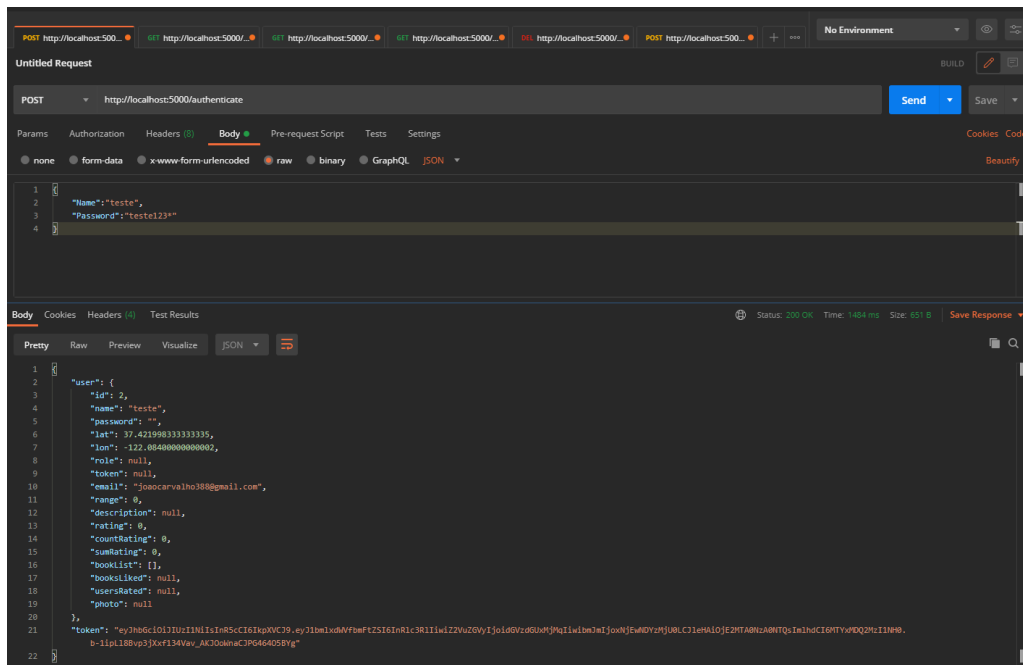
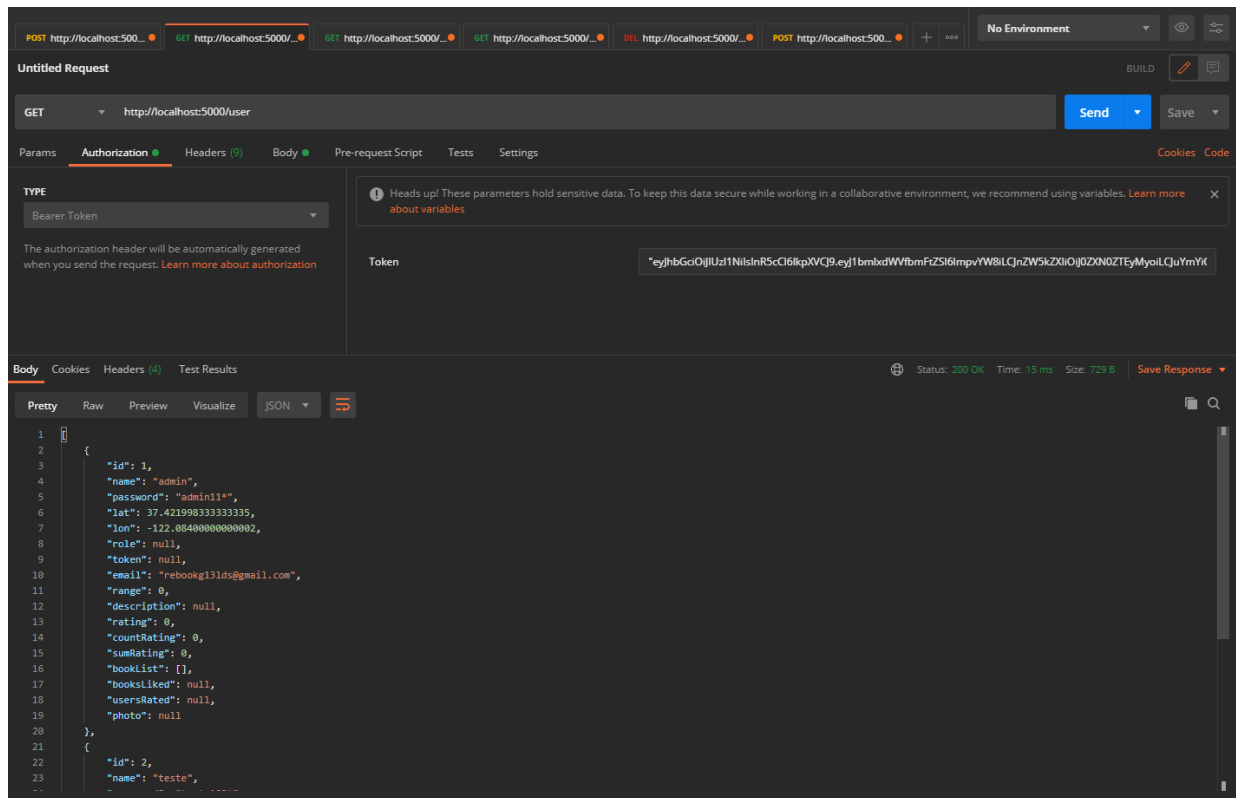


Figura 5 - Autenticação via JWT

## Relatório do Projeto

Após tudo isto, possuímos um token autenticado que nos dá acesso a várias rotas da aplicação que antes não tinha. Para isso é necessário informar esse token no header após a palavra Bearer, como é possível verificar na próxima imagem.



*Figura 6 - Authorization Bearer*

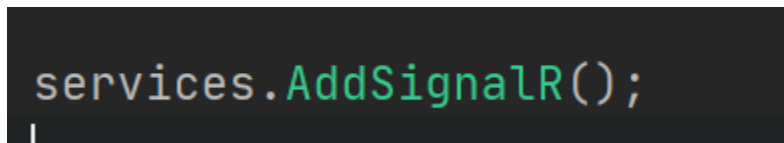
## 2.5 Implementação do chat em SignalR (Backend)

O Azure SignalR simplifica o processo e permite interatividade em tempo real das aplicações através de HTTP. Esta funcionalidade, permite que o serviço envie atualizações de conteúdo a clientes ligados, como uma única página Web ou aplicação móvel. Deste modo, os clientes são atualizados sem que tenham de consultar o servidor ou submeter novos pedidos HTTP para as atualizações, ou seja, podemos trabalhar praticamente com uma conexão sempre aberta e dessa maneira podemos detetar no servidor quando um novo cliente se conectou ou desconectou.

Através da biblioteca SignalR, é possível simplificar a criação e gestão de conexões persistentes entre cliente e servidor. Por consequência, facilita o desenvolvimento de aplicações que podem mostrar atualizações de dados mantidos no servidor em tempo real. (SignalR, 2021)

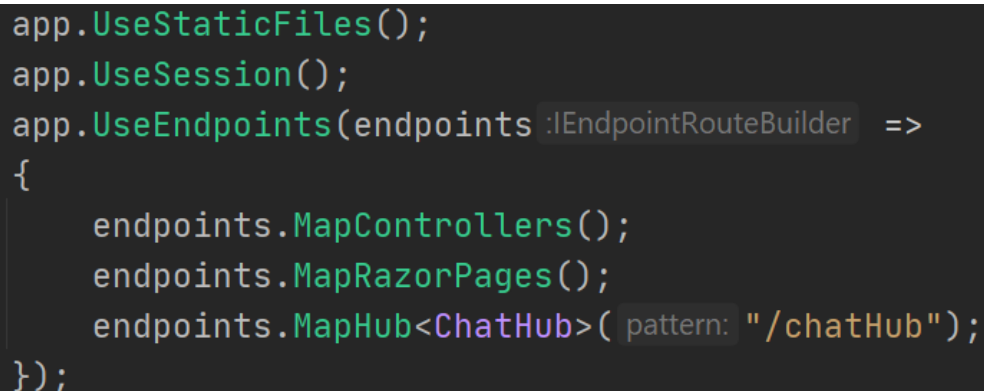
Portanto, o objetivo principal passa por implementar um chat usando a biblioteca SignalR para o efeito. Para criar a conexão utilizamos a classe Hub pois permite transmitir vários tipos de mensagens entre cliente/servidor.

Para a correta implementação do SignalR, necessitamos de atualizar o método `ConfigureServices` do ficheiro `Startup.cs`

A screenshot of a code editor showing the line `services.AddSignalR();` in a dark-themed environment. The text is in a monospaced font with syntax highlighting: `services` is in light blue, `.AddSignalR()` is in green, and the semicolon is in light blue.

*Figura 7 - Adicionar o SignalR*

Por sua vez, também foi necessário atualizar o método `Configure` como podemos verificar na próxima figura.

A screenshot of a code editor showing the `Configure` method in a dark-themed environment. The code is as follows:  

```
app.UseStaticFiles();  
app.UseSession();  
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapControllers();  
    endpoints.MapRazorPages();  
    endpoints.MapHub<ChatHub>(pattern: "/chatHub");  
});
```

  
The text is in a monospaced font with syntax highlighting: `app` is in light blue, `.UseStaticFiles()`, `.UseSession()`, and `.MapControllers()` are in green, `.MapRazorPages()` is in light blue, `.MapHub<ChatHub>` is in light blue, `(pattern: "/chatHub")` is in green, and `});` is in light blue.

*Figura 8 - Atualização do método Configure*

## Relatório do Projeto

Vários models foram criados para a implementação do chat em SignalR, como por exemplo, o chatConnections (model criado para representar as conexões), o chat(model que representa um chat), o chatInfo(model que armazena algumas informações dos utilizadores), o chatUsers(model que visa associar os utilizadores ao chat criado) e também tivemos que alterar o model das messages.

```
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Id { get; set; }

[Required] public int ChatId { get; set; }
[Required] public string Text { get; set; }

[Required] public DateTime Time { get; set; }

[Required] public string UserName { get; set; }

[ForeignKey("SenderId")]
public User User { get; set; }
public int SenderId { get; set; }
}
```

Figura 9 - Model que representa as mensagens

Posteriormente, criou-se a classe ClaimService e também o método que permite obter o id do utilizador através do token gerado.

```
public static int? GetIdFromClaimIdentity(ClaimsIdentity claimsIdentity)
{
    foreach (var claim in claimsIdentity.Claims)
    {
        if (claim.Type.Equals(ClaimTypes.SerialNumber))
        {
            return int.Parse(claim.Value);
        }
    }

    return null;
}
```

Figura 10 - Método GetIdFromClaimIdentity

## Relatório do Projeto

Após isso, criou-se a classe ChatHub.cs com os métodos SendMessage(permite enviar uma mensagem), OnConnectedAsync(cria a conexão para a transmissão de mensagens entre vários utilizadores) e o método OnDisconnectedAsync(visa remover a conexão criada).

```
public override async Task OnDisconnectedAsync(Exception ex)
{
    try
    {
        await Clients.All.SendAsync("UserDisconnected", Context.ConnectionId);
        int? userId = ClaimService.GetIdFromClaimIdentity((ClaimsIdentity)Context.User.Identity);
        if (userId != null) mRepo.RemoveUserHubConnection(Context.ConnectionId);
        await base.OnDisconnectedAsync(ex);
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}
```

Figura 11 - Método onDisconnectedAsync

As classes MessageRepo.cs e MessageController.cs foram completamente reformuladas pois foram implementados vários novos métodos para assim ser possível suportar o chat em tempo real através de SignalR, como podemos verificar nas próximas duas figuras.

```
public ActionResult<List<ChatInfo>> GetChats()
{
    try
    {
        int? loggedUser = ClaimService.GetIdFromClaimIdentity((ClaimsIdentity)this.ControllerContext.HttpContext.User.Identity);
        if (loggedUser == null)
        {
            return Unauthorized("Utilizador não autorizado ou inexistente!");
        }

        List<ChatInfo> chats = _messageRepo.GetChats((int) loggedUser);
        return Ok(chats);
    }
    catch (Exception e)
    {
        return BadRequest(e.Message);
    }
}
```

Figura 12 - Método GetChats do controller Messages

## ***Relatório do Projeto***

```
public List<Message> GetMessages(int chatId)
{
    List<Message> messages = _db.Messages.FromSqlRaw($"SELECT * FROM rebookdb.message WHERE ChatId = {chatId}")
        .ToList();

    if (messages.Count == 0 || messages == null)
    {
        throw new DatabaseIsEmptyException("No Messages in database");
    }

    messages.OrderByDescending(message => message.Time);

    return messages;
}
```

*Figura 13 - Método GetMessages da classe MessageRepo.cs*

## 2.6 Implementação do chat em SignalR (Frontend)

Do lado do cliente, criou-se alguns models para a correta implementação do SignalR. O model de chat foi adicionado e também optou-se por alterar o model das mensagens que já existia anteriormente, como é possível ver na próxima figura.

```
public class Message : ObservableObject
{

    public string Text { get; set; }

    public int SenderId { get; set; }

    public string userName { get; set; }
}
```

Figura 14 - Model Message (Frontend)

A lógica de match da nossa aplicação, foi forçosamente alterada devido às mudanças recentes na equipa de trabalho. Portanto, um utilizador quando clica no botão de de dar like num livro é criada uma hubConnection de modo a abrir um chat com o dono desse mesmo livro, como podemos observar na próxima figura.

```
private async void LikeButton_OnClicked(object sender, EventArgs e)
{
    Liked();
    await _hubConnection.StartAsync();
    await _hubConnection.InvokeAsync("SendMessage", arg1: Cards[_index].OwnerId, arg2: _id, arg3: "You have a new match!",
    await _hubConnection.StopAsync();
    _index++;
    CardStackView.Swipe(SwipeDirection.Right, animationLength: 500);
}
```

Figura 15 - Criação da HubConnection

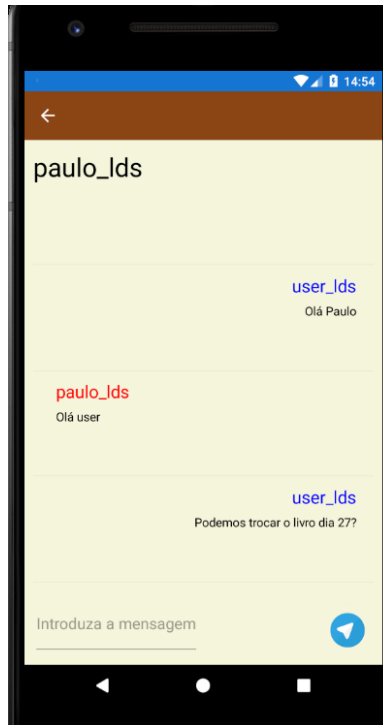
Por sua vez, quando o utilizador clica no botão de voltar para trás, é necessário parar a hubConnection, como demonstra a próxima figura.

```
protected override bool OnBackPressed()
{
    _hubConnection.StopAsync();
    return base.OnBackPressed();
}
```

Figura 16 - hubConnection.StopAsync

## ***Relatório do Projeto***

Como podemos verificar, na próxima figura conseguimos implementar um chat através da biblioteca SignalR.



*Figura 17 - Chat Frontend SignalR*



## 2.7 Ferramentas e Metodologias utilizadas

### 2.7.1 JetBrains Rider

O JetBrains Rider foi o ambiente de desenvolvimento (IDE) eleito para a elaboração do software. Na próxima figura, podemos observar o IDE e também um trecho de código produzido para este projeto. (JetBrains, 2020)

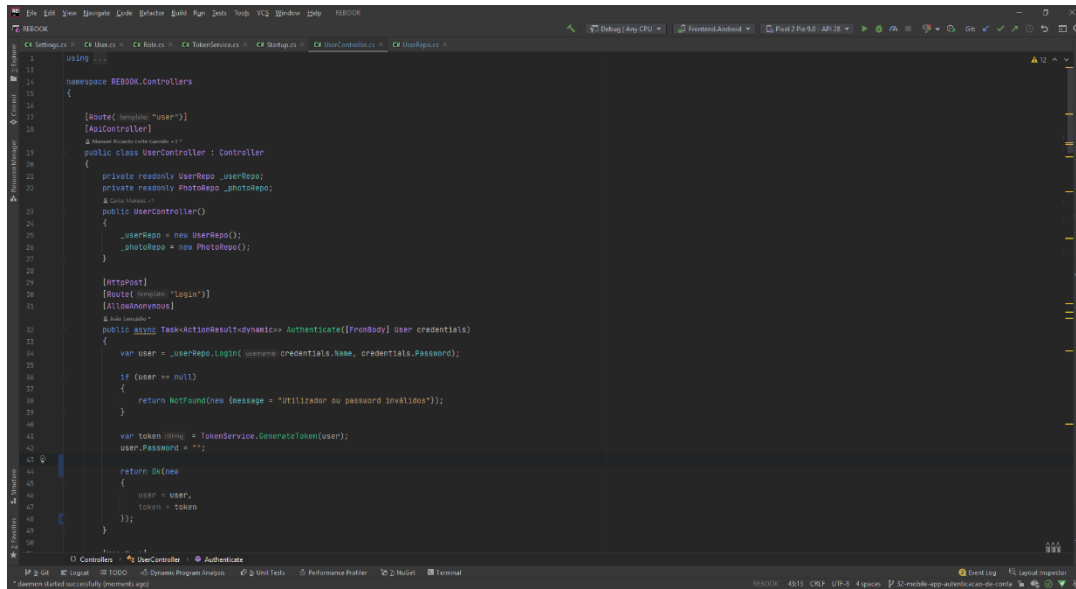


Figura 18 - JetBrains Rider (IDE)

### 2.7.2 Framework ASP.NET Core

ASP.NET Core é um framework de desenvolvimento web com vista ao desenvolvimento de aplicações web. Foi desenvolvido pela Microsoft, e por ser leve, rápido, modular e funcionar em conjunto com .NET Core tornou-se num framework popular entre a comunidade.

Por ser uma plataforma totalmente modular, possui também vários recursos que podem ser adicionados através de pacotes Nuget, o que permite à aplicação ser mais performática. Para além de tudo isto, o ASP.NET Core não necessita de um IDE específico, o que permite uma fácil portabilidade sem correr o risco de perder algum recurso ou funcionalidade. (Microsoft, 2020)

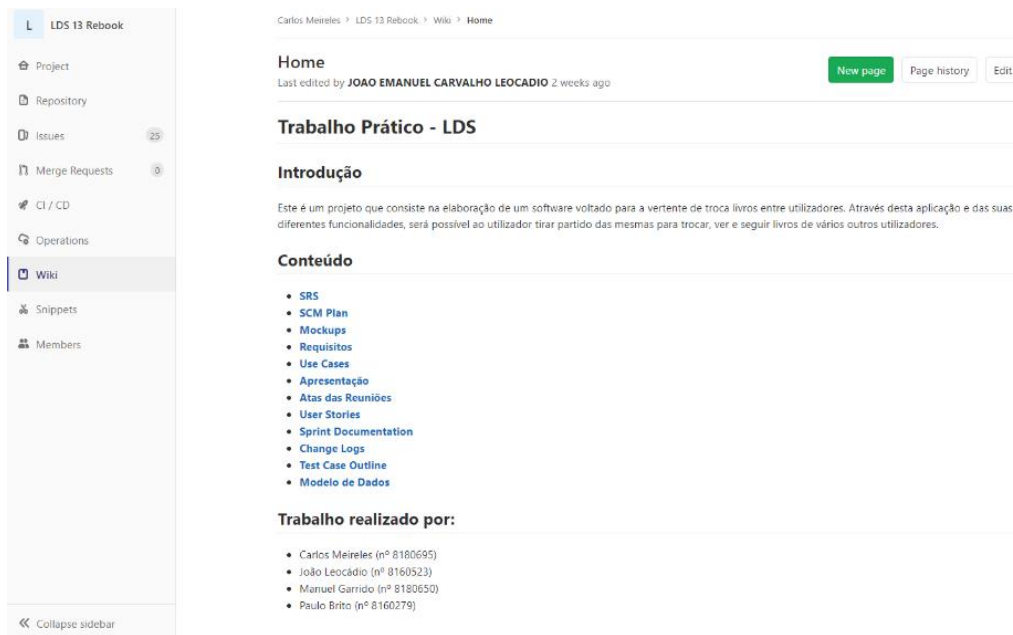


Figura 19 - ASP.NET Core

### 2.7.3 Gitlab

O GitLab é um repositório de software baseado em git, com suporte a Wiki, gestor de tarefas e possui CI (Continuous Integration) e também CD (Continuous Delivery). Deste modo, o GitLab é utilizado para controlo de versões.

No GitLab está presente uma wiki com a documentação do projeto, e será utilizado como repositório local no computador de cada Dev (desenvolvedor), permitindo a submissão das alterações quando desejado e trabalho independentemente do repositório online. A figura , mostra o ambiente do Gitlab mais propriamente a Wiki do projeto. (GitLab, 2020) (YouTube, 2020)



*Figura 20 - GitLab ESTG*

## 2.7.4 Documentação Swagger

Ter uma boa documentação da REST API é primordial para os desenvolvedores e testers compreenderem claramente o comportamento dela, pois são estas as responsáveis por desempenhar um papel fundamental na comunicação entre as aplicações.

O conceito de REST aplica-se ao desenvolvimento de aplicações que comunicam através de um serviço ou servidor por via WEB. Deste modo, REST API usa o protocolo HTTP (*HyperText Transfer Protocol*) utilizando os métodos POST(inserir), DELETE(Apagar), PUT(Editar) dados no servidor ou serviços.

Em suma, o Swagger foi a framework utilizada para a documentação da API do projeto. Assim, é possível vislumbrar de forma mais fácil a descrição, consumos e visualização de serviços da nossa REST API, como se pode observar na próxima figura.

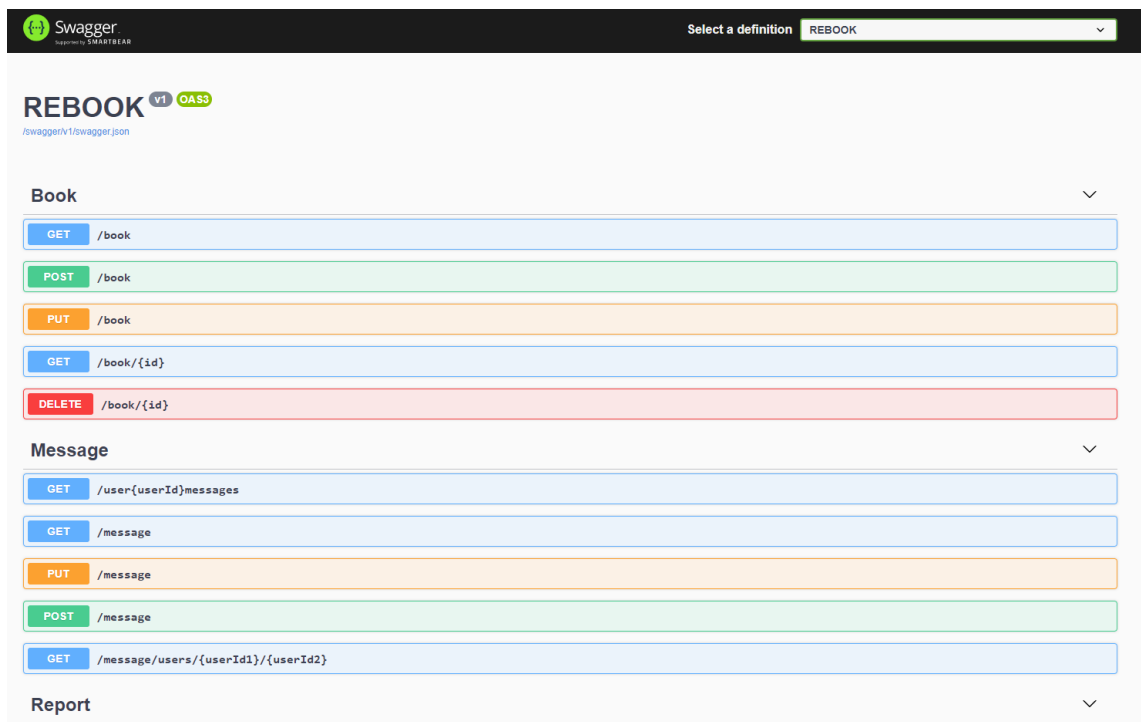


Figura 21 - Documentação Swagger

### **2.7.5 Xamarin (Frontend)**

Xamarin.Forms permite desenvolver e criar aplicações em Android, IOS e Windows a partir de uma única base de código partilhada, foi exatamente essa grande vantagem que nos levou a optar por Xamarin.

Deste modo, xamarin permite aos desenvolvedores criar interfaces de utilizador em XAML (C#). Essas interfaces são renderizadas como controlos nativos de desempenho em cada plataforma. (JetBrains, 2020)(Microsoft, 2020)



*Figura 22 - Xamarin logo*

### 3. Arquitetura do Projeto

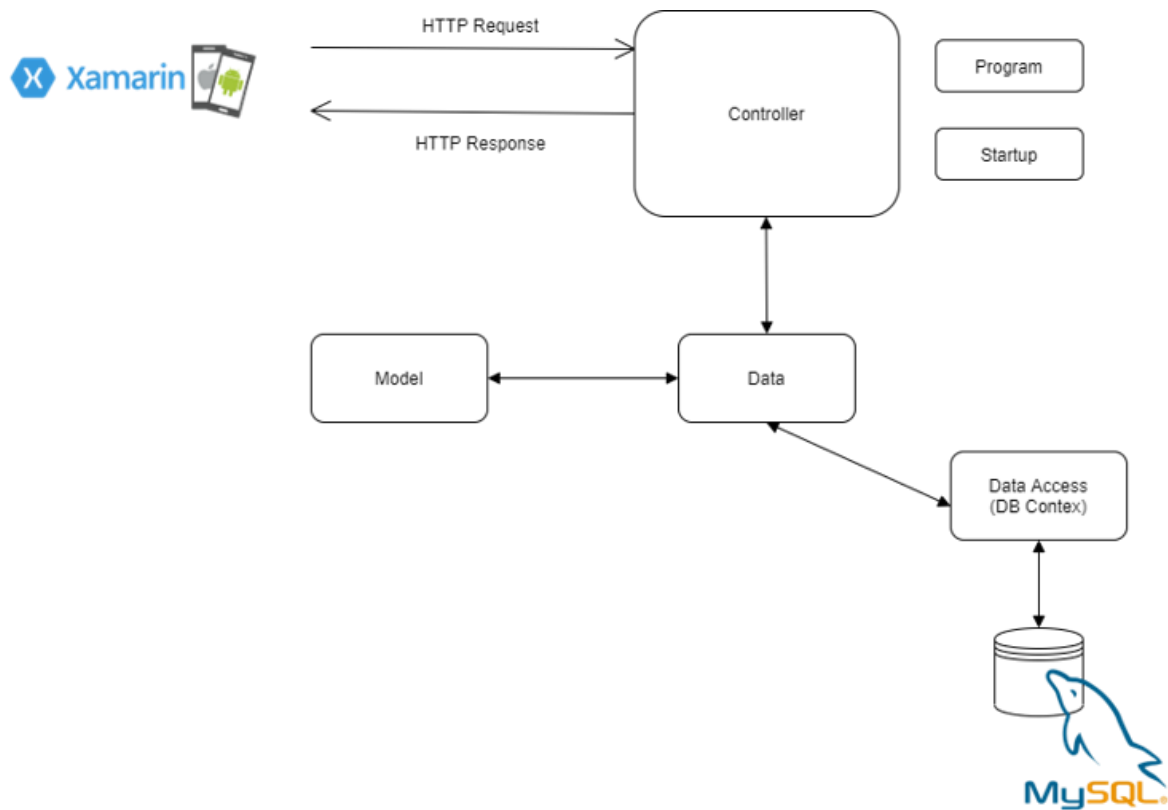


Figura 23 - Arquitetura do Projeto

Nesta imagem, é possível ver o desenho da arquitetura para a aplicação desenvolvida. No componente Data é onde se encontram as funções desenvolvidas para ler e escrever da base de dados. Os controllers regulam a comunicação com o frontend executando as funções que estão presentes no data dependendo dos HTTP requests. Por sua vez, no componente model é onde definimos as entidades associadas ao projeto. Por último, o frontend (Xamarin) consumirá todos os pedidos feitos ao servidor.

## 4. Diagrama de Classes

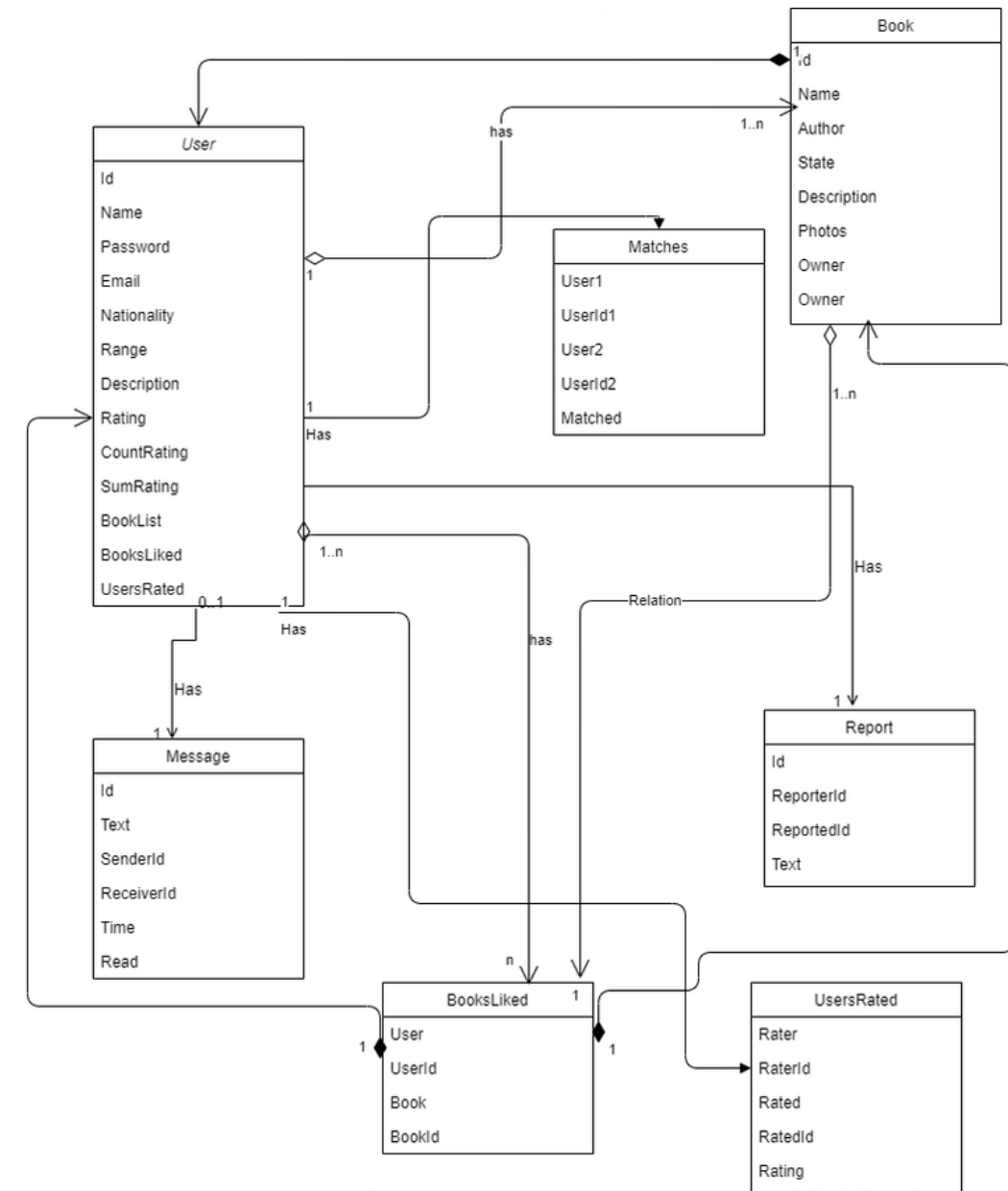
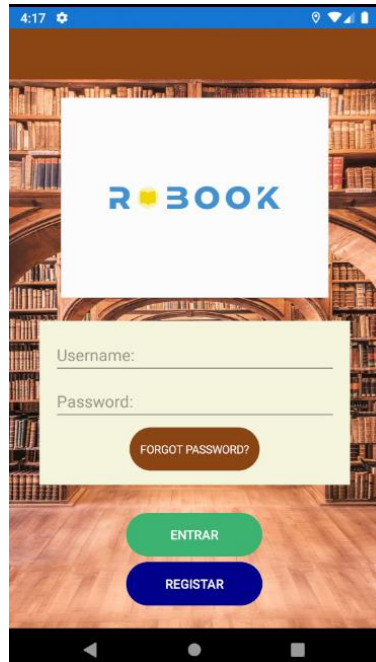


Figura 24 - Diagrama de Classes

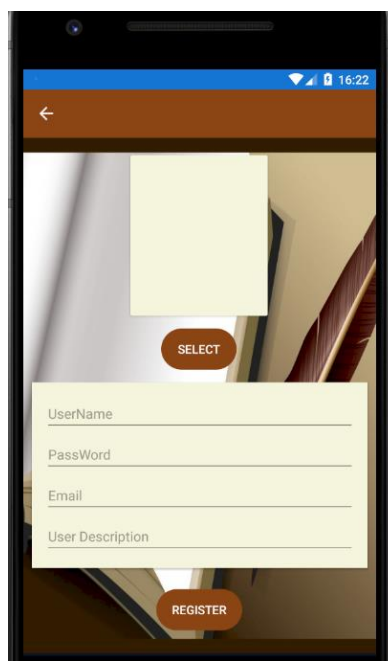
## 5. A aplicação do ponto de vista do Utilizador

A primeira interação com a aplicação, pode ser vislumbrada na figura seguinte. Onde é possível ao Utilizador criar conta (caso ainda não tenha) ou então fazer login.



*Figura 25 - LoginPage reBOOK*

Se não possuir uma conta, o Utilizador deverá criar utilizando a view que se segue na imagem.



*Figura 26 - RegistrationPage reBOOK*

## Relatório do Projeto

Depois de registrar conta com sucesso, o Utilizador receberá um email de boas vindas juntamente com as suas credenciais de acesso.



Figura 27 - Email com credenciais de acesso

Caso, o Utilizador se tenha esquecido da password poderá facilmente recuperá-la através da próxima imagem.

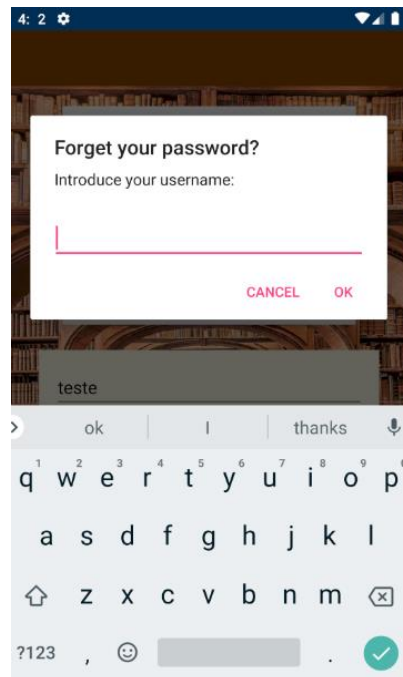
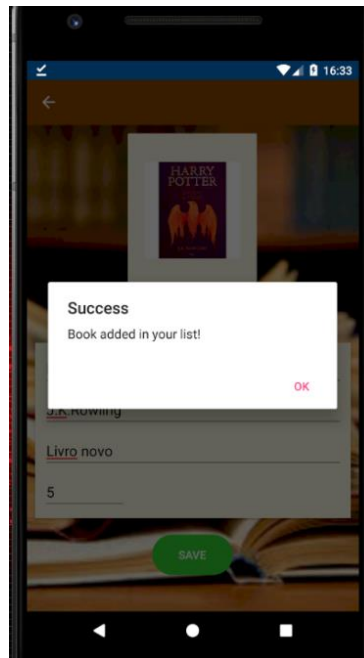


Figura 28 - Recuperar password



## *Relatório do Projeto*

Após registo, um utilizador pode adicionar à sua lista os livros que tem para troca, cada livro deve ter nome, autor e uma avaliação do seu estado (1 para muito mau estado, 5 para novo), assim como uma foto do mesmo.



*Figura 29 - Adicionar Livro*

De seguida, o Utilizador conseguirá ver a sua lista de livros, como demonstra a próxima imagem.

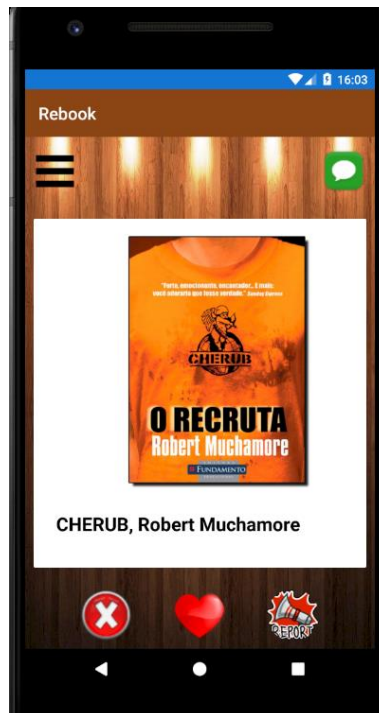


*Figura 30 - Lista de livros*

## *Relatório do Projeto*

Na view da figura 30, o Utilizador tem o poder de fazer o crud dos seus livros, ou seja, é possível adicionar mais livros, apagar, editar e ver mais informação de um livro à sua escolha.

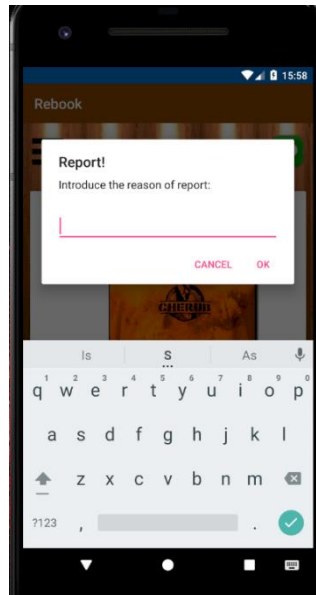
Por sua vez, e na página principal da aplicação o Utilizador conseguirá ver vários livros e adicioná-los aos favoritos, ou simplesmente passa-los, como é demonstrado na figura seguinte. Caso adicione aos favoritos, será criado automaticamente um chat com o proprietário desse mesmo livro.



*Figura 31 - Página principal*

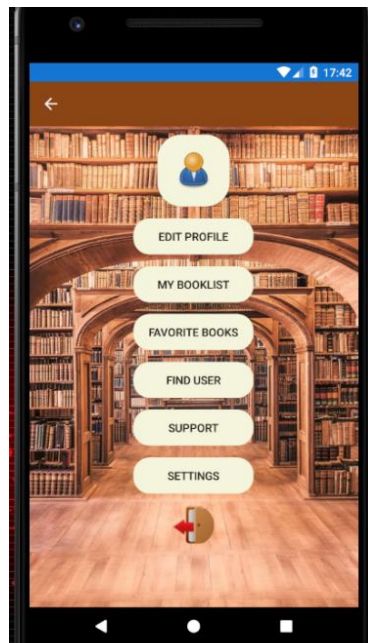
## *Relatório do Projeto*

A funcionalidade de reportar um livro de um determinado Utilizador, também está presente utilizando o respetivo botão. Dessa maneira, é possível reportar um livro como se pode verificar na próxima figura.



*Figura 32 - PopUp Report*

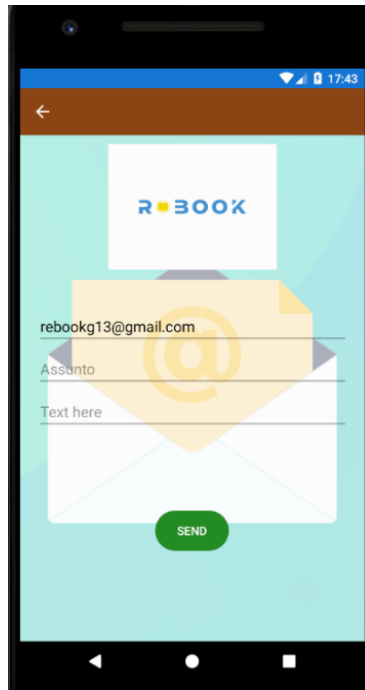
Na imagem seguinte, Podemos observar a dashboard da nossa aplicação. Funciona como se fosse um menu e permite ao Utilizador fazer a ação que mais desejar.



*Figura 33 - Dashboard da app*

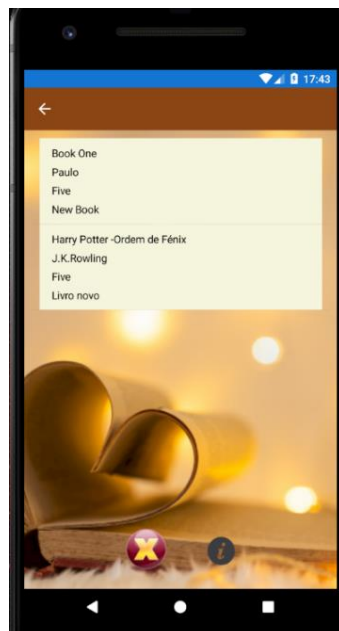
## *Relatório do Projeto*

O Utilizador também consegue contactar diretamente a equipa de suporte através de email, como demonstra a próxima figura.



*Figura 34 – Contactar Suporte*

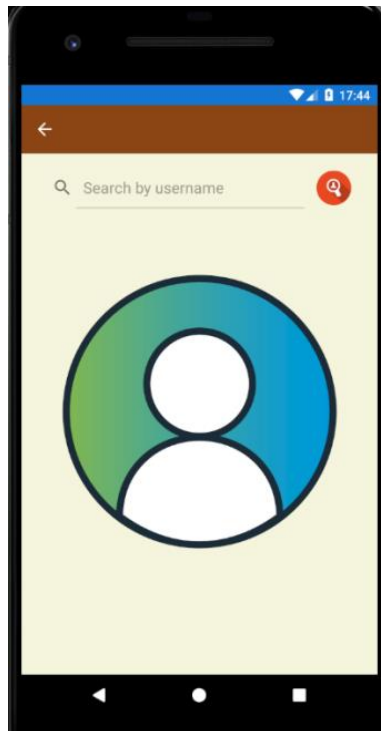
A próxima figura mostra a lista de favoritos com os respetivos livros adicionados pelo Utilizador, onde é possível ver mais informação à cerca do livro ou então elimina-lo da lista.



*Figura 35 - Lista de Favoritos*

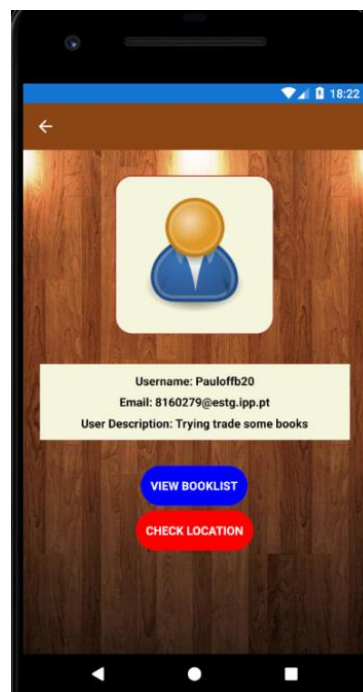
## *Relatório do Projeto*

O Utilizador, também consegue pesquisar outros utilizadores, como Podemos ver na próxima figura.



*Figura 36 - Procurar utilizador*

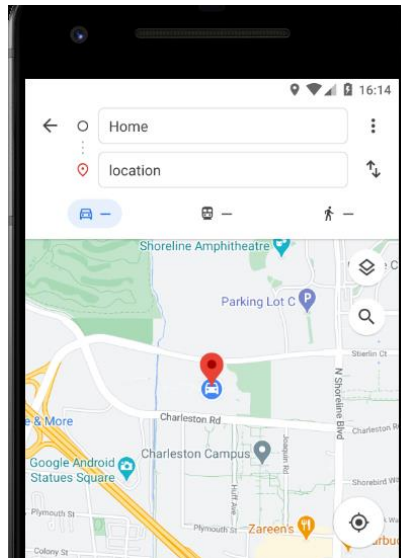
Posteriormente e após a pesquisa, o Utilizador conseguirá ver a localização do Utilizador pesquisado e também a lista de livros disponíveis para troca.



*Figura 37 - Pesquisar utilizador*

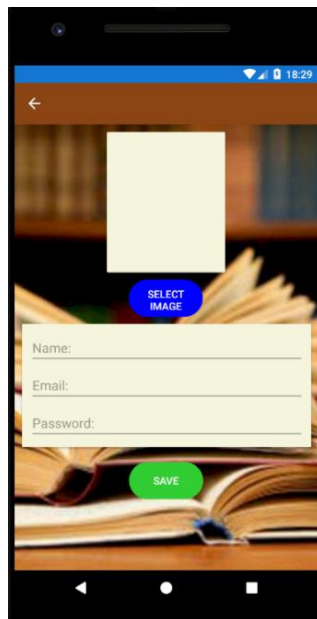
## *Relatório do Projeto*

Deste modo, é possível ver a localização do Utilizador através do google Maps, como Podemos ver na próxima figura.



*Figura 38 - Localização*

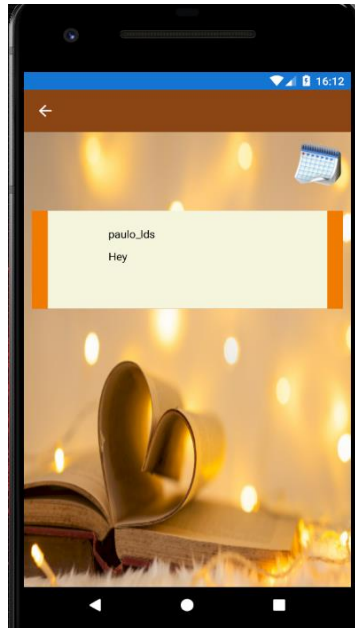
Por outro lado, o Utilizador também consegue alterar as informações do seu perfil, como é possível observar na próxima figura.



*Figura 39 - Editar perfil*

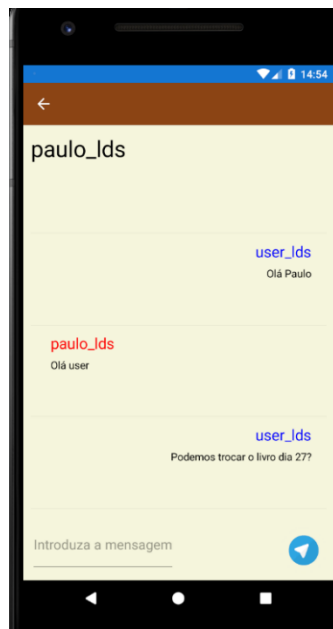
## *Relatório do Projeto*

O Utilizador quando carrega no botão das mensagens, conseguirá ver uma lista de matchs, ou seja, quando um livro é adicionado aos favoritos é criado automaticamente um chat, como mostra na figura 40.



*Figura 40 - Lista de Matchs*

A troca de mensagens entre utilizadores, não está esteticamente muito bem implementada devido à falta de tempo e de problemas que surgiram no seio do grupo mas ainda assim conseguimos melhorar no sexto sprint o Sistema de chat utilizando a biblioteca SignalR. Podemos ver o nosso Sistema de troca de mensagens na próxima figura.



*Figura 41 - Sistema de troca de mensagens*

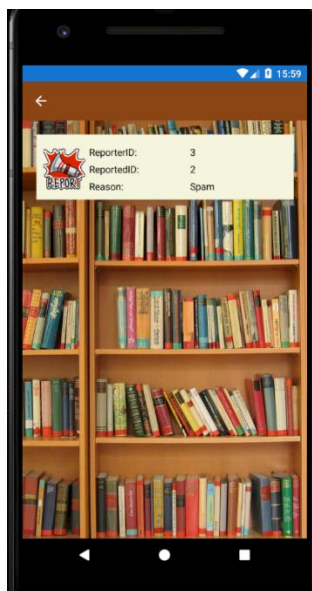
## 6. A aplicação do ponto de vista do Admin

O admin, é o utilizador que tem como objetivo principal gerir e dar suporte à nossa aplicação e na figura seguinte é possível vislumbrar e notar algumas diferenças entre o seu dashboard e de um Utilizador normal.



*Figura 42 - Dashboard Admin*

Como era expectável, o admin consegue ver os livros reportados e tomar algumas ações consoante o tipo de report. A próxima figura mostra exatamente uma lista de todos os reports efetuados por utilizadores “normais”.



*Figura 43 - Lista de todos os reports*



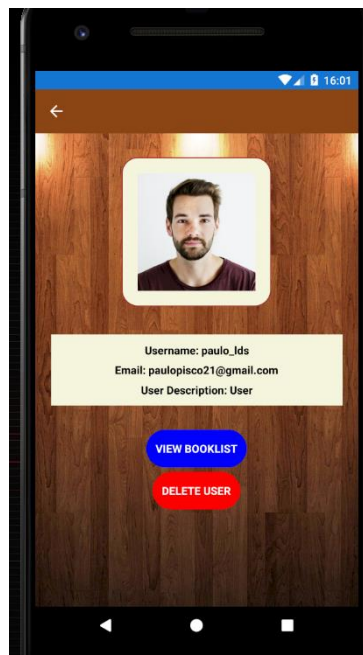
## *Relatório do Projeto*

O admin para conseguir analisar melhor o report e acompanhar melhor a situação, poderá pesquisar reports por Id, como podemos observar na próxima figura, e assim ver mais informação.



*Figura 44 - Procurar Reports por Id*

Após analisar corretamente a situação, o Admin tem o poder de alertar o Utilizador que foi reportado ou então apagar a conta do Utilizador em caso extremo.



*Figura 45 - Ações Admin*

## Relatório do Projeto

Outra das ações do Admin, passa por poder apagar um livro quando este não corresponde aos interesses da aplicação e de outros utilizadores. A próxima figura, mostra a swipe card view do lado do Admin e também o botão que permite apagar o livro.



Figura 466 – SwipeCardView Admin

Caso um livro seja eliminado pelo admin, o Utilizador proprietário será alertado via email, como se pode verificar na próxima figura.

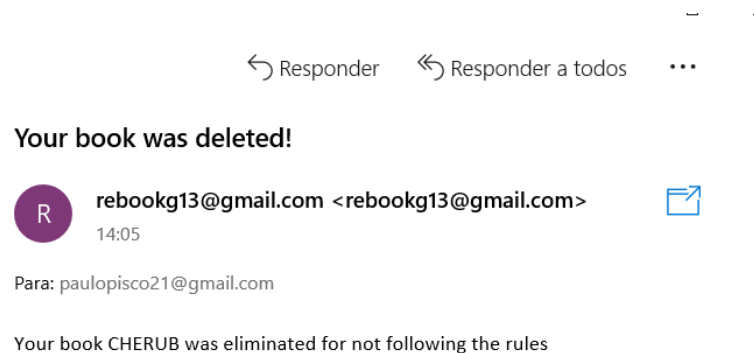


Figura 477 – Alerta por email

## **7. Conclusão e limitações**

Este projeto, teve como principal objetivo explorar os conceitos lecionados ao longo do curso como metodologias de testes, metodologias de desenvolvimento ágeis, Continuous Integration (CI) e planos de Software Configuration Management (SCM).

Assim e resumindo, foi elaborado uma aplicação mobile que funcionará na plataforma Android e permitirá a troca de livros entre vários utilizadores, ou seja, o utilizador consegue ver vários livros e poderá dar “like” e assim adicionar livros aos favoritos.

Deste modo e através do desenvolvimento desta aplicação, pode-se afirmar que expandimos as nossas respetivas competências de análise de um problema, identificação de requisitos, planear e implementar um plano de testes. Utilizar metodologias de desenvolvimento ágil como o SCRUM no desenvolvimento de software, usar ferramentas de suporte (ASP NET core, Swagger, Xunit, Gitlab) e perceber a importância de trabalhar em equipa assim como manter um ambiente saudável entre todos.

Neste contexto é de realçar todo o conhecimento transmitido pelos docentes da unidade curricular, que grande importância terá no futuro e que tão rico tornou o presente.

No que diz respeito às limitações sentidas ao longo de todo o percurso de desenvolvimento deste projeto, pode-se referir que foram algumas embora não impediram o bom desenvolvimento do mesmo.

A primeira limitação surgiu logo no início, que ficou marcado por algum receio, por desconhecimento dos métodos usados no trabalho, tais como, dotnet e ef core. Uma vez que nunca haviam sido utilizados outrora. Outra limitação, surgiu com o encurtamento do grupo devido à desistência de elementos que está diretamente ligado a mudanças no projeto e afetou claramente o planeamento do mesmo.

Na elaboração deste relatório, a principal limitação sentida prendeu-se com a gestão do tempo disponível para o mesmo.

Outras limitações ocorreram durante o decorrer do desenvolvimento do projeto, mas que rapidamente foram ultrapassadas graças ao espírito de equipa e companheirismo que é comum neste grupo de trabalho.

## **8. Bibliografia**

(14 de 01 de 2021). Obtido de SignalR: <https://docs.microsoft.com/pt-pt/azure/azure-signalr/signalr-overview>

GitLab. (01 de 11 de 2020). *ESTG*. Obtido de GitLab ESTG: <https://gitlab.estg.ipp.pt/8180695/lds-13-rebook/wikis/home>

JetBrains. (11 de 11 de 2020). *JetBrains*. Obtido de JetBrains: <https://www.jetbrains.com/>

Microsoft. (01 de 12 de 2020). *Microsoft*. Obtido de Microsoft: <https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms>

YouTube. (06 de 11 de 2020). *YouTube*. Obtido de YouTube: <https://www.youtube.com/>