Relatório

Comunicação de Dados

Alunos
João Apresentação (21152)
Pedro Simões (21140)
Augusto Pereira (21136)

Docente

Miguel Lopes

Licenciatura Engenharia Sistemas Informáticos

Barcelos, junho de 2022

1. Indice

2. Índice de ilustrações

Não foi encontrada nenhuma entrada do índice de ilustrações.

3. Resumo

Em suma, o trabalho prático da cadeira de Comunicação de Dados teve como propósito a montagem de uma API capaz de correr serviços web com determinadas funcionalidades. Era pretendido a implementação para o problema do escalonamento FJSSP (flexible job shop problem).

De uma forma mais descritiva, a existência de várias simulações, constituídas por Jobs, que por sua vez constituídos por operações, e cada operação será desempenhada por uma máquina com uma determinada duração. O escalonamento tem por objetivo encontrar um plano de produção mais eficiente ao nível de duração.

Esta aplicação oferece de uma forma geral os seguintes serviços:

- Gestão de Simulações;
- Tabela de Produção;
- Plano de Produção;
    - Automatizado;
    - Manual;
- Gestão de Utilizadores;

Este programa oferece a possibilidade de construção de simulações, á qual poderá se fazer download, simular manualmente e automaticamente, além de muitas outras funcionalidades.

Este terá duas categorias de permissões:

- Funcionário;
- Administrador;

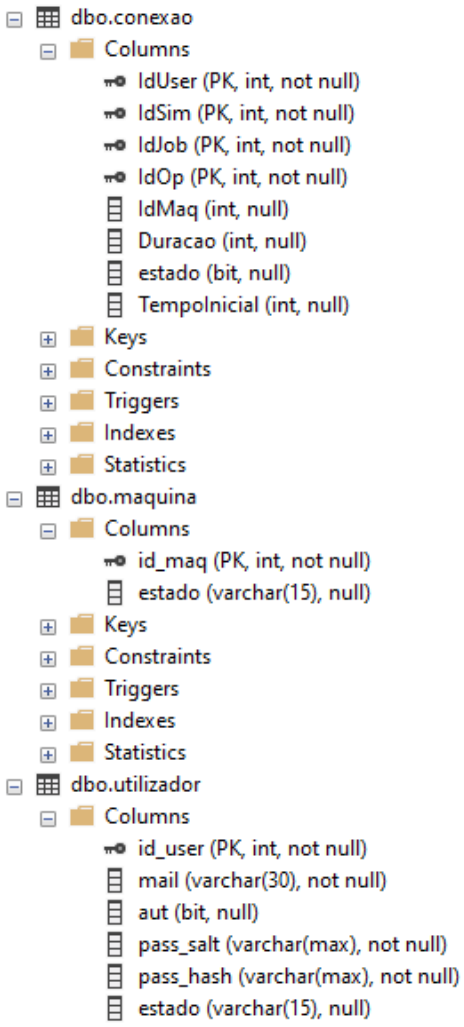
O Administrador tem todas as funcionalidades iguais ás do Funcionário, mas este ainda pode gerir os funcionários e as máquinas.

Para a resolução da mesma recorreu-se ao uso:

- linguagem C# para o Back-End realizado no Visual Studio 19;
- Html, Css e TypeScript para o Front-End elaborado em Visual Studio Code;
- Base de dados em Microsoft SQL Server.
- Testes de request realizados em Postman;

4. Base de dados

Para construção da base de dados, tal como o referido acima, construído em Microsoft SQL Server levou a seguinte estrutura:



Dentre as 3 tabelas existe a do utilizador e da máquina que são independentes da existência de dados na tabela das conexões. A tabela das conexões representa cada vez que um utilizador cria uma simulação com respetivas associações. Esta tabela será usada para a grande maioria das funcionalidades e apresentações.

5. Back-End

Para a criação do Back-End realizado em Visual Studio 19, foi criado uma ligação á base de dados através de

Este contem Controllers e os Models.

5.1. Models

Os models são criados através da linkagem com a base de dados e contem todos os atributos e respetiva tabela:

5.2. Controllers

Os Controllers são criados para se poder realizar os requests e poderem ser executados para o Front-End. Desta forma seguem-se os Controllers criados e suas respetivas funcionalidades:

5.2.1. Utilizador

A função que se segue, realiza um Get para devolder a lista dos utilizadores através da base de dados, e esta só será autorizada para um user que tenha token de administrador.

```
[HttpGet, Authorize(Roles = "Admin")]
0 references
public IEnumerable<Utilizador> Get()
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            List<Utilizador> utilizadores = context.Utilizador.Where(u => u.Estado == "Ativo").ToList();
            HidePassWord(utilizadores);

            return utilizadores;
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return null;
    }
}
```

Esta ainda usa uma função auxiliar para esconder passwords:

```
public void HidePassWord(Utilizador utilizador)
{
    try
    {
        utilizador.PassHash = "Hidden";
        utilizador.PassSalt = "Hidden";
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}

/// <summary> Método que esconde as palavras pass dos utilizadores todos
1 reference
public void HidePassWord(List<Utilizador> utilizadores)
{
    try
    {
        foreach (Utilizador utilizador in utilizadores)
        {
            utilizador.PassHash = "Hidden";
            utilizador.PassSalt = "Hidden";
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}
```

Temos uma funcionalidade Get, para buscar um utilizador através da token apanhada no Front-End através do login realizado. Este verifica através do seu mail e este é retornado.

```
[HttpGet("getuserbytoken"), Authorize(Roles = "Utilizador, Admin")]
0 references
public IActionResult GetUserByToken()
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            string MailUser = User.FindFirstValue(ClaimTypes.Email);
            Utilizador user = context.Utilizador.Where(u => u.Mail == MailUser).FirstOrDefault();
            if (user == null) return BadRequest();
            return new JsonResult(user);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }

}
```

Para quando se necessitar dos dados de um utilizador em especifico, é retornado através de um Get que leva por parâmetro o id do user em questão.

```
[HttpGet("{id}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Get(int id)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                Utilizador utilizador = context.Utilizador.Where(u => u.IdUser == id && u.Estado != "Inativo").FirstOrDefault();

                if (utilizador == null) return BadRequest();

                if (UserMail != utilizador.Mail)
                {
                    return Forbid();
                }
            }

            Utilizador user = context.Utilizador.Where(u => u.IdUser == id).FirstOrDefault();
            HidePassWord(user);

            return new JsonResult(user);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return null;
    }
}
```

Para o login do utilizador é feito um Post para autenticar este através das credenciais inseridas. Este verifica o utilizador, e no caso deste ser verificado, é gerada a token de sessão dependendo da sua rule.

```
[Route("Login")]
[HttpPost]
0 references
public IActionResult Login(Utilizador utilizador)
{
    using (var context = new EscalonamentoContext())
    {
        try
        {
            login login = new login();
            Utilizador user = context.Utilizador.FirstOrDefault(aux => aux.Mail == utilizador.Mail);

            VerifyAccount(utilizador);

            string token;

            if (user.Aut == false) token = CreateTokenUser(utilizador);
            else token = CreateTokenAdmin(utilizador);

            login.token = token;
            if (user.Aut == false) login.role = "Utilizador";
            else login.role = "Admin";

            return new JsonResult(login);
        }
        catch (ArgumentException ae)
        {
            Console.WriteLine(ae);
            return new JsonResult(ae.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            return BadRequest();
        }
    }
}
```

A funções auxiliares para a token e verificação da conta são as seguintes:

```
public static bool VerifyAccount(Utilizador utilizador)
{
    using (var context = new EscalonamentoContext())
    {
        Utilizador user = context.Utilizador.FirstOrDefault(aux => aux.Mail == utilizador.Mail && aux.Estado != "Inativo");

        if (user == null)
        {
            throw new ArgumentException("Utilizador não existe!", "account");
        }
        else
        {
            if (!HashSaltPW.VerifyPasswordHash(utilizador.PassHash, Convert.FromBase64String(user.PassHash), Convert.FromBase64String(user.PassSalt)))
            {
                throw new ArgumentException("Password Errada.", "account");
            }
        }

        return true;
    }
}
```

```
private string CreateTokenAdmin(Utilizador utilizador)
{
    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.Email, utilizador.Mail),
        new Claim(ClaimTypes.Role, "Admin")
    };

    var key = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(_configuration.GetSection("AppSettings:Token").Value));

    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.Now.AddDays(1),
        signingCredentials: creds);

    var jwt = new JwtSecurityTokenHandler().WriteToken(token);

    return jwt;
}
```

```
2 references
private string CreateTokenUser(Utilizador utilizador)
{
    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.Email, utilizador.Mail),
        new Claim(ClaimTypes.Role, "Utilizador")
    };

    var key = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(_configuration.GetSection("AppSettings:Token").Value));

    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.Now.AddDays(1),
        signingCredentials: creds);

    var jwt = new JwtSecurityTokenHandler().WriteToken(token);

    return jwt;
}
```

Para o utilizador ser criado é usado um POST do objeto Utilizador passado por parâmetro. Todos os seus dados são armazenados, a sua password é criptografada, e uma rule é lhe atribuído. No fim o utilizador é armazenado na lista.

```
[HttpPost]
0 references
public IActionResult Post(Utilizador uti)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Utilizador utilizador = new Utilizador();

            utilizador.IdUser = uti.IdUser;
            utilizador.Mail = uti.Mail;

            HashSaltPW.CreatePasswordHash(uti.PassHash, out byte[] passwordHash, out byte[] passwordSalt);
            utilizador.PassHash = Convert.ToBase64String(passwordHash);
            utilizador.PassSalt = Convert.ToBase64String(passwordSalt);

            utilizador.Aut = uti.Aut;
            utilizador.Estado = "Ati";

            context.Utilizador.Add(utilizador);
            context.SaveChanges();

            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

No caso de os dados do utilizador serem alterados é realizado Patch. É enviado por parâmetro o id do utilizador e um objeto com os novos dados a serem alterados. Este recebe uma verificação de autorizações, os dados são alterados e atualizados.

```
[HttpPatch("{id}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Patch(int id, [FromBody] Utilizador uti)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                Utilizador user = context.Utilizador.Where(u => u.IdUser == id && u.Estado != "Inativo").FirstOrDefault();

                if (user == null) return BadRequest();

                if (UserMail != user.Mail)
                {
                    return Forbid();
                }
            }

            Utilizador utilizador = context.Utilizador.Where(u => u.IdUser == id).FirstOrDefault();

            utilizador.Mail = uti.Mail is null ? utilizador.Mail : uti.Mail;
            utilizador.Aut = uti.Aut is null ? utilizador.Aut : uti.Aut;
            utilizador.Estado = uti.Estado is null ? utilizador.Estado : uti.Estado;

            if (utilizador.Aut == false) CreateTokenUser(utilizador);
            else CreateTokenAdmin(utilizador);

            context.SaveChanges();

            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para o caso de se alterar a password, a função que se segue cumpre. Este recebe o id do utilizador e o objeto utilizador que contem a nova password. A nova password é criptografa e armazenada.

```
[Route("recoverpassword/{id}")]
[HttpPatch, Authorize(Roles = "Admin ,Utilizador")]
0 references
public IActionResult RecoverPassword(int id, Utilizador utilizador)
{
    using (var context = new EscalonamentoContext())
    {
        try
        {
            Utilizador user = context.Utilizador.Where(c => c.IdUser == id && c.Estado != "Inativo").FirstOrDefault();

            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string mailUtilizador = User.FindFirstValue(ClaimTypes.Email);

                if (user == null) return BadRequest();

                if (mailUtilizador != user.Mail)
                {
                    return Forbid();
                }
            }
            else if (User.HasClaim(ClaimTypes.Role, "Admin"))
            {
                if (user == null) return BadRequest();
            }


            HashSaltPW.CreatePasswordHash(utilizador.PassHash, out byte[] passwordHash, out byte[] passwordSalt);
            user.PassHash = Convert.ToBase64String(passwordHash);
            user.PassSalt = Convert.ToBase64String(passwordSalt);

            context.SaveChanges();
            return Ok();
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            return BadRequest();
        }
    }
}
```

Para do utilizador querer querer remover a sua conta, este terá o estado da sua conta alterado para “Inativo”, ou seja, este terá a sua conta arquivada. É feito uma verificação da sua autorização, pois este so poderá remover a sua própria conta.

```
[HttpDelete("delete/{id}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult ArquivarUtilizador(int id)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Utilizador user = context.Utilizador.Where(u => u.IdUser == id && u.Estado != "Inativo").FirstOrDefault();

            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                if (user == null) return BadRequest();

                if (UserMail != user.Mail)
                {
                    return Forbid();
                }
            }
            else if (User.HasClaim(ClaimTypes.Role, "Admin"))
            {
                if (user == null) return BadRequest();
            }

            user.Estado = "Inativo";

            List<Conexao> conns = context.Conexao.Where(c => c.IdUser == user.IdUser).ToList();

            foreach (Conexao con in conns)
            {
                con.Estado = false;
            }

            context.SaveChanges();

            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Outro delete existente é da remoção de um utilizador através do admin. Este terá autorização para remover os seus funcionários.

```
[HttpDelete("{id}"), Authorize(Roles = "Admin")]
0 references
public IActionResult Delete(int id)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Utilizador utilizador = context.Utilizador.Where(u => u.IdUser == id).FirstOrDefault();

            context.Utilizador.Remove(utilizador);

            context.SaveChanges();
            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

5.2.2. Máquina

De forma a conseguir listar as máquinas todos com estado “Ativo” é utilizado um Get:

```
[HttpGet, Authorize(Roles = "Admin, Utilizador")]
0 references
public IEnumerable<Maquina> Get()
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            return context.Maquina.Where(m => m.Estado != "Inativo").ToList();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return null;
    }
}
```

Para a listagem de uma máquina em especifico é passado por parâmetro o seu id:

```
[HttpGet("{id_maquina}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Get(int id_maquina)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            return new JsonResult(context.Maquina.Where(m => m.IdMaq == id_maquina && m.Estado != "Inativo").FirstOrDefault());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para a obtenção de uma máquina e sua duração em uma operação, de um job, de uma simulação é enviado um objeto do tipo Conexao que contem os ids do user, simulação, job e operação. Pesquisando depois na base de dados este retorna a conexão que contem a máquina e a duração a serem apresentadas ao utilizador.

```
[HttpPost("getmaquinabyjobop"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult GetMaquinaByJobOp([FromBody] Conexao con)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Conexao con2 = context.Conexao.Where(m => m.IdUser == con.IdUser && m.IdSim == con.IdSim && m.IdJob == con.IdJob && m.IdOp == con.IdOp).FirstOrDefault();

            if (con2 == null || (con2.IdMaq == null)) return BadRequest();

            return new JsonResult(con2);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Inserção de uma máquina pelo Admin na base de dados:

```
[HttpPost, Authorize(Roles = "Admin")]
0 references
public IActionResult Post(Maquina maq)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Maquina maquina = new Maquina();

            maquina.IdMaq = maq.IdMaq;
            maquina.Estado = "Ativo";

            context.Maquina.Add(maquina);
            context.SaveChanges();

            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

O método que se segue atualiza os dados de uma máquina, mais especificamente o seu estado:

```
[HttpPatch("{id_maquina}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Patch(int id_maquina, [FromBody] Maquina maq)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Maquina maquina = context.Maquina.Where(m => m.IdMaq == id_maquina).FirstOrDefault();

            maquina.Estado = maq.Estado is null ? maquina.Estado : maq.Estado;

            context.SaveChanges();
            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para a remoção de uma máquina pelo Admin, este envia por parâmetro o id da máquina e altera o seu estado para “Inativo”. No caso desta ser desativada, todas as conexões que a contenham, o id maquina e a duração serão convertidas a null. Mais tarde o user pode verificar que tal ocorreu e substituir por máquinas e durações novas.

```
[HttpDelete("{id_maquina}"), Authorize(Roles = "Admin")]
0 references
public IActionResult Delete(int id_maquina)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Maquina maquina = context.Maquina.Where(m => m.IdMaq == id_maquina).FirstOrDefault();
            List<Conexao> conexoes = context.Conexao.Where(c => c.IdMaq == id_maquina).ToList();

            foreach (Conexao con in conexoes)
            {
                con.IdMaq = null;
                con.Duracao = null;
            }

            maquina.Estado = "Inativo";

            context.SaveChanges();
            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

5.2.3. Conexao

Para buscar as conexões temos os seguintes métodos:

Este retorna para o Admin as conexões cujo estado sejam verdade (por outras palavras ativa para uso).

```
[HttpGet("active"), Authorize(Roles = "Admin")]
0 references
public IActionResult GetActives()
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            return new JsonResult(context.Conexao.Where(c => c.Estado == true).ToList());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Este retorna todas as conexões falsas (ou inativas) para o Admin:

```
[HttpGet("inactive"), Authorize(Roles = "Admin")]
0 references
public IActionResult GetInactives()
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            return new JsonResult(context.Conexao.ToList());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para o caso do utilizador querer ver as suas conexões, este através do seu id, consegue retornar todas que lhe sejam pertencidas. O admin também as pode consultar:

```
[HttpGet("{id_utilizador}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult GetByIdUser(int id_utilizador)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                Utilizador user = context.Utilizador.Where(u => u.IdUser == id_utilizador && u.Estado != "Inativo").FirstOrDefault();

                if (user == null) return BadRequest();

                if (UserMail != user.Mail)
                {
                    return Forbid();
                }
            }

            return new JsonResult(context.Conexao.Where(c => c.IdUser == id_utilizador && c.Estado == true).ToList());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

No caso de se querer obter os dados de uma simulação pertencente a um utilizador, esta função envia os dados dessa simulação em especifico:

```
/// <returns></returns>
[HttpGet("{id_utilizador}/{id_simulacao}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult GetSimulacaoByUser(int id_utilizador, int id_simulacao)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                Utilizador user = context.Utilizador.Where(u => u.IdUser == id_utilizador && u.Estado != "Inativo").FirstOrDefault();

                if (user == null) return BadRequest();

                if (UserMail != user.Mail)
                {
                    return Forbid();
                }
            }

            return new JsonResult(context.Conexao.Where(c => c.IdUser == id_utilizador && c.Estado == true && c.IdSim == id_simulacao).ToList());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Na criação de uma nova conexão é feito um Post dos dados passados por um objeto do tipo Conexão:

```
[HttpPost, Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Post(Conexao con)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            Conexao conexao = new Conexao();

            conexao.IdUser = con.IdUser;
            conexao.IdSim = con.IdSim;
            conexao.IdJob = con.IdJob;
            conexao.IdOp = con.IdOp;
            conexao.IdMaq = con.IdMaq;
            conexao.Duracao = con.Duracao;
            conexao.Estado = true;
            conexao.TempoInicial = con.TempoInicial;

            context.Conexao.Add(conexao);
            context.SaveChanges();

            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para o caso de o utilizador querer alterar o id de máquina e duração de uma conexão em especifico, este terá de enviar todos os ids de user, sim, job e operação que o identifiquem e os novos valores a substituir:

```
[HttpPatch, Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Patch([FromBody]Conexao con)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                Utilizador user = context.Utilizador.Where(u => u.IdUser == con.IdUser && u.Estado != "Inativo").FirstOrDefault();

                if (user == null) return BadRequest();

                if (UserMail != user.Mail)
                {
                    return Forbid();
                }
            }

            Conexao conexao = context.Conexao.Where(c => c.IdUser == con.IdUser && c.IdSim == con.IdSim && c.IdJob == con.IdJob && c.IdOp == con.IdOp).FirstOrDefault();

            conexao.IdMaq = con.IdMaq is null ? conexao.IdMaq : con.IdMaq;
            conexao.Duracao = con.Duracao is null ? conexao.Duracao : con.Duracao;
            conexao.TempoInicial = con.TempoInicial is null ? conexao.TempoInicial : con.TempoInicial;

            context.SaveChanges();
            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para o caso de remover uma simulação de uma conexão, este recebe por parâmetros id do user e id da simulação e onde estes forem encontrados na base de dados é lhes atribuído o estado falso (ou inativo).

Para o Admin:

```
[HttpDelete("{id_utilizador}/{id_simulacao}"), Authorize(Roles = "Admin")]
0 references
public IActionResult DeleteByAdmin(int id_utilizador, int id_simulacao)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            List<Conexao> conexoes = context.Conexao.Where(c => c.IdSim == id_simulacao && c.IdUser == id_utilizador).ToList();

            foreach (Conexao con in conexoes)
            {
                con.Estado = false;
            }

            context.SaveChanges();
            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

Para Admin e Funcionario:

```
[HttpDelete("deletesim/{id_utilizador}/{id_simulacao}"), Authorize(Roles = "Admin, Utilizador")]
0 references
public IActionResult Delete(int id_utilizador, int  id_simulacao)
{
    try
    {
        using (var context = new EscalonamentoContext())
        {
            if (User.HasClaim(ClaimTypes.Role, "Utilizador"))
            {
                string UserMail = User.FindFirstValue(ClaimTypes.Email);

                Utilizador user = context.Utilizador.Where(u => u.IdUser == id_utilizador && u.Estado != "Inativo").FirstOrDefault();

                if (user == null) return BadRequest();

                if (UserMail != user.Mail)
                {
                    return Forbid();
                }
            }

            List<Conexao> conexoes = context.Conexao.Where(c => c.IdSim == id_simulacao && c.IdUser == id_utilizador).ToList();

            foreach (Conexao con in conexoes)
            {
                con.Estado = false;
            }

            context.SaveChanges();
            return Ok();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

5.3. Funcionalidades para plano de produção

5.3.1. Automático

Para automático, este chama uma função que representa o algoritmo (Encontrado no AssignedTask.cs) por trás do escalonamento e este retorna um objeto que será utilizado no Front-End:

```
[HttpGet("planear/{IdUser}/{IdSim}"), Authorize]
0 references
public IActionResult Planner(int IdUser, int IdSim)
{
    return AssignedTask.AlgoritmoEscalonamento(IdUser, IdSim);
}
```

Algoritmo para o automático:

Como uma fase inicial do algoritmo, de forma a adequar a nossa solução, foi feita uma construção de uma lista (allJobs) de listas, em que cada lista representa um job e contem id da máquina e duração. O resto do algoritmo é igual ao fornecido pela google.

```
public struct Row
{
    public int machine;
    public int duration;
}

2 references
public struct OutputFrontend
{
    public double DuracaoTotal;
    public string output;
    public double conflicts;
    public double branches;
    public double wallTime;
}

1 reference
public static IActionResult AlgoritmoEscalonamento(int IdUser, int IdSim)
{
    using (var context = new EscalonamentoContext())
    {
        var allJobs = new List<List<Row>>();
        Row op = new Row();
        List<Row> job_row;
        List<Conexao> simulacao = context.Conexao.Where(s => s.IdSim == IdSim && s.IdUser == IdUser).ToList();
        Conexao lastIDJob = context.Conexao.Where(c => c.IdSim == IdSim && c.IdUser == IdUser)
        .OrderBy(c => c.IdUser).ThenBy(c=> c.IdSim).ThenBy(c=> c.IdJob).ThenBy(c => c.IdOp).LastOrDefault();
        OutputFrontend outputFrontend = new OutputFrontend();

        for(int i = 1; i <= lastIDJob.IdJob; i++)
        {
            job_row = new List<Row>();
            foreach (Conexao con in simulacao)
            {
                if (con.IdJob == i)
                {
                    op.machine = (int)con.IdMaq;
                    op.duration = (int)con.Duracao;

                    job_row.Add(op);
                }
            }

            allJobs.Add(job_row);
        }
```

5.3.2. Manual

Para o algoritmo manual, visto que neste caso é inserido manualmente os instantes iniciais de cada operação no Front-End, este recebe como parâmetro uma lista de conexões, que contem todas as sequências e respetivo instante inicial dessa sequência (id user, id simulação, id job, id operação, id da máquina, duração e instante inicial). Como verificações dos valores inseridos dos instantes serem corretos para uma boa criação do plano, são verificadas as seguintes condições:

-

```
2 references
public struct status
{
    public int status_value;
    public string status_text;
}

[HttpPost("planearmanual"), Authorize]
0 references
public IActionResult PlannerManual(List<Conexao> sim)
{
    try
    {
        int TempoConclusao = 0;
        status status = new status();

        for (int i = 0; i < sim.Count; i++)
        {
            if ((sim[i].TempoInicial + sim[i].Duracao) > TempoConclusao)
                TempoConclusao = (int)sim[i].TempoInicial + (int)sim[i].Duracao;

            if (i != 0)
            {
                if ((sim[i].IdOp == (sim[i - 1].IdOp + 1)) && (sim[i].IdJob == sim[i - 1].IdJob))
                {
                    if (sim[i].TempoInicial < sim[i - 1].TempoInicial + sim[i - 1].Duracao)
                    {
                        status.status_value = 0;
                        status.status_text = "Job " + sim[i].IdJob + " Op " + sim[i].IdOp + " esta a comecar antes da operacao anterior acabar.";

                        return new JsonResult(status);
                    }
                }
            }
        }

        foreach(Conexao conexao in sim)
        {
            foreach(Conexao con in sim)
            {
                if((conexao.IdMaq == con.IdMaq) && !(conexao.IdSim == con.IdSim && conexao.IdJob == con.IdJob && conexao.IdOp == con.IdOp))
                {
                    if(conexao.TempoInicial >= con.TempoInicial && conexao.TempoInicial < con.TempoInicial + con.Duracao)
                    {
                        status.status_value = 0;
                        status.status_text = "Job " + conexao.IdJob + " Op " + conexao.IdOp + " tem uma maquina atribuida que esta ocupada.";

                        return new JsonResult(status);
                    }
                }
            }
        }

        status.status_value = 1;
        status.status_text = TempoConclusao.ToString();

        return new JsonResult(status);
    }
    catch(Exception e)
    {
        Console.WriteLine(e);
        return BadRequest();
    }
}
```

6. Front-End

7. Bibliografia

8. Conclusão