IPCA

Escola Superior de Tecnologia

Relatório do Trabalho Prático de Comunicação de Dados

Joel Philippe Melo Figueiras, nº 20809

Nuno Miguel Carvalho Araújo, nº 20078

Tiago João Coelho Azevedo, nº 21153

Vitor Hugo Sá Machado, nº 21158

Professor Miguel Lopes

2021/2022

Índice

Índice de Figuras

1 Introdução

1.1 Motivação e Objetivos

No âmbito da unidade curricular de Comunicação de dados, foi pedida a elaboração de um trabalho prático, de modo a implementar os conteúdos lecionados ao longo das aulas.

Este trabalho tem como principais objetivos, o estudo do funcionamento de uma API REST, a criação de uma aplicação servidor capaz de correr serviços web com várias funcionalidades, a criação de uma aplicação cliente que faça uso da API de serviços definida e apresente uma interface ao utilizador, a disponibilização de um serviço web que suporte a gestão de um plano de produção e a disponibilização da funcionalidade de uma biblioteca nativa através de um serviço web.

A solução a implementar deve consistir na aplicação servidor, que vai oferecer um conjunto de serviços web a clientes genéricos. Adicionalmente, deve ser apresentada uma aplicação cliente que faz o acesso a estes serviços.

1.2 Descrição do Problema

Considere o problema de Job-Shop, que é a alocação de recursos para a concretização de um trabalho. No problema de Job-Shop existem várias máquinas que conseguem realizar operações. Para se produzir um trabalho (job) é necessário realizar um conjunto de operações, por sequência, em várias máquinas. Como as máquinas não conseguem realizar todas as operações, cada trabalho é uma sequência de operações, em que cada operação tem de ser feita numa máquina específica.

Adicionalmente, porque cada trabalho é distinto dos restantes, a sequência de máquinas de cada trabalho pode não ser a mesma dos restantes trabalhos, i.e., cada trabalho tem a sua própria sequência de máquinas. Contudo, a ordem das máquinas para cada trabalho tem de ser respeitada, i.e., uma operação de um trabalho que deve ser executada numa máquina específica só pode começar quando a operação anterior desse mesmo trabalho já terminou, assim como a operação seguinte do trabalho só pode começar depois da operação actual terminar.

Finalmente, realizar uma operação numa máquina demora tempo. Cada operação de cada trabalho demora um tempo específico, que pode ser distinto para cada operação. Uma máquina só consegue fazer uma operação de cada vez, assim, quando começa uma operação, só fica livre para novas operações após esta terminar. O objetivo da gestão do plano de produção é produzir uma lista de operações, cada uma para ser realizada numa máquina num determinado instante.

2 Serviço de Gestão de Simulações

2.1 Objetivos

O primeiro ponto do trabalho solicitado, tem como objetivo principal o desenvolvimento de uma API de serviços para a gestão de simulações. Uma simulação consiste num plano de produção que serve para controlar máquinas de modo a produzirem uma lista de produtos.

O serviço deve ser capaz de realizar as seguintes tarefas:

- Criar uma nova simulação, com os parâmetros de número de máquinas, número de trabalhos e número de operações;
- Listar as simulações criadas;
- Remover uma simulação.

2.2 Implementação da solução

Para criar uma nova simulação, é necessária a existência de três atributos distintos, sendo eles, os jobs, as operações e as máquinas. Desta forma, criamos um controlador para manipulação de cada um dos atributos. Em todos os controladores criamos duas interações, a obtenção da lista de todos os objetos armazenados em base de dados e a inserção de um objeto.

```
#region Obter lista de jobs
// GET: api/Job
[HttpGet]
0 referências | Nuno Miguel Carvalho Araújo, há 22 dias | 1 autor, 1 alteração
public async Task<ActionResult<IEnumerable<Job>>> GetJob()
{
    return await _context.Job.ToListAsync();
}
#endregion
```

```
#region Inserir Job
// POST: api/Job
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPost]
0 referências | Nuno Miguel Carvalho Araújo, 1 dia atrás | 1 autor, 2 alterações
public async Task<ActionResult<Job>> PostJob(Job job)
{
    Response response = new();

    if (JobNameExists(job.Nome))
    {
        response._statusCode = 409;
        response._response = "Já existe um job com esse nome associado";
        return Ok(response);
    }
    if (VerificaNulo(job.Nome))
    {
        response._statusCode = 404;
        response._response = "Inserir nome do job";
        return Ok(response);
    }
    else
    {
        _context.Job.Add(job);
        await _context.SaveChangesAsync();
        response._statusCode = 201;
        response._response = "Job inserido com sucesso";
        return Ok(response);
    }
}
#endregion
```

Figura 1: Manipulação de objetos

A fim de listar as simulações criadas, desenvolvemos um request que seleciona a lista das simulações guardadas em base de dados e rotorna essa lista para a aplicação cliente.

```
#region Obter todas as simulações
// GET: api/Plano/simulacoes
[HttpGet("/simulacoes")]
0 referências | Joel Figueiras, há 18 horas | 2 autores, 3 alterações
public async Task<ActionResult<IEnumerable<Plano>>> GetAllSimulacoes()
{

    Response aux = new();
    var simulacoes = await _context.Plano.ToListAsync();
    if(simulacoes.Count == 0)
    {
        aux._statusCode = 404;
        aux._response = "Não foram encontradas simulações";

        return Ok(aux);
    }
    else return Ok(simulacoes);
}
#endregion
```

Figura 2: Listar simulações existentes

Para a remoção de uma simulação, criamos um método que recebe, da aplicação cliente, o id da simulação e procede à remoção da simulação e de todos os atributos a ela associados.

```
#region Eliminar uma Simulacao
// DELETE: api/Simulacao
[HttpDelete("/{IdSimulacao}")]
0 referências | Nuno Miguel Carvalho Araújo, há 4 horas | 2 autores, 2 alterações
public async Task<IActionResult> DeleteSimulacao(int IdSimulacao)
{
    Response response = new();
    var itens = _context.Plano.Where(c => c.IdSimulacao == IdSimulacao).ToList();
    var item = await _context.Simulacao.FindAsync(IdSimulacao);
    if (item == null)
    {
        response._statusCode = 404;
        response._response = "Simulação não encontrada";
        return Ok(response);
    }
    else if(itens.Count > 0)
    {
        foreach (var obj in itens)
        {
            _context.Plano.Remove(obj);
        }

        _context.Simulacao.Remove(item);
        await _context.SaveChangesAsync();

        response._statusCode = 201;
        response._response = "Simulação eliminada com sucesso";
        return Ok(response);
    }
    else
    {
        _context.Simulacao.Remove(item);
        await _context.SaveChangesAsync();

        response._statusCode = 201;
        response._response = "Simulação eliminada com sucesso";
        return Ok(response);
    }
}
#endregion
```

Figura 3: Remover determinada simulação

3 Serviço de Tabela de Produção

O segundo ponto do trabalho, tem como principal objetivo o desenvolvimento de uma API de serviços para a tabela de produção de uma simulação. Cada simulação é definida por um conjunto de máquinas, conjunto de trabalhos com as suas operações e, finalmente, por uma tabela de produção que indica para cada operação de cada trabalho qual a máquina que a consegue realizar e o tempo que demora a ser feita.

Segue-se uma tabela exemplo para o problema apresentado, em que os valores na tabela representam o par "máquina - tempo", para um caso com três jobs e diferentes operações para cada job.

Jobs	**Operation1**	**Operation2**	**Operation3**
Job0	(M0,3)	(M1,2)	(M2,2)
Job1	(M0,2)	(M2,1)	(M1,4)
Job2	(M1,4)	(M2,3)	-

3.1 Objetivos

- Construir a tabela que indica, para cada operação de cada trabalho, a máquina e a duração da operação. Esta tabela deve poder ser criada através de uma função que insere para uma operação específica de um trabalho, a sua máquina e o seu tempo;
- Indicar o término da construção da tabela e, com isto, obter duas respostas distintas, sendo elas:
  - Sucesso, caso tudo tenha sido preenchido em conformidade;
  - Erro, caso exista alguma operação com informação em falta (e a indicação da informação em falta).
- Após a inserção da tabela de operação, apenas o valor de cada operação pode ser alterado mas não podem ser inseridos novos valores;
- Consultar a tabela, através de uma consulta individual para cada operação de cada trabalho, com retorno da máquina e duração. Caso a operação não esteja ainda introduzida, indicar erro;
- Fazer o download de um ficheiro de texto que representa a tabela anterior. Cada linha representa um trabalho e, em cada linha, existe uma sequência de pares de números que representam a máquina e o tempo de execução, para cada operação de cada job.

3.2 Implementação da solução

De modo a conseguir contruir a tabela de produção, criamos um controlador que recebe como parâmetro um JSON com a identificação da simulação, job e plano a que vai ser atribuida a respetiva máquina e tempo. Como definido no enunciado, só é possível atribuir, para cada operação de cada job, uma máquina e o seu tempo e esta atribuição deve ser feita de forma singular, ou seja, uma atribuição por cada acesso do cliente ao servidor.

Criamos um conjunto de verificações, de modo a informar o utilizador de qualquer inconformidade ocorrida na criação da tabela de produção.

```
#region Construção da tabela de produção
// POST: api/Plano
[HttpPost]
0 referências | Nuno Miguel Carvalho Araújo, 1 dia atrás | 3 autores, 7 alterações
public async Task<ActionResult<Plano>> PostPlano(Plano plano)
{
    Response response = new();
    var id = GetUtilizadorID();
    var registo = GetRegistoUtilizador(id);
    if (registo == default)
    {
        response._statusCode = 404;
        response._response = "Utilizador não encontrador";
        return Ok(response);
    }

    if (plano.UnidadeTempo == 0)
    {
        response._statusCode = 409;
        response._response = "Deve inserir o tempo de trabalho da máquina!";
        return Ok(response);
    }
    if (await _context.Simulacao.FirstOrDefaultAsync(x => x.Id == plano.IdSimulacao) == default)
    {
        response._statusCode = 404;
        response._response = "Simulação não existe!";
        return Ok(response);
    }
    if (await _context.Job.FirstOrDefaultAsync(x => x.Id == plano.IdJob) == default)
    {
        response._statusCode = 404;
        response._response = "Job não existe!";
        return Ok(response);
    }
    if (await _context.Operacao.FirstOrDefaultAsync(x => x.Id == plano.IdOperacao) == default)
    {
        response._statusCode = 404;
        response._response = "Operação não existe!";
        return Ok(response);
    }
    if (await _context.Maquina.FirstOrDefaultAsync(x => x.Id == plano.IdMaquina) == default)
    {
        response._statusCode = 404;
        response._response = "Máquina não existe!";
        return Ok(response);
    }
```

```
    plano.IdUtilizador = registo.Id;
    plano.Estado = true;

    var checkPlano = _context.Plano.AsNoTracking().FirstOrDefault
        (c => c.IdSimulacao == plano.IdSimulacao &&
        c.IdJob == plano.IdJob &&
        c.IdOperacao == plano.IdOperacao);
    if(checkPlano == default)
    {
        var query = _context.Plano.AsQueryable();
        query = query.Where
            (c => c.IdUtilizador == registo.Id &&
            c.IdSimulacao == plano.IdSimulacao &&
            c.IdJob == plano.IdJob);
        var itens = await query.ToListAsync();

        plano.PosOperacao = 1;
        if(itens.Count > 0)
        {
            foreach (var item in itens)
            {
                if (plano.PosOperacao <= item.PosOperacao) plano.PosOperacao = item.PosOperacao + 1;
            }
        }

        _context.Plano.Add(plano);
        try { await _context.SaveChangesAsync(); }
        catch (DbUpdateException)
        {
            response._statusCode = 409;
            response._response = "Erro na inserção!";
            return Ok(response);
        }
        response._statusCode = 200;
        response._response = "Inserido com sucesso";
        return Ok(response);
    }
    else
    {
        response._statusCode = 409;
        response._response = $"O job {plano.IdOperacao} da simulação {plano.IdSimulacao} já contém a operação {plano.IdOperacao}";
        return Ok(response);
    }
}
```

Figura 4: Construção da tabela de produção

A fim de permitir ao utilizador alterar o valor de determinada operação, criamos um controlador que recebe um JSON com a simulação, o job e a operação que pretende alterar, bem como a nova máquina e o tempo a atribuir a essa operação.

Existe, no entanto, uma falha nas regras de negócio, pois, é possível um utilizador alterar o valor de uma operação após a atribuição dos tempos, o que pode gerar um conflito na boa implementação do algoritmo de atribuição dos tempos ao plano de produção (assunto abordado posteriormente).

Como este problema foi identificado tardiamente, não tivemos possibilidade de o resolver.

```
#region Alterar o valor de determinada operação
// PUT: api/Plano
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPut("/AlterarValor")]
0 referências | Nuno Miguel Carvalho Araújo, Há 19 minutos | 1 autor, 2 alterações
public async Task<IActionResult> PutPlano(Plano plano)
{
    Response response = new();
    var id = GetUtilizadorID();
    var registo = GetRegistoUtilizador(id);
    if (registo == default)
    {
        response._statusCode = 404;
        response._response = "Utilizador não encontrador";
        return Ok(response);
    }

    var item = await _context.Plano.AsNoTracking().FirstOrDefaultAsync
        (c => c.IdUtilizador == registo.Id &&
        c.IdSimulacao == plano.IdSimulacao &&
        c.IdJob == plano.IdJob &&
        c.IdOperacao == plano.IdOperacao);
    if (item == null)
    {
        response._statusCode = 404;
        response._response = "Operação não encontrada";
        return Ok(response);
    }

    item.IdMaquina = plano.IdMaquina;
    item.UnidadeTempo = plano.UnidadeTempo;

    _context.Entry(item).State = EntityState.Modified;

    try { await _context.SaveChangesAsync(); }
    catch (DbUpdateConcurrencyException)
    {
        response._statusCode = 409;
        response._response = "Erro na inserção!";
        return Ok(response);
    }
    response._statusCode = 200;
    response._response = "Dados alterados com sucesso!";
    return Ok(response);
}
#endregion
```

Figura 5: Alterar valor de cada operação

Em seguida, criamos um controlador com o objetivo de permitir ao utilizador consultar, individualmente, para cada operação de cada trabalho, a máquina e a respetiva duração. Este controlador recebe, por parâmetro, o id da simulação, o id do job e o id da operação e devolve, ao utilizador, a respetiva máquina e duração da execução da tarefa.

```
#region Consultar tabela através de operação
// GET: api/Plano
[HttpGet]
0 referências | Nuno Miguel Carvalho Araújo, 1 dia atrás | 2 autores, 3 alterações
public async Task<ActionResult<IEnumerable<Plano>>> GetPlano(int IdSimulacao, int IdJob, int IdOperacao)
{
    Response response = new();
    var id = GetUtilizadorID();
    var registo = GetRegistoUtilizador(id);
    if (registo == default)
    {
        response._statusCode = 404;
        response._response = "Utilizador não encontrador";
        return Ok(response);
    }

    var item = await _context.Plano.AsNoTracking().FirstOrDefaultAsync(c => c.IdUtilizador == registo.Id && c.IdSimulacao == IdSimulacao && c.IdJob == IdJob && c.IdOperacao == IdOperacao);
    if (item == null)
    {
        response._statusCode = 404;
        response._response = "Plano não encontrado";
        return Ok(response);
    }
    return Ok(item);
}
#endregion
```

Figura 6: Consultar tabela através da operação

Por fim, criamos um controlador que fornece, ao utilizador, a possibilidade de efetuar o download da tabela de produção criada anteriormente, em formato textual.

Para o sucesso desta implementação, o controlador recebe, por parâmetro, o id da simulação. Esse id é, depois, enviado para uma função externa responsável pela escrita do ficheiro. Após a conclusão da escrita, o ficheiro é disponibilizado ao utilizador para o respetivo download.

```
#region Download tabela de produção
[HttpGet("files/tabela/{IdSimulacao}")]
0 referências | Nuno Miguel Carvalho Araújo, há 23 horas | 1 autor, 1 alteração
public async Task<ActionResult> DownloadTabela(int IdSimulacao)
{
    Response response = new();
    var query = _context.Plano.AsQueryable();
    query = query.Where(plano => plano.IdSimulacao == IdSimulacao);
    var itens = await query.ToListAsync();
    if (itens == null || itens == default)
    {
        response._statusCode = 409;
        response._response = "Erro na obtenção da simulação";
        return Ok(response);
    }

    await WriteTabela(IdSimulacao, itens);
    var filePath = "Download/TabelaProducao.txt";
    var bytes = await System.IO.File.ReadAllBytesAsync(filePath);
    return File(bytes, "text/plain", Path.GetFileName(filePath));
}
1 referência | Nuno Miguel Carvalho Araújo, há 23 horas | 1 autor, 1 alteração
public static async Task WriteTabela(int IdSimulacao, List<Plano> itens)
{
    Response response = new();
    string text = $"Simulacao {IdSimulacao}\n\n";
    List<Plano> sortList = itens.OrderBy(o => o.IdJob).ToList();

    if(sortList.Count > 0)
    {
        Plano previous = default;

        foreach (Plano item in sortList)
        {
            if(previous == null) text += $"JOB {item.IdJob}: ({item.IdMaquina},{item.UnidadeTempo})";
            else if (previous.IdJob != item.IdJob || previous == null) text += $"\nJOB {item.IdJob}: ({item.IdMaquina},{item.UnidadeTempo})";
            else text += $" ({item.IdMaquina},{item.UnidadeTempo})";
            previous = item;
        }
    }
    using StreamWriter write = new StreamWriter("Download/TabelaProducao.txt");
    write.WriteLine(text);
}
#endregion
```

Figura 7: Download da tabela de produção

4 Serviço de Plano de Produção Manual

O terceiro ponto do trabalho, tem como principal objetivo o desenvolvimento de uma API de serviços para a definição de um plano de produção. Um plano de produção é uma lista que atribui, para cada operação de cada trabalho, um tempo de início.

A atribuição de um tempo de início está sujeita, neste problema, a duas restrições, sendo elas:

- As operações de cada trabalho têm de ser realizadas por ordem. A segunda operação de um trabalho só pode começar depois da primeira operação do mesmo trabalho estar concluída, e assim sucessivamente para todas as operações seguintes do mesmo trabalho;
- Uma máquina só consegue executar uma operação de cada vez, pelo que havendo duas operações para a mesma máquina, uma só pode começar depois da outra ter terminado.

Segue-se uma tabela exemplo para o resultado do escalonamento, em que cada operação de cada job tem um tempo de início.

Jobs	**Operation1**	**Operation2**	**Operation3**
Job0	2	8	10
Job1	0	2	4
Job2	0	4	-

4.1 Objetivos

- Atribuir um tempo de início, indicado manualmente pelo utilizador, para uma operação de um trabalho. Esta atribuição deverá ter em conta as seguintes condições:
  - O tempo de início da operação escolhida não pode começar antes da operação anterior desse mesmo trabalho terminar;
  - O tempo de início da operação escolhida não pode começar numa máquina que está ocupada a concluir outra operação de qualquer outro trabalho.
- Terminar a introdução manual dos tempos de início, validando se todas as operações têm um tempo de início atribuído;
- Indicar o tempo de conclusão da operação que termina em último lugar;
- Fazer o download de um ficheiro de texto com a informação do plano de produção. Cada linha representa uma operação e, em cada operação, está identificado o número do job, o número da operação, a máquina, o tempo de início, a duração e o tempo de fim.

4.2 Implementação da solução

De forma a permitir, ao utilizador, a inserção manual dos tempos de ínicio para cada operação de cada job de determinada operação, criamos um controlador que recebe um JSON com a lista de operações a serem modificadas, bem como o tempo de início definido pelo utilizador.

Após a receção do JSON, existe um algoritmo que verifica se os tempos inseridos pelo utilizador estão de acordo com as condições impostas anteriormente.

Caso uma operação de determinado job esteja a começar antes da operação anterior desse mesmo job terminar, o utilizador recebe a informação da operação que está a criar conflito.

Caso alguma máquina esteja ocupada, o utilizador recebe a informação da máquina que está ocupada em determinado momento.

Em caso de sucesso, o utilizador recebe a resposta com o tempo de conclusão da operação que termina em ultimo lugar.

```
foreach (var plano in List)
{
    item = await _context.Plano.AsNoTracking().FirstOrDefaultAsync
        (c => c.IdUtilizador == registo.Id &&
        c.IdSimulacao == IdSimulacao &&
        c.IdJob == plano.IdJob &&
        c.IdOperacao == plano.IdOperacao);
    if (item == null)
    {

        response._statusCode = 409;
        response._response = "Erro na leitura do plano";
        return Ok(response);
    }
    else plano.TempoFinal = plano.TempoInicial + plano.UnidadeTempo;
}

foreach (var plano_ant in List)
{
    maxTemp += plano_ant.UnidadeTempo;
    foreach(var plano_prox in List)
    {
        if ((plano_prox.PosOperacao == (plano_ant.PosOperacao + 1)) && (plano_prox.IdJob == plano_ant.IdJob))
        {
            if (plano_prox.TempoInicial < plano_ant.TempoFinal)
            {
                response._statusCode = 409;
                response._response= $"A operação {plano_prox.IdOperacao} do job {plano_prox.IdJob} começa antes da operação anterior terminar!";
                return Ok(response);
            }
        }
    }
}
```

Figura 8: Algoritmo verificação de operações por job

```
if(List.Count > 1)
{
    int[,] planArray = new int[nmaq+1, maxTemp+1];

    foreach (var plano in List)
    {
        tempo = plano.UnidadeTempo;
        while ((planArray[plano.IdMaquina, plano.TempoInicial] == 0) && (tempo > 0))
        {
            planArray[plano.IdMaquina, plano.TempoInicial] = 1;
            plano.TempoInicial++;
            tempo--;
        }
        if (tempo != 0)
        {
            response._statusCode = 409;
            response._response = $"Não é possivel atribuir a operação {plano.IdOperacao} do job {plano.IdJob}, pois a máquina {plano.IdMaquina} está a ser utilizada!";
            return Ok(response);
        }
    }
}
```

Figura 9: Algoritmo verificação de utilização de máquinas

Por fim, criamos um controlador que fornece, ao utilizador, a possibilidade de efetuar o download do plano de produção inserido anteriormente, em formato textual.

Para o sucesso desta implementação, o controlador recebe, por parâmetro, o id da simulação. Esse id é, depois, enviado para uma função externa responsável pela escrita do ficheiro. Após a conclusão da escrita, o ficheiro é disponibilizado ao utilizador para o respetivo download.

```
#region Download plano de produção
[HttpGet("files/plano/{IdSimulacao}")]
0 referências | Nuno Miguel Carvalho Araújo, 1 dia atrás | 1 autor, 1 alteração
public async Task<ActionResult> DownloadPlano(int IdSimulacao)
{
    Response response = new();
    var query = _context.Plano.AsQueryable();
    query = query.Where(plano => plano.IdSimulacao == IdSimulacao);
    var itens = await query.ToListAsync();
    if (itens == null || itens == default)
    {
        response._statusCode = 409;
        response._response = "Erro na obtenção da simulação";
        return Ok(response);
    }

    await WritePlano(IdSimulacao, itens);
    var filePath = "Download/PlanoProducao.txt";
    var bytes = await System.IO.File.ReadAllBytesAsync(filePath);
    return File(bytes, "text/plain", Path.GetFileName(filePath));
}
1 referência | Nuno Miguel Carvalho Araújo, 1 dia atrás | 1 autor, 1 alteração
public static async Task WritePlano(int IdSimulacao, List<Plano> itens)
{
    string text = $"Simulacao {IdSimulacao}\n\n";
    foreach (var iten in itens)
    {
        text = text + $"JOB: {iten.IdJob} | OP: {iten.PosOperacao} | MAQ: {iten.IdMaquina} | TI: {iten.TempoInicial} | DUR: {iten.UnidadeTempo} | TF: {iten.TempoFinal}\n";
    }
    using StreamWriter write = new StreamWriter("Download/PlanoProducao.txt");
    write.WriteLine(text);
}
#endregion
```

Figura 10: Download do plano de produção

5 Serviço de Plano de Produção Automatizado

O quarto ponto do trabalho, tem como principal objetivo o desenvolvimento de uma API de serviços para a definição de um plano de produção automatizado. Este plano automatizado pode ser obtido através da implementação de um algoritmo implementado na biblioteca OR-Tools, disponibilizada através da Google. Esta ferramenta gera, automaticamente, uma solução de um plano de produção capaz de cumprir com as restrições impostas no plano de produção manual.

5.1 Objetivos

- Atribuir um tempo de início, indicado automaticamente pela ferramenta, para todas as operações;
- Indicar o tempo de conclusão da operação que termina em último lugar;
- Fazer o download de um ficheiro de texto com a informação do plano de produção. Cada linha representa uma operação e, em cada operação, está identificado o número do job, o número da operação, a máquina, o tempo de início, a duração e o tempo de fim.

5.2 Implementação da solução

De modo a permitir, ao utilizador, a inserção automatizada dos tempos, criamos um controlador que recebe como parâmetro o id da simulação para a qual é possível calcular automaticamente o tempo de início de cada job.

Após a receção do id da simulação, obtemos uma lista com as operações que fazem parte dessa simulação.

Nesta parte, tivemos alguma dificuldade em modelar os dados recebidos de modo a poderem ser usados pelo algoritmo da Google, no entanto, com alguma pesquisa e ajuda externa, conseguimos formata-los corretamente.

Após a modelação dos dados, o algoritmo tenta obter o melhor escalonamento para o problema apresentado. Se esse escalonamento for obtido com sucesso, é retornada a informação tempo de conclusão e a operação que termina em último lugar.

Se o escalonamento não for solucionado, é enviada uma mensagem de erro a informar o utilizador que não foi possível chegar a uma conclusão.

```
Dictionary<int, List<Plano>> dictJobs = new Dictionary<int, List<Plano>>();
int macNumber = 0;
Dictionary<int, int> dictMac = new Dictionary<int, int>(); // Chave -> Id real da máquina | Valor -> Id "virtual" para o algoritmo da google
Dictionary<int, int> dictJob = new Dictionary<int, int>(); // Chave -> "virtual" do job para o algoritmo da google | Valor -> Id real do job

foreach (Plano plano in Planos)
{
    List<Plano> planoJob = dictJobs.GetValueOrDefault(plano.IdJob, new List<Plano>());
    planoJob.Add(plano);
    dictJobs[plano.IdJob] = planoJob;
}

List<List<googleOperation>> allJobs = new List<List<googleOperation>>();
foreach (int key in dictJobs.Keys.OrderBy(job => job))
{
    List<googleOperation> job = new List<googleOperation>();
    foreach (Plano plano in dictJobs.GetValueOrDefault(key, new List<Plano>()))
    {
        int nrMac;
        bool res = dictMac.TryGetValue(plano.IdMaquina, out nrMac);
        if (res == false) nrMac = macNumber++;
        job.Add(new googleOperation(nrMac, plano.UnidadeTempo));
        dictMac[plano.IdMaquina] = nrMac;
    }
    dictJob[allJobs.Count] = key;
    allJobs.Add(job);
}
```

Figura 11: Modelação de dados para algoritmo automático

Para fornecermos ao utilizador a possibilidade de download do ficheiro com o plano automatizado, utilizamos o controlador criado anteriormente, no plano de produção manual.

Como o ficheiro a ser gerado para download segue as mesmas regras, optamos por ter apenas um controlador, pois seria desnecessário ter dois controladores para o mesmo efeito.

```
#region Download plano de produção
[HttpGet("files/plano/{IdSimulacao}")]
0 referências | Nuno Miguel Carvalho Araújo, 1 dia atrás | 1 autor, 1 alteração
public async Task<ActionResult> DownloadPlano(int IdSimulacao)
{
    Response response = new();
    var query = _context.Plano.AsQueryable();
    query = query.Where(plano => plano.IdSimulacao == IdSimulacao);
    var itens = await query.ToListAsync();
    if (itens == null || itens == default)
    {
        response._statusCode = 409;
        response._response = "Erro na obtenção da simulação";
        return Ok(response);
    }

    await WritePlano(IdSimulacao, itens);
    var filePath = "Download/PlanoProducao.txt";
    var bytes = await System.IO.File.ReadAllBytesAsync(filePath);
    return File(bytes, "text/plain", Path.GetFileName(filePath));
}
1 referência | Nuno Miguel Carvalho Araújo, 1 dia atrás | 1 autor, 1 alteração
public static async Task WritePlano(int IdSimulacao, List<Plano> itens)
{
    string text = $"Simulacao {IdSimulacao}\n\n";
    foreach (var iten in itens)
    {
        text = text + $"JOB: {iten.IdJob} | OP: {iten.PosOperacao} | MAQ: {iten.IdMaquina} | TI: {iten.TempoInicial} | DUR: {iten.UnidadeTempo} | TF: {iten.TempoFinal}\n";
    }
    using StreamWriter write = new StreamWriter("Download/PlanoProducao.txt");
    write.WriteLine(text);
}
#endregion
```

Figura 12: Download do plano de produção

6 Gestão de utilizadores

O quinto e último ponto do trabalho, tem como principal objetivo o desenvolvimento de uma API de serviços para a gestão dos utilizadores do sistema. Esta gestão é feita e acessível apenas a utilizadores que estejam referenciados como administradores.

6.1 Objetivos

- Criar utilizadores, atribuindo uma palavra-chave individual;
- Permitir alterar a palavra-chave de um utilizador;
- Remover um utilizador, assim como todas as suas informações armazenadas no servidor.

6.2 Implementação da solução

Para a gestão de utilizadores, criamos um sistema de autenticação baseado em JWT (Json Web Token). Este sistema de autenticação permite verificar todas as informações do utilizador ativo sem necessidade de acesso à base de dados.

Para implementar este mecanismo, recorremos à utilização da biblioteca *Microsoft.IdentityModel.Tokens* e criamos um controlador responsável pela validação do login.

Se as credenciais introduzidas na página de inicio de sessão estiverem corretas, é retornado um JWT. Caso contrário, é indicado ao utilizador que as credenciais inseridas são inválidas, impedindo assim, o seu acesso a todos os endpoints seguintes.

```
public async Task<IActionResult> Post(string Username, string Password)
{
    Response response = new();
    if (string.IsNullOrEmpty(Username)) return Ok(response);
    if (string.IsNullOrEmpty(Password)) return Ok(response);

    if (Username != null && Password != null)
    {
        var userData = await GetUtilizador(Username, Password);
        var jwt = _configuration.GetSection("Jwt").Get<Jwt>();

        if (userData != null)
        {
            var claims = new[]
            {
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
                new Claim(JwtRegisteredClaimNames.Iat, DateTime.UtcNow.ToString()),
                new Claim("Id", userData.Id.ToString()),
                new Claim("Username", userData.Username),
                new Claim("Password", userData.Password),
                new Claim("Admin", userData.Admin.ToString()),
                new Claim("Estado", userData.Estado.ToString())
            };
            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwt.Key));
            var signIn = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var token = new JwtSecurityToken(
                jwt.Issuer,
                jwt.Audience,
                claims,
                expires: DateTime.Now.AddMinutes(60),
                signingCredentials: signIn);
            if (userData.Estado == false)
            {
                response._statusCode = 401;
                response._response = "O utilizador está desativado";
                return Ok(response);
            }
            var auxToken = new JwtSecurityTokenHandler().WriteToken(token);
            Token rspToken = new()
            {
                _token = auxToken
            };
            return Ok(rspToken);
        }
        else return Ok(response);
    }
    else return Ok(response);
}
```

Figura 13: Implementação do JWT

De forma a permitir a criação de novos utilizadores, criamos um controlador ao qual apenas utilizadores referenciados como administradores têm acesso. Esta referência é verificada através do JWT.

Os utilizadores que não têm acesso a esta página, recebem uma mensagem de erro sempre que a tentam aceder.

Em relação aos administradores, para a inserção de um novo utilizador, é enviado um JSON com as informações da conta a ser criada.

Em seguida, são feitas algumas verificações de modo a confirmar que todos os campos se encontram bem delineados.

Caso algum campo esteja incorreto, é enviada uma resposta ao administrador com o campo incorreto. No entanto, se tudo estiver de acordo com o pretendido, o administrador é informado à cerca da boa inserção do novo utilizador no servidor.

```
#region Criar utilizadores, atribuindo password (apenas administradores podem criar)
// POST: api/Utilizador
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPost]
0 referências | Nuno Miguel Carvalho Araújo, 1 dia atrás | 3 autores, 6 alterações
public async Task<ActionResult<Utilizador>> PostUtilizador(Utilizador utilizador)
{
    Response response = new();
    if (VerificaNulo(utilizador.Password))
    {
        response._statusCode = 406;
        response._response = "Deve inserir uma password";
        return Ok(response);
    }
    if (string.IsNullOrEmpty(utilizador.Password) || utilizador.Password.Length <= 6)
    {
        response._statusCode = 406;
        response._response = "A password deverá conter mais de 6 caracteres";
        return Ok(response);
    }
    if (VerificaNulo(utilizador.Username))
    {
        response._statusCode = 406;
        response._response = "Deve inserir um username!";
        return Ok(response);
    }
    if (utilizador.Username.Contains(' '))
    {
        response._statusCode = 406;
        response._response = "Username não deve conter espaços!";
        return Ok(response);
    }
    utilizador.Estado = true;

    var id = GetUtilizadorID();
    var registo = GetRegistoUtilizador(id);

    if (registo == default)
    {
        response._statusCode = 404;
        response._response = "Utilizador não encontrado";
        return Ok(response);
    }
    if (registo.Admin == false)
    {
        response._statusCode = 401;
        response._response = "Não tem permissões para criar novos utilizadores";
        return Ok(response);
    }
    if (registo.Admin == true)
    {
        var user = await _context.Utilizador.SingleOrDefaultAsync
            (e => e.Username == utilizador.Username);

        if (user != default)
        {
            response._statusCode = 409;
            response._response = "Username já registado!";
            return Ok(response);
        }
        else
        {
            _context.Utilizador.Add(utilizador);
            await _context.SaveChangesAsync();
            response._statusCode = 201;
            response._response = "Utilizador registado com sucesso";
            return Ok(response);
        }
    }
    response._statusCode = 409;
    response._response = "Erro no registo do utilizador";
    return Ok(response);
}
#endregion
```

Figura 14: Criação de novos utilizadores

De maneira a permitir aos utilizadores a alteração da sua password, criamos um controlador que recebe a password antiga do utilizador e verifica se ela está correta.

Se a password antiga estiver incorreta, uma mensagem de erro é enviada como resposta. No entando, se a password antiga estiver correta é permitida a alteração da password, dês de que a nova palavra-passe tenha um tamanho superior a 6 carateres e não seja igual à antiga.

Figura 15: Alteração da password

Por fim, permitimos aos administradores a possibilidade de desativar um determinado utilizador. Para tornar essa funcionalidade possível, criamos um controlador que recebe um JSON com as informações do utilizador a ser desativado.

O administrador recebe uma resposta positiva, caso o utilizador seja desativado com sucesso ou negativa, caso haja algum erro na remoção do utilizador.

Com a remoção de um utilizador, era suposto remover todas as informações guardadas no servidor para esse utilizador. No entanto, como regra de negócio, optamos por não remover as simulações, jobs ou operações criadas por esse utilizador. Achamos que não faz sentido remover simulações que possam estar a ser utilizadas por outros utilizadores, devido à remoção da conta que as criou.

Figura 16: Remoção de um utilizador

7 Conclusão

Na parte da gestão de simulações, conseguimos implementar todos os objetivos definidos sem grande dificuldade.

Em relação à tabela de produção, tivemos algumas dificuldades na delineação da estrutura da base de dados, no entanto, após alguma discussão em grupo, chegamos a um acordo referente à estrutura a implementar.

Quanto ao plano de produção manual, foi complicado obter um algoritmo capaz de identificar se o tempo definido para cada operação estava de acordo com as restrições impostas. No entanto, após batalharmos um bocado, conseguimos chegar a um algoritmo capaz de ir de encontro aos objetivos definidos.

No plano de produção automatizado, o maior desafio foi a manipulação dos nossos dados de forma a conseguirmos aplicar o algoritmo da google na nossa estrutura.

Já na parte da gestão de utilizadores, tivemos de aprofundar os nossos conhecimentos em relação aos métodos de autenticação existentes. Por fim, optamos por implementar um JWT e não sentimos grandes dificuldades em implementar soluções para os objetivos estabelecidos.

De uma forma geral, concluímos que a estrutura do trabalho implementada foi útil para a compreensão e interiorização dos conteúdos lecionados na unidade curricular de Comunicação de Dados.