2024
2025

UNIVERSIDADE
DA MAIA

Relatório de Estágio Curricular

Desenvolvimento da API do TeamWork

Licenciatura em Informática

Redes e Cibersegurança

Aluno: Tiago Leal Baganha – A041423

Orientador de Estágio: Docente Mário Serrão

Supervisor de Estágio: Eng. Bruno Silveira

Entidade acolhedora do Estágio: Garland

“A tecnologia é apenas uma ferramenta. O que importa é como a usamos..”

— Tim O’Reilly

Agradecimentos

Em primeiro lugar, agradecer profundamente aos meus pais, pelo apoio incondicional, paciência e incentivo constantes ao longo de todo o percurso académico, sem os quais este momento não seria possível.

Não esquecer do apoio, carinho, compreensão que a minha namorada foi capaz de oferecer nos momentos mais difíceis e desafiadores.

Aos meus amigos, agradeço pela amizade genuína, pelas palavras de encorajamento e pelos momentos de descontração que ajudaram a tornar esta jornada mais leve e equilibrada.

Um agradecimento especial ao engenheiro Bruno Silveira, orientador do estágio, pela orientação prática, disponibilidade e partilha de conhecimentos que foram fundamentais para a realização das atividades desenvolvidas.

Um sincero agradecimento ao colega de trabalho João Organista, cuja paciência, disponibilidade e partilha de conhecimentos foram essenciais para a minha evolução profissional e pessoal ao longo do estágio.

Não menos importante, manifestar a minha gratidão ao supervisor académico, docente Mário Serrão, pelo acompanhamento rigoroso, pelas sugestões construtivas e pelo apoio prestado durante todas as fases do estágio.

Desenvolvimento da API do TeamWork

Resumo

O estágio consiste no desenvolvimento de uma API para integrar a plataforma de gestão de projetos TeamWork com a ferramenta de Power BI.

Esta integração vem facilitar a monitorização e análise dos projetos da Garland, permitindo ao responsável centralizar os dados para uma melhor gestão entre projetos e colaboradores.

Durante o desenvolvimento utilizamos tecnologias já abordadas durante unidades curriculares, como por exemplo SQL e APIs, mas também utilizamos tecnologias novas que não tinha conhecimento, como o dapper que trata de toda a ligação entre os resultados da API com a base de dados.

Este estágio, com uma duração de 250 horas, foi importante na aprendizagem e na evolução do estudante para uma melhor perceção do mercado de trabalho e adição de conhecimentos técnicos.

Palavras-chaves: API do TeamWork, Servidor de SQL, Dapper, .NET, C#

TeamWork API Development

Abstract

The internship consists of developing an API to integrate the TeamWork project management platform with the Power BI tool.

This integration facilitates the monitoring and analysis of Garland's projects, allowing the person in charge to centralize data for better management between projects and employees.

During development, we used technologies already covered during curricular units, such as SQL and APIs, but we also used modern technologies that I was not familiar with, such as Dapper, which handles all the connections between the API results and the database.

This internship, which lasted 250 hours, was important for the student's learning and development, to better understand the job market and gain technical knowledge.

Keywords: TeamWork API, SQL Server, Dapper, .NET, C#

Índice

1. INTRODUÇÃO 9
2. ENQUADRAMENTO DO ESTÁGIO 11
2.1. Caracterização da Entidade Acolhedora 11
2.2. Contexto e Objetivos do Estágio 13
3. TECNOLOGIAS E FERRAMENTAS UTILIZADAS 14
4. ATIVIDADES DESENVOLVIDAS 17
4.1. Planeamento 17
4.2. Execução 18
4.3. Dificuldades e Soluções 26
5. COMPETÊNCIAS DESENVOLVIDAS 28
6. CONCLUSÕES E REFLEXÃO CRÍTICA 29
7. REFERÊNCIAS BIBLIOGRÁFICAS 30
8. ANEXOS 32
Apêndice 1 32

Índice de tabelas

Tabela 1. Cronograma de tarefas do estágio curricular 17

Índice de figuras

Figura 1. Gráfico de Ferramentas 16
Figura 2. Autenticação no appsettings 18
Figura 3. Variável do Tempo 18
Figura 4. Chamada Principal com Paginação 19
Figura 5. Ordenação e Métodos Auxiliares 19
Figura 6. Lista de Objetos 20
Figura 7.Apagar e Inserir Dados com Dapper 21
Figura 8. Dapper Insert Table 21
Figura 9. Dapper Trunkate Table 22
Figura 10. Base de Dados 22
Figura 11. Serviço SOA 23
Figura 12. Exemplo do BPEL 24
Figura 13. Criação do Calendário 24
Figura 14. Notificações 25
Figura 15. Diagrama de Blocos 25

Lista de Siglas e Abreviaturas

API - Application programming interface

BPEL - Business Process Execution Language

FSD – Functional Solution Designer

ORM - Object Relational Mapping

SQL - Structured Query Language

SOA - Service-oriented architecture

UMAIA – Universidade da Maia

1. INTRODUÇÃO

Pertinência do tema

O presente relatório descreve as atividades desenvolvidas no âmbito do estágio curricular da Licenciatura em Informática da Universidade da Maia, realizado na empresa Garland, uma empresa multinacional especializada em soluções de transporte terrestre, aéreo e marítimo. O estágio decorreu entre março e maio de 2025, com uma carga horária de 250 horas. O principal objetivo foi integrar uma equipa de desenvolvimento e utilizar conhecimentos adquiridos ao longo da formação académica, mais concretamente das unidades curriculares de desenvolvimento web I e II, base de dados e complementos de base de dados.

Neste estágio, foi desenvolvida uma integração com a API do TeamWork, com o objetivo de permitir à Garland aceder de forma fácil e interna a dados relevantes da sua ferramenta de gestão de projetos. Os dados importantes foram encaminhados por dapper para uma base de dados SQL através de um job automático.

Estrutura do Relatório

Na estrutura deste relatório é abordado o tema do enquadramento do estágio, onde é caracterizada a entidade acolhedora e descritos os objetivos do estágio.

Ultrapassando este ponto, foram discutidas as ferramentas utilizadas e uma pequena comparação com outras ferramentas no mercado que podiam ter sido utilizadas neste estágio.

De seguida, o relatório refere as atividades desenvolvidas, nomeadamente o planeamento, a execução, as dificuldades e as respetivas soluções.

Após as atividades desenvolvidas, nomeia-se as competências desenvolvidas durante as 250 horas do estágio curricular.

Por fim, efetua-se uma pequena reflexão crítica sobre o estágio e relata-se algumas conclusões.

2. ENQUADRAMENTO DO ESTÁGIO

2.1. Caracterização da Entidade Acolhedora

A Garland é uma empresa com uma história impressionante de 249 anos, sendo uma das mais antigas em Portugal. Ao longo do tempo, tem vindo a afirmar-se em vários setores, com especial destaque para a logística, os transportes e a navegação. (Garland, s.d.)

Com sede em Portugal, onde também se encontra a maioria dos seus colaboradores, a Garland conta com cinco escritórios no país, um em Espanha e outro em Marrocos. Em território nacional, dispõe de 154.000 m² de área de armazém, o que lhe permite oferecer soluções completas de gestão da cadeia de abastecimento em regime de outsourcing. (Garland, s.d.)

Na área dos transportes, a Garland garante soluções à medida, disponibilizando transporte aéreo, terrestre e marítimo a partir de qualquer um dos seus escritórios. Já no setor da navegação, destaca-se pelo serviço especializado que oferece entre todos os continentes, sendo ainda membro exclusivo da Multiport, uma rede internacional presente em mais de 1.500 portos. (Garland, s.d.)

A administração da empresa é constituída por três administradores, responsáveis pela gestão estratégica da empresa a nível global, nomeadamente:

- **Mark Dawson**, Presidente Executivo do Grupo, é responsável pelas áreas de Navegação, Finanças, Recursos Humanos e Qualidade;
- **Ricardo Sousa Costa** lidera as áreas de Logística, Marketing e Infraestruturas;
- **Giles Dawson** assume a responsabilidade pelos Transportes e pelos Sistemas de Informação. (Garland, s.d.)

Na Garland, mais em concreto o departamento de Sistemas de Informação está dividido em 6 equipas, nomeadamente:

1. Equipa de “Helpdesk” – O objetivo desta equipa é fornecer apoio de software e hardware a toda a empresa.
2. Equipa de Infraestrutura – Esta equipa gera toda a infraestrutura da Garland, assim como os servidores, “backups” e o uso geral do equipamento informático.
3. Equipa de Communications & Security – Gere toda a área de redes e cibersegurança.
4. Equipa de Desenvolvimento aplicacional – Esta equipa, onde o estagiário se integrou assegura toda a parte de desenvolvimento de novos projetos e também fornece suporte aplicacional.
5. Equipa de Integrações – Trabalha junto com a equipa de desenvolvimento, mas neste caso foca-se mais nas integrações com clientes e sistemas externos.
6. Equipa de FSD – São os “project owners” e tem como função definir com as equipas o software que é necessário para novos projetos e trabalham juntos com a equipa de desenvolvimento para planear e executar o projeto.

2.2. Contexto e Objetivos do Estágio

O desenvolvimento do projeto decorreu ao longo de várias semanas, tendo como metodologia principal a realização de "*commits*" frequentes no repositório Bitbucket e reuniões regulares ao fim de cada tarefa concluída, com uma periodicidade média semanal. Estas reuniões serviram para validar o progresso, identificar melhorias e consolidar boas práticas com o apoio da equipa.

O trabalho foi inicialmente guiado por uma abordagem exploratória, onde o programador aplicava conhecimentos prévios, complementando com apoio de ferramentas de inteligência artificial para encontrar soluções adequadas. Após esta fase, o progresso era validado em conjunto com o supervisor, que forneciam feedback sobre o que poderia ser melhorado e como.

As tarefas foram organizadas de forma progressiva, partindo de desafios mais simples até atingir funcionalidades mais complexas e específicas, como a integração com a base de dados através do Dapper.

3. TECNOLOGIAS E FERRAMENTAS UTILIZADAS

Bitbucket

Bitbucket é uma plataforma de controlo de versões baseada em Git, utilizada para armazenar e gerir o código-fonte do projeto. Durante o estágio, foram realizados "*commits*" quase diários para garantir o acompanhamento contínuo da evolução do trabalho e facilitar a colaboração e revisão por parte da equipa. Outra ferramenta que era possível de ser utilizada era o GitHub que é um serviço baseado em nuvem que hospeda um sistema de controle de versão. Foi escolhido o Bitbucket devido à uniformidade da empresa e ao facto de a empresa já conter o resto das aplicações em Bitbucket. (Bitbucket, s.d.) (Github, s.d.)

API do TeamWork

A API do TeamWork foi a principal interface externa integrada no projeto. Permitiu aceder a dados de tarefas, projetos e registos de tempo da plataforma TeamWork. A sua utilização exigiu a implementação de chamadas HTTP, tratamento de autenticação e adaptação a mudanças na estrutura da API ocorridas durante o desenvolvimento. (TeamWork, s.d.)

SQL Server

Foi utilizado o SQL Server como sistema de gestão de base de dados relacional (SGBD) para armazenar e organizar os dados obtidos da API do TeamWork. Uma outra ferramenta que podia ter sido utilizada era o MariaDB, um sistema open-source baseado no MySQL, desenvolvido para ser rápido, escalável e robusto. Optou-se pelo SQL Server, uma vez que já era utilizado noutras APIs da empresa, garantindo maior uniformidade e facilidade de manutenção. Foram criadas tabelas específicas para

tarefas e registos de tempo, com o objetivo de facilitar a análise e persistência dos dados. (SQL_Server, s.d.) (MariaDB, s.d.)

Dapper

Dapper é um micro ORM (Object-Relational Mapper) para .NET, que permite executar comandos SQL de forma leve, mantendo um bom desempenho. Foi utilizado para mapear os dados entre a aplicação e a base de dados SQL Server, possibilitando uma integração eficiente. Apesar da existência de alternativas como o Entity Framework Core, que também realiza o mapeamento objeto-relacional, optou-se pelo Dapper devido à experiência prévia da equipa com esta ferramenta, o que facilitou o desenvolvimento e reduziu o tempo necessário para adaptação. (Dapper, s.d.) (Entity Framework Core, s.d.)

Swagger

Swagger é uma ferramenta de documentação automática de APIs. Foi utilizada para expor de forma clara e interativa os "*endpoints"* da aplicação, facilitando os testes e a validação das funcionalidades implementadas. (Swagger, s.d.)

Documentação e Ficheiros de Configuração

- **README**: Inclui instruções técnicas sobre a configuração e utilização do projeto, facilitando o entendimento e a colaboração futura.
- **Testes Unitários**: Foram criados para validar funcionalidades específicas e garantir a estabilidade do código.
- **.editorconfig / .gitignore**: Ficheiros de configuração usados para padronizar o estilo de código entre diferentes ambientes de desenvolvimento e evitar o versionamento de ficheiros desnecessários no repositório.

Tal como se pode validar na Figura 1 estas foram as ferramentas e tecnologias utilizadas durante o estágio.

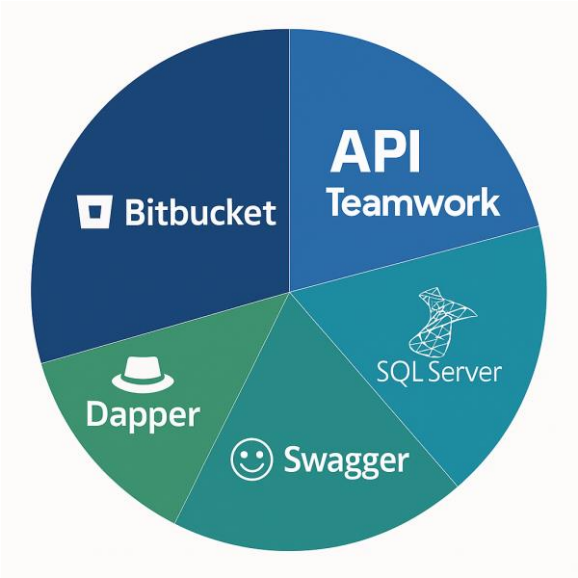

Figura 1. Gráfico de Ferramentas

4. ATIVIDADES DESENVOLVIDAS

4.1. Planeamento

As tarefas foram organizadas progressivamente, com validação constante por parte da equipa.

A metodologia adotada incluiu uma abordagem exploratória inicial, seguida de melhoria contínua com base em feedbacks.

As funcionalidades foram planeadas (ver Tabela 1) com base em prioridades, começando por testes simples com endpoints e evoluindo até à integração completa e refatoração do código.

Tabela 1. Cronograma de tarefas do estágio curricular

Semanas / Tarefas	**Fevereiro**				**Março**				**Abril**				**Maio**			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Estudar a Api do TeamWork																
Testes de endpoints																
Criação das Tabelas DB																
Elaboração das chamadas principais																
Criação de métodos secundários																
Atualização dos Endpoits																
Implementação das restrições de Tempo																
Resolução de problemas																
Criação do Job																
Pesquisa bibliográfica																
Elaboração do relatório																

4.2. Execução

A configuração inicial do projeto envolveu a reutilização da estrutura base de um projeto anterior da empresa que era a integração da API do SAP. No início foi realizado um primeiro teste com a API do TeamWork, seguido de uma implementação inicial com autenticação "*hardcoded*", como podemos visualizar nos anexos, Apêndice 1. Posteriormente foi reorganizada para adotar boas práticas de código e foi alojada na base appsetting.json, local apropriado para guardar logins e palavras-passe, como demonstra na Figura 2. Por motivos de segurança a autenticação foi escondida.

```
15      "TeamWorkSettings": {
16          "BaseUrl": "https://garland.teamwork.com",
17          "Username": "[redacted]@garland.pt",
18          "Password": "[redacted]"
19      },
```

Figura 2. Autenticação no appsettings

Após o problema de autenticação ter sido resolvido, iniciou-se a fase de realizar as chamadas à API do TeamWork para que seja possível o preenchimento da base de dados. Ambas as chamadas seguiram uma lógica semelhante onde apenas eram diferentes os "*endpoints*" envolvidos.

Na parte inicial do código era definida a variável temporal que os "*endpoints*" tinham de respeitar, como verificamos na Figura 3. Neste caso apenas devolviam tarefas ou tempos dos colaboradores que tivessem sido criados, modificados ou apagados nas últimas 24 horas.

```
DateTime now = DateTime.UtcNow;
var last24Hours = now.AddDays(-1);
string updatedAfterStr = last24Hours.ToString("yyyy-MM-ddTHH:mm:ssZ");
```

Figura 3. Variável do Tempo

Após a definição da variável de tempo foi também importante incorporar a paginação para que todas as tarefas consigam aparecer na base de dados sem problema, visto que a API apenas suporta 250 registos em cada página. Nesta chamada, além do parâmetro para corrigir a paginação, também implementamos o parâmetro "*showDeleted*" para apresentar na base de dados todos os registos incluindo os apagados, como podemos observar na Figura 4.

```
// Paginação
List<InlineResponse20020Todoitems> allRawTasks = [];
int currentPage = 1;
List<InlineResponse20020Todoitems> currentPageTasks;

do
{

    var response = _tasksApiNew.GETTasksJson(page: currentPage, pageSize: 250, showDeleted: "Yes", updatedAfterDate: updatedAfterStr);
    currentPageTasks = response?.TodoItems?.Where(t => t.Id.HasValue).ToList() ?? [];
    if (currentPageTasks.Count > 0)
        allRawTasks.AddRange(currentPageTasks);

    currentPage++;
} while (currentPageTasks.Count == 250);

if (allRawTasks.Count == 0)
{
    Console.WriteLine("Nenhuma tarefa encontrada.");
    return [];
}
```

Figura 4. Chamada Principal com Paginação

Com base na Figura 5 foi necessário ordenar as tarefas e os tempos respetivos por ordem decrescente tendo em conta a data de modificação, ou seja, do mais recente para o mais antigo. Importação de dados auxiliares foi também necessária. Este método contém informação pertinente para as duas tabelas, logo foi aconselhada a utilização de métodos auxiliares para que possam ser importados nas duas chamadas sem a necessidade de colocar mais nenhum "*endpoint*" no corpo principal da chamada, simplificando o processo.

```
// Ordenar as tarefas por data de atualização
var sortedTasks = allRawTasks
    .OrderByDescending(t => DateTime.TryParse(t.LastModifiedOn, out var updated) ? updated : (DateTime.TryParse(t.LastChangedOn, out var changed) ? changed : DateTime.MinValue))
    .ToList();

// Dados auxiliares
var projetos = _projectsApi.GETAllProjectsApi()?.Projects ?? [];
var todoItemsById = LoadTodoItemsByProject(projetos, _tasksApiNew);
var taskListNames = LoadTaskListNames(todoItemsById, _tasklistApi);
var projectCategories = LoadProjectCategories(projetos, _projectCategoriesApi);
var parentCategories = LoadParentCategories(projetos, (ProjectCategoriesApi)_projectCategoriesApi);
var (taskCompanies, taskTags) = LoadCompanyAndTags(todoItemsById);
var (todoItems20020, taskUpdaters) = LoadTodoItems20020AndUpdatersTasks(_tasksApiNew, updatedAfterStr);
```

Figura 5. Ordenação e Métodos Auxiliares

De seguida, é contruída a lista final de objetos, onde são preenchidos todos os campos da base de dados, por um lado com a resposta do "*endpoint*" principal e por outro lado com os métodos auxiliares, podemos comprovar isso observando a Figura 6.

```
var taskList = new List<TeamWorkTaskAndProject>();

foreach (var task in sortedTasks)
{
    int taskId = task.Id ?? 0;
    if (taskId == 0) continue;

    todoItemsById.TryGetValue(taskId, out var todoItem);
    var project = projetos.FirstOrDefault(p => p.Id == todoItem?.ProjectId);

    taskCompanies.TryGetValue(taskId, out var companyName);
    taskTags.TryGetValue(taskId, out var tags);
    projectCategories.TryGetValue(project?.Id ?? 0, out var projectCategoryName);
    parentCategories.TryGetValue(project?.Id ?? 0, out var parentCategoryName);
    taskUpdaters.TryGetValue(taskId, out var responsible);
    taskListNames.TryGetValue(todoItem?.TodoListId ?? 0, out var taskListName);

    var (parentTaskId, parentTask) = GetParentTaskInfo(taskId);

    taskList.Add(new TeamWorkTaskAndProject
    {
        TaskId = taskId,
        CreateDate = DateTime.TryParse(task.CreatedOn, out var createdOn) ? createdOn : null,
        UpdateDate = DateTime.TryParse(task.LastModifiedOn, out var lastChangedOn) ? lastChangedOn : null,
        StartDate = ParseDateFromCompactFormat(task.StartDate),
        DueDate = ParseDateFromCompactFormat(task.DueDate),
        EndDate = ParseDate(project?.EndDate),
        EstimatedMinutes = task.EstimatedMinutes,
        ProjectId = project?.Id,
        Project = project?.Name,
        ProjectCategory = projectCategoryName ?? string.Empty,
        ParentCategory = parentCategoryName,
        Company = companyName ?? string.Empty,
        TaskList = taskListName,
        Task = task.Content,
        ParentTaskId = parentTaskId ?? 0,
        ParentTask = parentTask ?? string.Empty,
        Tags = tags ?? string.Empty,
        BoardName = task.BoardColumn?.Name ?? string.Empty,
        Responsible = responsible ?? string.Empty,
        Status = task.Status,
        Priority = task.Priority,
        Progress = task.Progress,

    });
}
```

Figura 6. Lista de Objetos

Por fim, como podemos validar na Figura 7 foi necessário utilizar instruções para limpar a tabela já preenchida da última vez e procedeu-se à inserção dos novos dados com Dapper, devido ao facto de ser preciso que a base de dados seja apagada para não haver conflitos quando novos dados forem inseridos.

```
        await repository.TruncateTableTaskList();
        await repository.InsertTeamWorkTasksAndProjectsAsync(taskList);

        Console.WriteLine($">>> Total de Tasks inseridas: {taskList.Count}");
        return taskList;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro ao buscar as Tasks: {ex.Message}");
        return [];
    }
}
```

Figura 7.Apagar e Inserir Dados com Dapper

Na fase de estruturação da base de dados, foram criadas tabelas no SQL Server e utilizou-se o Dapper para garantir um mapeamento simples e eficiente entre os dados e os objetos da aplicação, percebemos isso observando a Figura 8. Assim não foi necessário utilizar um método de mapeamento manual, que consiste em identificar o resultado proveniente da API para fazer o encaminhamento manual para cada parâmetro da base de dados.

```
1 reference
21  public async Task InsertTeamWorkTasksAndProjectsAsync(IEnumerable<TeamWorkTaskAndProject> tasks)
22  {
23      const string query = @"
24          INSERT INTO dbo.TeamWorkTaskList (
25              TaskId, CreateDate, UpdateDate, StartDate, DueDate, EndDate,
26              EstimatedMinutes, ProjectId, Project, ProjectCategory, ParentCategory,
27              Company, TaskList, Task, ParentTaskId, ParentTask, Tags, BoardName,
28              Responsible, Status, Priority, Progress
29          )
30          VALUES (
31              @TaskId, @CreateDate, @UpdateDate, @StartDate, @DueDate, @EndDate,
32              @EstimatedMinutes, @ProjectId, @Project, @ProjectCategory, @ParentCategory,
33              @Company, @TaskList, @Task, @ParentTaskId, @ParentTask, @Tags, @BoardName,
34              @Responsible, @Status, @Priority, @Progress
35          )";
36
37      try
38      {
39          using var db = Connection;
40          await db.ExecuteAsync(query, tasks);
41      }
42      catch (Exception ex)
43      {
44          Console.WriteLine($"Erro ao inserir dados no banco: {ex.Message}");
45      }
46  }
47
```

Figura 8. Dapper Insert Table

Ao estabelecer a ligação com a base de dados, sempre que é efetuada uma chamada à API, em primeiro lugar é utilizado o método "*Trunkate*" na tabela respetiva, para que a mesma seja apagada e só depois é utilizado o método "*Insert*" para que os dados sejam inseridos na tabela. Assim sendo, garante-se a não existência de dados repetidos, nem a existência de falhas e erros na inserção dos dados nas respetivas tabelas, como podemos observar na Figura 9.

```
1 reference
public async Task TruncateTableTaskList()
{
    const string query = "TRUNCATE TABLE dbo.TeamWorkTaskList";
    try
    {
        using var db = Connection;
        await db.ExecuteAsync(query);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro ao truncar tabela: {ex.Message}");
    }
}
```

Figura 9. Dapper Trunkate Table

Depois de utilizarmos o Dapper para encaminhar os resultados da API para a base de dados estes aparecem na Figura 10, como podemos verificar. Todos os parâmetros da base de dados SQL foram preenchidos com respostas da API, mas em alguns casos esses resultados vêm vazios porque não são preenchidos na plataforma do TeamWork.

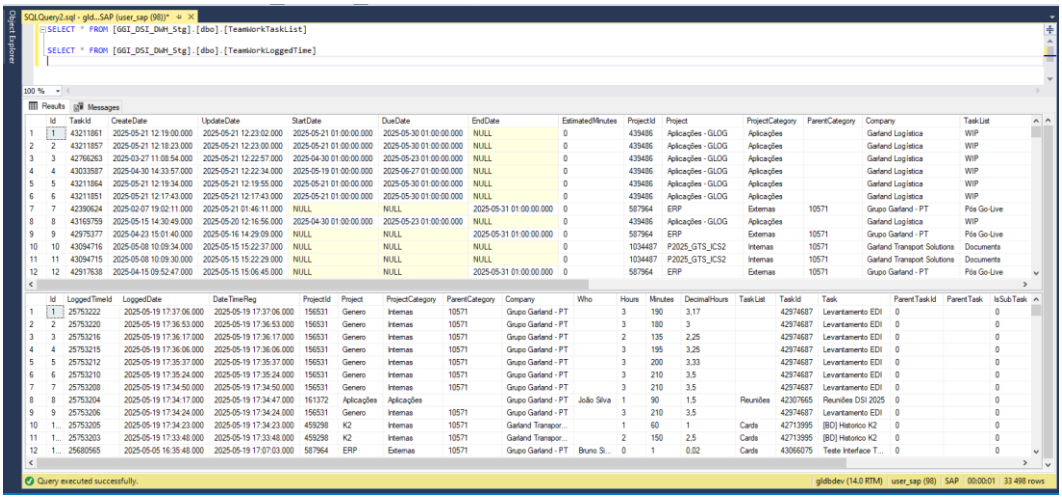

Figura 10. Base de Dados

A organização e estruturação do código seguiram uma arquitetura modular, distribuída em quatro camadas principais, nomeadamente: Api, Application, Domain e Infrastructure. Também foram incluídos a documentação e os ficheiros de configuração para facilitar a manutenção e expansão do projeto.

Execução do serviço SOA com o objetivo de uniformizar e agrupar as chamadas REST à API do TeamWork. Conforme ilustrado na Figura 11, do lado esquerdo estão os serviços expostos externamente, os quais são intermediados por dois componentes internos. (SOA, s.d.)

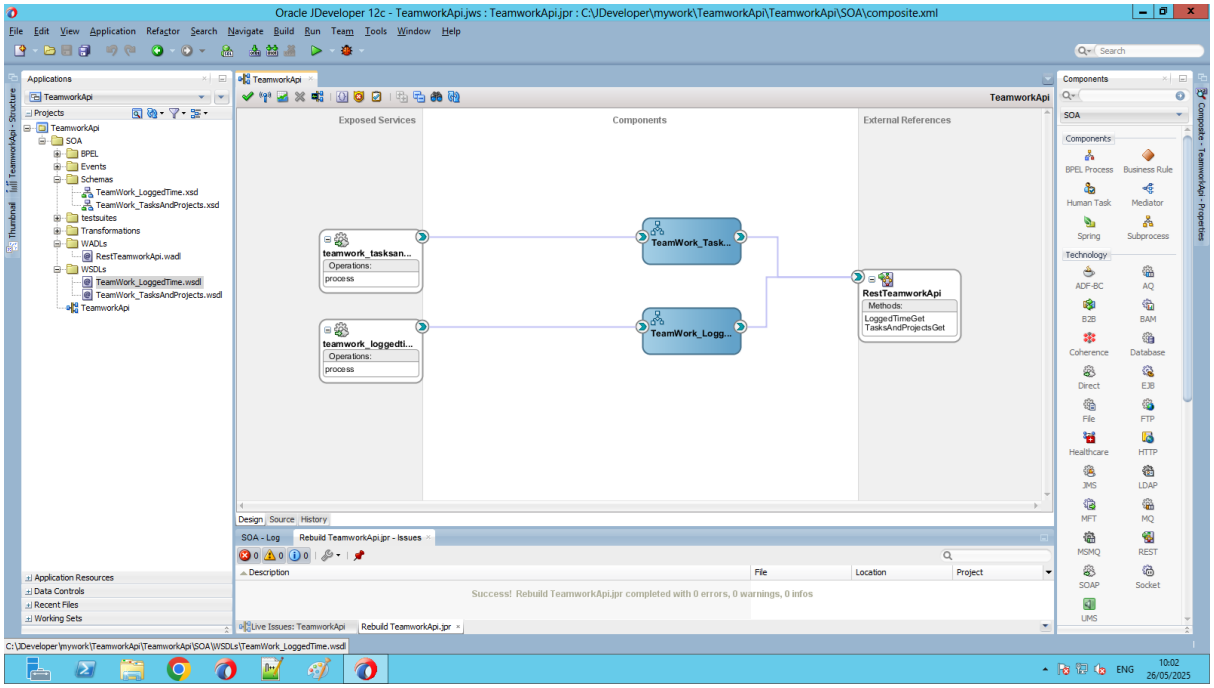

Figura 11. Serviço SOA

Na Figura 12 temos um exemplo de configuração de como o BPEL ficou implementado, neste caso é o exemplo da tabela das Tarefas, mas a tabela dos Tempos está estruturada da mesma forma.

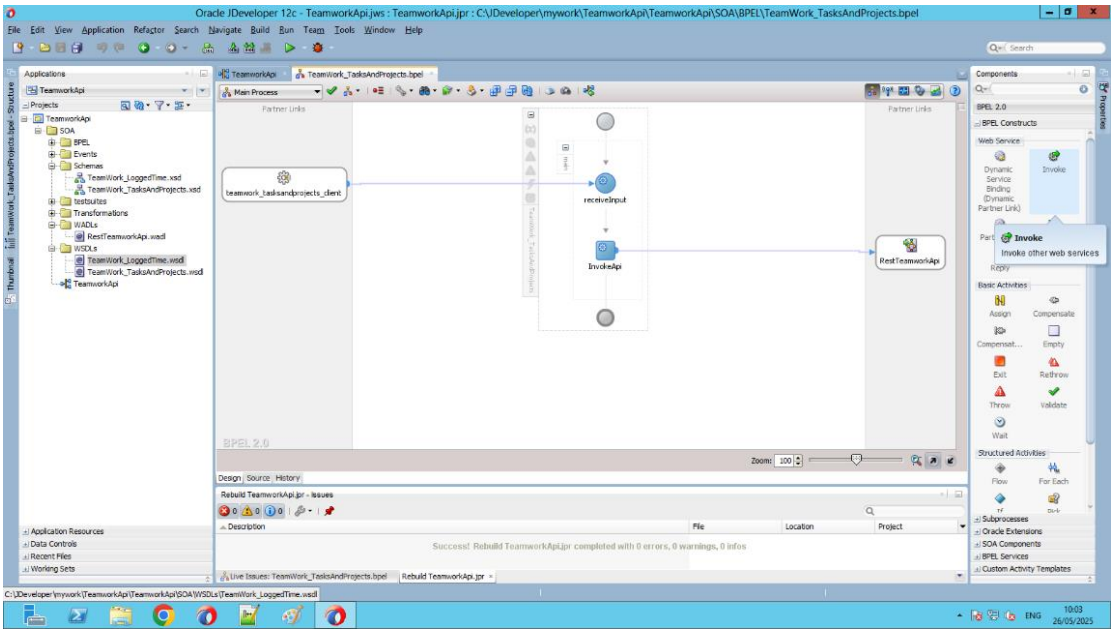

Figura 12. Exemplo do BPEL

Na Figura 13 conseguimos observar como foi implementado um *job* que atualiza diariamente as tabelas às 7:00 e às 7:30, garantindo que os dados se encontram atualizados antes do início do horário laboral dos colaboradores. Adicionalmente, foi integrado um sistema de notificações que envia um alerta por correio eletrónico sempre que ocorre um erro durante o processo, como podemos observar na Figura 14.

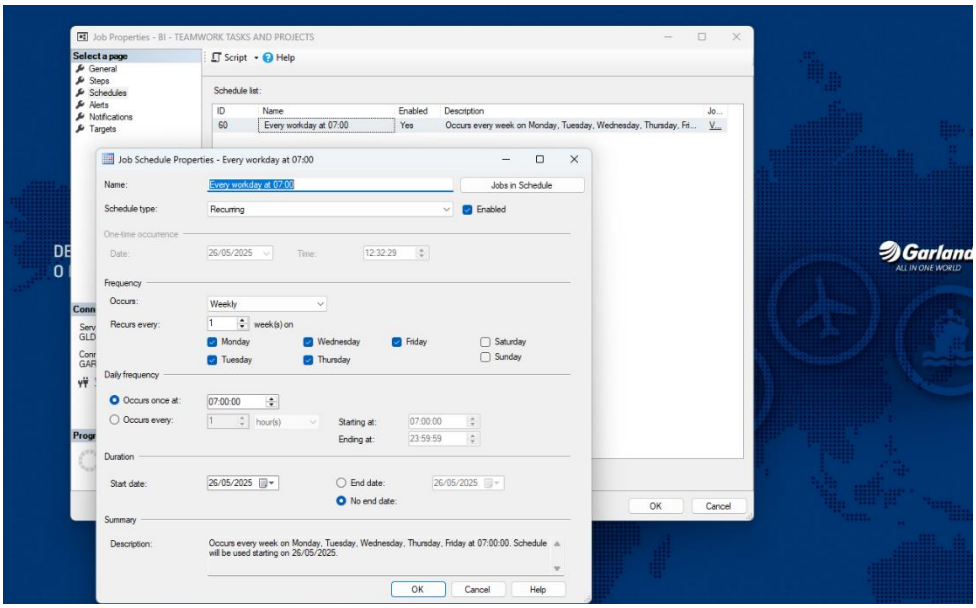

Figura 13. Criação do Calendário

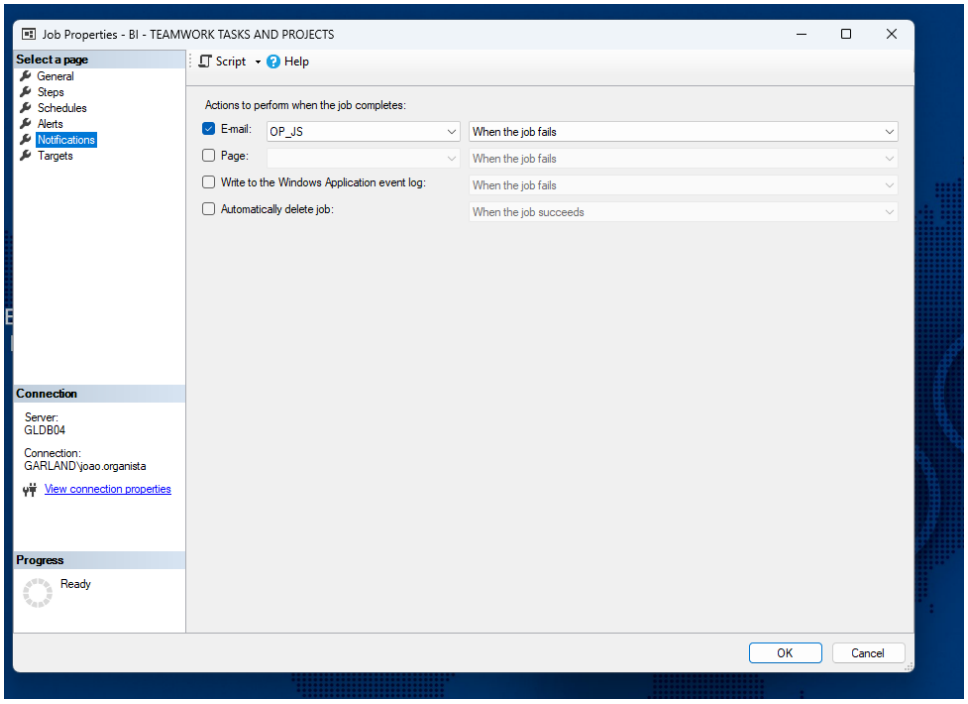

Figura 14. Notificações

Por fim, a organização e reutilização do código abrangeram a criação de métodos reutilizáveis, a extração de dados a partir de múltiplos endpoints, a centralização da configuração do Dapper e a aplicação de um modelo de tratamento de erros baseado em práticas utilizadas anteriormente.

Com base na Figura 15, que representa um diagrama de blocos, é possível observar as componentes que constituem a aplicação e as suas respetivas ações.

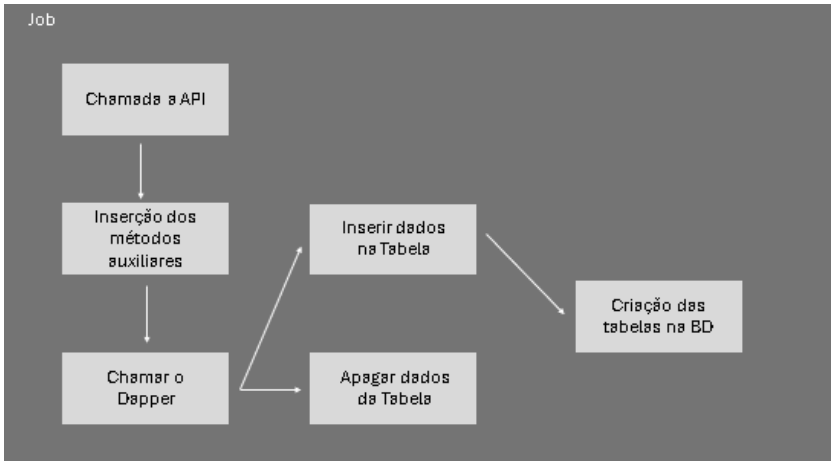

Figura 15. Diagrama de Blocos

4.3. Dificuldades e Soluções

Durante o desenvolvimento do projeto, enfrentaram-se várias dificuldades técnicas e estruturais, especialmente relacionadas com a integração da API e o entendimento dos procedimentos funcionais da empresa.

Numa fase inicial, o principal desafio foi compreender a estrutura do código existente, nomeadamente como é que a empresa organizava a comunicação com as APIs. O trabalho começou com base numa API já desenvolvida, a API do SAP e, a partir daí, foi necessário adaptar e compreender a lógica implementada para responder aos novos requisitos impostos pela API do TeamWork.

Outro obstáculo relevante foi perceber quais os parâmetros da base de dados que eram relevantes e como estes se relacionavam com os dados obtidos através das chamadas às APIs. Em muitos casos, os dados retornados não correspondiam diretamente ao que era pretendido armazenar, o que obrigava a realizar uma busca e posterior correspondência manual, tendo em conta o objetivo final da empresa ao registar essas informações.

Adicionalmente, surgiram dificuldades quando certos parâmetros esperados vinham "vazios" nas respostas da API, o que complicava ainda mais a tarefa de mapear os dados corretamente para a base de dados. Para contornar estas limitações, foi necessário recorrer a chamadas complementares de outros "*endpoints"* para obter os dados em falta.

Durante o processo de desenvolvimento, a API do TeamWork foi atualizada, o que representou um novo desafio. Esta atualização não só eliminou alguns "*endpoints*" anteriormente abrangentes, como também alterou a estrutura das chamadas e das respostas. Assim sendo, foi necessário readaptar o código, reformular as chamadas e rever toda a lógica de processamento de dados.

Outro aspeto essencial do projeto foi a curva de aprendizagem da aplicação do Dapper, uma biblioteca ORM para .NET que permite executar comandos SQL de forma simples e eficiente. O Dapper funciona como uma camada intermédia entre o código e a base de dados, facilitando a conversão de objetos em registos e vice-versa. (Dapper, s.d.)

Também foi essencial enfrentar o desafio de extrair apenas os dados relevantes de cada "*endpoint*" e consolidá-los numa única estrutura. Foi necessário cruzar dados provenientes de múltiplas fontes e ajustar para que todos os parâmetros fossem preenchidos com dados relevantes para a empresa.

Finalmente, houve uma evolução no uso de dicionários em C# como ferramenta para organizar e aceder rapidamente a dados em memória. Os dicionários revelaram-se particularmente úteis para melhorar a legibilidade e organização do código, mas também a não repetição dos mesmos "*endpoints*". (Dicionários em C#, s.d.)

Todas estas dificuldades foram ultrapassadas gradualmente através da consulta da documentação e do apoio da equipa. Esta experiência contribuiu significativamente para o desenvolvimento técnico do estagiário ao longo do projeto.

5. COMPETÊNCIAS DESENVOLVIDAS

Durante a realização deste projeto, desenvolvi diversas competências importantes tanto a nível técnico quanto profissional, visto que foi o meu primeiro contacto com o mercado de trabalho. Tive a oportunidade de compreender melhor o funcionamento de uma empresa, o que me permitiu adaptar ao ambiente de trabalho e às suas exigências. Aprendi nomeadamente a cumprir tarefas dentro dos prazos estabelecidos, uma capacidade essencial para a boa gestão do tempo e responsabilidade no contexto profissional.

Do ponto de vista técnico, aprofundei o meu conhecimento na estruturação de código, tornando-o mais organizado e eficiente. Compreendi também como realizar um job de forma eficaz, desde a sua conceção até à execução. Reforcei os conhecimentos previamente adquiridos nas unidades curriculares de Desenvolvimento Web I e II, especialmente no que toca à realização de chamadas a APIs. Além disso, aprendi a utilizar dicionários como ferramenta para criar métodos auxiliares, o que se revelou extremamente útil para reutilizar código em diferentes chamadas.

Foi também importante o aprimoramento do meu conhecimento em bases de dados, o que incluiu a criação e gestão de tabelas, bem como a manipulação eficiente dos dados. Conhecimentos estes, previamente adquiridos nas unidades curriculares de Base de dados I e complementos de base de dados.

Por fim, tive o primeiro contacto com o Dapper, uma ferramenta leve e eficaz para o mapeamento de dados em .NET, que me permitiu realizar operações com a base de dados de forma mais direta e eficaz.

6. CONCLUSÕES E REFLEXÃO CRÍTICA

A realização deste projeto permitiu-me aplicar e consolidar conhecimentos adquiridos ao longo do curso, enquanto desenvolvi novas competências técnicas e profissionais. Fui desafiado semanalmente a evoluir e esses mesmos desafios contribuíram significativamente para o meu crescimento enquanto programador. Além disso, aprendi a trabalhar com prazos, a colaborar de forma mais eficaz e a adaptar-me ao funcionamento de uma empresa, o que representa uma mais-valia para a minha inserção no mercado de trabalho.

Ao longo do projeto, enfrentei vários desafios técnicos que exigiram adaptação e aprendizagem contínua. Um dos principais obstáculos foi compreender o funcionamento da API do TeamWork e garantir a integridade dos dados recebidos, o que exigiu análise detalhada da documentação. Também percebi a importância de boas práticas na organização do código, algo que inicialmente negligenciei ao optar por soluções mais rápidas, mas que posteriormente organizei com base nos princípios adquiridos na empresa com ajuda constante.

Adicionalmente, percebi que, embora o conhecimento técnico seja fundamental, a capacidade de cumprir prazos, comunicar de forma eficaz e adaptar-me a novas ferramentas e ambientes são igualmente cruciais. Esta experiência destacou a importância da aprendizagem autónoma e contínua, uma vez que nem todos os problemas têm soluções diretas ou previamente ensinadas.

Em suma, este projeto foi uma oportunidade valiosa para refletir sobre o meu percurso académico, identificar áreas de melhoria e consolidar a minha motivação para continuar a evoluir como profissional.

7. REFERÊNCIAS BIBLIOGRÁFICAS

Bitbucket. (s.d.). Fonte: https://bitbucket.org/product/

Dapper. (s.d.). Fonte: https://www.learndapper.com/

Dicionários em C#. (s.d.). Fonte: https://www.macoratti.net/19/05/c_dicio1.htm

Entity Framework Core. (s.d.). Fonte: https://learn.microsoft.com/en-us/ef/core/

Garland. (s.d.). Fonte: https://www.garland.pt/pt/

Github. (s.d.). Fonte: https://www.hostinger.com/br/tutoriais/o-que-github

MariaDB. (s.d.). Fonte: https://mariadb.org/

SOA. (s.d.). Fonte: https://pt.wikipedia.org/wiki/Service-oriented_architecture

SQL_Server. (s.d.). Fonte: https://pt.wikipedia.org/wiki/SQL_Server_Management_Studio

Swagger. (s.d.). Fonte: https://swagger.io/

TeamWork. (s.d.). Fonte: https://www.teamwork.com/

Página intencionalmente deixada em branco.

8. ANEXOS

Apêndice 1

Foi feito um teste para verificar se é possível obter dados da API do TeamWork, como tarefas e tempos registados. Usou-se um código com autenticação básica ("*username*" e "*password*" em Base64). O código validava as credenciais, fazia um pedido GET à API e tratava possíveis erros (como autenticação falhada, recurso inexistente ou problemas de rede). O teste foi bem-sucedido e confirmou a viabilidade da integração com a API.

```
try

{

    string username = "";

    string password = "";



    if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))

    {

        return Unauthorized("Credenciais de autenticação não fornecidas.");

    }



    string credentials =
Convert.ToBase64String(Encoding.UTF8.GetBytes($"{username}:{password}"));



    using var client = _httpClientFactory.CreateClient();

    var request = new HttpRequestMessage(HttpMethod.Get,
"https://garland.teamwork.com/projects/api/v3/projects.json");



    request.Headers.Authorization = new AuthenticationHeaderValue("Basic",
credentials);



    var response = await client.SendAsync(request);

    if (!response.IsSuccessStatusCode)
```

```
    {
        return response.StatusCode switch
        {
            HttpStatusCode.BadRequest => BadRequest("Requisição inválida para o TeamWork."),
            HttpStatusCode.Unauthorized => Unauthorized("Não autorizado. Verifique suas credenciais."),
            HttpStatusCode.Forbidden => StatusCode(403, "Acesso proibido ao recurso solicitado."),
            HttpStatusCode.NotFound => NotFound("O recurso solicitado não foi encontrado."),
            HttpStatusCode.InternalServerError => StatusCode(500, "Erro interno no servidor do TeamWork."),
            _ => StatusCode((int)response.StatusCode, "Erro desconhecido ao buscar projetos.")
        };
    }

    var content = await response.Content.ReadAsStringAsync();
    return Content(content, "application/json");
}
catch (HttpRequestException ex)
{
    _logger.LogError(ex, "Erro de conexão ao buscar projetos do TeamWork.");
    return StatusCode((int)HttpStatusCode.ServiceUnavailable, "Erro de conexão ao buscar projetos.");
}
catch (TaskCanceledException ex)
{
    _logger.LogError(ex, "A requisição ao TeamWork expirou.");
    return StatusCode((int)HttpStatusCode.RequestTimeout, "A requisição ao TeamWork expirou.");
```

```
}

catch (Exception ex)

{

    _logger.LogError(ex, "Erro inesperado ao buscar projetos do TeamWork.");

    return StatusCode((int)HttpStatusCode.InternalServerError, "Erro inesperado ao
buscar projetos.");

}
```

UNIVERSIDADE
DA MAIA