



**INETE**  
Instituto de Educação Técnica

# Relatório de Prova de Aptidão Profissional

## Power Network-Timesheet & Project

**Gabriel Teixeira Pacheco**

N.º 190080, Turma: TGPSI19/02,

Técnicos de Gestão e Programação de Sistemas Informáticos

Lisboa, 15 de julho de 2022

# Índice

Lista de programas.....	i
Introdução.....	1
1 - Estágio .....	3
1.1. - Empresa de estágio .....	3
1.2. - Departamento de estágio .....	3
1.3. - Atividades durante o estágio .....	3
1.4. - Análise Crítica .....	4
2 - Fundamentação Técnico-Científica .....	6
2.1. – Lista de conceitos.....	6
.NET Framework .....	6
“aplicativo Web ASP.NET (.NET Framework)” .....	6
API (Application Programming Interface) - Interface de Programação de Aplicação .....	6
Azure websites.....	6
Base de Dados .....	6
Aplicações Canvas .....	7
Connectionstring .....	7
Conectores Personalizados .....	7
Entity Framework Core .....	7
JSON .....	7
Expressão Lambda .....	7
Linguagem de programação .....	7
Linguagem de programação c# .....	8
Microsoft Teams .....	8
Microsoft SQL Server Management Studio.....	8
Power Apps.....	8
Programa .....	8
Programação (em informática).....	8
SQL.....	9
Visual Studio .....	9
Worklog.....	9
2.2. - Descrição do sistema de informação – Power Network-Timesheet & Project .....	9
2.2.1. – Conceção da Base de Dados .....	10
2.2.2. – Criação do <i>Repository</i> e do <i>Controller</i> .....	14

2.2.3. – Formação do Conector Personalizado .....	15
2.2.4. – Personalização do sistema de informação através da aplicação <i>Canvas</i> do <i>Power Apps</i> .....	18
3 – Apresentação do sistema de informação Power Network-Timesheet & Project .	21
Condicionalismos .....	28
Conclusão.....	29
Webgrafia .....	30
Anexos .....	31
Anexo A: appsettings.json .....	31
Anexo B: Program.cs .....	31
Anexo C: _DbContext.cs .....	32
Anexo D: Activity.cs.....	32
Anexo E: Activity_File.cs .....	33
Anexo F: ActivityState.cs.....	33
Anexo G: ActivityType.cs .....	34
Anexo H: BillingType.cs .....	34
Anexo I: Client.cs .....	34
Anexo J: Expense.cs.....	35
Anexo K: Expense_File.cs .....	35
Anexo L: ExpenseState.cs .....	36
Anexo M: ExpenseType.cs.....	36
Anexo N: File.cs .....	36
Anexo O: FileContent.cs .....	37
Anexo P: FileContentType.cs.....	37
Anexo Q: Line.cs .....	37
Anexo R: LineCity.cs .....	38
Anexo S: LineType.cs .....	38
Anexo T: Project.cs .....	39
Anexo U: ProjectState.cs .....	39
Anexo V: Team.cs .....	39
Anexo W: User.cs.....	40
Anexo X: User_Activity.cs .....	40
Anexo Y: UserFunction.cs.....	41
Anexo Z: Worklog.cs .....	41
Anexo AA: WorklogState.cs .....	42

Anexo AB: ActivityHours.cs .....	42
Anexo AC: ActivityIdName.cs .....	42
Anexo AD: ActivityInfo.cs .....	42
Anexo AE: ClientEmailName.cs .....	43
Anexo AF: Date.cs .....	43
Anexo AG: ProjectHours.cs .....	43
Anexo AH: ProjectInfo.cs .....	43
Anexo AI: ProjectIdName.cs .....	44
Anexo AJ: TeamInfo.cs .....	44
Anexo AK: TimesheetUserInfo.cs .....	44
Anexo AL: TimesheetWorklog.cs .....	44
Anexo AM: UserActivityWorklogs.cs .....	44
Anexo AN: UserInfo.cs .....	45
Anexo AO: WorklogCompleteInfo.cs .....	45
Anexo AP: WorklogInfo.cs .....	45
Anexo AQ: TimesheetRepository.cs – ITimesheetRepository .....	45
Anexo AR: TimesheetRepository.cs – variáveis e default-constructor .....	46
Anexo AS: TimesheetRepository.cs – GetUserId .....	46
Anexo AT: TimesheetRepository.cs – GetWorklog .....	47
Anexo AU: TimesheetRepository.cs – CreateWorklog .....	47
Anexo AV: TimesheetRepository.cs – CreateWorklogByPeriod .....	48
Anexo AW: TimesheetRepository.cs - UpdateWorklog .....	49
Anexo AX: TimesheetRepository.cs - DeleteWorklog .....	50
Anexo AY: TimesheetRepository.cs – GetActivityUser .....	50
Anexo AZ: TimesheetRepository.cs – GetProjectUser .....	51
Anexo BA: TimesheetRepository.cs – GetUserWeekWorklog .....	52
Anexo BB: TimesheetRepository.cs – GetActivitiesInfo .....	53
Anexo BC: TimesheetRepository.cs – GetProjectInfo .....	54
Anexo BD: TimesheetRepository.cs – GetBillingTypes .....	56
Anexo BE: TimesheetRepository.cs – GetWorklogState .....	56
Anexo BF: TimesheetRepository.cs – AddDays .....	57
Anexo BG: TimesheetRepository.cs – GetProjectHour .....	58
Anexo BH: TimesheetRepository.cs - GetActivityHours .....	59
Anexo BI: TimesheetRepository.cs – GetUserActivityWorklogs .....	59
Anexo BJ: TimesheetController.cs .....	60

Anexo BK: 2022070712220_1thMigration.cs – Up.....	63
Anexo BL: Power Network – btn_Timesheet_LogTime – OnSelect .....	82
Anexo BM: Power Network – btn_popup_Save – OnSelect.....	83
Anexo BN: Power Network – btn_popup_Delete – OnSelect.....	94
Anexo BO: Power Network – rct_glry_ActivityBackGround – OnSelect.....	95
Anexo BP: Power Network – btn_popup_ProjectName – OnSelect.....	95
Anexo BQ: Power Network – btn_glry_UserInfo – OnSelect.....	95
Anexo BR: Power Network – btn_glry_ProjectBackGround – OnSelect.....	96
Anexo BS: Power Network – btn_glry_ActBackGround – OnSelect.....	96
Anexo BT: Power Network – btn_glry_WIBackGround – OnSelect.....	97
Anexo BU: Power Network – btn_popup_Sv – OnSelect .....	97
Anexo BV: Power Network - dtp_Timesheet_SearchDate – OnSelect.....	99
Anexo BW: Power Network – btn_popup_Dlt – OnSelect .....	100
Anexo BX: Power Network – btn_glry_ActName – OnSelect.....	101

## Lista de programas

- Microsoft Visual Studio 2022
- Microsoft SQL Server Management Studio 18
- Power Apps

## Introdução

O presente projeto encontra-se integrado no Curso Técnico de Gestão e Programação de Sistemas Informáticos e foi realizado no âmbito da Prova de Aptidão Profissional (PAP). O relatório do projeto contempla a construção e explica o funcionamento do sistema de informação realizado ao longo do estágio na empresa Arquiconsult.

O projeto Power Network-Timesheet foi proposto pelo coordenador de estágio, Engenheiro Ricardo Casaca, e, o seu nome resulta da junção de *Power Apps* com o navegador *Network* usado pela Arquiconsult. Neste navegador inserem-se várias ferramentas, entre as quais uma *Timesheet* que possibilita otimizar a gestão do fluxo de trabalho e do tempo gasto em cada projeto por cada colaborador da empresa. A este projeto foi adicionada outra ferramenta da *Network*, designada de *Project* que permite aceder aos projetos dos vários colaboradores da empresa.

Por sugestão do coordenador de estágio, foram feitas alterações ao projeto inicial, permitindo ampliar as funcionalidades do sistema de informação construído. Assim, ao adicionar a ferramenta *Project*, este passou a designar-se de Power Network-Timesheet & Project que consiste num sistema de informação baseado nas ferramentas *Timesheet* e *Project* inseridas na *Network* utilizada pela Arquiconsult, com a intenção de ser integrada no sistema de informação *Power Network* (ainda em construção) e, posteriormente inserida no *Microsoft Teams* (utilizado na empresa para a comunicação entre os colaboradores), permitindo, assim, centralizar todos os sistemas de informação usados na Arquiconsult nesta plataforma.

O projeto Power Network-Timesheet & Project pretende ser um sistema para otimizar a gestão do fluxo de trabalho e do tempo gasto em cada projeto por cada colaborador da empresa e, em relação à *Timesheet*, tem como objetivos: (1) registar e consultar as horas que um colaborador investe em cada atividade de um determinado projeto; (2) registar e consultar os comentários feitos pelo colaborador numa atividade de um determinado projeto; (3) consultar a atividade selecionada; (4) consultar o projeto da atividade selecionada; (5) consultar a(s) atividade(s) do projeto e a(s) equipa(s) de colaboradores do mesmo, permitindo, também, consultar informações sobre os colaboradores. Tal como foi referido anteriormente, adicionou-se a ferramenta *Project* ao sistema de informação apresentando também, por isso os seguintes objetivos: (1) aceder aos projetos e respetivas horas de um colaborador; (2) consultar as atividades

e horas do projeto selecionado; (3) visualizar o registo de horas num determinado dia da atividade selecionada; (4) editar/apagar um registo selecionado.

Para a realização do projeto Power Network-Timesheet & Project tiveram-se em conta quatro etapas distintas. A primeira etapa consistiu na criação da base de dados ou seja tabelas e respetivos modelos relacionais no *Microsoft SQL Server Management Studio* através do *Microsoft Visual Studio*, criando-se um projeto de modelo “aplicativo Web ASP.NET (.NET Framework)” no qual foram instalados três pacotes para soluções, *Microsoft.EntityFrameworkCore*, *Microsoft.EntityFrameworkCore.SqlServer* e *Microsoft.EntityFrameworkCore.Tools*. Salienta-se que, esta base de dados foi criada em conjunto com o colega de estágio, Lucas Carvalho e, por esta razão, contém algumas tabelas que não foram necessárias para a realização deste projeto. A segunda etapa concentrou-se no desenvolvimento de uma *Application Programming Interface (API)*, que possibilita a comunicação entre plataformas, onde se pode encontrar diversos métodos criados para serem utilizados neste projeto e que permitem a manipulação de dados na base de dados referida na etapa anteriormente. A terceira etapa teve como objetivo a criação de um conector personalizado no *Power Apps*, que permite conectar a *API* com o *Power Apps*, possibilitando o uso dos métodos da *API* no *Power Apps*. E, por último, a quarta etapa, que consistiu na criação de uma aplicação *Canvas* no *Power Apps*, na qual se desenvolveu o *design* da aplicação e a programação de alguns objetos. Nestes objetos foram utilizados alguns dos métodos do conector personalizado criado na etapa anterior, para que desta forma se possa atualizar, remover, adicionar e consultar os dados das diferentes tabelas criadas na primeira etapa.



# 1 - Estágio

## 1.1. - Empresa de estágio

O estágio foi realizado na **Arquiconsult** (soluções *Microsoft Dynamics 365*), uma empresa de consultoria de sistemas de informação, assente em tecnologias *Microsoft*, com o título de “*Gold Certified Partner [Microsoft Gol Enterprise Resource Planning, Microsoft Gold Application Development, Microsoft Gold Cloud Platform]*”.



Fig. 1.1. – Logotipo da empresa Arquiconsult

## 1.2. - Departamento de estágio

O estágio foi realizado no departamento “Dynamics 365” na equipa “Power Platform”, coordenada pelo Engenheiro Ricardo Casaca que também coordenou o estágio.

A equipa “Power Platform” é responsável por desenvolver *Power Apps* através de ferramentas *Microsoft* que fazem parte da *Power Platform*, entre as quais, o *Power BI*, *Power Automate*...

## 1.3. - Atividades durante o estágio

O estágio realizado na empresa Arquiconsult, teve a duração de cerca de três meses, durante o qual se concretizaram as atividades que se seguem, realizou-se uma formação em *Power Apps*, que consistiu numa pesquisa e estudo autónomos e teve a duração de, sensivelmente, quatro semanas. A primeira e segunda semanas desta formação incidiram no estudo autónomo, onde se pesquisou sobre o funcionamento geral do *Power Apps*; a terceira semana prosseguiu com o estudo autónomo das diferentes aplicações do *Power Apps*, que incidiram mais na aplicação *Canvas*; e na última semana, foi frequentada uma formação da *Microsoft* através de aulas em tempo

real via *Zoom*, cujo foco incidiu sobre a complexidade e as diferentes funcionalidades do *Power Apps*.

As restantes semanas destinaram-se à construção da base de dados, à definição da estrutura, ao desenvolvimento e à conclusão do sistema de informação, *Power Network-Timesheet & Project*.

#### 1.4. - Análise Crítica

O estágio realizado na empresa Arquiconsult, teve a duração de cerca de três meses, e possibilitou o conhecimento do *Power Apps* e de outras aplicações integradas neste conjunto, como aplicações *Canvas* e conectores personalizados, bem como o desenvolvimento de *APIs*, permitindo alargar o leque de conhecimentos adquiridos ao longo do curso.

Salienta-se que o início do estágio foi bastante motivador, tendo existido uma boa receção e acolhimento por parte dos colaboradores da Arquiconsult.

As formações permitiram aprender a trabalhar com o *Power Apps* e as suas potencialidades, assim como desenvolver código no *Microsoft Visual Studio* utilizando o *Microsoft.EntityFrameworkCore*, *Microsoft.EntityFrameworkCore.SqlServer* e *Microsoft.EntityFrameworkCore.Tools* para soluções integradas no projeto desenvolvido.

Destaca-se que, durante o primeiro mês, as reuniões diárias com o coordenador de estágio foram gratificantes e possibilitaram o esclarecimento de dúvidas e a organização de ideias à medida que a formação avançava.

Após a proposta do coordenador de estágio para a realização do projeto *Power Network-Timesheet*, as reuniões começaram a tornar-se menos frequentes, deixando de existir. Lamenta-se esta situação, uma vez que seria fundamental para apresentar e discutir ideias, tirar dúvidas para que o projeto pudesse evoluir de outra forma.

No dia de apresentação do projeto ao coordenador de estágio foram feitas algumas críticas construtivas em relação ao *design* do mesmo, tendo sido realçado o facto de poder ter sido mais ambicioso e juntar ao projeto a ferramenta *Project* que está integrada na *Network* utilizada pela empresa. De facto, a ideia tinha já ocorrido algumas vezes, mas a pouca experiência e a perceção de que os objetivos definidos

havam já sido atingidos enquanto estagiário, limitou a sua prossecução. No entanto, as críticas apresentadas pelo coordenador de estágio, fizeram ressurgir, dar forma e força a essa ideia e, considerando o tempo ainda disponível, optou-se por enriquecer o trabalho, incluindo no sistema de informação a ferramenta *Project* da *Network*.

## 2 - Fundamentação Técnico-Científica

### 2.1. – Lista de conceitos

Considerou-se importante a apresentação de uma lista de conceitos que são usados ao longo do relatório, onde se explica a construção e o funcionamento do sistema de informação Power Network-Timesheet & Project.

#### **.NET Framework**

Plataforma que permite o desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para *.NET* pode ser executado em qualquer dispositivo que possua um *Framework* de tal plataforma.

#### **“aplicativo Web ASP.NET (.NET Framework)”**

É uma tecnologia da *.Net Framework* que permite criar aplicativos *Web*.

#### **API (Application Programming Interface) - Interface de Programação de Aplicação**

É uma *interface* que faz a interligação entre diferentes aplicações, de forma segura e eficaz.

#### **Azure websites**

Oferece serviços que podem ser usados para hospedar *websites* e aplicações *web*.

#### **Base de Dados**

É um conjunto de dados estruturados num determinado modelo que permite a sua utilização por outras aplicações.

## **Aplicações Canvas**

Estas pertencem ao *Power Apps*, permitem a criação de *interfaces* simples e são de fácil construção, não utilizam nenhum tipo de linguagem de programação tradicional.

## **ConnectionString**

Permite passar as informações necessárias para que a *API* estabeleça uma conexão com a base de dados criada no *Microsoft SQL Server Management Studio*.

## **Conectores Personalizados**

São utilizados para estabelecer uma ligação entre a *API* e o *Power Apps*, de modo que possam ser utilizadas as funções da *API* em qualquer aplicação do *Power Apps*.

## **Entity Framework Core**

Permite que os desenvolvedores do *.NET* trabalhem com uma base de dados utilizando objetos *.NET*.

## **JSON**

É um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto.

## **Expressão Lambda**

É usada no código de linguagem de programação *c#* para pesquisa baseada em métodos.

## **Linguagem de programação**

Conjunto de regras e instruções que permitem escrever um programa.

## **Linguagem de programação c#**

Faz parte da plataforma *.NET*, e é uma linguagem de programação orientada a objetos.

## **Microsoft Teams**

É uma plataforma de comunicação e colaboração que permite enviar/receber mensagens em tempo real, videoconferências, armazenamento de ficheiros (incluindo colaboração na realização de documentos partilhados) e integração de aplicações.

## **Microsoft SQL Server Management Studio**

É um *software* usado para desenvolver e administrar base de dados num certo servidor.

## **Power Apps**

É um conjunto de aplicações, serviços e conectores, semelhante a uma plataforma de dados, que possibilita um ambiente de programação rápida para que se possam criar aplicações personalizadas às necessidades empresariais ou do utilizador.

## **Programa**

Conjunto de instruções que faz um computador executar um certo número de tarefas. Os programas são escritos numa determinada linguagem de programação, ou divididos em múltiplas partes escritas em diversas linguagens.

## **Programação (em informática)**

Processo pelo qual um programador escreve, numa linguagem de programação, o código de um programa.

## SQL

Abreviatura de *Structure Query Language*, que é uma linguagem para aceder e manipular dados numa base de dados.

## Visual Studio

É um *software* usado para desenvolver aplicações especialmente dedicadas ao *.NET Framework*, linguagem de programação *c#*, e entre outros.

## Worklog

É um conjunto de registos, dos quais dizem respeito a uma atividade de um projeto que um colaborador esteja a desenvolver. Esses registos são as horas, um pequeno comentário do que foi feito nas horas registadas, o tipo de arrecadação monetária do tempo gasto e o dia em que foram realizadas essas mesmas horas.

## 2.2. - Descrição do sistema de informação – Power Network-Timesheet & Project

O sistema de informação, Power Network-Timesheet & Project, é composto por uma base de dados o qual dispõe de diversas tabelas que foram criadas e relacionadas entre si com diversos tipos de relações, muitos para um (N:1), um para um (1:1) e muitos para muitos (N:N). Cada tabela tem os seus campos com a sua devida importância e características.

As tabelas foram criadas pela *API*, no *Visual Studio* através da *Entity Framework Core*, que permite trabalhar com a base de dados através de objetos *.NET Framework* e programar utilizando a linguagem de programação *c#*. Na *API* que pertence a este sistema de informação apresentam-se vários métodos que permitem adicionar, editar e remover dados. Aqui, pode-se encontrar uma *connectionstring* que permite o acesso à base de dados criada. Ao publicar a *API* num hospede do *Azure* podem-se usar esses mesmos métodos através do *website*.





Salienta-se que nem todas as tabelas ilustradas na figura 2.2.1.1.1., dizem respeito ao sistema de informação.

### 2.2.1.2. – Criação dos campos das Tabelas e Relações entre si

Para a criação dos campos das tabelas e estabelecimento das relações entre si, utilizou-se o *Microsoft Visual Studio 2022*, tendo sido construído um projeto de modelo “aplicativo Web ASP.NET (.NET Framework)”, à solução do projeto foram instalados os seguintes pacotes para soluções, *Microsoft.EntityFrameworkCore*, *Microsoft.EntityFrameworkCore.SqlServer* e *Microsoft.EntityFrameworkCore.Tools*. Em seguida, criaram-se várias “Classes”, correspondentes a cada tabela, esta ação realizou-se dentro de uma pasta da solução do projeto, a pasta “Models”. Estas “Classes” permitem a programação na linguagem em c#.

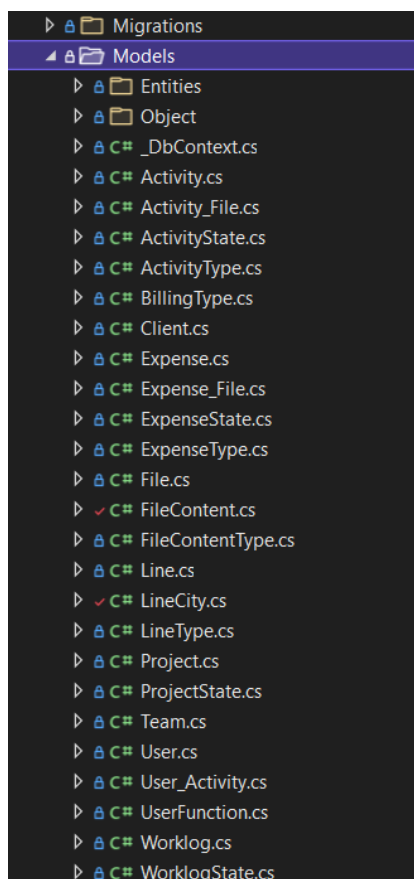
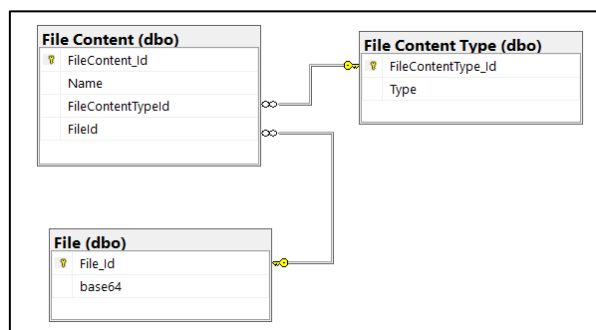


Figura 2.2.1.2.1 – pasta “Models” e “Classes”

Em cada “Classe” foram adicionados os respetivos campos e o tipo dos mesmos (por exemplo, *int*, *string*, *decimal*, *date*, ...) de acordo com as suas características. Destaca-se, também, a importância do *namespace* “System.ComponentModel.DataAnnotations” que possibilita definir se o campo é de preenchimento obrigatório, bem como o seu tamanho,... e mencionar qual dos campos é a chave primária da tabela, que consiste num identificador único para cada registo da tabela (pode-se encontrar o código para a definição dos campos do Anexo D ao Anexo AA). Salienta-se que nas situações onde houve necessidade de estabelecer uma relação de muitos para muitos (N:N) foi necessária a criação de uma nova tabela com a definição de uma chave composta, que consiste na utilização das chaves primárias das outras tabelas com as quais se relaciona. Pode-se encontrar uma chave composta nas tabelas: “Team”, “Activity\_File”, “Expense\_File” e “User\_Activity”, através do método “OnModelCreating” com utilização de expressões *lambda* (conforme Anexo C).

Depois de se criarem os campos das tabelas, é feita a relação entre alguns campos. Estas relações são feitas apenas entre os campos definidos como chaves primárias. Para tal foram estabelecidas relações de um para um (1:1), de muitos para um (N:1) e de muitos para muitos (N:N), como apresentam as figuras 2.2.1.2.2., 2.2.1.2.3. e 2.2.1.2.4.. As relações permitem estabelecer uma ligação entre os campos de uma ou mais tabelas, realça-se que nas relações de muitos para muitos (N:N), acresce a necessidade de ser criada uma nova tabela que estabelece essas relações entre duas tabelas (as relações estabelecidas entre os campos das tabelas encontram-se no código descrito do Anexo D ao Anexo AA, após um comentário “Navigation Properties”).



**Figura 2.2.1.2.2. – exemplo de uma relação de um para um (1:1), entre as tabelas “File Content” e “File”**

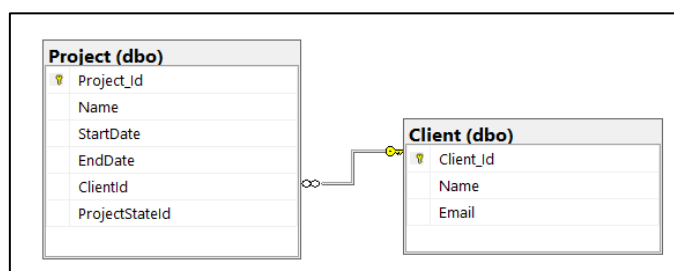


Figura 2.2.1.2.3. – exemplo de uma relação de um para muitos (1:N), entre as tabelas “Project” e “Client”

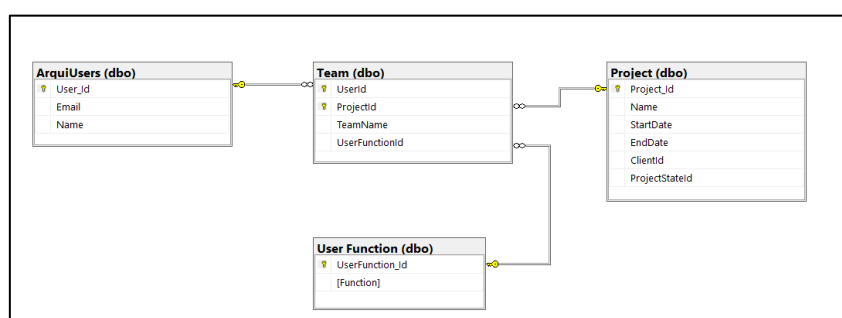


Figura 2.2.1.2.4. – exemplo de uma relação de muitos para muitos (N:N), entre as tabelas “ArquiUsers” e “Project”

### 2.2.1.3. – Migration e Update-DataBase

Depois de se definirem os campos das tabelas e as relações entre os mesmos através do “aplicativo Web ASP.NET (.NET Framework)”, prosseguiu-se com a *Migration*, que permite manter sincronizada a base de dados com o projeto, atualizando o esquema da mesma. É possível realizar uma migração através de uma linha de comando executada a partir do “Console”, como mostra a figura 2.2.1.3.1.. Realça-se que as *Migrations* efetuadas não podem ter o mesmo nome que as anteriores.

```

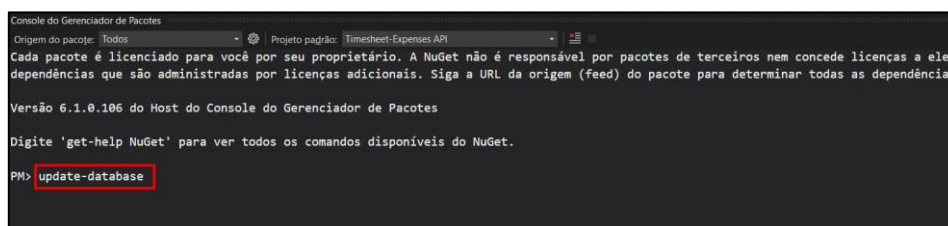
Console do Gerenciador de Pacotes
Origem do pacote: Todos | Projeto padrão: Timesheet-Expenses API
Cada pacote é licenciado para você por seu proprietário. A NuGet não é responsável por pacotes de terceiros nem concede dependências que são administradas por licenças adicionais. Siga a URL da origem (feed) do pacote para determinar todos os pacotes de dependência.
Versão 6.1.0.106 do Host do Console do Gerenciador de Pacotes
Digite 'get-help NuGet' para ver todos os comandos disponíveis do NuGet.
PM> add-migration firstMigration

```

Figura 2.2.1.3.1. – exemplo de linha de comando para criar uma *Migration*

Após efetuada a linha de comando é criada uma pasta com o nome de “Migrations”, caso seja a primeira *Migration*. Esta pasta contém um ficheiro .cs com código que, permite atualizar a base de dados (Anexo BK), sendo criado ao executar a linha de comando ilustrada na figura 2.2.1.3.1..

Seguidamente, executou-se uma outra linha de comando com mostra a figura 2.2.1.3.2.:



**Figura 2.2.1.3.2. – linha de comando *update-database***

Através desta linha de comando são dadas ordens para a criação/atualização do esquema da base de dados localizada no *Microsoft SQL Server Management Studio 18*. Se a base de dados ainda não estiver criada, ao executar a linha do comando demonstrada na figura 2.2.1.3.2., a base de dados será gerada de acordo com o nome dado na *connectionstring* (consultar Anexo A) e criar-se-á do zero o esquema da mesma.

## **2.2.2. – Criação do *Repository* e do *Controller***

Começou-se por criar o *Repository* na *API* numa pasta com nome “Repositories”. A esse *Repository* deu-se o nome de “TimesheetRepository”. Este serve de meio para remover, editar, adicionar e/ou enviar dados da base de dados através de métodos criados em linguagem de programação c#, estes encontram-se do Anexo AQ ao Anexo BI.

Em seguida, criou-se o *Controller* na *API* numa pasta com o nome “Controllers”. A esse *Controller* deu-se o nome de “TimesheetController” que contém vários métodos dos quais permite receber dados do “TimesheetRepository”, através da interface


“ITimesheetRepository” criada na “TimesheetRepository”. Este método ao ser executado vai devolver valores e/ou objetos *.NET* ao “TimesheetController”, que os apresenta em forma de *JSON*. Estes métodos criados na “TimesheetController” na linguagem de programação *c#* podem ser encontrados no Anexo BJ.

### 2.2.3. – Formação do Conector Personalizado

O conector personalizado foi criado, dentro do *Power Apps*, no ambiente “D365CE & Portals - Academy”. Este é utilizado pela Arquiconsult para a criação de projetos experimentais. A função deste conector personalizado é estabelecer uma conexão entre o *Power Apps* e os métodos da *API*.

A seguinte figura apresenta a criação de um conector personalizado no *Power Apps*, onde o anfitrião é o site de *Azure*, onde se publicou a *API*.

Informações gerais

 Carregar ícone do conector  
Os formatos de ficheiro suportados são PNG e JPG. (< 1 MB)

Carregar

Cor de fundo do ícone

#007ee5

Descrição

Atribua uma breve descrição ao conector personalizado

☐ Ligar através do gateway de dados no local Saiba mais

Esquema \*

☒ HTTPS ☐ HTTP

Anfitrião \*

inete2022.azurewebsites.net

URL Base

/

**Figura 2.2.3.1. – criação de um conector personalizado no *Power Apps***

As figuras 2.2.3.2., 2.2.3.3. e 2.2.3.4. ilustram os passos para a criação de uma ação no conector personalizado, possibilitando assim o uso dos métodos criados na *API* através da criação de ações no conector personalizado. No fim de todas as ações

serem criadas ficam concluídas todas as funções necessárias para o conector personalizado funcionar.

The screenshot shows the 'Geral' tab of a configuration interface. It contains the following fields and options:

- Resumo** (Summary): A text box containing 'GetBillingTypes'.
- Descrição** (Description): A text box containing 'gets all billing types'.
- ID da operação \*** (Operation ID): A text box containing 'GetBillingTypes'. Below it, a note states: 'Esta é a cadeia exclusiva utilizada para identificar a operação.' (This is the exclusive chain used to identify the operation.)
- Visibilidade** (Visibility): Four radio buttons labeled 'none', 'advanced', 'internal', and 'important'. The 'none' button is selected.

**Figura 2.2.3.2. – exemplo, geral de uma ação de um conector personalizado**

The screenshot shows the 'Pedido' tab of a configuration interface. It contains the following fields and options:

- Verbo \*** (Verb): A dropdown menu showing 'GET'. A note above it says: 'O verbo descreve as operações disponíveis num único caminho.' (The verb describes the operations available in a single path.)
- URL \*** (URL): A text box containing 'https://inete2022.azurewebsites.net/api/Timesheet/GetBillingTypes'. A note above it says: 'Este é o URL do pedido.' (This is the request URL.)
- Caminho** (Path): A text box is empty. A note above it says: 'O caminho é utilizado em conjunto com Modelos de Caminho, onde o valor do parâmetro faz realmente parte do URL da operação.' (The path is used together with Path Models, where the parameter value is actually part of the operation URL.)
- Consulta** (Query): A text box is empty. A note above it says: 'Os parâmetros de consulta são anexados ao URL. Por exemplo, em /items?id=####, o parâmetro de consulta é o ID.' (Query parameters are attached to the URL. For example, in /items?id=####, the query parameter is the ID.)
- Cabeçalhos** (Headers): A text box is empty. A note above it says: 'Estes são cabeçalhos personalizados que fazem parte do pedido.' (These are custom headers that are part of the request.)
- Corpo** (Body): A text box is empty. A note above it says: 'O corpo é o payload que está anexado ao pedido de HTTP. Só pode existir um parâmetro de corpo.' (The body is the payload that is attached to the HTTP request. Only one body parameter can exist.)

At the top right of the tab, there is a button labeled '+ Importar a partir da amostra' (Import from sample).

**Figura 2.2.3.3. – exemplo, pedido de uma ação de um conector personalizado**

**Resposta** + Importar a partir da amostra

Nome \*

default

Cabeçalhos

Estes são cabeçalhos personalizados que faz parte da resposta.

Referências Utilizadas

Seguem-se as referências utilizadas por esta entidade

Corpo

O payload que está disponível na resposta. Estes são os tokens que vão ser apresentados como saídas no estruturador.

\* Type

**Figura 2.2.3.4. – exemplo, resposta de uma ação de um conector personalizado**

Na figura 2.2.3.4. ilustra-se como se pode importar um determinado corpo de um *JSON*, facilitando assim a criação do corpo da resposta.

**Pedido** + Importar a partir da amostra

Verbo \*

O verbo descreve as operações disponíveis num único caminho.

GET

URL \*

Este é o URL do pedido.

https://inete2022.azurewebsites.net/api/Timesheet/GetWorklog/{worklogId}

Caminho

O caminho é utilizado em conjunto com Modelos de Caminho, onde o valor do parâmetro faz realmente parte do URL da operação.

\* worklogId

Consulta

Os parâmetros de consulta são anexados ao URL. Por exemplo, em /items?id=####, o parâmetro de consulta é o ID.

Cabeçalhos

Estes são cabeçalhos personalizados que fazem parte do pedido.

Corpo

O corpo é o payload que está anexado ao pedido de HTTP. Só pode existir um parâmetro de corpo.

**Figura 2.2.3.5. – exemplo, pedido de uma ação de um conector personalizado, com parâmetro**

A figura 2.2.3.5., para além de exemplificar o pedido da ação de um conector personalizado, como se pode verificar no *URL*, mostra como também é possível editar o tipo de parâmetro(s) referido(s) no *URL* através do caminho do pedido quando aparece um parâmetro definido no mesmo.

## 2.2.4. – Personalização do sistema de informação através da aplicação *Canvas* do *Power Apps*

Esta etapa finaliza todos os procedimentos para o desenvolvimento deste sistema de informação, assim, através da aplicação *Canvas* do *Power Apps* procedeu-se à personalização do *design* deste sistema de informação.

Primeiramente, criou-se uma aplicação *Canvas* no mesmo ambiente do conector personalizado (referido anteriormente no ponto 2.2.3.), “D365CE & Portals - Academy”, para o utilizar na aplicação *Canvas* criada, uma vez que se encontram no mesmo ambiente, como se apresenta na figura 2.2.4.1..

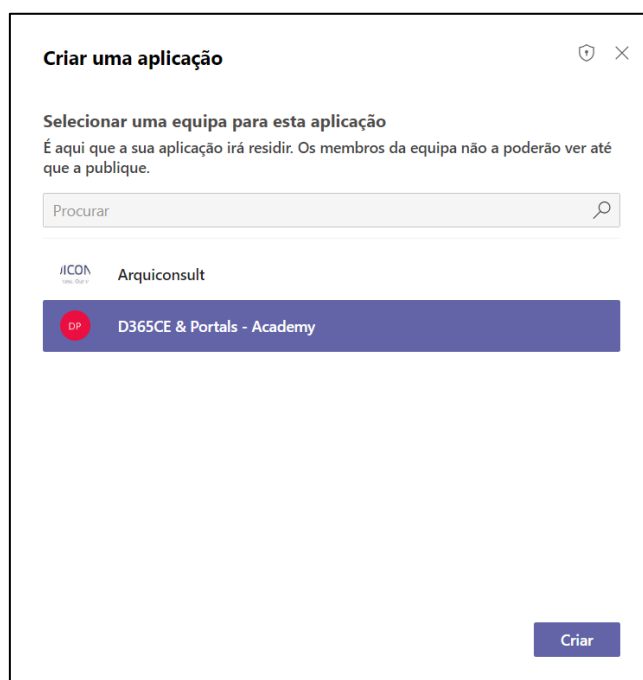


Figura 2.2.4.1. – criação da aplicação *Canvas*

### 2.2.4.1. – Fusão do Conector Personalizado

Visto que o conector personalizado e a aplicação *Canvas* são criadas ambas no mesmo ambiente, anteriormente referido, é possível adicionar o conector personalizado a esta aplicação *Canvas*.



A seguinte figura ilustra os passos para a instalação do conector personalizado na aplicação *Canvas* criada.

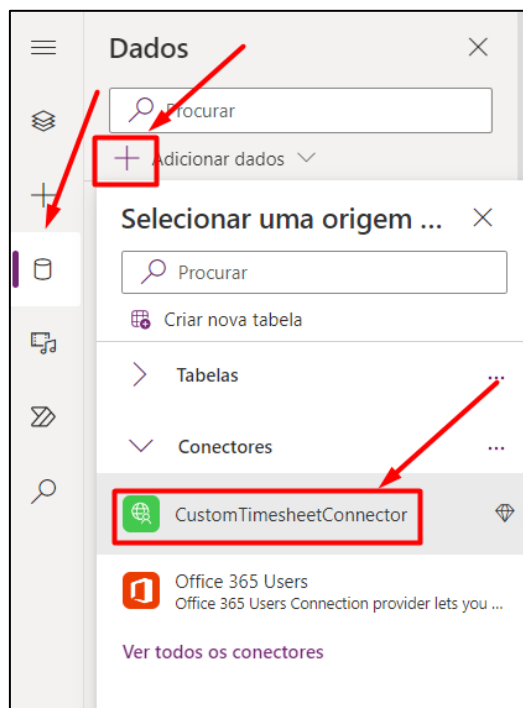


Figura 2.2.4.1.1. – adicionar conector personalizado à aplicação *Canvas* criada

#### 2.2.4.2. – Desenvolvimento da aplicação *Canvas*

Na aplicação *Canvas* acabada de criar, adicionaram-se três ecrãs, respetivamente “Timesheet Screen”, “Project Screen” e “Settings Screen”, como mostra a seguinte figura:

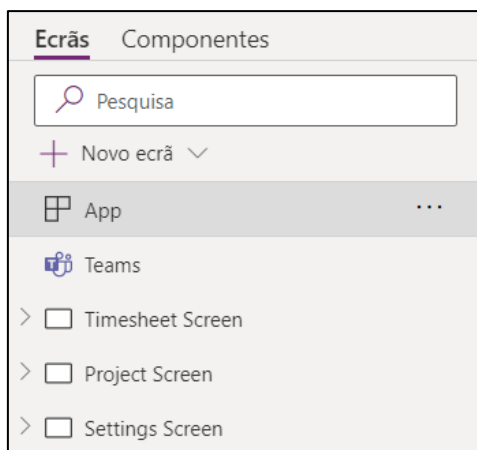
















Figura 2.2.4.2.1. – ecrãs da aplicação *Canvas*

Em seguida, no ecrã “Timesheet Screen”, deu-se início à personalização do mesmo, utilizando-se uma galeria, um botão, algumas imagens, um seletor de datas, entre outros. A galeria inserida serve para suporte de informação em massa, ou seja, quando se têm vários registos, ao utilizar uma galeria, os mesmos são organizados e separados uns dos outros, como demonstra a seguinte figura:

Activity	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Design Web	3 		3 		19 	7 	
Estágio Gabriel			12 	7 	5 		

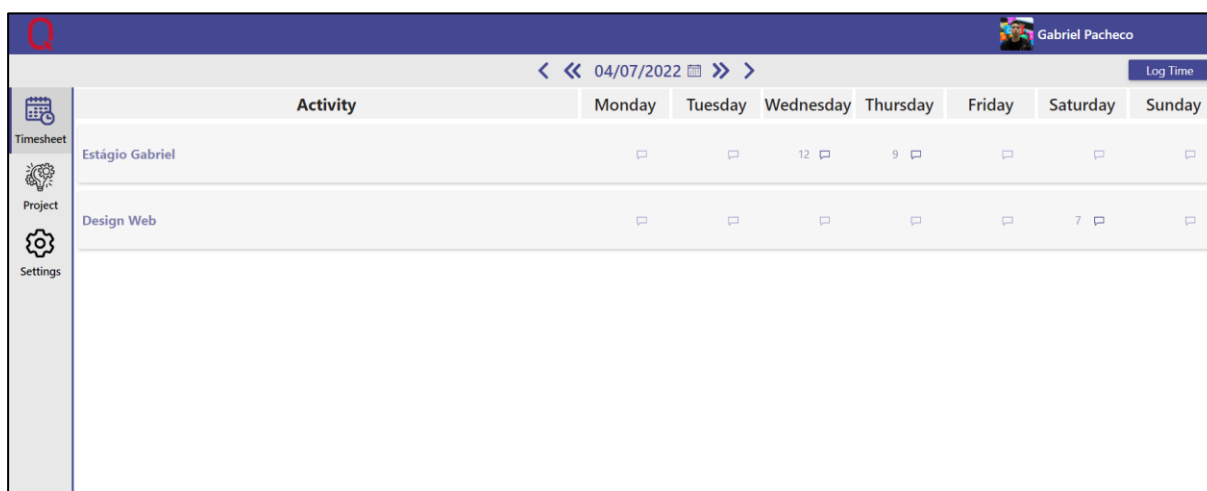
**Figura 2.2.4.2.2. – exemplo de uma galeria com registos diferentes**

No ecrã “Project Screen” encontra-se como objeto mais relevante uma galeria que suporta diversos outros registos, na qual a *interface* interage de acordo com o registo selecionado.

E por último, o ecrã “Settings Screen” onde foram inseridos como principais objetos utilizados, uma foto e duas etiquetas pertencentes ao colaborador a utilizar o sistema de informação.

Destaca-se que, na personalização da *interface* do sistema de informação, se teve em conta a utilização de cores idênticas às do *Microsoft Teams*, plataforma onde funcionará este sistema de informação.

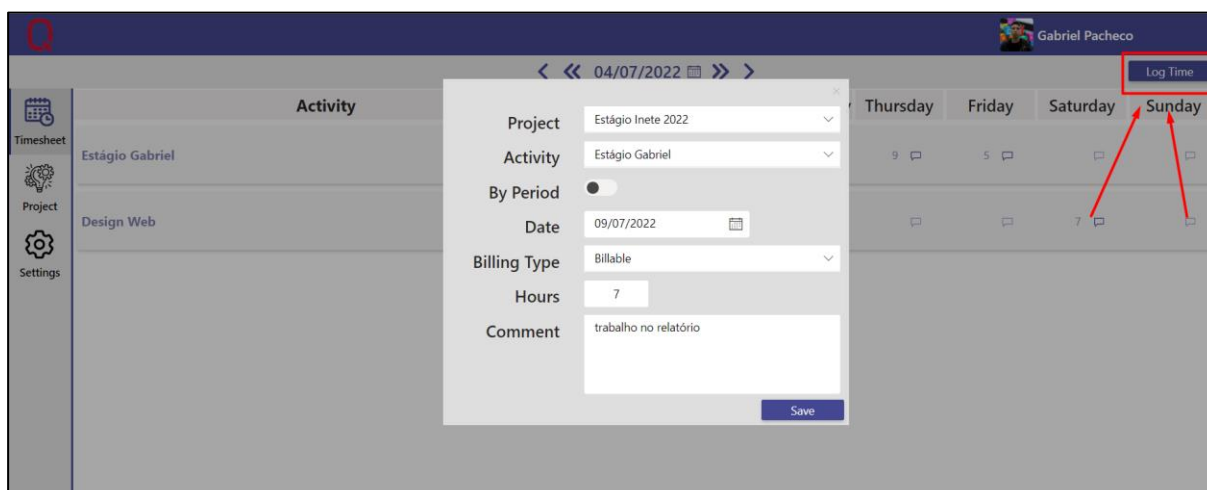
### 3 – Apresentação do sistema de informação Power Network-Timesheet & Project



Activity	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Estágio Gabriel			12	9			
Design Web						7	

Figura 3.1. – sistema de informação, *Timesheet*

Neste ecrã, selecionado a partir do *Tab Page, Timesheet*, surgem as atividades que têm horas registadas pelo colaborador, na semana indicada (neste caso 04/07/2022). Os registos mudam de acordo com a semana selecionada, esta ação aparece indicada no Anexo BV.

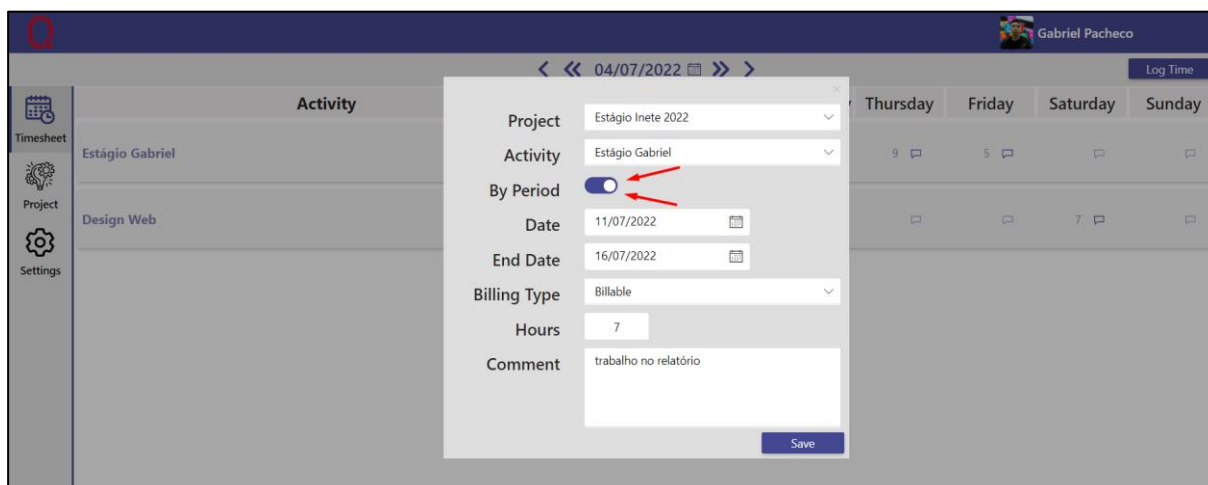


Project: Estágio Inete 2022  
Activity: Estágio Gabriel  
By Period: ☐  
Date: 09/07/2022  
Billing Type: Billable  
Hours: 7  
Comment: trabalho no relatório  
Save

Figura 3.2. – criação de uma *worklog*, através do *Log Time*

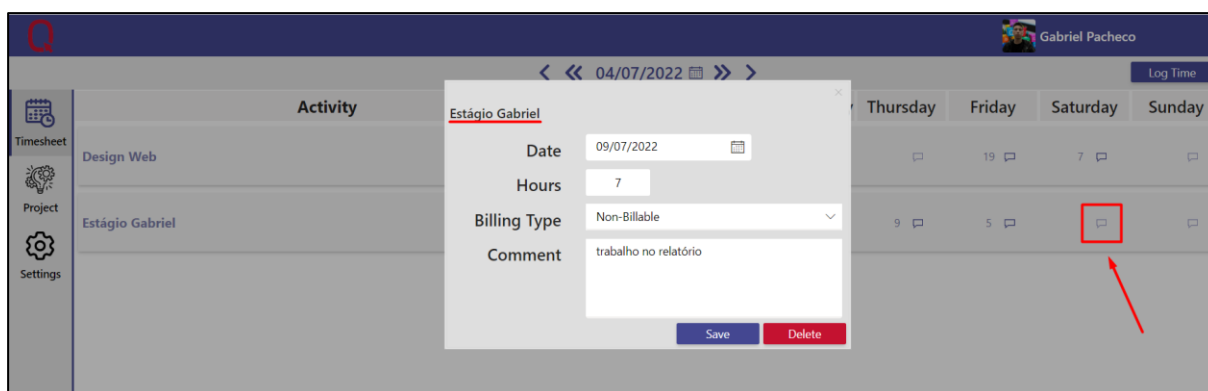
A figura 3.2. ilustra a criação de uma *worklog*, que consiste no registo de horas, comentário num determinado dia associado à atividade de um projeto, ao clicar no botão “Log Time”, ver Anexo BL com o código desta ação. Para executar esta ação,

basta clicar no botão “Save”, após o preenchimento de todos os campos, deve-se ter em conta que o número de horas registadas não pode exceder vinte e quatro horas nem ser inferior ou igual a zero, quer no registo individual, quer no somatório de todas as horas nas diferentes atividades referentes àquele dia. Após a ação, a *popup* desaparece, atualizando assim os registos de acordo com a ação efetuada, ver Anexo BM para código das ações mencionadas.



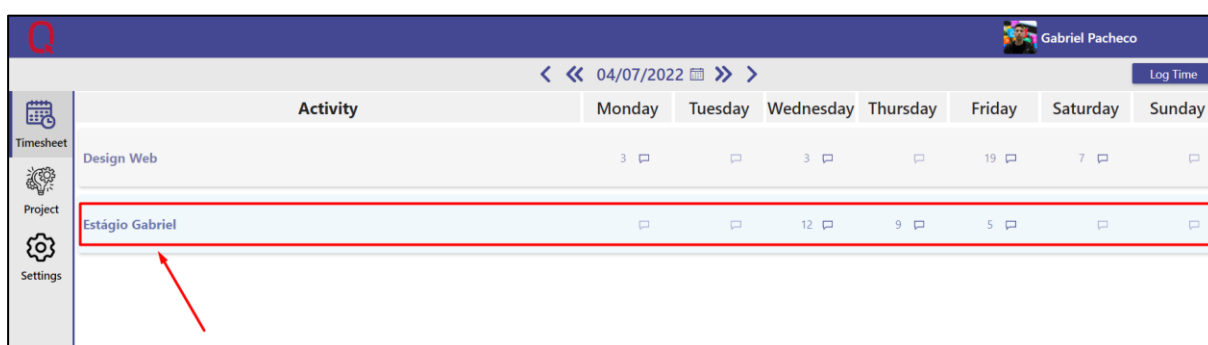
**Figura 3.3. - criação de uma *worklog by period*, através do *Log Time***

A figura 3.3. demonstra que também é possível criar várias *worklogs* ao mesmo tempo, ativando o *troggle button*, “By Period”, onde se deve escolher uma data de início e de fim, tendo em conta que a diferença entre as duas não pode exceder o período de sete dias. Caso exista um registo no período selecionado da mesma atividade, ao executar a ação ocorrerá um erro e a criação das *worklogs* não será feita, ver Anexo BM para código das ações mencionadas.



**Figura 3.4. – criação/edição/eliminação de uma *worklog*, de uma atividade num determinado dia**

A figura 3.4. apresenta a criação de uma *worklog* associada a uma atividade no dia selecionado. Ao clicar no botão destacado na figura, surge uma *popup* na qual o dia e a atividade aparecem predefinidos, não podendo ser alterados. Os campos “Hours”, “Billing Type” e “Comment” devem ser preenchidos e guardados ao clicar no botão “Save”. Para editar a *worklog* associada a uma atividade no dia selecionado, o procedimento é o mesmo. Para eliminar a *worklog* associada a uma atividade no dia selecionado esta tem de estar previamente criada, e posteriormente clicar no botão “Delete”, ver Anexo BN para código do botão “Save” e ver Anexo BM para código do botão “Delete”.



**Figura 3.5. – seleção de uma atividade**

É possível consultar a informação sobre a atividade clicando sobre a mesma, como demonstra a figura 3.5., ver Anexo BO.

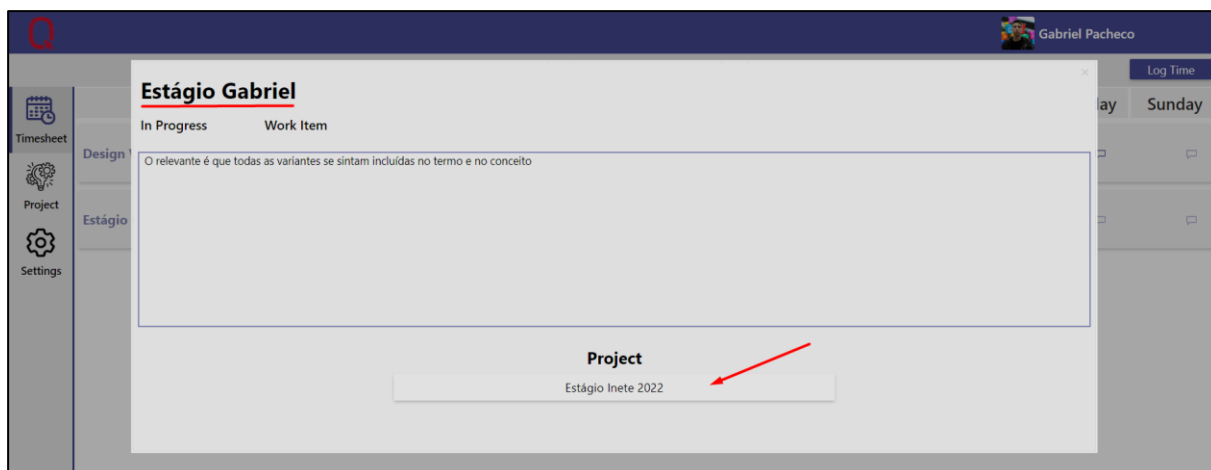


Figura 3.6. – informação sobre a atividade selecionada

A figura 3.6. mostra a janela *popup* com a informação sobre a atividade selecionada, nomeadamente o estado, o tipo, a descrição e o projeto relacionado com a atividade. Para obter informações mais detalhadas, como o nome e cargo dos colaboradores envolvidos no projeto da atividade selecionada, deve clicar-se no botão com o nome do projeto, como se exibe na figura abaixo (ver Anexo BP para código da ação).

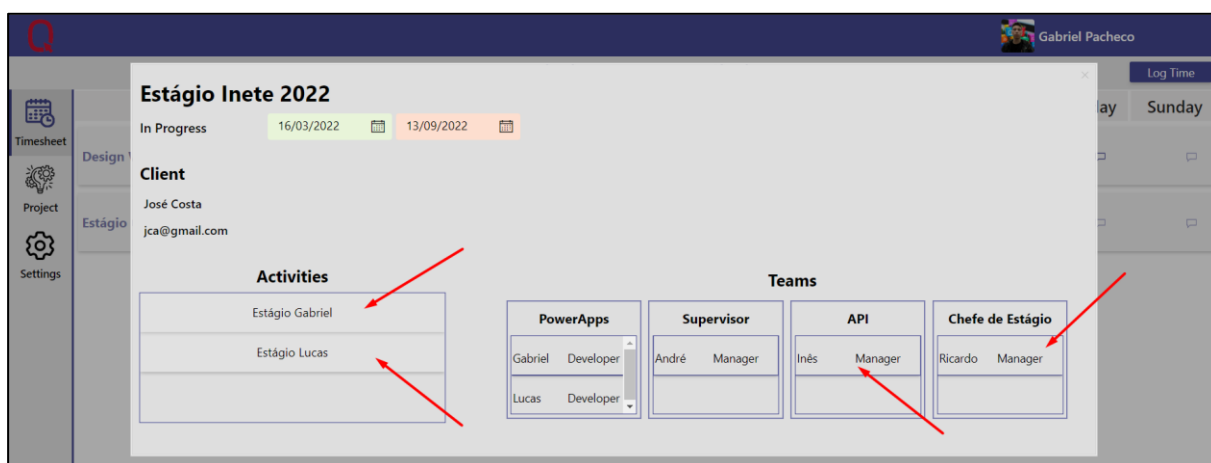


Figura 3.7. – informação sobre o projeto selecionado

O tipo de informação apresentado na figura 3.7., diz respeito às atividades e às equipas envolvidas no projeto, onde se pode visualizar o nome completo e *email* do cliente que solicitou o projeto à empresa, assim como o estado do projeto, a data de início e a data prevista de conclusão do mesmo. Também permite consultar as

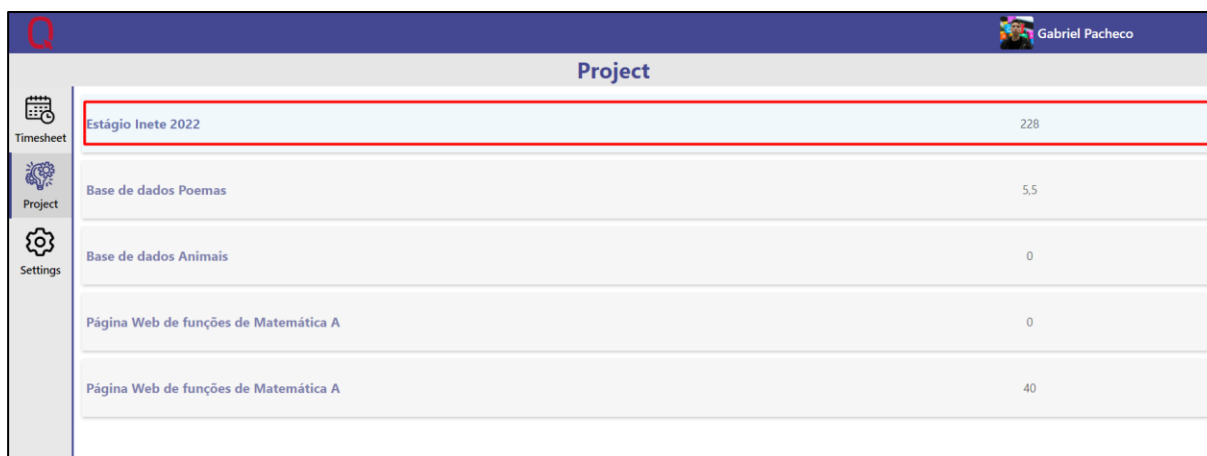
diferentes informações das atividades relacionadas com o projeto, bem como ver os dados (nome completo e email da empresa) do colaborador selecionado (ver Anexo BX e Anexo BQ, respetivamente).



Project	
Estágio Inete 2022	228
Base de dados Poemas	5,5
Base de dados Animais	0
Página Web de funções de Matemática A	0
Página Web de funções de Matemática A	40

**Figura 3.8. – sistema de informação, *Project***

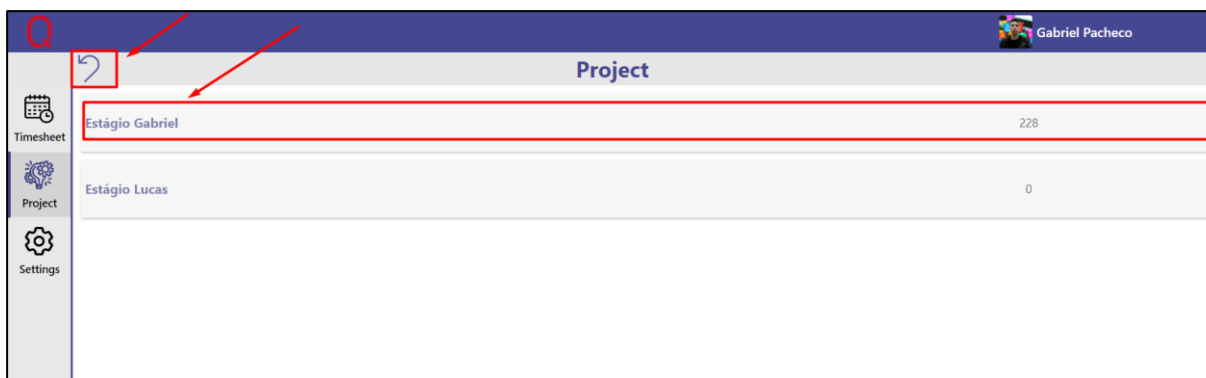
Neste ecrã, selecionado a partir do *Tab Page, Project*, apresentam-se os projetos e as respetivas horas de um colaborador.



Project	
Estágio Inete 2022	228
Base de dados Poemas	5,5
Base de dados Animais	0
Página Web de funções de Matemática A	0
Página Web de funções de Matemática A	40

**Figura 3.9. – seleção de um projeto**

Como a figura 3.9. mostra, ao selecionar um projeto, ação indicada no Anexo BR, surge a figura seguinte:



Project	
Estágio Gabriel	228
Estágio Lucas	0

**Figura 3.10. – atividades e respectivas horas do projeto selecionado**

Na figura 3.10. encontram-se todas as atividades relacionadas com o projeto selecionado, bem como as horas despendidas pelo colaborador nas mesmas. Surge também um botão no canto superior esquerdo do ecrã que permite voltar ao estado anterior do ecrã. Ao selecionar uma atividade o ecrã altera o seu estado, por meio da ação registada no Anexo BS, para a figura seguinte:

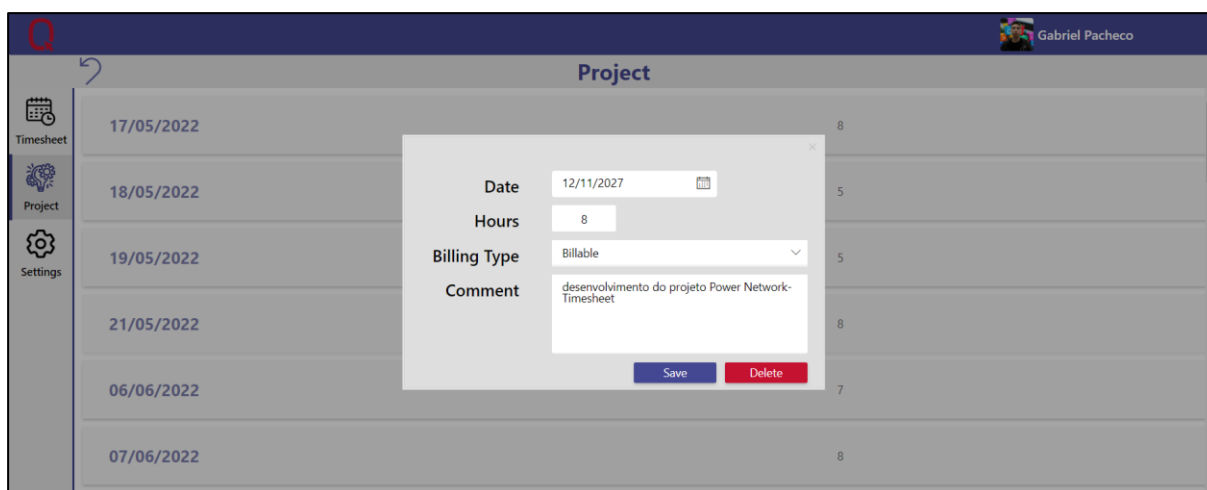


Project	
17/05/2022	8
18/05/2022	5
19/05/2022	5
21/05/2022	8
06/06/2022	7
07/06/2022	8

**Figura 3.11. – lista do registo das datas e horas de uma atividade**

A figura 3.11. permite visualizar o registo das datas e horas realizadas pelo colaborador na *Timesheet* através da criação de *worklogs*. Selecionando-se um determinado registo, por ação do código do Anexo BT, aparece numa *popup* com as informações sobre essa mesma *worklog*.





**Figura 3.12. – editar/eliminar a *worklog* selecionada**

A figura 3.12. exibe uma *popup* da *worklog* referente ao dia selecionado. Neste ambiente pode-se editar ou eliminar a *worklog* clicando no botão “Save”, após as alterações, ou “Delete” respetivamente, o código para estas ações apresenta-se no Anexo BU e no Anexo BW. Salienta-se que as alterações realizadas nesta opção são atualizadas na *Timesheet*.



**Figura 3.13. - sistema de informação, *Settings***

Neste ecrã, selecionado a partir do *Tab Page*, *Settings*, apresenta-se a respetiva imagem de perfil, nome completo e email do colaborador que se encontra a usar o sistema de informação, Power Network-Timesheet & Project.

## Condicionalismos

Ao longo do desenvolvimento do sistema de informação, como acontece em todas as aprendizagens e percursos profissionais e/ou pessoais, foram sentidas algumas dificuldades e adversidades que foram superadas com pesquisas autónomas.

Destacam-se dificuldades na utilização de *lambda* no uso da *connectionstring*, que foi contornada através da visualização de exemplos que levaram à aprendizagem sobre as suas funções e sobre a forma de serem aplicadas no código da linguagem *c#*. Também, se verificaram algumas dificuldades na realização do *Controller* e *Repository* que se conseguiram contornar através de pesquisas efetuadas e através da visualização de muitos vídeos sobre o desenvolvimento de *APIs*, acabando por se compreender como estas funcionam de forma geral. A criação de chaves compostas foi ultrapassada com pesquisa sobre como fazer todos os tipos de relações com o *EntityFrameworkCore*, visto que a chave composta aparece numa tabela originada de uma relação de muitos para muitos (N:N). Estas tabelas que têm duas ou mais chaves primárias, ou seja, uma chave composta, leva a um meio diferente de criar essas chaves primárias, como se pode ver no método “OnModelCreating” presente no Anexo C.

Mostra-se que todas as dificuldades foram ultrapassadas através de pesquisas autónomas sobre as mesmas, à medida que o sistema de informação se desenvolvia. Realça-se a vantagem da formação no início do estágio, pois possibilitou entender as principais funções do *Power Apps*, eliminando a existência de possíveis dúvidas que poderiam ter surgido.

De salientar também, e como foi referido anteriormente, que, depois de um período de reuniões e acompanhamento diário, foi realizado um trabalho mais autónomo que, por inexperiência e receio, se sentiu como limitador. No entanto, consideram-se as críticas apresentadas pelo coordenador de estágio, na apresentação do projeto, como um importante momento de crescimento, que estimulando a capacidade de resiliência, possibilitou uma melhoria no trabalho já realizado, bem como a perceção de que se deve discutir e desenvolver as ideias construídas no decurso dos projetos, que os possam enriquecer, mesmo que os seus objetivos iniciais já tenham sido alcançados.

## Conclusão

O sistema de informação Power Network, como se referiu inicialmente, pretende ser um sistema mais complexo com outras funções para além das realizadas. Contudo, considera-se que os objetivos definidos foram atingidos no que respeita às ferramentas *Timesheet* e *Project*, que estão desenvolvidas e que se encontram aptas para utilizar. Deste modo, conclui-se que o presente sistema de informação designado por Power Network-Timesheet & Project se encontra terminado.

Reconhece-se o facto de ainda existir a necessidade de melhorar as questões relacionadas com o *design* do sistema de informação, uma vez que os interesses se acentuaram mais no desenvolvimento de código para atingir os objetivos propostos para as ferramentas *Timesheet* e *Project*.

Importa aqui referir que os conhecimentos adquiridos ao longo do curso foram essenciais para a realização deste projeto, uma vez que possibilitam não só conhecer, analisar e interpretar diferentes ferramentas de trabalho, como também saber como procurar, de forma autónoma, resposta para as dúvidas e dificuldades que se foram encontrando. Faz-se, por isso, um percurso do conhecimento orientado, pelos professores, á autonomização, já no estágio e desenvolvimento do projeto, importante para o desenvolvimento de uma atividade profissional ou prosseguimento de estudos.

Também se destaca a importância de realizar estágio em ambiente profissional, pois a realidade do desenvolvimento de projetos ao longo do curso, nomeadamente, nas aulas de Programação e Sistemas de Informação (PSI) é muito diferente. A formação em ambiente profissional permitiu uma melhoria significativa na autonomia e responsabilidade na gestão do tempo e na resolução de problemas e adversidades encontradas, que se traduziram em experiências enriquecedoras ao nível profissional e pessoal.

## Webgrafia

<http://cadeiras.iscte-iul.pt/PTecIII/Docs/GLOSSARY.pdf>

<https://docs.microsoft.com/pt-pt/power-apps/powerapps-overview>

<https://support.microsoft.com/pt-pt/office/no%C3%A7%C3%B5es-b%C3%A1sicas-da-base-de-dados-a849ac16-07c7-4a31-9948-3c8c94a7c204>

<https://www.techtudo.com.br/tudo-sobre/drawio/>

<https://www.inforpedia.pt/dicionarios/lingua-portuguesa/basededados>

<https://brasil.softlinegroup.com/sobre-a-empresa/blog/microsoft-teams-entenda-o-que-e-e-como-funciona>

<https://docs.microsoft.com/pt-br/ef/core/>

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/JSON>

## Anexos

### Anexo A: appsettings.json

```
{
  "ConnectionStrings": {
    "Default": "Server=inete2022.database.windows.net; Database=inete2022; uid=admininete ;pwd=Arqui!123."
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

### Anexo B: Program.cs

```
using Timesheet_Expenses_API.Models;
using Microsoft.EntityFrameworkCore;
using Timesheet_Expenses_API.Repositories;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

#region Modified
//ConnectionString
builder.Services.AddDbContext<_DbContext>(x =>
x.UseSqlServer(builder.Configuration.GetConnectionString("Default")));

builder.Services.AddScoped<ITimesheetRepository, TimesheetRepository>();
builder.Services.AddScoped<IExepensesObjectsRep, ExepensesObjectsRep>();
#endregion

// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

## Anexo C: \_DbContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace Timesheet_Expenses_API.Models
{
    public class _DbContext: DbContext
    {
        public _DbContext(DbContextOptions<_DbContext> options) :
base(options)
        { }

        protected override void OnModelCreating (ModelBuilder builder)
        {
            base.OnModelCreating (builder);
            builder.Entity<Team>().HasKey(t => new { t.ProjectId, t.UserId
});
            builder.Entity<Activity_File>().HasKey(af => new {
af.ActivityId, af.FileContentId });
            builder.Entity<Expense_File>().HasKey(ef => new {
ef.FileContentId, ef.ExpenseId });
            builder.Entity<User_Activity>().HasKey(ua => new {ua.UserId,
ua.ActivityId});

        }

        public DbSet<User> users { get; set; }
        public DbSet<WorklogState> worklogStates { get; set; }
        public DbSet<BillingType> billingTypes { get; set; }
        public DbSet<UserFunction> userFunction { get; set; }
        public DbSet<ProjectState> projectStates { get; set; }
        public DbSet<Client> client { get; set; }
        public DbSet<ActivityState> activityState { get; set; }
        public DbSet<ActivityType> activityType { get; set; }
        public DbSet<FileContentType> fileContType { get; set; }
        public DbSet<ExpenseType> expenseType { get; set; }
        public DbSet<ExpenseState> expenseState { get; set; }
        public DbSet<Project> projects { get; set; }
        public DbSet<Team> teams { get; set; }
        public DbSet<Activity> activities { get; set; }
        public DbSet<Worklog> worklogs { get; set; }
        public DbSet<FileContent> fileContents { get; set; }
        public DbSet<Activity_File> activities_files { get; set; }
        public DbSet<Expense> expenses { get; set; }
        public DbSet<Expense_File> expenses_files { get; set; }
        public DbSet<Line> lines { get; set; }
        public DbSet<File> files { get; set; }
        public DbSet<User_Activity> activities_users { get; set; }
        public DbSet<LineType> lineType { get; set; }
        public DbSet<LineCity> lineCity { get; set; }
    }
}
```

## Anexo D: Activity.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"Activity")]
    public class Activity
    {
        [Key]
        public int Activity_Id { get; set; }
        [Required, MaxLength(150)]
        public string Name { get; set; }
        public string Description { get; set; }

        //Navigation Properties
        public int ProjectId { get; set; }
        public Project Project { get; set; }

        public int ActivityStateId { get; set; }
        public ActivityState ActivityState { get; set; }

        public int ActivityTypeId { get; set; }
        public ActivityType ActivityType { get; set; }

        public List<User_Activity> user_Activities { get; set; }
        public List<Worklog> Worklog { get; set; }
        public List<Activity_File> Activity_File { get; set; }
    }
}

```

## Anexo E: Activity\_File.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"Activity_File")]
    public class Activity_File
    {
        //Navigation Properties
        public int ActivityId { get; set; }
        public Activity Activity { get; set; }

        public int FileContentId { get; set; }
        public FileContent FileContent { get; set; }
    }
}

```

## Anexo F: ActivityState.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"Activity State")]
    public class ActivityState
    {
        [Key]
        public int ActivityState_Id { get; set; }
    }
}

```

```

        [Required, MaxLength(30)]
        public string State { get; set; }

        //Navigation Properties
        public List<Activity> Activity { get; set; }
    }
}

```

## Anexo G: ActivityType.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Activity Type")]
    public class ActivityType
    {
        [Key]
        public int ActivityType_Id { get; set; }
        [Required, MaxLength(30)]
        public string Type { get; set; }

        //Navigation Properties
        public List<Activity> Activity { get; set; }
    }
}

```

## Anexo H: BillingType.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Billing Type")]
    public class BillingType
    {
        [Key]
        public int BillingType_Id { get; set; }
        [Required, MaxLength(50)]
        public string Type { get; set; }

        //Navigation Properties
        public List<Worklog> Worklog { get; set; }
    }
}

```

## Anexo I: Client.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Client")]
    public class Client

```



```

    {
        [Key]
        public int Client_Id { get; set; }
        [Required, MaxLength(250)]
        public string Name { get; set; }
        [Required, MaxLength(254)]
        public string Email { get; set; }

        //Navigation Properties
        public List<Project> Project { get; set; }
    }
}

```

## Anexo J: Expense.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Expense")]
    public class Expense
    {
        [Key]
        public int Expense_Id { get; set; }
        public string Expense_Name { get; set; }
        public string Month { get; set; }
        public int Year { get; set; }
        public decimal TotalMoney { get; set; }

        //Navigation Properties

        public string ExpenseState { get; set; }

        public int UserId { get; set; }

        public int ProjectId { get; set; }

        public List<Line> Line { get; set; }
    }
}

```

## Anexo K: Expense\_File.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Expense_File")]
    public class Expense_File
    {
        //Navigation Properties
        public int FileContentId { get; set; }
        public FileContent FileContent { get; set; }
    }
}

```

```

        public int ExpenseId { get; set; }
        public Expense Expenses { get; set; }
    }
}

```

## Anexo L: ExpenseState.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Expense State")]
    public class ExpenseState
    {
        [Key]
        public int ExpenseState_Id { get; set; }
        [Required, MaxLength(30)]
        public string State { get; set; }

        //Navigation Properties
        public List<Expense> Expenses { get; set; }
    }
}

```

## Anexo M: ExpenseType.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Expense Type")]
    public class ExpenseType
    {
        [Key]
        public int ExpenseType_Id { get; set; }
        [Required, MaxLength(30)]
        public string Type { get; set; }

        //Navigation Properties
        public List<Expense> Expenses { get; set; }
    }
}

```

## Anexo N: File.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "File")]
    public class File
    {
        [Key]
        public int File_Id { get; set; }
        [Required]

```

```

        public string base64 { get; set; }

    }
}

```

## Anexo O: FileContent.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"File Content")]
    public class FileContent
    {
        [Key]
        public int FileContent_Id { get; set; }
        [Required, MaxLength(250)]
        public string Name { get; set; }

        //Navigation Properties
        public int FileContentTypeId { get; set; }
        public FileContentType FileContentType { get; set; }

        public int FileId { get; set; }
        public File File { get; set; }

        public List<Activity_File> Activity_File { get; set; }
        public List<Expense_File> Expense_File { get; set; }
    }
}

```

## Anexo P: FileContentType.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"File Content Type")]
    public class FileContentType
    {
        [Key]
        public int FileContentType_Id { get; set; }
        [Required, MaxLength(30)]
        public string Type { get; set; }

        //Navigation Properties
    }
}

```

## Anexo Q: Line.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

```

```

namespace Timesheet_Expenses_API.Models
{[Table(name:"Line")]
    public class Line
    {
        [Key]
        public int Cod_Line { get; set; }
        public decimal UnityPrice { get; set; }
        public DateTime Date { get; set; }
        public string Discription { get; set; }
        public int lineCity { get; set; }
        public int lineType { get; set; }
        //Navigation Properties
        public int ExpenseId { get; set; }
        public Expense Expense { get; set; }
    }
}

```

## Anexo R: LineCity.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models

{[Table(name: "LineCity")]
    public class LineCity
    {
        [Key]
        public int LineCityID { get; set; }
        public string City { get; set; }

        //Navigation Properties
        public List<Line> line { get; set; }
    }
}

```

## Anexo S: LineType.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models

{
    [Table(name: "LineType")]
    public class LineType
    {
        [Key]
        public int LineTypeID { get; set; }
        public string Type { get; set; }
        // migrations
        public List<Line> line { get; set; }
    }
}

```

```
}
```

## Anexo T: Project.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Project")]
    public class Project
    {
        [Key]
        public int Project_Id { get; set; }
        [Required, MaxLength(150)]
        public string Name { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }

        //Navigation Properties
        public int ClientId { get; set; }
        public Client Client { get; set; }

        public int ProjectStateId { get; set; }
        public ProjectState ProjectState { get; set; }

        public List<Team> Team { get; set; }
        public List<Activity> Activity { get; set; }
        public List<Expense> Expenses { get; set; }
    }
}
```

## Anexo U: ProjectState.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Project State")]
    public class ProjectState
    {
        [Key]
        public int ProjectState_Id { get; set; }
        [Required, MaxLength(30)]
        public string State { get; set; }

        //Navigation Properties
        public List<Project> Project { get; set; }
    }
}
```

## Anexo V: Team.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Team")]
    public class Team
    {
        [Required, MaxLength(100)]
        public string TeamName { get; set; }

        //Navigation Properties
        public int UserId { get; set; }
        public User user { get; set; }

        public int ProjectId { get; set; }
        public Project project { get; set; }

        public int UserFunctionId { get; set; }
        public UserFunction UserFunction { get; set; }
    }
}

```

## Anexo W: User.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "ArquiUsers")]
    public class User
    {
        [Key]
        public int User_Id { get; set; }
        [Required, MaxLength(254)]
        public string Email { get; set; }
        [Required, MaxLength(250)]
        public string Name { get; set; }

        //Navigation Properties
        public List<Team> Team { get; set; }
        public List<User_Activity> user_Activities { get; set; }
        public List<Worklog> Worklog { get; set; }
        public List<Expense> Expenses { get; set; }
    }
}

```

## Anexo X: User\_Activity.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "User_Activity")]
    public class User_Activity
    {
        //Navigation Properties
    }
}

```

```

        public int UserId { get; set; }
        public User user { get; set; }

        public int ActivityId { get; set; }
        public Activity activity { get; set; }
    }
}

```

## Anexo Y: UserFunction.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"User Function")]
    public class UserFunction
    {
        [Key]
        public int UserFunction_Id { get; set; }
        [Required, MaxLength(30)]
        public string Function { get; set; }

        //Navigation Properties
        List<Team> teams { get; set; }
    }
}

```

## Anexo Z: Worklog.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name:"Worklog")]
    public class Worklog
    {
        [Key]
        public int Cod_Worklog { get; set; }
        [Required]
        public DateTime Date { get; set; }
        [Required]
        public decimal Hours { get; set; }
        public string Comment { get; set; }

        //Navigation Properties
        public int UserId { get; set; }
        public User User { get; set; }

        public int ActivityId { get; set; }
        public Activity Activity { get; set; }

        public int BillingTypeId { get; set; }
        public BillingType BillingType { get; set; }

        public int WorklogStateId { get; set; }
        public WorklogState WorklogState { get; set; }
    }
}

```

```
}
```

## Anexo AA: WorklogState.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Timesheet_Expenses_API.Models
{
    [Table(name: "Worklog State")]
    public class WorklogState
    {
        [Key]
        public int WorklogState_Id { get; set; }
        [MaxLength(50)]
        public string State { get; set; }

        //Navigation Properties
        public List<Worklog> Worklog { get; set; }
    }
}
```

## Anexo AB: ActivityHours.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ActivityHours
    {
        public ActivityIdName activityIdName { get; set; }
        public decimal hours { get; set; }
    }
}
```

## Anexo AC: ActivityIdName.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ActivityIdName
    {
        public string ActivityName { get; set; }
        public int ActivityId { get; set; }
    }
}
```

## Anexo AD: ActivityInfo.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ActivityInfo
    {
        public ActivityIdName ActivityIdName { get; set; }
        public string ActivityState { get; set; }
        public string ActivityType { get; set; }
        public string ActivityDescription { get; set; }
        public ProjectsIdName ProjectInfo { get; set; }
    }
}
```



## Anexo AE: ClientEmailName.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ClientEmailName
    {
        public string Email { get; set; }
        public string Name { get; set; }
    }
}
```

## Anexo AF: Date.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class Date
    {
        public int Day { get; set; }
        public int Month { get; set; }
        public int Year { get; set; }
    }
}
```

## Anexo AG: ProjectHours.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ProjectHours
    {
        public ProjectsIdName projectsIdName { get; set; }
        public decimal hours { get; set; }
    }
}
```

## Anexo AH: ProjectInfo.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ProjectInfo
    {
        public ProjectsIdName ProjectsIdName { get; set; }
        public List<ActivityIdName> ProjectActivities { get; set; }
        public string ProjectState { get; set; }
        public ClientEmailName Client { get; set; }
        public int StartDay { get; set; }
        public int StartMonth { get; set; }
        public int StartYear { get; set; }
        public int EndDay { get; set; }
        public int EndMonth { get; set; }
        public int EndYear { get; set; }
        public List<TeamInfo> Teams { get; set; }
    }
}
```

## Anexo AI: ProjectIdName.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class ProjectsIdName
    {
        public int projectId { get; set; }
        public string projectName { get; set; }
    }
}
```

## Anexo AJ: TeamInfo.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class TeamInfo
    {
        public string Name { get; set; }
        public List<TimesheetUserInfo> UserInfo { get; set; }
    }
}
```

## Anexo AK: TimesheetUserInfo.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class TimesheetUserInfo
    {
        public int UserId { get; set; }
        public string UserName { get; set; }
        public string Function { get; set; }
    }
}
```

## Anexo AL: TimesheetWorklog.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class TimesheetWorklog
    {
        public ActivityIdName Activity { get; set; }
        public WorklogInfo Monday { get; set; }
        public WorklogInfo Tuesday { get; set; }
        public WorklogInfo Wednesday { get; set; }
        public WorklogInfo Thursday { get; set; }
        public WorklogInfo Friday { get; set; }
        public WorklogInfo Saturday { get; set; }
        public WorklogInfo Sunday { get; set; }
    }
}
```

## Anexo AM: UserActivityWorklogs.cs

```
namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class UserActivityWorklogs
```

```

    {
        public int worklogId { get; set; }
        public int Day { get; set; }
        public int Month { get; set; }
        public int Year { get; set; }
        public decimal hours { get; set; }
    }
}

```

### Anexo AN: UserInfo.cs

```

namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class UserInfo
    {
        public string UserName { get; set; }
        public string UserEmail { get; set; }
    }
}

```

### Anexo AO: WorklogCompleteInfo.cs

```

namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class WorklogCompleteInfo
    {
        public string ActivityName { get; set; }
        public int WorklogId { get; set; }
        public int ActivityId { get; set; }
        public string Comment { get; set; }
        public string WorklogState { get; set; }
        public string BillingType { get; set; }
        public decimal Hours { get; set; }
        public int Day { get; set; }
        public int Month { get; set; }
        public int Year { get; set; }
    }
}

```

### Anexo AP: WorklogInfo.cs

```

namespace Timesheet_Expenses_API.Models.Object.Timesheet
{
    public class WorklogInfo
    {
        public int worklogId { get; set; }
        public decimal Hours { get; set; }
    }
}

```

### Anexo AQ: TimesheetRepository.cs – ITimesheetRepository

```

public interface ITimesheetRepository
{
    public int GetUserId(string email);
    public WorklogCompleteInfo GetWorklog(int worklogId);
}

```

```

        public bool CreateWorklog(DateTime date, decimal hours, string
comment, int activity, string billingType, string worklogState, int
userId);
        public bool CreateWorklogByPeriod(DateTime EndDate, DateTime
StartDate, decimal hours, string comment, int activity, string billingType,
string worklogState, int userId);
        public bool UpdateWorklog(int worklogId, decimal hours, string
comment, string billingType, string worklogState);
        public bool DeleteWorklog(int worklogId);
        public List<ProjectsIdName> GetProjectUser(int userId);
        public List<ActivityIdName> GetActivityUser(int userId, int
projectId);
        public List<TimesheetWorklog> GetUserWeekWorklog(DateTime date, int
userId);
        public ActivityInfo GetActivitiesInfo(int activityId);
        public ProjectInfo GetProjectInfo(int projectId);
        public UserInfo GetUserInfo(int userId);
        public List<string> GetBillingTypes();
        public List<string> GetWorklogState();
        public Date AddDays(DateTime date, int AddDays);
        public List<ProjectHours> GetProjectHour(int userId);
        public List<ActivityHours> GetActivityHours(int userId, int
projectId);
        public List<UserActivityWorklogs> GetUserActivityWorklogs(int
userId, int activityId);
    }

```

## Anexo AR: TimesheetRepository.cs – variáveis e default-constructor

```

#region variables
private readonly _DbContext db;
#endregion

//Default-Constructor
public TimesheetRepository(_DbContext _db)
{
    db = _db;
}

```

## Anexo AS: TimesheetRepository.cs – GetUserId

```

//recebe o email do utilizador e devolve o id do mesmo (o id vai
ser guardado em cache na app utilizada)
public int GetUserId(string email)
{
    try
    {
        int userId = db.users.Where(u =>
u.Email.Equals(email)).FirstOrDefault().User_Id;

        return userId;
    }
    catch
    {
        return 0;
    }
}

```

## Anexo AT: TimesheetRepository.cs – GetWorklog

```
//recebe um worklogId e devolve as informações sobre o mesmo
indicado
public WorklogCompleteInfo GetWorklog(int worklogId)
{
    try
    {
        var worklog = db.worklogs.Find(worklogId);
        WorklogCompleteInfo wlInfo = new WorklogCompleteInfo();
        wlInfo.WorklogId = worklogId;
        wlInfo.ActivityName =
db.activities.Find(worklog.ActivityId).Name;
        wlInfo.ActivityId = worklog.ActivityId;
        wlInfo.BillingType =
db.billingTypes.Find(worklog.BillingTypeId).Type;
        wlInfo.WorklogState =
db.worklogStates.Find(worklog.WorklogStateId).State;
        wlInfo.Hours = worklog.Hours;
        wlInfo.Comment = worklog.Comment;
        int day = worklog.Date.Day;
        int month = worklog.Date.Month;
        int year = worklog.Date.Year;
        wlInfo.Day = day;
        wlInfo.Month = month;
        wlInfo.Year = year;

        return wlInfo;
    }
    catch
    {
        return new WorklogCompleteInfo();
    }
}
```

## Anexo AU: TimesheetRepository.cs – CreateWorklog

```
//cria um objecto do tipo Worklog e adiciona os dados do mesmo à
base de dados
public bool CreateWorklog(DateTime date, decimal hours, string
comment, int activity, string billingType, string worklogState, int userId)
{
    try
    {
        //verifica se esta worklog existe
        var verifWL1 = db.worklogs.Where(wl => wl.Date.Equals(date)
&& wl.ActivityId.Equals(activity) && wl.UserId.Equals(userId)).ToList();
        if (verifWL1.Count() != 0)
            throw new Exception("This worklog already exists!!");

        //verifica se neste dia o total de horas já excedeu as 24
horas
        var verifWL2 = db.worklogs.Where(wl => wl.Date.Equals(date)
&& wl.UserId.Equals(userId)).ToList();
        decimal dayHours = hours;
        foreach (Worklog wl in verifWL2)
        {
            dayHours += wl.Hours;
        }
    }
}
```

```

    }
    if (dayHours > 24)
    {
        throw new Exception("Excedeu as horas de um dia!");
    }

    //cria a worklog
    var worklog_db = new Worklog
    {
        Date = date,
        Hours = hours,
        Comment = comment,
        User = db.users.Find(userId),
        Activity = db.activities.Find(activity),
        WorklogState =
db.worklogStates.Find(db.worklogStates.Where(ws =>
ws.State.Equals(worklogState)).FirstOrDefault().WorklogState_Id),
        BillingType =
db.billingTypes.Find(db.billingTypes.Where(bt =>
bt.Type.Equals(billingType)).FirstOrDefault().BillingType_Id)
    };
    //adiciona à base de dados e salva as alterações
    db.worklogs.Add(worklog_db);
    db.SaveChanges();

    return true;
}
catch
{
    return false;
}
}

```

## Anexo AV: TimesheetRepository.cs – CreateWorklogByPeriod

```

    //cria um objecto do tipo Worklog para cada dia de entre duas datas
    e adiciona os dados do mesmo à base de dados
    public bool CreateWorklogByPeriod(DateTime EndDate, DateTime
StartDate, decimal hours, string comment, int activity, string billingType,
string worklogState, int userId)
    {
        try
        {
            //verifica se existe alguma worklog no periodo escolhido
            DateTime DateAux = StartDate;
            while (DateAux != EndDate)
            {
                var verifWL = db.worklogs.Where(wl =>
wl.Date.Equals(DateAux) && wl.ActivityId.Equals(activity) &&
wl.UserId.Equals(userId)).ToList();
                if (verifWL.Count() != 0)
                    throw new Exception("This worklog already
exists!!");

                //verifica se neste dia o total de horas já excedeu as
24 horas
                var verifWL2 = db.worklogs.Where(wl =>
wl.Date.Equals(DateAux) && wl.UserId.Equals(userId)).ToList();
                decimal dayHours = hours;
                foreach (Worklog wl in verifWL2)

```

```

        {
            dayHours += wl.Hours;
        }
        if (dayHours > 24)
        {
            throw new Exception("Excedeu as horas de um dia!");
        }

        DateAux = DateAux.AddDays(1);
    }

    while (StartDate != EndDate)
    {
        // cria a worklog
        var worklog_db = new Worklog
        {
            Date = StartDate,
            Hours = hours,
            Comment = comment,
            User = db.users.Find(userId),
            Activity = db.activities.Find(activity),
            WorklogState =
db.worklogStates.Find(db.worklogStates.Where(ws =>
ws.State.Equals(worklogState)).FirstOrDefault().WorklogState_Id),
            BillingType =
db.billingTypes.Find(db.billingTypes.Where(bt =>
bt.Type.Equals(billingType)).FirstOrDefault().BillingType_Id)
        };
        //adiciona à base de dados e salva as alterações
        db.worklogs.Add(worklog_db);
        db.SaveChanges();
        //adiciono um dia à data inicial
        StartDate = StartDate.AddDays(1);
    }

    return true;
}
catch
{
    return false;
}
}

```

## Anexo AW: TimesheetRepository.cs - UpdateWorklog

```

//recebe os novos valores de uma worklog e atualiza os mesmos
public bool UpdateWorklog(int worklogId, decimal hours, string
comment, string billingType, string worklogState)
{
    try
    {
        //procura o registo com o id indicado
        var worklog_db = db.worklogs.Find(worklogId);
        //atualiza os dados igualando os mesmos
        worklog_db.Hours = hours;
        worklog_db.Comment = comment;
        worklog_db.WorklogState =
db.worklogStates.Find(db.worklogStates.Where(ws =>
ws.State.Equals(worklogState)).FirstOrDefault().WorklogState_Id);
    }
}

```

```

        worklog_db.BillingType =
db.billingTypes.Find(db.billingTypes.Where(bt =>
bt.Type.Equals(billingType)).FirstOrDefault().BillingType_Id);
        db.SaveChanges();

        return true;
    }
    catch
    {
        return false;
    }
}

```

## Anexo AX: TimesheetRepository.cs - DeleteWorklog

```

//recebe um id de uma worklog e elimina a mesma da base de dados
public bool DeleteWorklog(int worklogId)
{
    try
    {
        //procura a worklog com o id recebido
        var worklog_db = db.worklogs.Find(worklogId);
        db.worklogs.Remove(worklog_db);
        db.SaveChanges();

        return true;
    }
    catch
    {
        return false;
    }
}

```

## Anexo AY: TimesheetRepository.cs – GetActivityUser

```

//recebe um id do user e um id do projecto seleccionado, devolve uma
lista com o nome e id das atividades relacionadas com o user
public List<ActivityIdName> GetActivityUser(int userId, int
projectId)
{
    try
    {
        List<ActivityIdName> ActivityUsers = new
List<ActivityIdName>();

        //lista de User_Activity com apenas o userId indicado
        var userActivity_db = db.activities_users.Where(ua =>
ua.UserId.Equals(userId)).ToList();
        //adicionar os ids das atividades relacionadas ao user
        //incicados a uma lista de números inteiros
        List<int> activitiesIds = new List<int>();
        foreach (User_Activity ua in userActivity_db)
        {
            activitiesIds.Add(ua.ActivityId);
        }

        //lista de Activity com apenas o projectId indicado
    }
}

```



```

        var activities = db.activities.Where(a =>
a.Project.Project_Id.Equals(projectId)).ToList();
        //comparar os ids das atividades relacionadas com o user
        indicado com os ids das atividades relacionadas com o projecto indicado
        foreach (Activity a in activities)
        {
            foreach (int i in activitiesIds)
            {
                //caso seja igual adicionar o mesmo à lista
                if (a.Activity_Id == i)
                {
                    if (a.ActivityStateId != 2)
                    {
                        ActivityIdName actUser = new
ActivityIdName();

                        actUser.ActivityId = a.Activity_Id;
                        actUser.ActivityName = a.Name;
                        ActivityUsers.Add(actUser);
                    }
                }
            }

            return ActivityUsers;
        }
    }
    catch
    {
        return new List<ActivityIdName>();
    }
}

```

## Anexo AZ: TimesheetRepository.cs – GetProjectUser

```

        //recebe o id do user e devolve uma lista com o nome e id de todos
os projetos ao qual está relacionado
        public List<ProjectsIdName> GetProjectUser(int userId)
        {
            try
            {
                //procura as equipas com o userId indicado
                var team = db.teams.Where(t =>
t.UserId.Equals(userId)).ToList();
                //cria e devolve a lista de projectos que se relacionam com
o user

                List<ProjectsIdName> projectsInfo = new
List<ProjectsIdName>();
                foreach (Team t in team)
                {
                    var proj = db.projects.Find(t.ProjectId);
                    if (proj.ProjectStateId == 1)
                    {
                        var pInfo = new ProjectsIdName();
                        pInfo.projectId = proj.Project_Id;
                        pInfo.projectName = proj.Name;
                        projectsInfo.Add(pInfo);
                    }
                }

                return projectsInfo;
            }
        }
    }
}

```

```

        catch
        {
            return new List<ProjectsIdName>();
        }
    }
}

```

## Anexo BA: TimesheetRepository.cs – GetUserWeekWorklog

```

//recebe a data indicada e devolve uma lista com todas as worklogs
do user
public List<TimesheetWorklog> GetUserWeekWorklog(DateTime date, int
userId)
{
    try
    {
        List<TimesheetWorklog> TimesheetWorklogs = new
List<TimesheetWorklog>();

        //vai percorrer a semana toda e adicionar todas as worklogs
da semana do user indicado a uma lista
        List<Worklog> worklog = new List<Worklog>();
        for (int i = 0; i < 7; i++)
        {
            List<Worklog> worklogAux = db.worklogs.Where(wl =>
wl.User.User_Id.Equals(userId) &&
wl.Date.Equals(date.AddDays(i))).ToList();
            foreach (Worklog w in worklogAux)
            {
                worklog.Add(w);
            }
        }

        //todos os ids e nomes das diferentes atividades da worklog
        List<ActivityIdName> activitiesInfos = new
List<ActivityIdName>();
        foreach (Worklog wl in worklog)
        {
            bool aux = false;
            foreach (ActivityIdName ai in activitiesInfos)
            {
                if (ai.ActivityId == wl.ActivityId)
                {
                    aux = true;
                    break;
                }
            }
            //caso o id da worklog ainda não esteja alucado na
lista activitiesInfos vai criar um objeto ActivityInfo e adicionar o mesmo
            if (aux == false)
            {
                ActivityIdName actInfo = new ActivityIdName();
                actInfo.ActivityId = wl.ActivityId;
                actInfo.ActivityName =
db.activities.Find(wl.ActivityId).Name;
                activitiesInfos.Add(actInfo);
            }
        }

        //vai criar um objeto TimesheetWorklog e adicionar o mesmo
à lista a que vamos dar return

```

```

foreach (ActivityIdName ai in activitiesInfos)
{
    TimesheetWorklog tw = new TimesheetWorklog();
    tw.Activity = ai;
    foreach (Worklog wl in worklog)
    {
        if (ai.ActivityId == wl.ActivityId)
        {
            WorklogInfo wli = new WorklogInfo();
            wli.worklogId = wl.Cod_Worklog;
            wli.Hours = wl.Hours;
            //ver o dia da semana
            switch (wl.Date.DayOfWeek)
            {
                case DayOfWeek.Monday:
                    tw.Monday = wli;
                    break;
                case DayOfWeek.Tuesday:
                    tw.Tuesday = wli;
                    break;
                case DayOfWeek.Wednesday:
                    tw.Wednesday = wli;
                    break;
                case DayOfWeek.Thursday:
                    tw.Thursday = wli;
                    break;
                case DayOfWeek.Friday:
                    tw.Friday = wli;
                    break;
                case DayOfWeek.Saturday:
                    tw.Saturday = wli;
                    break;
                case DayOfWeek.Sunday:
                    tw.Sunday = wli;
                    break;
            }
        }
    }
    TimesheetWorklogs.Add(tw);
}

return TimesheetWorklogs;
}
catch
{
    return new List<TimesheetWorklog>();
}
}

```

## Anexo BB: TimesheetRepository.cs – GetActivitiesInfo

```

//recebe um id de uma atividade e devolve a informação da
atividade, assim como uma lista de todos os ficheiros(nome e id, o proprio
ficheiro não é enviado aqui) relacionados com essa atividade
public ActivityInfo GetActivitiesInfo(int activityId)
{
    try
    {
        //procura a Activity com o id recebido
        var activity = db.activities.Find(activityId);
    }
}

```

```

do mesmo //cria um objeto do tipo ActivityInfo e preenche os campos

        ActivityInfo activityInfos = new ActivityInfo();
        ActivityIdName actIdName = new ActivityIdName();
        actIdName.ActivityName = activity.Name;
        actIdName.ActivityId = activity.Activity_Id;
        activityInfos.ActivityIdName = actIdName;

        activityInfos.ActivityState =
db.activityState.Find(activity.ActivityStateId).State;
        activityInfos.ActivityType =
db.activityType.Find(activity.ActivityTypeId).Type;
        activityInfos.ActivityDescription = activity.Description;

        ProjectsIdName projIdName = new ProjectsIdName();
        projIdName.projectId =
db.projects.Find(activity.ProjectId).Project_Id;
        projIdName.projectName =
db.projects.Find(activity.ProjectId).Name;
        activityInfos.ProjectInfo = projIdName;

        return activityInfos;
    }
    catch
    {
        return new ActivityInfo();
    }
}

```

## Anexo BC: TimesheetRepository.cs – GetProjectInfo

```

//recebe o id do project indicado e devolve as informações do
mesmo, tal como as equipas que têm algum tipo de relação com o mesmo
public ProjectInfo GetProjectInfo(int projectId)
{
    try
    {
        ProjectInfo projectInfo = new ProjectInfo();
        //encontra o Project com o projectId indicado
        var project = db.projects.Find(projectId);

        //adicionar o ProjectName e projectId do Project
        ProjectsIdName projectIdName = new ProjectsIdName();
        projectIdName.projectId = project.Project_Id;
        projectIdName.projectName = project.Name;
        projectInfo.ProjectsIdName = projectIdName;

        //cria uma lista de Activities que estão relacionadas ao
Project
        var activitiesProj = db.activities.Where(a =>
a.ProjectId.Equals(projectId)).ToList();
        List<ActivityIdName> activities = new
List<ActivityIdName>();
        foreach (Activity a in activitiesProj)
        {
            ActivityIdName actIdName = new ActivityIdName();
            actIdName.ActivityId = a.Activity_Id;
            actIdName.ActivityName = a.Name;
            activities.Add(actIdName);
        }
    }
}

```

```

    }
    projectInfo.ProjectActivities = activities;
    //adiciona ProjectState e StartDate e EndDate
    projectInfo.ProjectState =
db.projectStates.Find(project.ProjectStateId).State;
    projectInfo.StartDay = project.StartDate.Day;
    projectInfo.StartMonth = project.StartDate.Month;
    projectInfo.StartYear = project.StartDate.Year;
    projectInfo.EndDay = project.EndDate.Day;
    projectInfo.EndMonth = project.EndDate.Month;
    projectInfo.EndYear = project.EndDate.Year;

    //adiciona as informações sobre o Client
    ClientEmailName client = new ClientEmailName();
    client.Email = db.client.Find(project.ClientId).Email;
    client.Name = db.client.Find(project.ClientId).Name;
    projectInfo.Client = client;

    //cria um objecto de TeamInfo e adiciona o mesmo a
projectInfo
    var projectTeams = db.teams.Where(t =>
t.ProjectId.Equals(projectId)).ToList();
    List<TeamInfo> teams = new List<TeamInfo>();
    foreach (Team t in projectTeams)
    {
        bool aux = false;
        foreach (TeamInfo tf in teams)
        {
            if (tf.Name == t.TeamName)
            {
                aux = true;
                break;
            }
        }
        //vai criar um objeto do tipo TeamInfo caso o TeamName
seja novo para a coleção teams
        if (aux == false)
        {
            //adiciona ao objeto TeamInfo o TeamName
            TeamInfo teamInfo = new TeamInfo();
            teamInfo.Name = t.TeamName;
            //procura todos os registos das teams com o Name
igual e adiciona a uma List<Team>
            var teamNames = db.teams.Where(ta =>
ta.TeamName.Equals(teamInfo.Name)).ToList();
            List<TimesheetUserInfo> userInfos = new
List<TimesheetUserInfo>();
            foreach (Team tm in teamNames)
            {
                //para cada User dentro da Team vai adicionar
as informações do mesmo
                TimesheetUserInfo timesheetUserInfo = new
TimesheetUserInfo();
                timesheetUserInfo.UserId =
db.users.Find(tm.UserId).User_Id;
                timesheetUserInfo.UserName =
db.users.Find(tm.UserId).Name;
                timesheetUserInfo.Function =
db.userFunction.Find(tm.UserFunctionId).Function;
                userInfos.Add(timesheetUserInfo);
            }
        }
    }

```

```

//adiciona a TimesheetUserInfo e por fim adiciona a
teams o objecto teamInfo
        teamInfo.UserInfo = userInfos;
        teams.Add(teamInfo);
    }
}
projectInfo.Teams = teams;

return projectInfo;
}
catch
{
    return new ProjectInfo();
}
}

```

## Anexo BD: TimesheetRepository.cs – GetBillingTypes

```

//retorna todos os billing types
public List<string> GetBillingTypes()
{
    try
    {
        List<string> BillingTypes = new List<string>();
        var BillingTypesObj = db.billingTypes.ToList();
        foreach (BillingType bt in BillingTypesObj)
        {
            BillingTypes.Add(bt.Type);
        }

        return BillingTypes;
    }
    catch
    {
        return new List<string>();
    }
}

```

## Anexo BE: TimesheetRepository.cs – GetWorklogState

```

//retorna todos os worklog states
public List<string> GetWorklogState()
{
    try
    {
        List<string> WorklogState = new List<string>();
        var WorklogStateObj = db.worklogStates.ToList();
        foreach (WorklogState bt in WorklogStateObj)
        {
            WorklogState.Add(bt.State);
        }

        return WorklogState;
    }
    catch
    {
        return new List<string>();
    }
}

```

}

## Anexo BF: TimesheetRepository.cs – AddDays

```
//adiciona o numero de dias referidos na data recebida, retorna a
segunda feira da data já com os dias adicionados
public Date AddDays(DateTime date, int AddDays)
{
    try
    {
        Date mondayDate = new Date();

        date = date.AddDays(AddDays);
        switch (date.DayOfWeek)
        {
            case DayOfWeek.Monday:
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
            case DayOfWeek.Tuesday:
                date = date.AddDays(-1);
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
            case DayOfWeek.Wednesday:
                date = date.AddDays(-2);
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
            case DayOfWeek.Thursday:
                date = date.AddDays(-3);
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
            case DayOfWeek.Friday:
                date = date.AddDays(-4);
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
            case DayOfWeek.Saturday:
                date = date.AddDays(-5);
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
            case DayOfWeek.Sunday:
                date = date.AddDays(-6);
                mondayDate.Day = date.Day;
                mondayDate.Month = date.Month;
                mondayDate.Year = date.Year;
                break;
        }

        return mondayDate;
    }
}
```

```

        catch
        {
            return new Date();
        }
    }
}

```

## Anexo BG: TimesheetRepository.cs – GetProjectHour

```

//recebe um userId e retorna todos os projetos com as respectivas
horas gastas
public List<ProjectHours> GetProjectHour(int userId)
{
    try
    {
        List<ProjectHours> projects = new List<ProjectHours>();

        //procura as equipas com o userId indicado
        var team = db.teams.Where(t =>
t.UserId.Equals(userId)).ToList();
        //cria e devolve a lista de projectos que se relacionam com
o user
        List<ProjectsIdName> projectsInfo = new
List<ProjectsIdName>();
        foreach (Team t in team)
        {
            var proj = db.projects.Find(t.ProjectId);
            var pInfo = new ProjectsIdName();
            pInfo.projectId = proj.Project_Id;
            pInfo.projectName = proj.Name;
            projectsInfo.Add(pInfo);
        }

        //adicionar a cada projeto relacionado ao user as horas que
gastou
        foreach (ProjectsIdName pin in projectsInfo)
        {
            var activities = db.activities.Where(a =>
a.ProjectId.Equals(pin.projectId)).ToList();
            decimal hours = 0;
            foreach (Activity act in activities)
            {
                var worklogs = db.worklogs.Where(wl =>
wl.ActivityId.Equals(act.Activity_Id) &&
wl.UserId.Equals(userId)).ToList();
                foreach (Worklog wl in worklogs)
                {
                    hours += wl.Hours;
                }
            }
            //criar o objeto ProjectHours
            ProjectHours projHours = new ProjectHours();
            projHours.hours = hours;
            projHours.projectsIdName = pin;
            projects.Add(projHours);
        }

        return projects;
    }
    catch
    {

```



```

        return new List<ProjectHours>();
    }
}

```

## Anexo BH: TimesheetRepository.cs - GetActivityHours

```

//recebe um userId e um projectId e retorna todas as atividades e
as respectivas horas
public List<ActivityHours> GetActivityHours(int userId, int
projectId)
{
    try
    {
        List<ActivityHours> activityHours = new
List<ActivityHours>();

        //procura todas as activities relacionadas com aquele
projeto
        var activities = db.activities.Where(a =>
a.ProjectId.Equals(projectId)).ToList();
        foreach (Activity act in activities)
        {
            decimal hours = 0;
            //procura todas as worklogs relacionadas com os dois
            var worklogs = db.worklogs.Where(wl =>
wl.ActivityId.Equals(act.Activity_Id) &&
wl.UserId.Equals(userId)).ToList();
            foreach (Worklog wl in worklogs)
            {
                hours += wl.Hours;
            }
            ActivityIdName actIdName = new ActivityIdName();
            actIdName.ActivityId = act.Activity_Id;
            actIdName.ActivityName = act.Name;
            //criar objeto para adicionar à lista
            ActivityHours actHours = new ActivityHours();
            actHours.hours = hours;
            actHours.activityIdName = actIdName;
            activityHours.Add(actHours);
        }

        return activityHours;
    }
    catch
    {
        return new List<ActivityHours>();
    }
}

```

## Anexo BI: TimesheetRepository.cs – GetUserActivityWorklogs

```

//recebe um userId e um activityId e retorna todas as worklogs
relacionadas com os dois
public List<UserActivityWorklogs> GetUserActivityWorklogs(int
userId, int activityId)
{
    try
    {

```

```

        List<UserActivityWorklogs> userActivityWorklogs = new
List<UserActivityWorklogs>();

        //procura todas as worklogs relacionadas com os dois
        var worklogs = db.worklogs.Where(wl =>
wl.ActivityId.Equals(activityId) && wl.UserId.Equals(userId)).ToList();
        foreach (Worklog wl in worklogs)
        {
            //cria o objeto e adiciona à lista
            UserActivityWorklogs userActWl = new
UserActivityWorklogs();
            userActWl.worklogId = wl.Cod_Worklog;
            userActWl.hours = wl.Hours;
            userActWl.Day = wl.Date.Day;
            userActWl.Month = wl.Date.Month;
            userActWl.Year = wl.Date.Year;
            userActivityWorklogs.Add(userActWl);
        }

        return userActivityWorklogs;
    }
    catch
    {
        return new List<UserActivityWorklogs>();
    }
}

```

## Anexo BJ: TimesheetController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using Timesheet_Expenses_API.Models.Object.Timesheet;
using Timesheet_Expenses_API.Repositories;

namespace Timesheet_Expenses_API.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class TimesheetController : Controller
    {
        private readonly ITimesheetRepository repos;

        public TimesheetController(ITimesheetRepository _repos)
        {
            repos = _repos;
        }

        [HttpGet]
        public IActionResult GetBillingTypes()
        {
            var BillingTypes_db = repos.GetBillingTypes();
            return Ok(BillingTypes_db);
        }

        [HttpGet]
        public IActionResult GetWorklogState()
        {
            var WorklogState_db = repos.GetWorklogState();
            return Ok(WorklogState_db);
        }
    }
}

```

```

[HttpGet("{ userEmail}")]
public IActionResult GetUserId([FromRoute] string userEmail)
{
    var uId_db = repos.GetUserId(userEmail);
    return Ok(uId_db);
}

[HttpGet("{ userId}")]
public IActionResult GetProjectUser([FromRoute] int userId)
{
    var projects_db = repos.GetProjectUser(userId);
    return Ok(projects_db);
}

[HttpGet("{ userId}")]
public IActionResult GetProjectHour([FromRoute] int userId)
{
    var ProjectHours_db = repos.GetProjectHour(userId);
    return Ok(ProjectHours_db);
}

[HttpGet("{ userId}; {projectId}")]
public IActionResult GetActivityHours([FromRoute] int userId, int
projectId)
{
    var ActivityHours_db = repos.GetActivityHours(userId,
projectId);
    return Ok(ActivityHours_db);
}

[HttpGet("{ userId}; {activityId}")]
public IActionResult GetUserActivityWorklogs([FromRoute] int
userId, int activityId)
{
    var GetUserActivityWorklogs_db =
repos.GetUserActivityWorklogs(userId, activityId);
    return Ok(GetUserActivityWorklogs_db);
}

[HttpGet("{ userId}; {projectId}")]
public IActionResult GetActivityUser([FromRoute] int userId, int
projectId)
{
    var ActivityUser_db = repos.GetActivityUser(userId, projectId);
    return Ok(ActivityUser_db);
}

[HttpGet("{ date}; {userId}")]
public IActionResult GetUserWeekWorklog([FromRoute] DateTime date,
int userId)
{
    var weekWorklog_db = repos.GetUserWeekWorklog(date, userId);
    return Ok(weekWorklog_db);
}

[HttpGet("{ date}; {AddDays}")]
public IActionResult AddDays([FromRoute] DateTime date, int
AddDays)
{
    var newDate_db = repos.AddDays(date, AddDays);

```

```

        return Ok(newDate_db);
    }

    [HttpGet("{activityId}")]
    public IActionResult GetActivitiesInfo([FromRoute] int activityId)
    {
        var activityInfo_db = repos.GetActivitiesInfo(activityId);
        return Ok(activityInfo_db);
    }

    [HttpGet("{projectId}")]
    public IActionResult GetProjectInfo([FromRoute] int projectId)
    {
        var projectInfo_db = repos.GetProjectInfo(projectId);
        return Ok(projectInfo_db);
    }

    [HttpGet("{userId}")]
    public IActionResult GetUserInfo([FromRoute] int userId)
    {
        var userInfo_db = repos.GetUserInfo(userId);
        return Ok(userInfo_db);
    }

    [HttpGet("{worklogId}")]
    public IActionResult GetWorklog([FromRoute] int worklogId)
    {
        var worklogId_db = repos.GetWorklog(worklogId);
        return Ok(worklogId_db);
    }

    [HttpPost("{date};{hours};{comment};{activity};{billingType};{worklogState};{userId}")]
    public IActionResult CreateWorklog([FromRoute] DateTime date,
    decimal hours, string comment, int activity, string billingType, string
    worklogState, int userId)
    {
        var createWl = repos.CreateWorklog(date, hours, comment,
    activity, billingType, worklogState, userId);
        return Ok(createWl);
    }

    [HttpPost("{EndDate};{StartDate};{hours};{comment};{activity};{billingType};{worklogState};{userId}")]
    public IActionResult CreateWorklogByPeriod([FromRoute] DateTime
    EndDate, DateTime StartDate, decimal hours, string comment, int activity,
    string billingType, string worklogState, int userId)
    {
        var createWl = repos.CreateWorklogByPeriod(EndDate, StartDate,
    hours, comment, activity, billingType, worklogState, userId);
        return Ok(createWl);
    }

    [HttpPut("{worklogId};{hours};{comment};{billingType};{worklogState}")]
    public IActionResult UpdateWorklog([FromRoute] int worklogId,
    decimal hours, string comment, string billingType, string worklogState)
    {

```

```

        var updateWl = repos.UpdateWorklog(worklogId, hours, comment,
billingType, worklogState);
        return Ok(updateWl);
    }

    [HttpDelete("{worklogId}")]
    public IActionResult DeleteWorklog([FromRoute] int worklogId)
    {
        var deleteWl = repos.DeleteWorklog(worklogId);
        return Ok(deleteWl);
    }
}

```

## Anexo BK: 2022070712220\_1thMigration.cs – Up

```

protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Activity State",
        columns: table => new
        {
            ActivityState_Id = table.Column<int>(type: "int",
nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            State = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Activity State", x =>
x.ActivityState_Id);
        });

    migrationBuilder.CreateTable(
        name: "Activity Type",
        columns: table => new
        {
            ActivityType_Id = table.Column<int>(type: "int",
nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Type = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)

```

```

        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Activity Type", x =>
x.ActivityType_Id);
        });

migrationBuilder.CreateTable(
    name: "ArquiUsers",
    columns: table => new
    {
        User_Id = table.Column<int>(type: "int", nullable:
false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Email = table.Column<string>(type: "nvarchar(254)",
maxLength: 254, nullable: false),
        Name = table.Column<string>(type: "nvarchar(250)",
maxLength: 250, nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_ArquiUsers", x => x.User_Id);
    });

migrationBuilder.CreateTable(
    name: "Billing Type",
    columns: table => new
    {
        BillingType_Id = table.Column<int>(type: "int",
nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Type = table.Column<string>(type: "nvarchar(50)",
maxLength: 50, nullable: false)
    },
    constraints: table =>
    {

```

```

        table.PrimaryKey("PK_Billing Type", x =>
x.BillingType_Id);

    });

    migrationBuilder.CreateTable(
        name: "Client",
        columns: table => new
        {
            Client_Id = table.Column<int>(type: "int", nullable:
false)

                .Annotation("SqlServer:Identity", "1, 1"),
            Name = table.Column<string>(type: "nvarchar(250)",
maxLength: 250, nullable: false),
            Email = table.Column<string>(type: "nvarchar(254)",
maxLength: 254, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Client", x => x.Client_Id);
        });

    migrationBuilder.CreateTable(
        name: "Expense State",
        columns: table => new
        {
            ExpenseState_Id = table.Column<int>(type: "int",
nullable: false)

                .Annotation("SqlServer:Identity", "1, 1"),
            State = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Expense State", x =>
x.ExpenseState_Id);
        });

```

```

migrationBuilder.CreateTable(
    name: "Expense Type",
    columns: table => new
    {
        ExpenseType_Id = table.Column<int>(type: "int",
nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Type = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Expense Type", x =>
x.ExpenseType_Id);
    });

migrationBuilder.CreateTable(
    name: "File",
    columns: table => new
    {
        File_Id = table.Column<int>(type: "int", nullable:
false)
            .Annotation("SqlServer:Identity", "1, 1"),
        base64 = table.Column<string>(type: "nvarchar(max)",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_File", x => x.File_Id);
    });

migrationBuilder.CreateTable(
    name: "File Content Type",
    columns: table => new
    {
        FileContentType_Id = table.Column<int>(type: "int",
nullable: false)

```



```

        .Annotation("SqlServer:Identity", "1, 1"),
        Type = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_File Content Type", x =>
x.FileContentType_Id);
    });

migrationBuilder.CreateTable(
    name: "LineCity",
    columns: table => new
    {
        LineCityID = table.Column<int>(type: "int", nullable:
false)
        .Annotation("SqlServer:Identity", "1, 1"),
        City = table.Column<string>(type: "nvarchar(max)",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_LineCity", x => x.LineCityID);
    });

migrationBuilder.CreateTable(
    name: "LineType",
    columns: table => new
    {
        LineTypeID = table.Column<int>(type: "int", nullable:
false)
        .Annotation("SqlServer:Identity", "1, 1"),
        Type = table.Column<string>(type: "nvarchar(max)",
nullable: false)
    },
    constraints: table =>
    {

```

```

        table.PrimaryKey("PK_LineType", x => x.LineTypeID);
    });

migrationBuilder.CreateTable(
    name: "Project State",
    columns: table => new
    {
        ProjectState_Id = table.Column<int>(type: "int",
nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        State = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Project State", x =>
x.ProjectState_Id);
    });

migrationBuilder.CreateTable(
    name: "User Function",
    columns: table => new
    {
        UserFunction_Id = table.Column<int>(type: "int",
nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Function = table.Column<string>(type: "nvarchar(30)",
maxLength: 30, nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_User Function", x =>
x.UserFunction_Id);
    });

migrationBuilder.CreateTable(
    name: "Worklog State",

```

```

        columns: table => new
        {
            WorklogState_Id = table.Column<int>(type: "int",
nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            State = table.Column<string>(type: "nvarchar(50)",
maxLength: 50, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Worklog State", x =>
x.WorklogState_Id);
        });

migrationBuilder.CreateTable(
    name: "File Content",
    columns: table => new
    {
        FileContent_Id = table.Column<int>(type: "int",
nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(type: "nvarchar(250)",
maxLength: 250, nullable: false),
        FileContentTypeId = table.Column<int>(type: "int",
nullable: false),
        FileId = table.Column<int>(type: "int", nullable:
false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_File Content", x =>
x.FileContent_Id);
        table.ForeignKey(
            name: "FK_File Content_File Content
Type_FileContentTypeId",
            column: x => x.FileContentTypeId,
            principalTable: "File Content Type",
            principalColumn: "FileContentType_Id",

```

```

        onDelete: ReferentialAction.Cascade);
    table.ForeignKey(
        name: "FK_File Content_File_FileId",
        column: x => x.FileId,
        principalTable: "File",
        principalColumn: "File_Id",
        onDelete: ReferentialAction.Cascade);
    });

    migrationBuilder.CreateTable(
        name: "Project",
        columns: table => new
        {
            Project_Id = table.Column<int>(type: "int", nullable:
false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Name = table.Column<string>(type: "nvarchar(150)",
maxLength: 150, nullable: false),
            StartDate = table.Column<DateTime>(type: "datetime2",
nullable: false),
            EndDate = table.Column<DateTime>(type: "datetime2",
nullable: false),
            ClientId = table.Column<int>(type: "int", nullable:
false),
            ProjectStateId = table.Column<int>(type: "int",
nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Project", x => x.Project_Id);
            table.ForeignKey(
                name: "FK_Project_Client_ClientId",
                column: x => x.ClientId,
                principalTable: "Client",
                principalColumn: "Client_Id",
                onDelete: ReferentialAction.Cascade);
            table.ForeignKey(

```

```

        name: "FK_Project_Project State_ProjectStateId",
        column: x => x.ProjectStateId,
        principalTable: "Project State",
        principalColumn: "ProjectState_Id",
        onDelete: ReferentialAction.Cascade);

    });

migrationBuilder.CreateTable(
    name: "Activity",
    columns: table => new
    {
        Activity_Id = table.Column<int>(type: "int", nullable:
false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(type: "nvarchar(150)",
maxLength: 150, nullable: false),
        Description = table.Column<string>(type:
"nvarchar(max)", nullable: false),
        ProjectId = table.Column<int>(type: "int", nullable:
false),
        ActivityStateId = table.Column<int>(type: "int",
nullable: false),
        ActivityTypeId = table.Column<int>(type: "int",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Activity", x => x.Activity_Id);
        table.ForeignKey(
            name: "FK_Activity_Activity State_ActivityStateId",
            column: x => x.ActivityStateId,
            principalTable: "Activity State",
            principalColumn: "ActivityState_Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Activity_Activity Type_ActivityTypeId",
            column: x => x.ActivityTypeId,

```

```

        principalTable: "Activity Type",
        principalColumn: "ActivityType_Id",
        onDelete: ReferentialAction.Cascade);
    table.ForeignKey(
        name: "FK_Activity_Project_ProjectId",
        column: x => x.ProjectId,
        principalTable: "Project",
        principalColumn: "Project_Id",
        onDelete: ReferentialAction.Cascade);
    });

    migrationBuilder.CreateTable(
        name: "Expense",
        columns: table => new
        {
            Expense_Id = table.Column<int>(type: "int", nullable:
false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Expense_Name = table.Column<string>(type:
"nvarchar(max)", nullable: false),
            Month = table.Column<string>(type: "nvarchar(max)",
nullable: false),
            Year = table.Column<int>(type: "int", nullable: false),
            TotalMoney = table.Column<decimal>(type:
"decimal(18,2)", nullable: false),
            ExpenseState = table.Column<string>(type:
"nvarchar(max)", nullable: false),
            UserId = table.Column<int>(type: "int", nullable:
false),
            ProjectId = table.Column<int>(type: "int", nullable:
false),
            ExpenseState_Id = table.Column<int>(type: "int",
nullable: true),
            ExpenseType_Id = table.Column<int>(type: "int",
nullable: true)
        },
        constraints: table =>
        {

```

```

        table.PrimaryKey("PK_Expense", x => x.Expense_Id);
        table.ForeignKey(
            name: "FK_Expense_ArquiUsers_UserId",
            column: x => x.UserId,
            principalTable: "ArquiUsers",
            principalColumn: "User_Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Expense_Expense State_ExpenseState_Id",
            column: x => x.ExpenseState_Id,
            principalTable: "Expense State",
            principalColumn: "ExpenseState_Id");
        table.ForeignKey(
            name: "FK_Expense_Expense Type_ExpenseType_Id",
            column: x => x.ExpenseType_Id,
            principalTable: "Expense Type",
            principalColumn: "ExpenseType_Id");
        table.ForeignKey(
            name: "FK_Expense_Project_ProjectId",
            column: x => x.ProjectId,
            principalTable: "Project",
            principalColumn: "Project_Id",
            onDelete: ReferentialAction.Cascade);
    });

    migrationBuilder.CreateTable(
        name: "Team",
        columns: table => new
        {
            UserId = table.Column<int>(type: "int", nullable:
false),
            ProjectId = table.Column<int>(type: "int", nullable:
false),
            TeamName = table.Column<string>(type: "nvarchar(100)",
maxLength: 100, nullable: false),

```

```

        UserFunctionId = table.Column<int>(type: "int",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Team", x => new { x.ProjectId,
x.UserId });

        table.ForeignKey(
            name: "FK_Team_ArquiUsers_UserId",
            column: x => x.UserId,
            principalTable: "ArquiUsers",
            principalColumn: "User_Id",
            onDelete: ReferentialAction.Cascade);

        table.ForeignKey(
            name: "FK_Team_Project_ProjectId",
            column: x => x.ProjectId,
            principalTable: "Project",
            principalColumn: "Project_Id",
            onDelete: ReferentialAction.Cascade);

        table.ForeignKey(
            name: "FK_Team_User Function_UserFunctionId",
            column: x => x.UserFunctionId,
            principalTable: "User Function",
            principalColumn: "UserFunction_Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "Activity_File",
    columns: table => new
    {
        ActivityId = table.Column<int>(type: "int", nullable:
false),
        FileContentId = table.Column<int>(type: "int",
nullable: false)
    },

```



```

        constraints: table =>
        {
            table.PrimaryKey("PK_Activity_File", x => new {
x.ActivityId, x.FileContentId });

            table.ForeignKey(
                name: "FK_Activity_File_Activity_ActivityId",
                column: x => x.ActivityId,
                principalTable: "Activity",
                principalColumn: "Activity_Id",
                onDelete: ReferentialAction.Cascade);

            table.ForeignKey(
                name: "FK_Activity_File_File
Content_FileContentId",
                column: x => x.FileContentId,
                principalTable: "File Content",
                principalColumn: "FileContent_Id",
                onDelete: ReferentialAction.Cascade);

        });

migrationBuilder.CreateTable(
    name: "User_Activity",
    columns: table => new
    {
        UserId = table.Column<int>(type: "int", nullable:
false),
        ActivityId = table.Column<int>(type: "int", nullable:
false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_User_Activity", x => new {
x.UserId, x.ActivityId });

        table.ForeignKey(
            name: "FK_User_Activity_Activity_ActivityId",
            column: x => x.ActivityId,
            principalTable: "Activity",
            principalColumn: "Activity_Id",

```

```

        onDelete: ReferentialAction.Cascade);
    table.ForeignKey(
        name: "FK_User_Activity_ArquiUsers_UserId",
        column: x => x.UserId,
        principalTable: "ArquiUsers",
        principalColumn: "User_Id",
        onDelete: ReferentialAction.Cascade);
    });

    migrationBuilder.CreateTable(
        name: "Worklog",
        columns: table => new
        {
            Cod_Worklog = table.Column<int>(type: "int", nullable:
false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Date = table.Column<DateTime>(type: "datetime2",
nullable: false),
            Hours = table.Column<decimal>(type: "decimal(18,2)",
nullable: false),
            Comment = table.Column<string>(type: "nvarchar(max)",
nullable: false),
            UserId = table.Column<int>(type: "int", nullable:
false),
            ActivityId = table.Column<int>(type: "int", nullable:
false),
            BillingTypeId = table.Column<int>(type: "int",
nullable: false),
            WorklogStateId = table.Column<int>(type: "int",
nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Worklog", x => x.Cod_Worklog);
            table.ForeignKey(
                name: "FK_Worklog_Activity_ActivityId",
                column: x => x.ActivityId,
                principalTable: "Activity",

```

```

        principalColumn: "Activity_Id",
        onDelete: ReferentialAction.Cascade);

table.ForeignKey(
    name: "FK_Worklog_ArquiUsers_UserId",
    column: x => x.UserId,
    principalTable: "ArquiUsers",
    principalColumn: "User_Id",
    onDelete: ReferentialAction.Cascade);

table.ForeignKey(
    name: "FK_Worklog_Billing Type_BillingTypeId",
    column: x => x.BillingTypeId,
    principalTable: "Billing Type",
    principalColumn: "BillingType_Id",
    onDelete: ReferentialAction.Cascade);

table.ForeignKey(
    name: "FK_Worklog_Worklog State_WorklogStateId",
    column: x => x.WorklogStateId,
    principalTable: "Worklog State",
    principalColumn: "WorklogState_Id",
    onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(
    name: "Expense_File",
    columns: table => new
    {
        FileContentId = table.Column<int>(type: "int",
nullable: false),
        ExpenseId = table.Column<int>(type: "int", nullable:
false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Expense_File", x => new {
x.FileContentId, x.ExpenseId });
        table.ForeignKey(

```

```

        name: "FK_Expense_File_Expense_ExpenseId",
        column: x => x.ExpenseId,
        principalTable: "Expense",
        principalColumn: "Expense_Id",
        onDelete: ReferentialAction.Cascade);

    table.ForeignKey(
        name: "FK_Expense_File_File Content_FileContentId",
        column: x => x.FileContentId,
        principalTable: "File Content",
        principalColumn: "FileContent_Id",
        onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(
    name: "Line",
    columns: table => new
    {
        Cod_Line = table.Column<int>(type: "int", nullable:
false)
            .Annotation("SqlServer:Identity", "1, 1"),
        UnityPrice = table.Column<decimal>(type:
"decimal(18,2)", nullable: false),
        Date = table.Column<DateTime>(type: "datetime2",
nullable: false),
        Discription = table.Column<string>(type:
"nvarchar(max)", nullable: false),
        lineCity = table.Column<int>(type: "int", nullable:
false),
        lineType = table.Column<int>(type: "int", nullable:
false),
        ExpenseId = table.Column<int>(type: "int", nullable:
false),
        LineCityID = table.Column<int>(type: "int", nullable:
true),
        LineTypeID = table.Column<int>(type: "int", nullable:
true)

    },
    constraints: table =>

```

```

{
    table.PrimaryKey("PK_Line", x => x.Cod_Line);
    table.ForeignKey(
        name: "FK_Line_Expense_ExpenseId",
        column: x => x.ExpenseId,
        principalTable: "Expense",
        principalColumn: "Expense_Id",
        onDelete: ReferentialAction.Cascade);
    table.ForeignKey(
        name: "FK_Line_LineCity_LineCityID",
        column: x => x.LineCityID,
        principalTable: "LineCity",
        principalColumn: "LineCityID");
    table.ForeignKey(
        name: "FK_Line_LineType_LineTypeID",
        column: x => x.LineTypeID,
        principalTable: "LineType",
        principalColumn: "LineTypeID");
});

migrationBuilder.CreateIndex(
    name: "IX_Activity_ActivityStateId",
    table: "Activity",
    column: "ActivityStateId");

migrationBuilder.CreateIndex(
    name: "IX_Activity_ActivityTypeId",
    table: "Activity",
    column: "ActivityTypeId");

migrationBuilder.CreateIndex(
    name: "IX_Activity_ProjectId",
    table: "Activity",
    column: "ProjectId");

```

```
migrationBuilder.CreateIndex(  
    name: "IX_Activity_File_FileContentId",  
    table: "Activity_File",  
    column: "FileContentId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Expense_ExpenseState_Id",  
    table: "Expense",  
    column: "ExpenseState_Id");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Expense_ExpenseType_Id",  
    table: "Expense",  
    column: "ExpenseType_Id");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Expense_ProjectId",  
    table: "Expense",  
    column: "ProjectId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Expense_UserId",  
    table: "Expense",  
    column: "UserId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Expense_File_ExpenseId",  
    table: "Expense_File",  
    column: "ExpenseId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_File_Content_FileContentTypeId",  
    table: "File_Content",  
    column: "FileContentTypeId");
```

```
migrationBuilder.CreateIndex(  
    name: "IX_File_Content_FileId",  
    table: "File Content",  
    column: "FileId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Line_ExpenseId",  
    table: "Line",  
    column: "ExpenseId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Line_LineCityID",  
    table: "Line",  
    column: "LineCityID");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Line_LineTypeID",  
    table: "Line",  
    column: "LineTypeID");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Project_ClientId",  
    table: "Project",  
    column: "ClientId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Project_ProjectStateId",  
    table: "Project",  
    column: "ProjectStateId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Team_UserFunctionId",  
    table: "Team",  
    column: "UserFunctionId");
```

```
migrationBuilder.CreateIndex(  
    name: "IX_Team_UserId",  
    table: "Team",  
    column: "UserId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_User_Activity_ActivityId",  
    table: "User_Activity",  
    column: "ActivityId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Worklog_ActivityId",  
    table: "Worklog",  
    column: "ActivityId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Worklog_BillingTypeId",  
    table: "Worklog",  
    column: "BillingTypeId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Worklog_UserId",  
    table: "Worklog",  
    column: "UserId");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Worklog_WorklogStateId",  
    table: "Worklog",  
    column: "WorklogStateId");  
  
}
```

## Anexo BL: Power Network – btn\_Timesheet\_LogTime – OnSelect

```
Set(  
    LogTime;
```



```

1
);;
ClearCollect(
    UserProjects;
    CustomTimesheetConnector.GetProjectUser(UserId)
);;
Reset(cbb_popup_Project);;
Reset(cbb_popup_BillingType);;
Reset(dtp_popup_WorklogDate);;
Reset(txt_popup_HoursEdit);;
Reset(dtp_popup_EndDate);;

```

## Anexo BM: Power Network – btn\_popup\_Save – OnSelect

```

If(
    LogTime = 1;
    If(
        IsBlank(dtp_popup_WorklogDate.Value) = true;
        //tem de selecionar uma data para a worklog
        Notify(
            "Select a date!";
            Error
        );
        //tem uma data selecionada
        If(
            IsBlank(cbb_popup_Project.Selected) = true;
            //tem de selecionar um projeto
            Notify(
                "Select a project!";
                Error
            );
            //tem um projeto selecionado
            If(
                IsBlank(cbb_popup_Activity.Selected) = true;
                //tem de selecionar uma atividade

```

```

Notify(
    "Select an Activity!";
    Error
);
//tem uma atividade selecionada
If(
    tgl_popup_ByPeriod.Checked = false;
    //não é por periodo
    If(
        Value(txt_popup_HoursEdit.Text) <= 0;
        //não pode registrar uma worklog com menos de 0
        horas

        Notify(
            "The hours field needs to be bigger than 0!";
            Error
        );
        //tem mais de 0 horas
        If(
            Value(txt_popup_HoursEdit.Text) >= 24;
            //não pode registrar uma worklog com mais de 24
            horas

            Notify(
                "The hours field needs to be lower than
24!";

                Error
            );
            //tem mais de 0 horas e menos de 24
            If(
                IsBlank(cbb_popup_BillingType.Selected);
                //tem de ter um tipo de billing
                Notify(
                    "Billing type missing!";
                    Error
                );
                If(

```

```

true;

IsBlank(txt_popup_CommentEdit.Value) =

//tem de ter um comentário
Notify(
    "You need to do a comment!";
    Error
);
//criar worklog
If(
CustomTimesheetConnector.CreateWorklog(
    Text (
dtp_popup_WorklogDate.Value;
        "yyyy-mm-dd"
    );
Value(txt_popup_HoursEdit.Text);
    txt_popup_CommentEdit.Value;
cbb_popup_Activity.Selected.activityId;
cbb_popup_BillingType.Selected.Value;
    "To Approve";
    UserId
) = true;
Notify(
    "Worklog created!";
    Success
);;
ClearCollect(
    WeekWL;
CustomTimesheetConnector.GetUserWeekWorklog(
    Text (
dtp_Timesheet_SearchDate.Value;
        "yyyy-mm-dd"

```

```

);
    UserId
)
);;
Set(
    LogTime;
    0
);;
Set(
    EditWL;
    0
);
Notify(
    "Worklog not created!";
    Error
)
)
)
)
);
//é por periodo
If(
    DateDiff(
        dtp_popup_WorklogDate.Value;
        dtp_popup_EndDate.Value
    ) > 7;
    //as datas não têm menos de uma semana de diferença
    Notify(
        "The two dates need to be from the same week!";
        Error
    );
    //as datas têm menos de uma semana de diferença
    If(

```

```

        dtp_popup_WorklogDate.Value >
dtp_popup_EndDate.Value;

        //a data de inicio é maior que a data de fim
        Notify(
            "The end date needs to be bigger than the
start date!";

            Error
        );
        //a data de fim é maior que a de inicio
        If(
            Value(txt_popup_HoursEdit.Text) <= 0;
            //não pode registrar uma worklog com menos
de 0 horas

            Notify(
                "The hours field needs to be bigger
than 0!";

                Error
            );
            //tem mais de 0 horas
            If(
                Value(txt_popup_HoursEdit.Text) >= 24;
                //não pode registrar uma worklog com
mais de 24 horas

                Notify(
                    "The hours field needs to be lower
than 24!";

                    Error
                );
                //tem mais de 0 horas e menos de 24
                If(
                    IsBlank(cbb_popup_BillingType.Selected);

                    //tem de ter um tipo de billing
                    Notify(
                        "Billing type missing!";
                        Error
                    );
                );
            );
        );
    );

```

```

If(

IsBlank(txt_popup_CommentEdit.Value) = true;

//tem de ter um comentário
Notify(
    "You need to do a
comment!";

    Error
);
//criar worklog by period
If(

CustomTimesheetConnector.CreateWorklogByPeriod(

    Text(

dtp_popup_EndDate.Value;

        "yyyy-mm-dd"
    );

    Text(

dtp_popup_WorklogDate.Value;

        "yyyy-mm-dd"
    );

Value(txt_popup_HoursEdit.Text);

txt_popup_CommentEdit.Value;

cbb_popup_Activity.Selected.activityId;

cbb_popup_BillingType.Selected.Value;

    "To Approve";

    UserId
) = true;
Notify(
    "Worklog created by
period!";

    Success
);;

```

```

CustomTimesheetConnector.GetUserWeekWorklog(
    ClearCollect(
        WeekWL;
        Text(
            dtp_Timesheet_SearchDate.Value;
            "yyyy-mm-dd"
        );
        UserId
    )
);
Set(
    LogTime;
    0
);
Set(
    EditWL;
    0
);
Notify(
    "Worklog not created by
period!";
    Error
)
)
)
)
)
)
)
)
)
)
);

```

```

If(
    WLinfo.activityId = 0;
    //criar worklog
    If(
        Value(txt_popup_HoursEdit.Text) <= 0;
        //não pode registrar uma worklog com menos de 0 horas
        Notify(
            "The hours field needs to be bigger than 0!";
            Error
        );
        //tem mais de 0 horas
        If(
            Value(txt_popup_HoursEdit.Text) >= 24;
            //não pode registrar uma worklog com mais de 24 horas
            Notify(
                "The hours field needs to be lower than 24!";
                Error
            );
            If(
                IsBlank(cbb_popup_BillingType.Selected);
                //tem de ter um tipo de billing
                Notify(
                    "Billing type missing!";
                    Error
                );
                If(
                    IsBlank(txt_popup_CommentEdit.Value) = true;
                    //tem de ter um comentário
                    Notify(
                        "You need to do a comment!";
                        Error
                    );
                    //criar worklog
                    If(
                        CustomTimesheetConnector.CreateWorklog(

```



```

        Text (
            dtp_popup_DateEdit.Value;
            "yyyy-mm-dd"
        );
        Value(txt_popup_HoursEdit.Text);
        txt_popup_CommentEdit.Value;
        WLActivityId;
        cbb_popup_BillingType.Selected.Value;
        "To Approve";
        UserId
    ) = true;
    Notify(
        "Worklog created!";
        Success
    );;
    ClearCollect(
        WeekWL;

CustomTimesheetConnector.GetUserWeekWorklog(
        Text (
            dtp_Timesheet_SearchDate.Value;
            "yyyy-mm-dd"
        );
        UserId
    )
);;
    Set(
        LogTime;
        0
    );;
    Set(
        EditWL;
        0
    );
    Notify(

```

```

        "Worklog not created!";
        Error
    )
    )
    )
    )
    )
);
//atualizar worklog
If(
    Value(txt_popup_HoursEdit.Text) <= 0;
    //não pode registrar uma worklog com menos de 0 horas
    Notify(
        "The hours field needs to be bigger than 0!";
        Error
    );
    //tem mais de 0 horas
    If(
        Value(txt_popup_HoursEdit.Text) >= 24;
        //não pode registrar uma worklog com mais de 24 horas
        Notify(
            "The hours field needs to be lower than 24!";
            Error
        );
        If(
            IsBlank(cbb_popup_BillingType.Selected);
            //tem de ter um tipo de billing
            Notify(
                "Billing type missing!";
                Error
            );
            If(
                IsBlank(txt_popup_CommentEdit.Value) = true;
                //tem de ter um comentário
                Notify(

```

```

        "You need to do a comment!";
        Error
    );
    //atualiza worklog
    If(
        CustomTimesheetConnector.UpdateWorklog(
            WLinfo.worklogId;
            Value(txt_popup_HoursEdit.Text);
            txt_popup_CommentEdit.Value;
            cbb_popup_BillingType.Selected.Value;
            WLinfo.worklogState
        ) = true;
        Notify(
            "Worklog updated!";
            Success
        );
        ClearCollect(
            WeekWL;

CustomTimesheetConnector.GetUserWeekWorklog(
            Text(
                dtp_Timesheet_SearchDate.Value;
                "yyyy-mm-dd"
            );
            UserId
        )
    );
    Set(
        LogTime;
        0
    );
    Set(
        EditWL;
        0
    );

```

100

INETE – Instituto de Educação Técnica

100

```
0  
)
```

### **Anexo BO: Power Network – rct\_glry\_ActivityBackGround – OnSelect**

```
Set(  
    ActInfo;  
    1  
);;  
Set(  
    ActivityInfo;  
  
CustomTimesheetConnector.GetActivitiesInfo(ThisItem.activity.activityId)  
);;
```

### **Anexo BP: Power Network – btn\_popup\_ProjectName – OnSelect**

```
Set(  
    ProjectInfo;  
  
CustomTimesheetConnector.GetProjectInfo(ActivityInfo.projectInfo.projectId)  
);;  
Set(  
    ProjInfo;  
    1  
);;  
Set(  
    ActInfo;  
    0  
);;
```

### **Anexo BQ: Power Network – btn\_glry\_UserInfo – OnSelect**

```
Set(  
    UserInfo;  
  
CustomTimesheetConnector.GetUserInfo(ThisItem.userId)  
);;
```

```

Set (
    UInf;
    1
);;

```

## Anexo BR: Power Network – btn\_glry\_ProjectBackGround – OnSelect

```

ClearCollect (
    ActivitiesHours;
    CustomTimesheetConnector.GetActivityHours (
        UserId;
        ThisItem.projectsIdName.projectId
    )
);;
Set (
    AuxProjId;
    ThisItem.projectsIdName.projectId
);;
Set (
    ActHours;
    1
);;
Set (
    ProjHours;
    0
);;

```

## Anexo BS: Power Network – btn\_glry\_ActBackGround – OnSelect

```

ClearCollect (
    WorklogHours;
    CustomTimesheetConnector.GetUserActivityWorklogs (
        UserId;
        ThisItem.activityIdName.activityId
    )
)

```

```

);;
Set(
    AuxActId;
    ThisItem.activityIdName.activityId
);;
Set(
    ActHours;
    0
);;
Set(
    WlHours;
    1
);;

```

### Anexo BT: Power Network – btn\_glry\_WIBackGround – OnSelect

```

Set(
    PopWl;
    1
);;
Set(
    WorklogInfo;
    CustomTimesheetConnector.GetWorklog(ThisItem.worklogId)
);;

```

### Anexo BU: Power Network – btn\_popup\_Sv – OnSelect

```

//atualizar worklog
If(
    Value(txt_popup_Hrs.Text) <= 0;
    //não pode registrar uma worklog com menos de 0 horas
    Notify(
        "The hours field needs to be bigger than 0!";
        Error
    );
    //tem mais de 0 horas

```

```

If(
    Value(txt_popup_Hrs.Text) >= 24;
    //não pode registrar uma worklog com mais de 24 horas
    Notify(
        "The hours field needs to be lower than 24!";
        Error
    );
If(
    IsBlank(cbb_popup_BillingTp.Selected);
    //tem de ter um tipo de billing
    Notify(
        "Billing type missing!";
        Error
    );
If(
    IsBlank(txt_popup_Cmnt.Value) = true;
    //tem de ter um comentário
    Notify(
        "You need to do a comment!";
        Error
    );
    //atualiza worklog
    If(
        CustomTimesheetConnector.UpdateWorklog(
            WorklogInfo.worklogId;
            Value(txt_popup_Hrs.Text);
            txt_popup_Cmnt.Value;
            cbb_popup_BillingTp.Selected.Value;
            WorklogInfo.worklogState
        ) = true;
        Notify(
            "Worklog updated!";
            Success
        );
        ClearCollect(

```





```

CustomTimesheetConnector.AddDays(
    Text(dtp_Timesheet_SearchDate.Value; "yyyy-mm-dd");
    0
)
);;
Set(
    SearchDate;
    Date(
        DateAUX.year;
        DateAUX.month;
        DateAUX.day
    )
);;
ClearCollect(
    WeekWL;
    CustomTimesheetConnector.GetUserWeekWorklog(
        Text(SearchDate; "yyyy-mm-dd");
        UserId
    )
);;

```

## Anexo BW: Power Network – btn\_popup\_Dlt – OnSelect

```

If(
    CustomTimesheetConnector.DeleteWorklog(WorklogInfo.worklogId) = true;
    Notify(
        "Worklog deleted!";
        Success
    );
    Notify(
        "Worklog not deleted!";
        Error
    )
);;
ClearCollect(

```

```

        WorklogHours;
        CustomTimesheetConnector.GetUserActivityWorklogs (
            UserId;
            AuxActId
        )
    );
);
ClearCollect(
    ActivitiesHours;
    CustomTimesheetConnector.GetActivityHours (
        UserId;
        AuxProjId
    )
);
ClearCollect(
    ProjectHours;
    CustomTimesheetConnector.GetProjectHour (UserId)
);
Set (
    PopWl;
    0
);
);

```

## Anexo BX: Power Network – btn\_glry\_ActName – OnSelect

```

Set (
    ProjInfo;
    0
);
Set (
    ActInfo;
    1
);
Set (
    ActivityInfo;
    CustomTimesheetConnector.GetActivitiesInfo (ThisItem.activityId)
);

```

