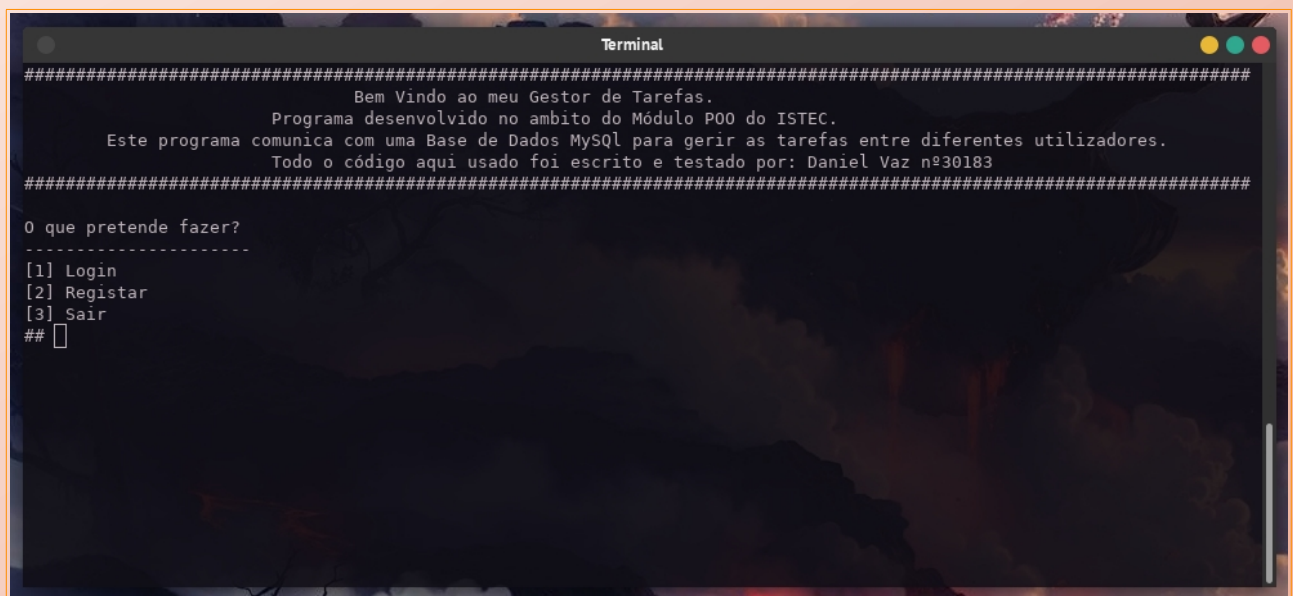


# Relatório

Aplicação para Gestão de Tarefas com recurso a *MySQL* e *PHP*.

Escrito em C#

A screenshot of a macOS Terminal window titled "Terminal". The window displays a text-based menu for a task management application. The text is as follows:

```
#####  
                Bem Vindo ao meu Gestor de Tarefas.  
                Programa desenvolvido no ambito do Módulo P00 do ISTECS.  
                Este programa comunica com uma Base de Dados MySQL para gerir as tarefas entre diferentes utilizadores.  
                Todo o código aqui usado foi escrito e testado por: Daniel Vaz nº30183  
#####  
  
0 que pretende fazer?  
-----  
[1] Login  
[2] Registrar  
[3] Sair  
## 
```

The terminal window has standard macOS window controls (red, yellow, green buttons) in the top right corner. The background of the terminal is dark with a subtle, abstract pattern.

# Introdução

Este trabalho surge a pedido do Professor Sandro Ferreira no módulo de Programação Orientada a Objetos, do *Instituto Superior de Tecnologias Avançadas*.

Com este trabalho prático, procurava-se que fosse documentado e criado um programa escrito em C# capaz de gerir as tarefas de diferentes utilizadores , sendo que toda a informação tanto das tarefas como dos utilizadores teria de estar armazenada numa Base de Dados [\*MySQL\*](#). De modo a este programa ser aprovado, teria de cumprir os seguintes requisitos:

- Permitir a Autenticação e Registo de utilizadores;
- Criação, edição e eliminação de tarefas.

Tendo estas linha orientadoras, neste mesmo relatório poderá ser verificado todo o meu processo em torno do planeamento e criação desta mesma solução. Note-se que decidi desde partida construir a mesma através da FrameWork “*Console Apps*” , ou seja, será um programa sem qualquer interface gráfico, sendo que todo o seu manuseamento será efetuado com recurso a um terminal.

De destacar também que esta aplicação foi construída num sistema de base *Windows10*, onde todo o código foi escrito no [\*Visual Studio Community\*](#), e o programa foi compilado e executado com ajuda dos utilitários provenientes do mesmo. Para servir a minha Base de Dados foi usada a solução grátis da [\*000WebHost\*](#), onde todas as comunicações para esta BD foram efetuadas através de Scripts escritos em *PHP*.

## Source Code

Todo o código existente neste projeto foi desenhado, construído e testado por mim, tomando recurso da documentação oficial tanto da linguagem C# como de todas as bibliotecas usadas.

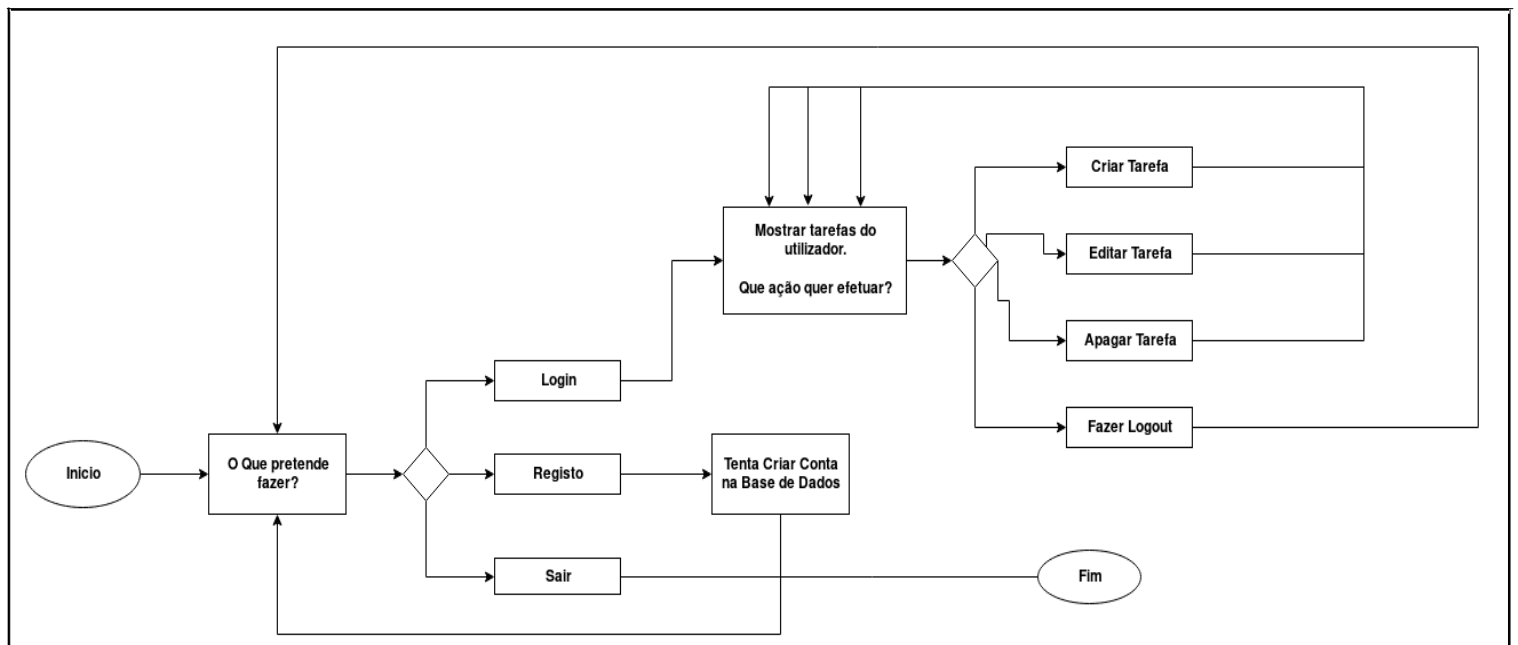
<a href="#"><u>Newtonsoft.Json.NET</u></a>		<a href="#"><u>System.Net.Http</u></a>
<a href="#"><u>MySQL Improved Extension</u></a>		<a href="#"><u>MySQL Syntax</u></a>

Todo o código fonte deste projeto poderá ser consultado através do mesmo *.zip* onde este *.pdf* se encontrava, no ficheiro de nome “ ***Program.cs*** ”, contudo e por questões de redundância também poderá ser consultado através do seguinte [link](#).

# Planeamento e Estrutura Lógica do Programa

Antes de sequer começarmos a desenvolver/explicar qualquer código, devemos perceber e delinear como será o funcionamento geral do programa. Para isso, podemos visualmente desenhar o mesmo num diagrama de ações, de forma a percebermos como é que devemos passar estas, para uma implementação em C# .

Portanto, de uma forma global e para cumprirmos os requisitos impostos inicialmente por este exercício, podemos assentar o desenvolvimento numa esquematização deste género:



Analisando o diagrama acima, percebemos que o programa irá funcionar com 2 'Loops' muito básicos, que irão na sua simplicidade prender o utilizador em 2 Menus, o de escolha inicial de Login\Registo e o Menu pós-Login que mostra ao utilizador as suas tarefas e as ações que pode exercer nas mesmas.

Contudo isto será apenas a estrutura do programa que irá interagir diretamente com o utilizador, sendo que para esta estrutura funcionar será ainda necessário construirmos a nossa Base de Dados. Neste exemplo em questão, a nossa *BD*, está armazenada e a ser servida através do fornecedor [000WebHost](http://000WebHost.net).

Nesta plataforma podemos criar Bases de Dados [MySQL](#) através do [PHPMyAdmin](#), sendo que apenas podemos comunicar remotamente com a nossa Base de Dados, através de scripts *PHP* que estejam armazenados no nosso servidor.

Na sua essência estes Scripts apenas comunicam com a nossa BD e tentam executar SQL Querys, sendo que retornam algum tipo de identificador pendente o sucesso ou falha da execução dos mesmos.

Dado que o foco desta documentação não assenta na estrutura dos scripts de *PHP*, não será exaustivamente explicado a estrutura destes, contudo todo o código fonte pode ser consultado [aqui](#).

Finalmente e para o leitor poder perceber melhor como é que os dados estão armazenados na nossa *BD*, deixo aqui o [link](#) onde pode ser consultado o Query de SQL que criará as tabelas necessárias a armazenar os nossos dados.

# Análise do Código

Nesta secção deste relatório vou abordar diferentes segmentos de código que tornam este programa funcional. Sendo que serão abordadas as diferentes secções de forma individual permitindo assim uma gradual interpretação do código em torno deste programa.

## Classes que contêm Métodos:

Existem neste programa 2 classes que se destacam, sendo estas a **MetodosBD** e a **MetodosAjuda**. Estas tal como os nomes referenciam armazenam os diferentes métodos que serão usados ao longo da execução do programa, sendo que na primeira temos os Métodos que comunicam diretamente com a nossa BD e a segunda são apenas todos os outros métodos que vão sendo necessários, ou para garantir coesão, corrigir problemas, ou execução de ainda outros métodos de formas diferentes.

### Classe MetodosBD

Nesta Classe vamos encontrar 7 métodos bastante idênticos entre eles: *Registar()*, *Login()*, *UserID()*, *Tarefas()*, *criarTarefa()*, *apagarTarefa()*, *editarTarefa()*.

E devido as suas semelhanças, aqui será apenas explicado a lógica entorno de um destes métodos, sendo que nos restantes apenas estamos a mudar algumas variáveis que enviamos via *PHP*, e os ficheiros que estamos a chamar.

Abaixo deixo o código do método *Registar()*, sendo a sua estrutura vai ser praticamente replicado nos restantes acima indicados.

```
public async Task<string> Registar(string username, string password)
{
    const string Url =
"http://todoaplicacao.000webhostapp.com/phpScripts/registerUser.php";
    HttpClient httpClient = new HttpClient();
```

```

var values = new List<KeyValuePair<string, string>>{
    new KeyValuePair<string, string>("username", username),
    new KeyValuePair<string, string>("password", password)
};

var content = new FormUrlEncodedContent(values);
var response = httpClient.PostAsync(Url, content).Result;

if (response.IsSuccessStatusCode)
{
    var responseContent = await
response.Content.ReadAsStringAsync();
    return responseContent;
}
else { return "Erro a Estabelecer Comunicação com o Servidor."; }
}

```

A essência deste e dos restantes métodos deste género passa pela seguinte lógica:

- Receber dados nos seus parâmetros;
- Formatar os mesmos de forma a estes serem aceites no passo seguinte;
- Comunicar com a BD chamando o respetivo ficheiro *.php*, onde serão passados os dados anteriores;
- Pendente a execução ou não do ficheiro *.php* é retornado uma mensagem/digito identificador do seu sucesso ou falha.

## Classe MetodosAjuda

Nesta Classe vamos encontrar um total de 10 Métodos diferentes, sendo que estes podem ser separados em 2 grupos: Os métodos que são “**utilitários**”, ou seja os métodos que vão ser recorrente-mente chamados por questões práticas de modo a auxiliarem a execução do programa sem erros; e os métodos de “**execução**” sendo estes apenas responsáveis por executarem os outros diferentes métodos que comunicam com a BD.

Estes últimos são essenciais dado que os métodos que comunicam com a BD são todos eles *Tasks* assíncronas que têm pouco ou nenhum sistema de reconhecimento de erro. Deste modo nos métodos “**ExecutarX**” podemos trabalhar e manipular alguns dos dados que são passados para os métodos

superiores, garantindo assim mais um nível de manipulação, tornando o programa mais flexível e consistente.

Vamos começar por analisar os métodos utilitários. Estes são compostos por os seguintes 3: *Banner()*, *sha256\_hash()*, *NumCheck()*.

O Método *Banner()*, é responsável por limpar o ecrã e colocar alguma informação sobre o mesmo sempre no topo da tela. Este método apenas existe por questões estéticas.

O Método *sha256\_hash()*, recebe um string e retorna o *Hash* do mesmo. Este método é chamado sempre que existe um manuseamento das passwords dos utilizadores, sendo que estas estão armazenadas no formato *SHA256* na nossa Base de Dados e deste modo sempre que um user faz login ou um registo necessitamos de converter a password que este introduziu para o respetivo *Hash*, de modo a compararmos com os valores na nossa *BD*.

```
public String sha256_hash(String value)
{
    StringBuilder Sb = new StringBuilder();

    using (SHA256 hash = SHA256Managed.Create())
    {
        Encoding enc = Encoding.UTF8;
        Byte[] result = hash.ComputeHash(enc.GetBytes(value));

        foreach (Byte b in result)
            Sb.Append(b.ToString("x2"));
    }
    return Sb.ToString();
}
```

O Método *NumCheck()*, está preparado para receber uma string e tentar converter a mesma para um numero inteiro, sendo que caso seja possível o mesmo irá retornar “true” , caso oposto “false” .

```
public bool NumCheck(string str) {
    return int.TryParse(str, out int num);
}
```



Os restantes 7 métodos nesta classe são os já referidos “*ExecutarX*”, sendo que estes têm uma relação de 1 para 1 com os métodos existentes na classe **MetodosBD**.

De forma geral todos estes métodos agem segundo a mesma lógica:

- Solicitar dados ao utilizador;
- Trabalhar os dados introduzidos e enviar os mesmo para os métodos superiores de comunicação com a BD;
- Receber a resposta dos métodos superiores, e ou retornam algum valor\mensagem, ou trabalham ainda os dados de forma a serem devidamente apresentados ao utilizador.

De forma a simplificar a leitura desta mesma documentação deixo a explicação de apenas 2 destes métodos, o *ExecutarRegistar()* e o *ExecutarTarefas()*, dado que os restantes aceitam uma lógica idêntica a estes.

O *ExecutarRegistar()* tem a seguinte sintaxe:

```
public async void ExecutarRegistar(string user, string pass)
{
    MetodosBD MBD = new MetodosBD();
    var tarefa = MBD.Registar(user, pass);
    string resultado = await tarefa;
    if (Int32.Parse(resultado) == 1)
    {
        Console.WriteLine("Os Dados Introduzidos já existem em
sistema.");
    }
    else if (Int32.Parse(resultado) == 0)
    {
        Console.WriteLine("O Utilizador foi criado com Sucesso.");
    }
    else {
        Console.WriteLine("Ocorreu um erro aquando criação da conta.
Tente Novamente mais tarde.");
    }
}
```

Ou seja, podemos ver que este método vai receber dois parâmetros, e vai enviar os mesmos para o método superior *Registar()*, sendo que pendente o resultado devolvido pelo mesmo, este método irá escrever no ecrã se ocorreu algum problema ou não.

O método *ExecutarTarefas()*, tem a seguinte estrutura:

```
public async void ExecutarTarefas(string userID)
{
    MetodosBD MBD = new MetodosBD();
    var tarefa = MBD.Tarefas(userID);
    var resultado = await tarefa;
    List<Modelo_Tarefas> resultado_Traduzido =
JsonConvert.DeserializeObject<List<Modelo_Tarefas>>(resultado);

Console.WriteLine("=====
");
    foreach (var i in resultado_Traduzido) {
        Console.WriteLine("\t\t\t\t\t[{0}]", i.id);
        Console.WriteLine("Titulo: {0}", i.titulo);
        Console.WriteLine("Data: {0}", i.data);
        Console.WriteLine("Descrição: {0}", i.descricao);

Console.WriteLine("=====
");
    }
}
```

Neste método já agimos um pouco de forma diferente, ou seja, *recebemos a mesma um valor como parâmetro*, contudo desta vez não estamos apenas a tentar registar um utilizador de forma a sabermos se foi bem sucedido ou não.

Aqui estamos à mesma a *comunicar com o respetivo método superior Tarefas()*, contudo este método irá nos retornar (graças ao script de *php* que está associado a esta ação), um Json com as diferentes Tarefas que estão associadas ao user que esteja logado naquele instante. Para isso necessitamos de associar estes dados obtidos a um Objeto Modelo, sendo que este foi logo definido no início do programa, apresentado a seguinte estrutura:

```
public class Modelo_Tarefas {
    public int id { get; set; }
    public int dono { get; set; }
    public string titulo { get; set; }
    public string data { get; set; }
    public string descricao { get; set; }
}
```

NOTA: Com os outros métodos sempre que era necessário de proceder de forma idêntica, era semelhante-mente criado uma Classe Modelo, para os valores obtidos pelo respetivo Json.

É com base neste modelo que criamos uma Lista que vai armazenar todos os resultados.

Finalmente, para apresentar estas mesmas tarefas ao utilizador, é efetuado um *foreach* em todas as linhas devolvidas de forma a ficar mais apresentável os dados obtidos.

## Main:

Aqui encontramos o “corpo” principal deste programa.

É aqui que são chamados todos os outros métodos e é criada a lógica e a estrutura base do funcionamento desta solução. Em relação a este programa, não será colocado o código do método **Main** completo, dado que o mesmo ainda tem um tamanho considerável.

Contudo podemos explicar o seu funcionamento lógico, recorrendo a alguns excertos :

```
static void Main(string[] args)
{
    MetodosAjuda MA = new MetodosAjuda();
    MetodosBD MBD = new MetodosBD();

    bool loopPrimario = true;
    bool loopLogin = false;
    while (loopPrimario)
    {
        MA.Banner();
        Console.WriteLine("O que pretende fazer?");
        Console.WriteLine("-----");
        Console.WriteLine("[1] Login");
        Console.WriteLine("[2] Registar");
        Console.WriteLine("[3] Sair");
        Console.Write("## ");
    }
    (...)
}
```

O método começa por **instanciar as classes referidas acima**, e **criar algumas variáveis** necessárias para a criação de **While** Loops que irão prender o utilizador em determinados menus de escolha de ações.

Pode ser analisado que o **primeiro Menu** que o user ficar retido até executar ações futuras, é o titulado *MenuInicial*, sendo que aqui o utilizador escolhe se quer fazer um Login/Registo ou sair da aplicação.

Pendente a escolha do utilizador é efetuado um **Switch/Case** que irá ter as diferentes opções, sendo que apenas caso o utilizador decida efetuar Login e seja

bem sucedido a o fazer, é que é iniciado o Segundo **While** Loop que vai desta vez prender o utilizador no **menu de escolha de ações sobre as tarefas** que este possui.

```
(...)
switch (escolhaMenuInicialINT)
{
    case 1:
        MA.Banner();
        Console.WriteLine("Introduza os seus dados de acesso:");
        Console.WriteLine("-----");
        Console.Write("Username: ");
        string user = Console.ReadLine();
        Console.Write("Password: ");
        string pass = MA.sha256_hash(Console.ReadLine());
        Task<string> resposta = Task.Run(async () => await
MA.ExecutarLogin(user, pass));
        resposta.Wait();

        if (resposta.Result == "true") {
            loopLogin = true;

            (...)

            while (loopLogin)
            {
                MA.Banner();

Console.WriteLine("=====
=");

                Console.WriteLine("|\\t\\t\\tAs tuas tarefas\\t\\t\\t
|");

                Console.WriteLine("|\\t\\t\\t\\t\\t\\t\\t\\t\\t      |");
                MA.ExecutarTarefas(userID);
                Console.WriteLine("");
                Console.WriteLine("Que ações é que pretende
executar?");

Console.WriteLine("-----");

                Console.WriteLine("[1]- Criar Tarefa;");
                Console.WriteLine("[2]- Editar Tarefa;");
                Console.WriteLine("[3]- Apagar Tarefa;");
                Console.WriteLine("[4]- Fazer LogOff;");
                Console.Write("## ");

                (...)
            }
        }
    }
}
```

Fora esta estrutura, o código existente no **Main** é de fácil compreensão, sendo que ao analisarmos o mesmo vemos que apenas funciona pendente a escolha que o utilizador faça às perguntas colocadas, sendo de seguida chamados os diferentes métodos que foram definidos previamente e criados especificamente para as respetivas situações.

# Melhorias Futuras e Aspetos a considerar

Apesar de completo este programa tem ainda algumas falhas que em iterações futuras deverão ser consideradas e se possível corrigidas.

Algumas destas falhas são:

- **Controlo de Erros;**
  - Apesar de existirem alguns métodos como o **NumCheck()**, e apesar de os scripts de *php* estarem devidamente preparados para reportar eventuais erros, existem ainda aspetos na execução deste programa que fazem com que o mesmo possa falhar ou deparar-se com Bugs que o obriguem a terminar. Será necessário adicionar um sistema mais detalhado de controlo de Erros, estando associado a eventuais problemas ou de conexão ou de falhas em queries SQL. No mínimo, seria necessário um melhor método para reportar mensagens mais específicas para DeBugging.
- **Funcionalidades mais detalhadas;**
  - Mesmo cumprindo os requisitos impostos inicialmente, poderiam ser adicionadas novos métodos de manipulação de tarefas, quer seja de forma individual ou massivamente;
  - De notar que outro aspeto ainda em falta é a capacidade de gerir\controlar não só as datas das Tarefas como as horas a que estas estão associadas, permitindo assim uma maior personalização das tarefas na nossa aplicação.
- **Melhorias à Base de Dados;**
  - Apesar de funcional, a Base de Dados utilizada podia estar mais composta de forma a armazenar mais dados não só dos utilizadores como das tarefas, deste modo podendo aumentar a personalização do conteúdo a ser apresentado na aplicação.

## Conclusão

Mesmo tendo alguns aspetos necessários a rever, nesta solução podemos ver uma sólida estrutura em torno dos requisitos impostos inicialmente. Não só cumpre os mesmo eficientemente, como também está preparada para corrigir eventuais dificuldades de modo a que o programa não esteja sempre a terminar, por erros encontrados ou na Base de Dados, ou na comunicação com a mesma.

Espera-se que associado com esta documentação se tenha uma fácil abordagem ao código fonte do programa, permitindo assim a sua total interpretação e análise.

Obrigado pela sua leitura.