

Licenciatura em Engenharia Informática  
Laboratório de Planeamento e Desenvolvimento de  
Software

David Patrício Luna

Luís Filipe Leite Barbosa

**Autores**

Pedro Melo — 74160

Vítor Gonçalves — 78315

Tiago Costa — 78501

Tiago Silva — 78417

Vila Real, 2024

# Índice

- Introdução
- Mockups
- Modelos de Dados
- Bibliografia

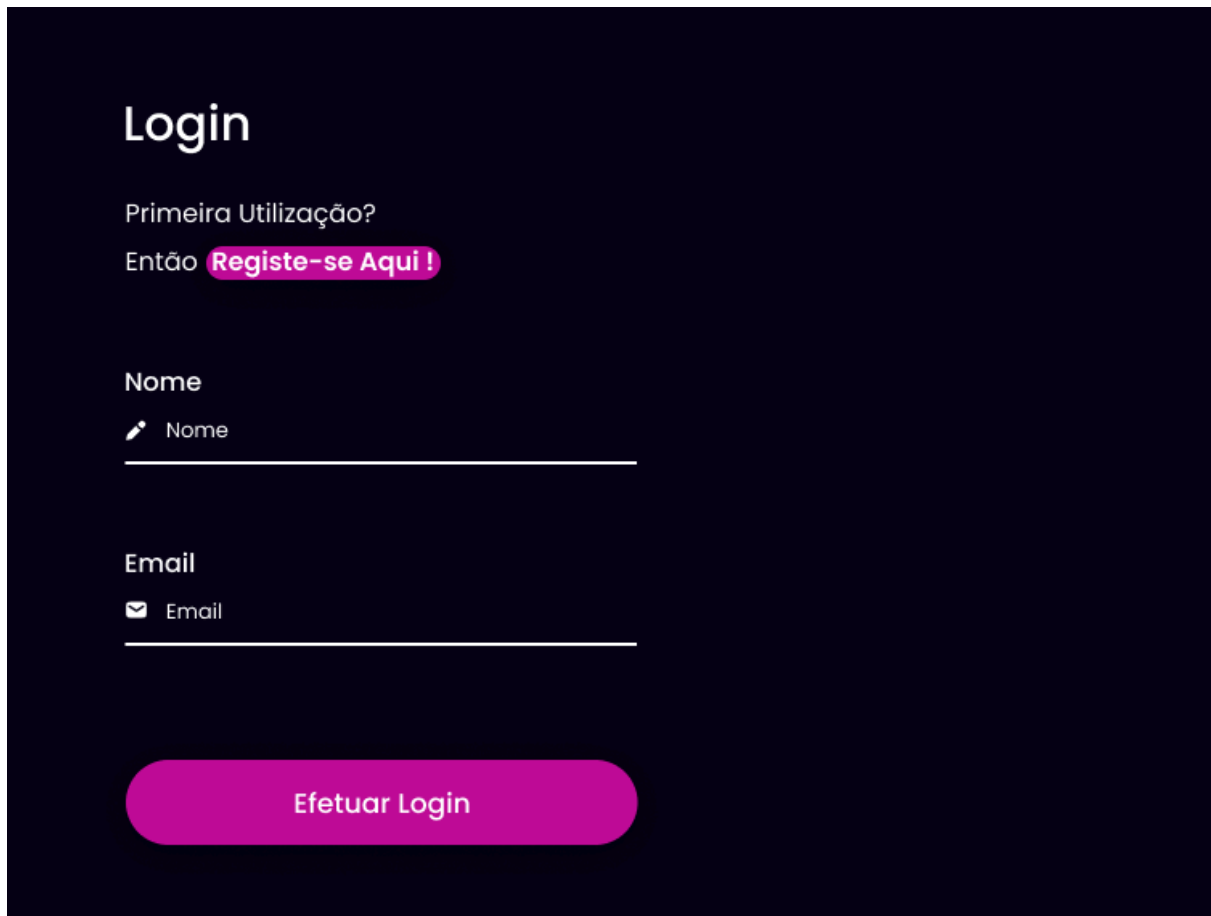
# Introdução

O trabalho proposto é uma aplicação de “*To-do List*”, ou seja, uma aplicação que centralize e organize todos os projetos/tarefas a fazer para organizar e tornar mais eficiente a sua realização, esta aplicação possui também alertas que ajudam o utilizador a lembrar-se destas mesmas tarefas.

Nesta fase do projeto criámos as mockups das views com recurso ao *figma.com*, tendo também implementado o modelo de dados em *C#* com a framework *.NET 8.0* no *Visual Studio*.

Ambos foram desenvolvidos com base no Diagrama de Casos de uso e Diagrama de Classes que nos foi fornecido.

# Mockups



The mockup shows a login interface on a dark blue background. At the top, the word "Login" is written in white. Below it, the text "Primeira Utilização?" is followed by "Então" and a pink button labeled "Registe-se Aqui!". There are two input fields: one for "Nome" with a pink icon and label, and one for "Email" with a pink icon and label. At the bottom, there is a large pink button labeled "Efetuar Login".


Ecrã de Login

A página de login é a primeira view do utilizador na aplicação e é nela que ele deve autenticar-se, inserindo as suas credenciais, neste caso, o e-mail e o nome, assim como uma fotografia, sendo esta opcional.


Após a autenticação, o utilizador é direcionado para a sua dashboard.


# Registo

Nome

 Nome

Email

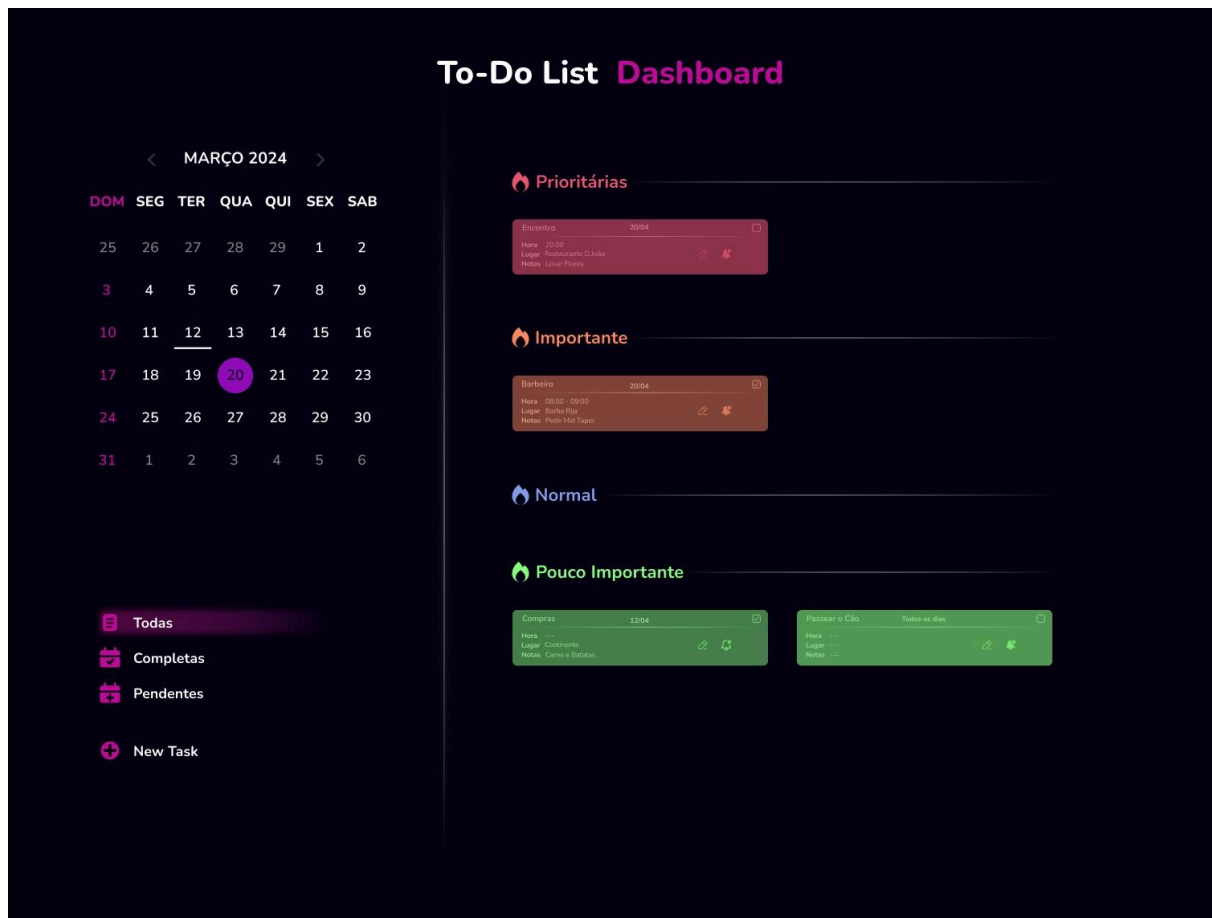
 Email

  
Inserir Imagem

Registar

Criar Conta

Este ecrã é apresentado caso seja a primeira vez do utilizador na aplicação e ele selecione a opção de criar conta no ecrã de login, depois dos campos serem preenchidos, a sua conta é criada e este é redirecionado para a sua dashboard.



Dashboard

A Dashboard é a página principal que permite ao utilizador ver as tarefas, podendo seleccionar um dia em específico para consultar as tarefas.

Esta é a view geral que permite adicionar, editar e remover tarefas, assim como consultá-las.

As tarefas são apresentadas com data de início e de fim, estado e descrição.

É também possível filtrar as tarefas por estado de realização (Pendentes/Completas/Todas) e desativar os seus alertas.

Foi criada uma hierarquia de níveis que vai de Pouco Importante até Prioritária, as tarefas com maior prioridade estão equipadas com alertas antes e depois da hora de realização da tarefa.

## Nova Tarefa

MARÇO 2024

DOM	SEG	TER	QUA	QUI	SEX	SAB
25	26	27	28	29	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Título

Inserir Título

Hora

Hora : Minutos / Hora : Minutos

Lugar

Inserir Lugar

Estado

☐ Por Iniciar

☐ Em execução

☐ Por Terminar

Notas

Inserir Notas

Periodicidade

☐ Diário

☐ Semanal

☐ Mensal

Alerta de Antecipação

☐ 15 minutos

☐ 30 minutos

☐ 60 minutos

Alerta de Não-Realização

☐ 15 minutos

☐ 30 minutos

☐ 60 minutos

Confirmar

### Criar Tarefa

Este ecrã é apresentado quando o utilizador pretende adicionar uma tarefa nova.

Após o preenchimento dos campos a tarefa é adicionada e o utilizador volta para a dashboard.

## Editar Tarefa

MARÇO 2024

DOM	SEG	TER	QUA	QUI	SEX	SAB
25	26	27	28	29	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Título

Encontro

Hora

20 : 00 / Hora : Minutos

Lugar

Restaurante D. João

Estado

☐ Por Iniciar  
☐ Em execução  
☐ Por Terminar

Notas

Levar Flores

Periodicidade

☐ Diário  
☐ Semanal  
☐ Mensal

Alerta de Antecipação

☐ 15 minutos  
☐ 30 minutos  
☐ 60 minutos

Alerta de Não-Realização

☐ 15 minutos  
☐ 30 minutos  
☐ 60 minutos

Confirmar


### Editar Tarefa

Semelhante ao ecrã de criação de tarefas, mas neste os campos estão preenchidos com as informações da tarefa a alterar, após a sua modificação, o utilizador é redirecionado para a dashboard.




## Perfil

Nome

 Jacinto Mateus


---

Email

 jotinham@gmail.com

---

Salvar



Editar Perfil

Esta view aparece caso o utilizador queira fazer alguma alteração ao seu perfil.

Caso alguma alteração seja efetuada as suas informações são modificadas no respetivo ficheiro e o utilizador volta para a dashboard.

# Modelos de Dados

```
1 namespace UTAD.ToDoList.WPF.Models.Shared
2 {
3     public abstract class BaseModel
4     {
5         public string id { get; set; }
6
7         protected BaseModel()
8         {
9             if(string.IsNullOrEmpty(id))
10                 id = Guid.NewGuid().ToString();
11         }
12     }
13 }
14
```

BaseModel.cs

Classe base de todos os modelos com identificadores (IDs), facilita a implementação de modelos futuros através da relação de herança.

```

1  using UTAD.ToDoList.WPF.Models.Shared;
2
3  namespace UTAD.ToDoList.WPF.Models
4  {
5      public class Tarefa : BaseModel
6      {
7          public string titulo { get; set; }
8          public DateOnly dataInicio { get; set; }
9          public DateOnly dataTermino { get; set; }
10
11          // nível de importância (0 - Sem Nível | 1 - Pouco Importante | 2 - Normal | 3 - Importante | 4 - Prioritária)
12          public int nivelImportancia { get; set; }
13          public bool estado { get; set; }
14
15          // atributos opcionais
16          public string? descricao { get; set; }
17          public Periodicidade? periodicidade { get; set; }
18          public Alerta? alertaAntecipacao { get; set; }
19          public Alerta? alertaExecucao { get; set; }
20
21          // construtor por defeito
22          public Tarefa()
23          {
24              titulo = string.Empty;
25              descricao = string.Empty;
26              dataInicio = DateOnly.MinValue;
27              dataTermino = DateOnly.MaxValue;
28              nivelImportancia = 0;
29              estado = false;
30              periodicidade = new Periodicidade();
31              alertaAntecipacao = new Alerta();
32              alertaExecucao = new Alerta();
33          }
34
35          // construtor com parâmetros
36          public Tarefa(string _titulo, string _descricao, DateOnly _dataInicio, DateOnly _dataTermino, int _nivelImportancia, bool _estado,
37              Periodicidade _periodicidade, Alerta _alertaAntecipacao, Alerta _alertaExecucao)
38          {
39              titulo = _titulo;
40              descricao = _descricao;
41              dataInicio = _dataInicio;
42              dataTermino = _dataTermino;
43              nivelImportancia = _nivelImportancia;
44              estado = _estado;
45              periodicidade = _periodicidade;
46              alertaAntecipacao = _alertaAntecipacao;
47              alertaExecucao = _alertaExecucao;
48          }
49      }
50  }

```

## Tarefa.cs

Classe com o modelo de dados que representa uma tarefa na aplicação, possui uma relação de herança com a classe BaseModel uma vez que cada tarefa tem um identificador único.

Tem atributos obrigatórios como o título, as datas, o estado e o nível de importância, assim como atributos opcionais como a descrição, os alertas e a periodicidade, sendo estes últimos dois objetos das suas próprias classes.

Foram também implementados construtores por defeito e com parâmetros.

```

1  using UTAD.ToDoList.WPF.Models.Shared;
2
3  namespace UTAD.ToDoList.WPF.Models
4  {
5      public class Periodicidade : BaseModel
6      {
7          public string tipo { get; set; }
8          public IList<String> diasSemana { get; set; }
9
10         // construtor por defeito
11         public Periodicidade()
12         {
13             tipo = string.Empty;
14             diasSemana = new List<String>();
15         }
16
17         // construtor com parâmetros
18         public Periodicidade(string _tipo, IList<string> _diasSemana)
19         {
20             tipo = _tipo;
21             diasSemana = _diasSemana;
22         }
23     }
24 }
25

```

Periodicidade.cs

Classe com o modelo de dados que representa uma tarefa na aplicação, possui uma relação de herança com a classe BaseModel uma vez que cada periodicidade tem um identificador único.

Possui um tipo (string) que serve para identificar quando a tarefa se deverá repetir (semanalmente, mensalmente, anualmente) e uma lista de dias a repetir.

Foram também implementados construtores por defeito e com parâmetros.

```

1  using UTAD.ToDoList.WPF.Models.Shared;
2
3  namespace UTAD.ToDoList.WPF.Models
4  {
5      public class Alerta: BaseModel
6      {
7          public string mensagem { get; set; }
8          public DateTime dataHora { get; set; }
9
10         // tipo de alerta (1 - Alerta Windows | 2 - Email)
11         public int tipo { get; set; }
12         public bool estado { get; set; }
13
14         // construtor por defeito
15         public Alerta()
16         {
17             mensagem = string.Empty;
18             dataHora = DateTime.MinValue;
19             tipo = 1;
20             estado = false;
21         }
22         // construtor com parâmetros
23         public Alerta(string _mensagem, DateTime _dataHora, int _tipos, bool _estado)
24         {
25             mensagem = _mensagem;
26             dataHora = _dataHora;
27             tipo = _tipos;
28             estado = _estado;
29         }
30     }
31 }

```

Alerta.cs

Classe com o modelo de dados que representa uma tarefa na aplicação, possui uma relação de herança com a classe BaseModel uma vez que cada alerta tem um identificador único.

Cada alerta, seja este de antecipação ou de não realização, é composto por uma mensagem e uma data para apresentar o alerta, um tipo caso queiramos mandar um email ou apresentar um pop-up (default) e o estado para verificarmos se o alerta já foi apresentado.

Foram também implementados construtores por defeito e com parâmetros.

```

1  using System.Windows.Media;
2  using System.Windows.Media.Imaging;
3
4  namespace UTAD.ToDoList.WPF.Models
5  {
6      public class Perfil
7      {
8
9
10         public string nome { get; set; }
11         public string email { get; set; }
12
13         // fotografia do perfil (bitmap)
14         public BitmapImage fotografia { get; set; }
15
16
17         // construtor por defeito
18         public Perfil()
19         {
20             nome = string.Empty;
21             email = string.Empty;
22             fotografia = new BitmapImage();
23         }
24
25         // construtor com parâmetros
26         public Perfil(string _name, string _email, BitmapImage _fotografia)
27         {
28             nome = _name;
29             email = _email;
30             fotografia = _fotografia;
31         }
32     }
33 }

```

Perfil.cs

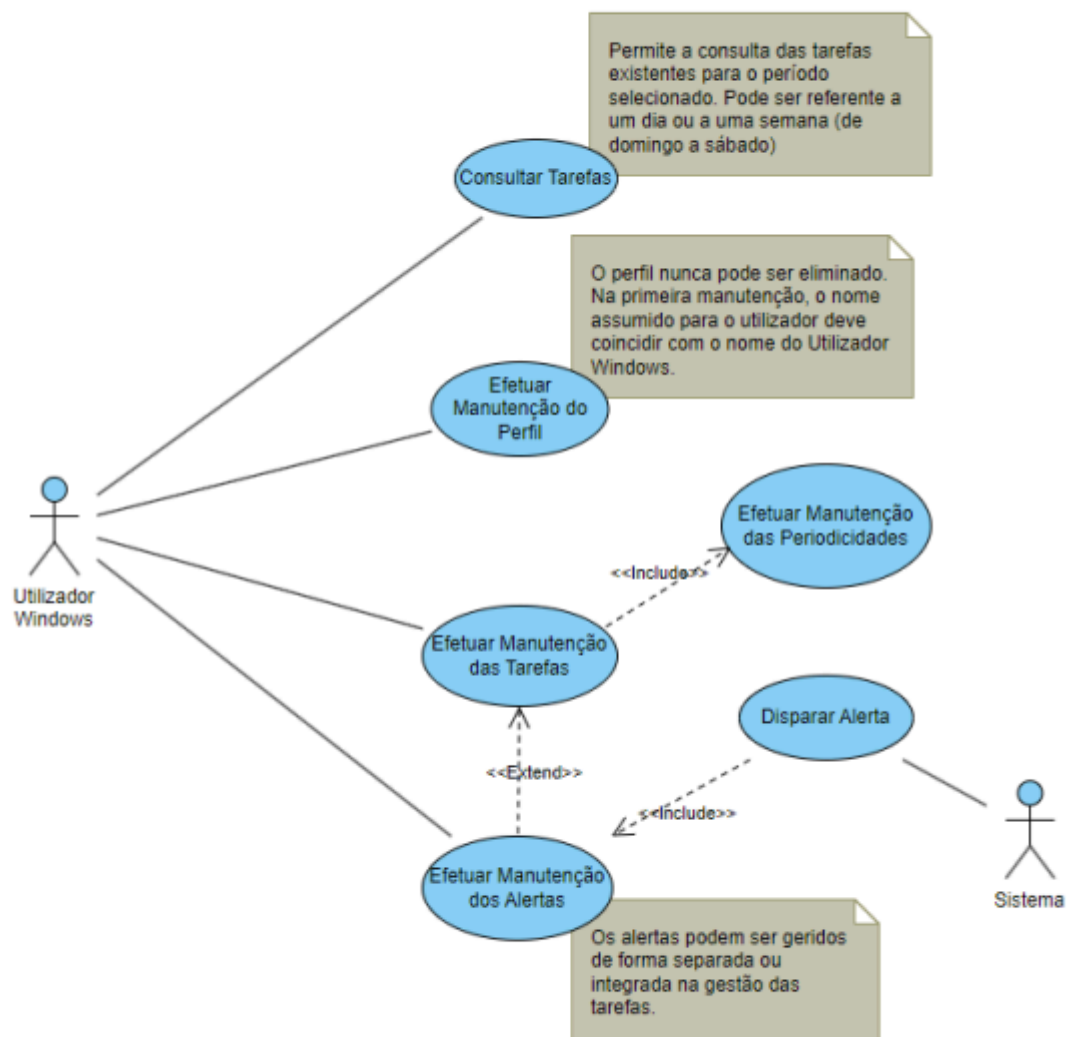
Modelo de dados que representa um utilizador, é composto pelo seu nome(string), email(string) e fotografia(Bitmap Image).

Único modelo que não tem herança com o BaseModel por não possuir um ID.

Foram implementados construtores por defeito e com parâmetros.

# Bibliografia

Anexo 1 – Diagrama de Casos de uso



## Anexo 2 – Diagrama de Classes

