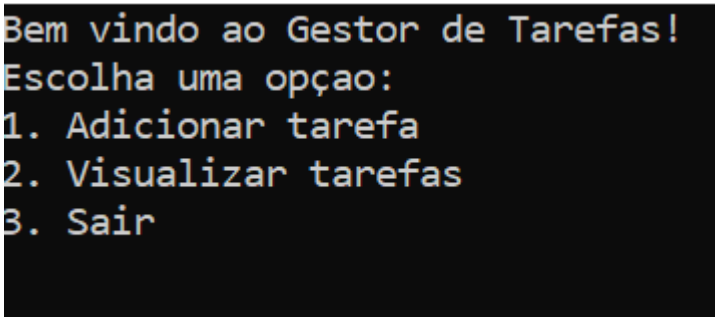
APLICAÇÃO DEMONSTRADORA

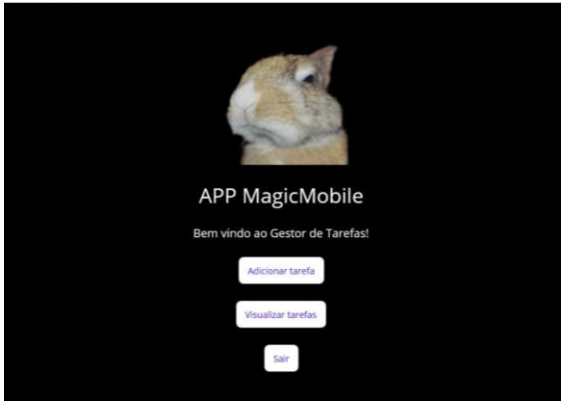
GESTOR DE TAREFAS

Equipa MagicMobile: Alix Paulino, Ana Guerreiro, André Alface, Joel Ginga e Marisa Bernardo

O que faz/permite fazer

+	Inserir tarefas	O utilizador pode inserir as suas tarefa a desenvolver
+	Visualizar tarefas	O utilizador pode consultar a lista de tarefas atual, em qualquer momento (antes ou depois da inserção de tarefas novas
+	Sair da aplicação	O utilizador também tem acesso a uma opção para sair da aplicação demonstradora, encerrando-a
	A aplicação efetua a interação com o utilizador através da consola	





Separação de Responsabilidade entre Componentes

CRONTROLLER

- Gere o fluxo de controlo da aplicação demonstradora
- Reage aos inputs do utilizador, que faz a sua escolha através de um menu de opções
- Faz a gestão de erros e reage às exceções
- Encerra a aplicação demonstradora
- Artefactos:
  - Controller.cs
  - Model.cs
  - Iview.cs

MODEL

- Iniciazia a lista de tarefas
- Adiciona as tarefas novas à tarefas à lista de tarefas
- Atualiza a lista de tarefas
- Lança uma exceção, caso a lista de tarefas esteja vazia
- Artefactos:
  - Model.cs
  - Iview.cs

VIEW

- Apresenta mensagens ao utilazor, através de um interface (Iview):
  - Exibir mensagem de boas vindas
  - Exibir lista de tarefas
  - Exibir mensagens de erro
- Artefactos:
  - View.cs
  - Iview.cs

Aspetos interessantes e desafiantes do uso de exceções face às características da API .NetMaui

- **Desacoplamento de componentes** - Através do uso de eventos delegados conseguimos promover o desacoplamento entre os diferentes componentes da aplicação. Desta forma temos o Controller a comunicar com a View e com o Model através de eventos delegados, facilitando também a manutenção do código.

- **Comunicação assíncrona** - Os eventos delegados permitem este tipo de comunicação, ou seja o Controller emite eventos para a classe View e Model, que depois podem lidar com esses eventos, o que se torna bastante útil quando a resposta de um componente demora muito tempo.

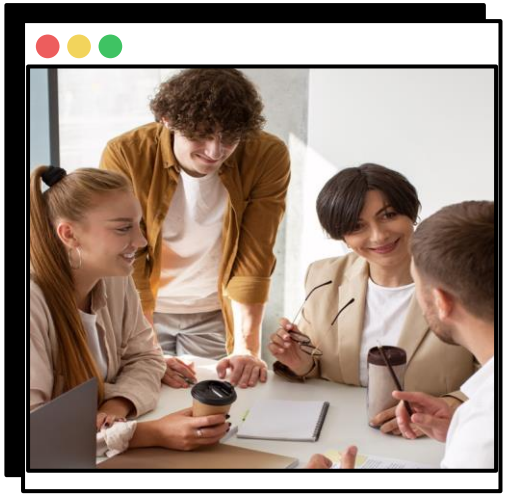
- **Tratamento de erros e exceções** - Aqui o uso de eventos delegados também mostra-se útil. No nosso caso, temos, por exemplo o Controller a lançar um evento de erro (através de uma mensagem de erro), sendo que a View é responsável por tratar esse evento ao exibir uma mensagem correta ao utilizador.

Aspetos interessantes e desafiantes do uso de exceções face às características da API .NetMaui

- **Tratamento de exceções de forma "personalizada"** - O código apresentado permite lançar exceções personalizadas, ou seja, conseguimos lidar com situações especificas, como por exemplo, uma lista de tarefas vazia. Desta forma, conseguimos ter uma identificação mais detalhada dos erros e garantir que conseguimos lidar com isso da melhor forma. Isto é visível no facto do Controller "capturar" a exceção e chamar os métodos da View para emitir mensagens de erro adequadas ao usuário.

- **Interação do usuário com a interface** - Ao capturarmos exceções através do Controller, conseguimos interagir com a interface do usuário, através dos métodos da View. Desta forma conseguimos exibir mensagens de erro ao utilizador, solicitar novas entradas ou lidar com as exceções de forma adequada. Este tipo de interação também garante uma melhor experiência ao utilizador, quando a resposta de um componente demora muito tempo.

Aspetos interessantes e desafiantes do uso de exceções face às características da API .NetMaui



- **Abstração e Independência** - O uso de interfaces numa API .NET MAUI permite abstrair a implementação de um componente. Desta forma há independência entre os componentes, permitindo que se comuniquem de forma desacordada. Se olharmos para o código, verificamos que isso acontece na interface IView, que define os métodos ExibirMensagem, ExibirLista e ExibirErro, que devem ser implementados pela classe View. Essa interface permite que outros componentes como o Controller, comunique com a View sem depender diretamente da implementação concreta. Assim o Controller não precisa de conhecer os detalhes da View, mas apenas a interface para interagir com ela.
- **Polimorfismo** - Com esta aplicação foi possível "aplicar" o conceito de polimorfismo. Isto é útil uma vez que conseguimos ter componentes reutilizáveis e que podem ser adaptados a contextos diferentes. Por exemplo, neste caso observamos polimorfismo através do uso de interfaces e dos eventos delegados que "ocorrem" no Controller.

Conclusões, após o processo

O projeto foi desafiante.
Ainda que à primeira vista podesse parecer simples, todo o processo envolveu vários passos de implemnetação, verificação, re-avaliação do que estava feito e só assim conseguimos desenvolver a aplicação demonstradora apresentada.
No final sentimos que conseguimos evoluir, e compreender a importância de cada fase no desenvolvimento de uma aplicação demonstradora.

O que poderia ter ficado melhor?

Data da tarefa	Eleminar tarefas	Menu simples após erro
Implementar uma opção para o utilizador inserir a data prevista para a realização da tarefa, possibilitando assim uma melhor gestão das tarefas	Implementar uma opção para eliminar tarefas já concluídas, por forma a poder ter acesso a listas sempre atualizadas das tarefas ainda em curso	Quando é lançado o erro e o utilizador clica em qualquer tecla para continuar, poderia ser apresentado um Menu apenas com as opções de “Adicionar tarefa” e “Sair”, por forma a evitar que o utilizador volte a pedir uma lista vazia e a gerar o erro novamente

OBRIGADO!

Dúvidas?

Estamos aqui para responder a todas as dúvidas sobre o nosso projeto.

Feedback?

Queremos ouvir o feedback de todos! Essa é a melhor forma de identificar melhorias a implementar.