

Programação Orientada a Objetos (POO)- LESI(PL)

Relatório do trabalho da disciplina de programação
orientada a objetos

Gonçalo Pereira Gomes 25455

EST- IPCA

Barcelos

Novembro de 2024

Índice

Índice	2
Índice de Figuras.....	3
Introdução.....	4
Objetivos	5
1ª Fase	6
Estrutura das Classes.....	6
1. Animal	6
2. Alimentação	7
3. Assistência Veterinária	7
4. Bilhete	8
5. Cliente.....	8
6. Espetáculo.....	9
7. Funcionário.....	9
8. Utilizador	10
9. TipoComida	10
10. LimpezaHabitat.....	11
11. Habitat.....	11
Estruturas de dados	12
Implementação essencial de uma classe.....	13
2ª Fase	15
Organização em N-Tier	15
Pilares da programação orientada a objetos.....	16
Herança	16
Polimorfismo	16
Abstração.....	16
Encapsulamento	16
Desenvolvimento do Windows Forms	17
Conclusão.....	19
Bibliografia	20

Índice de Figuras

Figura 1- Construtor da classe Animal.....	13
Figura 2- Método "EscolherAnimalAleatório da classe Animal.....	14
Figura 3 - Organização N-Tier	15
Figura 4 - Menu Funcionário.....	17
Figura 5 - Menu Adicionar Animal	17
Figura 6 - Menu Mostrar Listas	18
Figura 7 - Camada Frontend e todos os menus.....	18

Introdução

No âmbito da Unidade Curricular de Programação Orientada a Objetos integrada no 1º semestre do 2º ano do curso de LESI, lecionada pelo docente Ernesto Casanova, foi proposto desenvolver um projeto em linguagem C# onde se pretendia soluções para problemas reais de complexidade moderada.

Optou-se pelo tema sugerido pelo docente, gestão de jardim zoológico, onde será implementado um sistema que permite a gestão de tarefas de um jardim zoológico.

Objetivos

Este projeto tem, para além do interesse académico inerente, uma relevância prática considerável, destacando-se diversos objetivos fundamentais. Em primeiro lugar, pretende consolidar os conceitos essenciais do Paradigma Orientado a Objetos, proporcionando uma base sólida e aplicável neste modelo de programação.

Adicionalmente, visa a análise de problemas reais, permitindo a aplicação prática dos conhecimentos adquiridos e incentivando a resolução de desafios concretos. A utilização da linguagem de programação C# como parte integrante do projeto tem como objetivo desenvolver as competências de programação neste ambiente específico.

Para além das vantagens práticas, o projeto procura também potenciar a experiência no desenvolvimento de software, o que inclui a aplicação de técnicas específicas, bem como o desenvolvimento de competências de programação, gestão de projetos e resolução eficiente de problemas.

Por fim, o projeto procura assegurar a assimilação completa dos conteúdos da Unidade Curricular, garantindo não só a compreensão dos conceitos teóricos, mas também a capacidade de os aplicar de forma eficaz em contextos práticos.

1ª Fase

Estrutura das Classes

Nesta primeira fase do projeto, foi realizada a identificação e definição das principais classes que compõem o sistema, estabelecendo a base para o desenvolvimento de funcionalidades mais complexas. As classes foram delineadas para representar os elementos e operações essenciais do sistema, garantindo uma estrutura coerente e robusta. Segue a descrição das classes identificadas:

1. Animal

Descrição: A classe Animal representa os animais no zoológico, incluindo detalhes como dieta, identificação, nome, espécie, idade e peso. Esta classe mantém uma lista estática de todos os animais criados, o que facilita a sua gestão e consulta no sistema.

Atributos:

- **DIETA dieta:** Enumeração que define o tipo de dieta (Carnívoro, Herbívoro, Omnívoro).
- **int id:** Identificador único do animal.
- **static int idstatic:** Atributo estático usado para gerar IDs únicos.
- **string nome:** Nome do animal.
- **string especie:** Espécie do animal.
- **int idade:** Idade do animal em anos.
- **double peso:** Peso do animal.
- **static List<Animal> animais:** Lista estática que armazena todos os animais criados.

Funcionalidades: Permite a criação e gestão de objetos Animal, adicionando-os à lista geral para um fácil acesso e manipulação.

2. Alimentação

Descrição: A classe Alimentacao regista eventos de alimentação dos animais, incluindo informações sobre a data e hora da alimentação, o animal envolvido, a quantidade de comida e o tipo de alimento fornecido.

Atributos:

- **int id:** Identificador único do registo de alimentação.
- **static int idstatic:** Atributo estático para geração automática de IDs.
- **Animal animal:** Referência ao objeto Animal que foi alimentado.
- **DateTime tempoAlimentacao:** Data e hora em que a alimentação ocorreu.
- **double quantidade:** Quantidade de comida fornecida, em unidades apropriadas.
- **TipoComida tipocomida:** Tipo de comida utilizada na alimentação.
- **static List<Alimentacao> alimentacoes:** Lista estática de todos os registos de alimentação.

Funcionalidades: Permite criar registos detalhados de alimentação e associar esses eventos a animais específicos.

3. Assistência Veterinária

Descrição: A classe AssistênciaVeterinária representa os registos de tratamento veterinário realizados para os animais. Armazena informações sobre o animal tratado, a data do tratamento e o habitat associado.

Atributos:

- **Habitat habitat:** Referência ao habitat onde o animal se encontra.
- **Animal animal:** Referência ao objeto Animal que recebeu assistência.
- **DateTime datadotratamento:** Data em que o tratamento foi realizado.
- **static List<AssistênciaVeterinária> assistenciaveterinária:** Lista estática que guarda todos os registos de assistência veterinária.

Funcionalidades: Regista eventos de tratamento e mantém um histórico acessível para consultas futuras.

4. Bilhete

Descrição: A classe Bilhete representa os bilhetes emitidos para os visitantes do zoológico, especificando a zona a que o bilhete dá acesso, o tipo de bilhete (passeio, espetáculo ou completo) e o preço. Inclui uma lista estática de bilhetes emitidos.

Atributos:

- **ZONA zona:** Enumeração que indica a zona do zoológico (Savana, Deserto, Aquática, Floresta).
- **TIPOBILHETE tipobilhete:** Enumeração que define o tipo de bilhete (Passeio, Espetáculo, Completo).
- **double preço:** Preço do bilhete.
- **int id:** Identificador único do bilhete.
- **static int idstatic:** Atributo estático para geração de IDs únicos.
- **static List<Bilhete> bilhetes:** Lista estática que armazena todos os bilhetes emitidos.

Funcionalidades: Permite criar e gerir bilhetes, registando cada um com os seus detalhes específicos e facilitando a consulta e a gestão de vendas.

5. Cliente

Descrição: A classe Cliente herda de Utilizador e representa os visitantes do zoológico, com atributos específicos relacionados ao seu saldo monetário.

Atributos:

- **double saldo:** Valor monetário disponível na conta do cliente.

Funcionalidades: Permite criar clientes e gerir o seu saldo para operações de compra de bilhetes e outros serviços.

6. Espetáculo

Descrição: A classe Espetáculo representa as apresentações ou exibições que ocorrem no zoológico, com informações sobre o tipo de espetáculo, o nome, o horário e o animal participante.

Atributos:

- **int id:** Identificador único do espetáculo.
- **TIPOESPETÁCULO tipoespetaculo:** Enumeração que define o tipo de espetáculo (Elefante, Golfinho, Tubarão, Leão).
- **string nome:** Nome do espetáculo.
- **DateTime horario:** Data e hora em que o espetáculo ocorre.
- **Animal animalespetaculo:** Animal que participa do espetáculo.
- **static List<Espetáculo> espetaculos:** Lista estática que armazena todos os espetáculos.

Funcionalidades: Gerencia a criação, programação e listagem de espetáculos.

7. Funcionário

Descrição: A classe Funcionário herda de Utilizador e representa os trabalhadores do zoológico. Possui uma lista estática para gerir todos os funcionários.

Atributos:

- **static List<Funcionário> funcionários:** Lista que armazena os funcionários do sistema.

Funcionalidades: Permite criar funcionários e registar as suas funções e operações.

8. Utilizador

Descrição: Classe abstrata que representa um utilizador do sistema, seja ele um cliente ou um funcionário. Contém atributos comuns a todos os tipos de utilizadores.

Atributos:

- **int id:** Identificador único do utilizador.
- **static int idstatic:** Utilizado para gerar IDs únicos.
- **string username:** Nome de utilizador.
- **string password:** Senha para autenticação.
- **string email:** Endereço de email.
- **string nome:** Nome completo do utilizador.
- **string nif:** Número de Identificação Fiscal.
- **static List<Utilizador> utilizadores:** Lista estática de todos os utilizadores.

Funcionalidades: Serve como classe base para Cliente e Funcionário, proporcionando métodos comuns e atributos partilhados.

9. TipoComida

Descrição: A classe TipoComida gere diferentes tipos de comida que podem ser fornecidos aos animais, incluindo detalhes nutricionais e a dieta associada.

Atributos:

- **int id:** Identificador único do tipo de comida.
- **static int idstatic:** Para geração automática de IDs.
- **DIETA dieta:** Enumeração que define a dieta (Carnívoro, Herbívoro, Omnívoro).
- **string nomecomida:** Nome da comida.
- **double calorias:** Valor calórico da comida.
- **static List<TipoComida> tipocomidas:** Lista que contém todos os tipos de comida.

Funcionalidades: Métodos para adicionar, mostrar e remover tipos de comida da lista.

10. LimpezaHabitat

Descrição: A classe LimpezaHabitat regista a limpeza de habitats, indicando o habitat específico e a hora em que a limpeza foi realizada.

Atributos:

- **Habitat habitat:** Referência ao habitat que foi limpo.
- **DateTime horalimpeza:** Hora em que a limpeza ocorreu.
- **static List<LimpezaHabitat> limpezahabitats:** Lista de todos os registos de limpezas.

Funcionalidades: Permite registar eventos de limpeza, associando cada um a um habitat e a uma data específica.

11. Habitat

Descrição: A classe Habitat representa os espaços onde os animais vivem, incluindo informações sobre a zona, o nome do habitat e os animais que lá habitam.

Atributos:

- **ZONA zona:** Enumeração que define a zona do habitat (Savana, Deserto, Aquática, Floresta).
- **int idhabitat:** Identificador único do habitat.
- **string nomehabitat:** Nome do habitat.
- **List<Animal> animais habitat:** Lista de animais que vivem no habitat.
- **static List<Habitat> habitats:** Lista estática de todos os habitats criados.

Funcionalidades: Gerencia a alocação e a gestão dos habitats, incluindo a listagem de animais por habitat e o controlo de zonas.

Estruturas de dados

As estruturas de dados são fundamentais para organizar e manipular dados de forma eficiente no desenvolvimento de software. Elas determinam como os dados são armazenados na memória e acedidos, sendo que as escolhas de estruturas variam consoante as necessidades específicas do sistema. No contexto deste projeto, a estrutura de dados principal utilizada foi a lista, mais especificamente a lista estática.

As listas em C# permitem armazenar coleções de objetos de forma dinâmica. A classe **List<T>** oferece funcionalidades para adicionar, remover e aceder aos elementos, o que a torna útil para armazenar dados que podem crescer ou ser modificados ao longo do tempo. A lista estática é uma variação onde os dados são partilhados entre todas as instâncias de uma classe, permitindo o acesso global a essa coleção. Neste projeto, foram utilizadas listas estáticas para armazenar objetos em várias classes, como na classe **Animal**, onde é guardada a lista de todos os animais.

A vantagem principal das listas estáticas é que elas facilitam o acesso e a manipulação dos dados em qualquer parte do sistema, sem a necessidade de instanciar objetos. Isso torna o código mais organizado e permite uma gestão centralizada dos dados, ideal para o tipo de informações que o sistema precisa manipular, como registos de animais, alimentação e espetáculos.

Implementação essencial de uma classe

A implementação essencial de uma classe envolve a definição dos seus **atributos**, que são as características ou propriedades que representam o estado da classe, e dos **métodos**, que definem os comportamentos ou ações que a classe pode realizar. Além disso, a classe pode incluir um **construtor**, responsável por inicializar os seus atributos com valores específicos quando uma nova instância da classe é criada. A classe pode também conter métodos estáticos ou instâncias de outras classes, criando interações e funcionalidades dentro do sistema.

Por exemplo, na classe **Animal**, a implementação essencial inclui **atributos** como **id**, **nome**, **especie**, **peso**, **idade** e **dieta**, que definem as características fundamentais de cada animal no zoológico. A classe também possui um **construtor**, que permite inicializar esses atributos de forma personalizada para cada instância criada.

```
/// <summary>
/// Constructor para criar um animal com os atributos principais.
/// </summary>
/// <param name="nome">Nome do animal.</param>
/// <param name="especie">Espécie do animal.</param>
/// <param name="idade">Idade do animal em anos.</param>
/// <param name="peso">Peso do animal em quilogramas.</param>
/// <param name="dieta">Tipo de dieta do animal.</param>

public Animal(string nome, string especie, int idade, double peso, DIETA dieta)
{
    idstatic++;
    id = idstatic;
    Nome = nome;
    Especie = especie;
    Idade = idade;
    Peso = peso;
    Dieta = dieta;
}
```

Figura 1- Construtor da classe Animal

Para além disso, um exemplo de método implementado na classe é o método **EscolherAnimalAleatorio**, que permite selecionar aleatoriamente um animal da lista de animais com base na sua espécie.

```
/// <summary>
/// Seleciona aleatoriamente um animal de uma espécie específica.
/// </summary>
/// <param name="especie">Espécie desejada do animal.</param>
/// <returns>Retorna um animal aleatório da espécie; null se não houver animais da espécie.</returns>
1 referência
public static Animal? EscolherAnimalAleatorio(string especie)
{
    // Filtra os animais pela espécie desejada
    var animaisDaEspecie = animais.Where(a => a.Especie == especie).ToList();

    // Verifica se há animais disponíveis para a espécie especificada
    if (animaisDaEspecie.Count == 0)
    {
        Console.WriteLine($"Nenhum animal disponível para a espécie {especie}.");
        return null;
    }

    // Seleciona aleatoriamente um animal da lista filtrada
    Random random = new Random();
    int index = random.Next(animaisDaEspecie.Count);
    return animaisDaEspecie[index];
}
```

Figura 2- Método "EscolherAnimalAleatório" da classe *Animal*

2ª Fase

A segunda fase do projeto teve como objetivos principais a implementação da organização em N-Tier, o desenvolvimento da interface gráfica utilizando Windows Forms e assegurar os 4 pilares da programação orientada a objetos (POO): herança, encapsulamento, polimorfismo e abstração.

Organização em N-Tier

A organização em N-tier (ou camadas) é uma arquitetura de software que divide o sistema em diferentes camadas lógicas e funcionais. As camadas típicas incluem o frontend(interface do utilizador), regras zoo e acesso a dados. Essa abordagem promove a escalabilidade e facilita a manutenção do sistema. As interligações entre as camadas devem ser cuidadosamente implementadas, permitindo que cada camada funcione de forma independente e seja substituível sem afetar as outras.

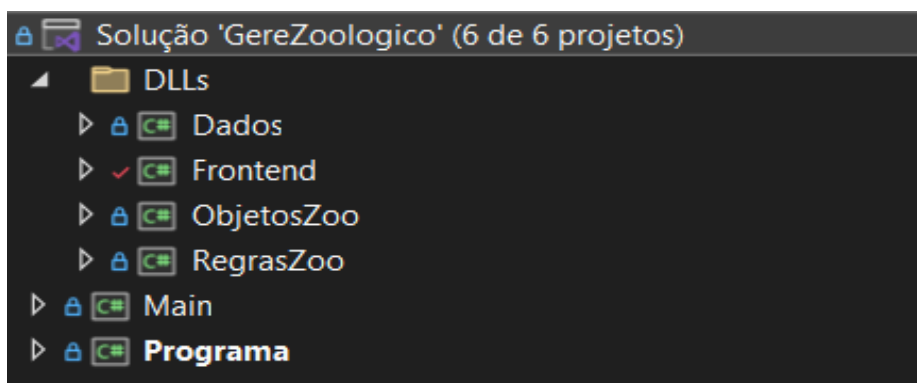


Figura 3 - Organização N-Tier

Pilares da programação orientada a objetos

Para aprimorar o código e torná-lo mais modular e reutilizável, a aplicação dos 4 pilares da programação orientada a objetos foi um aspecto crucial na segunda parte do projeto.

Herança

A herança é um conceito fundamental na programação orientada a objetos que permite criar uma nova classe baseada em uma classe existente, aproveitando as propriedades e comportamentos da classe original. A classe recém-criada é chamada de classe derivada ou subclasse. A herança promove a reutilização de código, evitando a duplicação de implementações e facilitando a manutenção. Além disso, ela suporta a criação de hierarquias de classes, onde as classes mais específicas herdam características mais gerais.

Polimorfismo

Polimorfismo refere-se à capacidade de um objeto se comportar de diferentes maneiras com base no contexto. O polimorfismo permite que diferentes métodos tenham o mesmo nome, mas aceitem parâmetros diferentes. O polimorfismo aumenta a flexibilidade e extensibilidade do código, permitindo que diferentes partes do sistema interajam de maneira consistente e adaptável.

Abstração

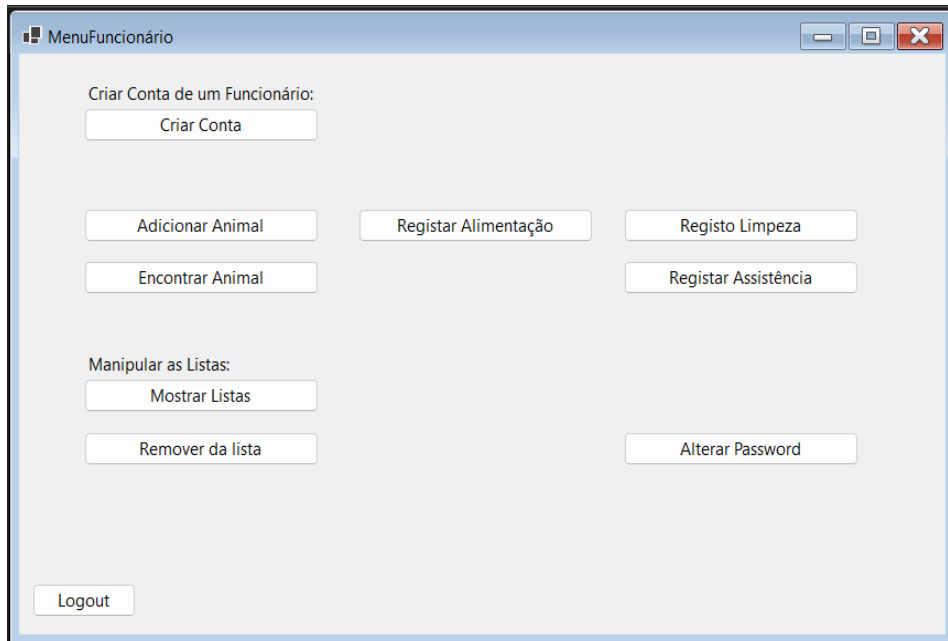
A abstração é um conceito que envolve a simplificação e representação de objetos do mundo real em um nível mais elevado de generalização. É utilizada em conjunto com a herança, permitindo a reutilização de código, diferindo desta, pois a classe derivada tem que obrigatoriamente reescrever os métodos abstratos da classe base.

Encapsulamento

Encapsulamento é o princípio de agrupar os dados (atributos) e os métodos (comportamentos) relacionados a um objeto em uma única classe. O encapsulamento restringe o acesso direto aos detalhes internos de um objeto, permitindo que o seu estado seja modificado apenas por meio de métodos cuidadosamente controlados (públicos). Isso ajuda a proteger a integridade dos dados e a ocultar a complexidade interna, facilitando a manutenção do código.

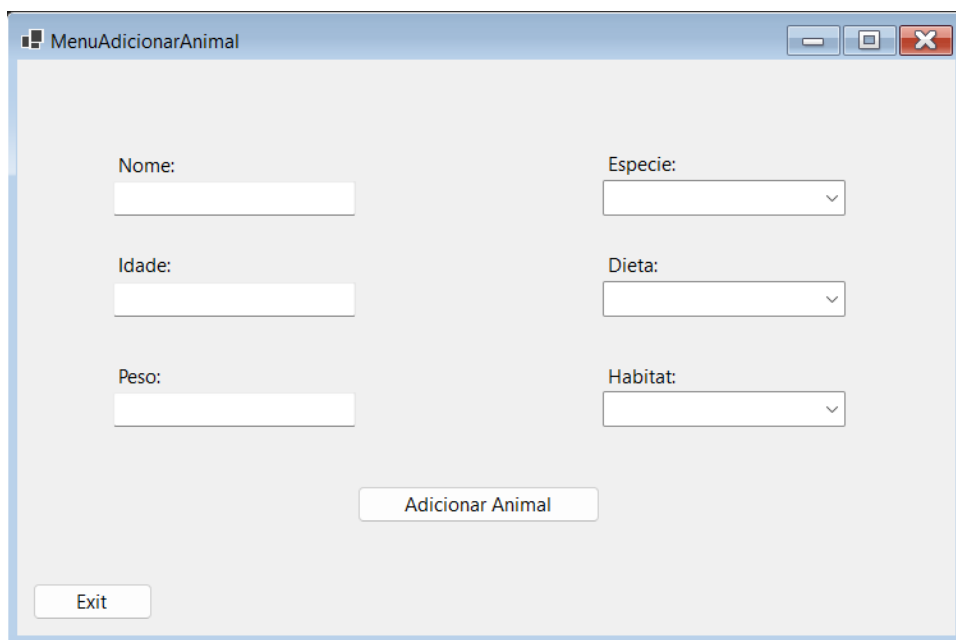
Desenvolvimento do Windows Forms

Após a divisão por camadas, a próxima etapa foi o desenvolvimento da interface gráfica utilizando a ferramenta **Windows Forms**. Foram implementados vários forms que permitem ao utilizador realizar diversas operações, como por exemplo:



The screenshot shows a Windows Form titled "MenuFuncionário". It contains several buttons for user management and system operations. The buttons are arranged in a grid-like fashion. At the top, there is a button "Criar Conta" under the label "Criar Conta de um Funcionário:". Below this, there are three buttons in a row: "Adicionar Animal", "Registrar Alimentação", and "Registo Limpeza". Underneath these, there are two buttons: "Encontrar Animal" and "Registrar Assistência". Further down, under the label "Manipular as Listas:", there are two buttons: "Mostrar Listas" and "Remover da lista". At the bottom right, there is a button "Alterar Password". Finally, at the bottom left, there is a "Logout" button. The form has a standard Windows title bar with minimize, maximize, and close buttons.

Figura 4 - Menu Funcionário



The screenshot shows a Windows Form titled "MenuAdicionarAnimal". It is designed for adding a new animal to the system. It features six input fields arranged in two columns. The left column contains text boxes for "Nome:", "Idade:", and "Peso:". The right column contains dropdown menus for "Especie:", "Dieta:", and "Habitat:". Each field is preceded by its respective label. At the bottom center, there is a button labeled "Adicionar Animal". At the bottom left, there is an "Exit" button. The form has a standard Windows title bar with minimize, maximize, and close buttons.

Figura 5 - Menu Adicionar Animal

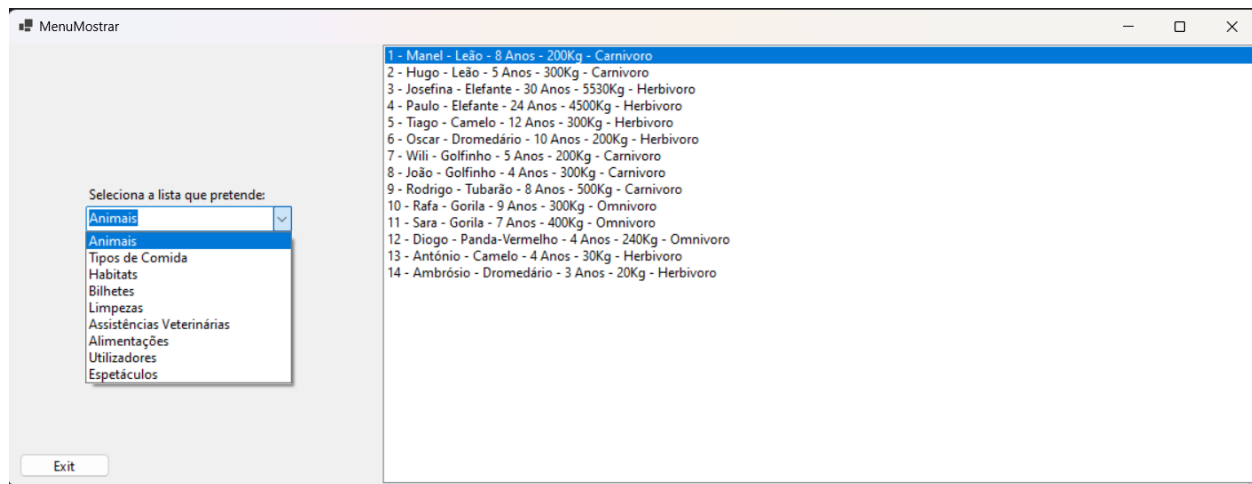


Figura 6 - Menu Mostrar Listas

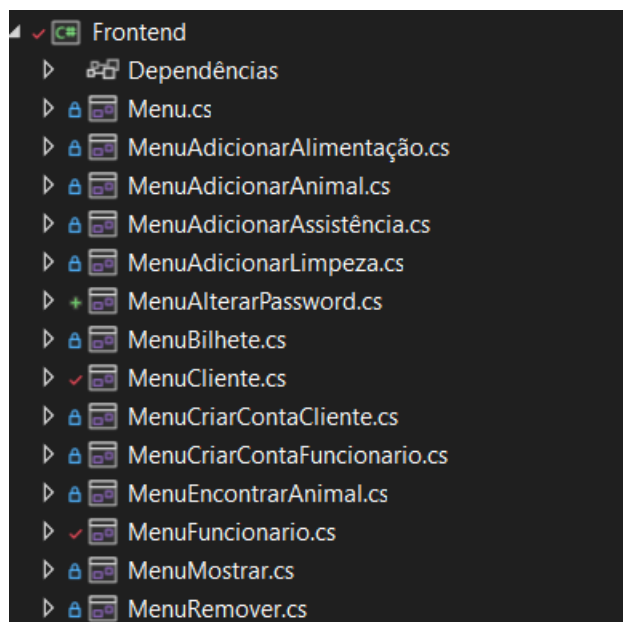


Figura 7 - Camada Frontend e todos os menus

Conclusão

O desenvolvimento deste projeto permitiu a aquisição de novas práticas no desenvolvimento de um projeto na linguagem de programação C#, assim como a consolidação de conhecimentos adquiridos nas aulas da cadeira de Programação Orientada a Objetos.

Este projeto forneceu uma base sólida para futuros empreendimentos e na área do desenvolvimento de software, trazendo certeza de que as lições aprendidas serão lembradas em futuros projetos, tanto a nível acadêmico como a nível profissional.

Bibliografia

Microsoft Learn (dotnet.microsoft.com): <https://dotnet.microsoft.com/pt-br/learn/csharp>

C# Programming Guide (docs.microsoft.com): <https://learn.microsoft.com/en-us/dotnet/csharp/>

TutorialsPoint (tutorialspoint.com): <https://www.tutorialspoint.com/csharp/index.htm>

W3Schools (w3schools.com): <https://www.w3schools.com/cs/>