



RSGYMP T

Projeto II (2) – RSGymPT

OBJECT-ORIENTED PROGRAMMING

Alexandra Mendes

Introdução

Este relatório documenta o desenvolvimento do negócio RSGymPT e das suas aplicações de apoio à gestão, projetada para ampliar os serviços do ginásio RSGym oferecendo aulas de personal trainer ao domicílio.

O relatório, em formato PowerPoint, apresenta uma visão geral do projeto, destacando a migração do módulo 'Gestão de Clientes' para um ambiente web e o desenvolvimento do módulo 'Administração de Utilizadores'.

Esta documentação visa fornecer um entendimento claro do progresso e das funcionalidades das vertentes de apoio da aplicação, facilitando a comunicação dos resultados alcançados e dos próximos passos no projeto.

NOTA: as aplicações estão estruturalmente em inglês para fácil escala e adaptação, no entanto, para precaver diferenças do enunciado, o nome dos projetos no Visual Studio permanecem em português para fácil associação assim como o relatório.



Descrição da aplicação

O ginásio RSGym está a expandir os seus serviços, oferecendo aulas de personal trainer (PT) ao domicílio. Este novo acrescento de valor e serviços são suportados pela aplicação RSGymPT, que possui duas vertentes principais:

interface do cliente e o back-end.

A Vertente Back-End - a desenvolver nesta fase: back-end da aplicação é composto por dois módulos principais:

- **Módulo 'Gestão de Clientes':** Originalmente desenvolvido como uma aplicação de consola, este módulo gere a inscrição dos clientes e os pagamentos. No âmbito deste projeto, o módulo está sendo atualizado para um ambiente web, mantendo todas as funcionalidades existentes.
- **Módulo 'Administração de Utilizadores':** Este novo módulo, desenvolvido em sistema de consola, permite a administração dos utilizadores da aplicação, incluindo a gestão de permissões e acesso.

Objetivos da aplicação RSGymPT: experiência eficiente e conveniente tanto para os clientes quanto para os administradores/empregados, facilitando a gestão dos serviços de personal trainer e satisfação dos intervenientes.

Estrutura geral do projeto

Vertente back-end

'Gestão de Clientes'

(ambiente web) MVC Core 6, Entity Framework Core, padrão model first

'Administração de Utilizadores'

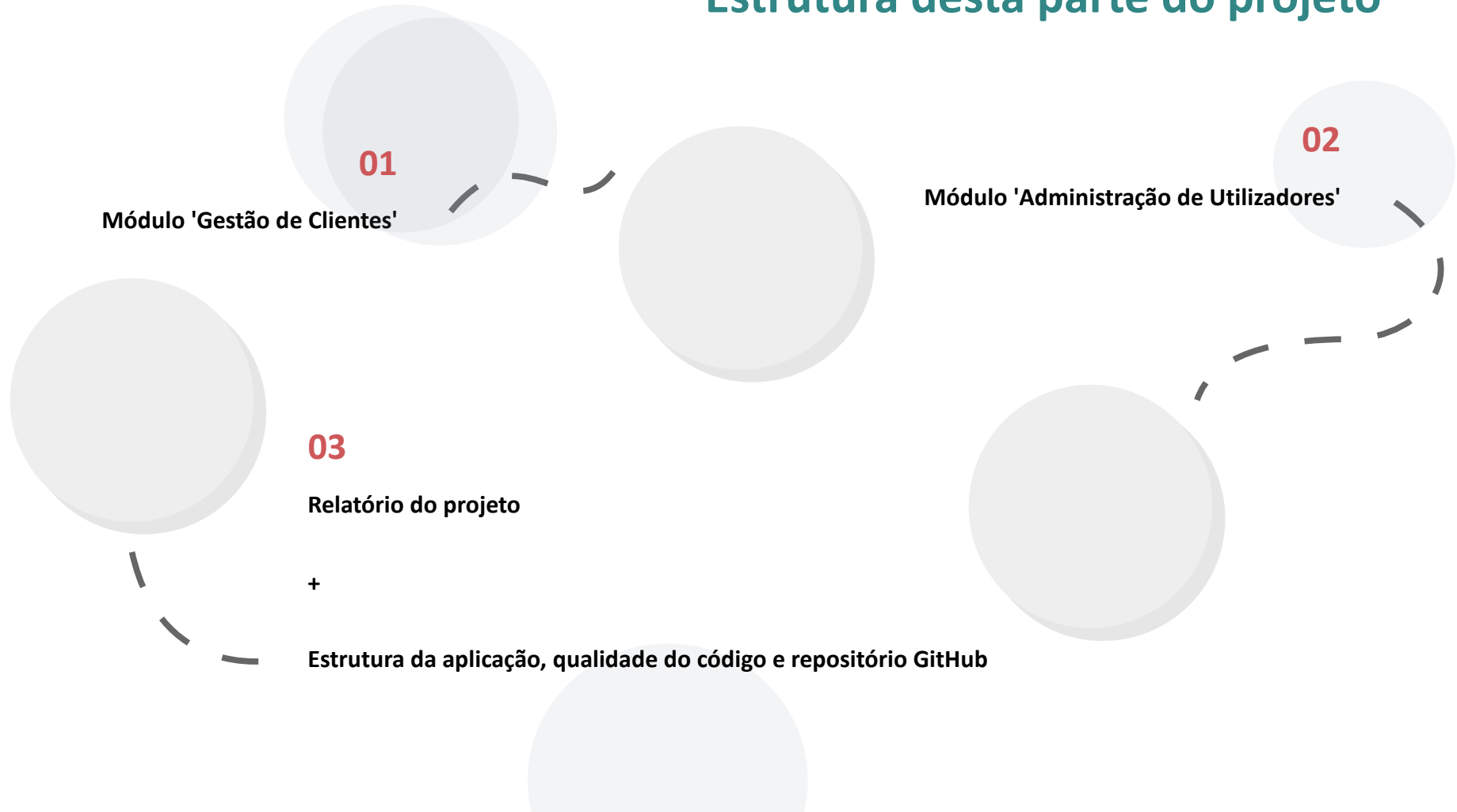
(aplicação de consola) .NET Framework 4.7.2

Vertente interface do cliente

A interface do cliente permite que os utilizadores se inscrevam no serviço de personal trainer, possam ter acesso a informações sobre as aulas e façam agendamentos diretamente pela aplicação.

(aplicação de consola - 1ª parte do projeto)

Estrutura desta parte do projeto





//

Módulo 'Gestão de Clientes'

//



Estrutura

- Objetivo 1: **trata da inscrição dos clientes** na base de dados
 - Tratar da informação dos dados dos clientes ao inscrever como contactos, localização, informações pessoais como o NIF, tipo de contrato, fidelização escolhida e que produto têm, ou seja, serviço.
- Objetivo 2: **controlo dos pagamentos**
 - Ter informação dos pagamentos até então com os dados do cliente a que diz respeito, método de pagamento, data de pagamento e valor.

O enunciado não inclui emissão de fatura ou outras formas de registo contabilístico e operacional sendo que podem existir várias aplicações criadas para tratar de outros objetivos do negócio e complementarem-se na aplicação.



Criação do logotipo da aplicação

Relações

| Client | | | | | | | | | | | | |
|----------|----------------|--------------------|-----------|----------|---------------|--------------|---------|------------|------------|-----------------------------|----------------|------------------|
| ClientId | ContractTypeld | FidelizationTypeld | FirstName | LastName | Street | DoorAndFloor | City | PostalCode | FiscalCode | Email | Phone | RegistrationDate |
| 1 | 1 | 1 | Francisco | Morgado | Rua D. Maria | 63 | Porto | 4000-001 | 987654321 | F_morgado67@gmail.com | +3519191589899 | 30.01.2024 |
| 2 | 2 | 1 | Inês | Soares | Rua D. Manuel | 45 | Coimbra | 3000-001 | 192837465 | ij_soares@gmail.com | +3519191589855 | 01.04.2024 |
| 3 | 1 | 2 | Madalena | Tavares | Rua D. Joao V | 5 | Lisboa | 1000-001 | 123456789 | madalena90_soares@gmail.com | +3519191589888 | 12.05.2024 |

| ClientService | | |
|-----------------|----------|-----------|
| ClientServiceId | ClientId | ServiceId |
| 1 | 1 | 8 |
| 2 | 2 | 1 |
| 3 | 3 | 7 |

| ContractType | |
|----------------|------------------------------|
| ContractTypeId | Description |
| 1 | Fixed monthly fee |
| 2 | One-time payment per service |

| Service | | |
|-----------|-------------------------------------|-----------|
| ServiceId | Description | UnitPrice |
| 1 | Personal Training | 50,00 € |
| 2 | Group Fitness Classes | 15,00 € |
| 3 | Open functional training zone | 20,00 € |
| 4 | Open specialized weightlifting zone | 25,00 € |
| 5 | Nutritional Counseling | 15,00 € |
| 6 | Massage Therapy | 15,00 € |
| 7 | All services pack | 20,00 € |
| 8 | Two services pack | 20,00 € |

| FidelizationType | | | | |
|--------------------|-------------|----------|----------|-----------------|
| FidelizationTypeId | IsFidelized | Duration | Discount | ServicesMaximum |
| 1 | No | 0 | 0 | 2 |
| 2 | Yes | 4 | 15,00% | null |

| Payment | | | | |
|-----------|----------|-------------|------------------------|---------|
| PaymentId | ClientId | PaymentDate | PaymentMethod | Value |
| 1 | 1 | 30.02.2024 | Mobile Payment Service | 20,00 € |
| 2 | 1 | 30.03.2024 | Credit Card | 20,00 € |
| 3 | 2 | 30.04.2024 | Cash | 50,00 € |
| 4 | 1 | 30.04.2024 | Mobile Payment Service | 20,00 € |
| 5 | 3 | 30.05.2024 | Bank Transfer | 17,00 € |
| 6 | 1 | 30.05.2024 | Debit Card | 20,00 € |
| 7 | 3 | 30.06.2024 | Cash | 17,00 € |
| 8 | 1 | 30.06.2024 | Credit Card | 20,00 € |

Notas

Lógica das relações

- 1 serviço pode ter n clientes e n clientes podem ter 1 serviço (n-n)
- 1 tipo de contrato pode ter n clientes e 1 cliente só pode ter 1 tipo de contrato (1-n)
- 1 tipo de fidelização pode ter n clientes e 1 cliente só pode ter 1 tipo de fidelização (1-n)
- 1 cliente pode ter n pagamentos associados e 1 pagamento tem 1 cliente associado (1-n)

Uso de dados meramente fictícios e sugestivos para racionalizar e chegar às tabelas finais, com base em decisões de regras de negócio como:

ex. tabela de clientes com informação de localidade e contactos/detalhes (mantidos propositadamente – no meu ponto de vista o contexto desta aplicação não beneficiaria de tabelas adicionais e, então, decidi **manter dependências**)

Aplicação

ASP.NET Core Web Application (Model-View-Controller)

- Instalação de packages:
 1. Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 6.0.10
 2. Install-Package Microsoft.EntityFrameworkCore.Tools -Version 6.0.10
 3. Install-Package Microsoft.EntityFrameworkCore.Design -Version 6.0.10

(atualização posterior para 6.0.32 para resolução de incompatibilidades)

- **Criação das classes/entidades - Models, conforme as relações existentes entre as tabelas decididas**
 - ContractType: ContractTypeId, Description;
 - FidelizationType: FidelizationTypeId, Description, Duration, Discount, ServicesMaximum;
 - Client: ClientId, ContractTypeId, FidelizationTypeId, FirstName, LastName, Street, DoorAndFloor, City, PostalCode, FiscalCode, Email, Phone, RegistrationDate;
 - CientService: CientServiceId, ClientId, ServiceId;
 - Service: ServiceId, Description, Unit Price;
 - Payment: PaymentId, ClientId, PaymentDate, PaymentMethod, Value;

```

// Phone
[Required(ErrorMessage = "Phone is required")]
[StringLength(13, ErrorMessage = "Cannot be longer than 13 characters")]
[RegularExpression(@"^(+351)?\d{9}$", ErrorMessage = "Phone number must be 9 digits long, optionally starting with +351")]
[Phone(ErrorMessage = "Invalid Phone Number")]
[Column(TypeName = "nvarchar(13)")]
[Display(Name = "Phone")]
13 references
public string Phone { get; set; }

// RegistrationDate
[DataType(DataType.Date)]
[Display(Name = "Registration date")]
7 references
public DateTime RegistrationDate { get; set; } = DateTime.Now;

#endregion

#region Navigation
9 references
public ContractType? ContractType { get; private set; }
9 references
public FidelizationType? FidelizationType { get; private set; }
0 references
public ICollection<Payment> Payment { get; } = new List<Payment>();
0 references
public ICollection<ClientService> ClientService { get; } = new List<ClientService>();

#endregion

```

exemplo de data annotations e navigation properties

- **Decisão/uso de Data Annotations**

- **Não required** na classe FidelizationType na Duration (pode não ter especificamente fixo), ServicesMaximum (pode não ter/ser nul), Discount (pode não ter benefício direto desta forma)

- **Criação de navigation properties**

relações n-n (outra tabela) e relações 1-n

Ex. public Type type { get; set; }

public ICollection<Type> Type { get; set; }

- Configuração da `ConnectionString` no `appsettings.json` com `server localhost`. É benéfico ter a `connection string` guardada no `appsettings.json` em vez de ter a `connection string` guardada na classe `program`.
- Criação da pasta `DAL` (`Data Access Layer`) e adicionar a classe contexto : `DbContext` com `DbSet<T>` que representam uma tabela na base de dados.
- Com a criação do projeto MVC com template, há configurações necessárias já no `program`. Com o uso do `Entity Framework Core`, há uma configuração a fazer para adicionar um contexto, através do método `AddDbContext`. Este método usa o tipo genérico que foi criado que herda da classe `DbContext`, que como parâmetro indica que as `DbContextOptions` desse contexto usam o `SqlServer` (que precisa de uma `connection string` para saber que base de dados usar). No construtor da classe do `DbContext` define que as options são do tipo do `DbContext` criado herdando da base. Para conseguirmos ter o valor da `connection string` que está no `appsettings` usa-se o `WebApplicationBuilder`.
- Criação da migração inicial com uma pasta criada chamada `Migrations` introduzida na pasta `DAL` e fazer `update-database` no package manager console para criar a base de dados.
- Criar os controllers com scaffolding: "MVC Controller with views, using Entity Framework".
- Alterar o layout e incluir os links, na navigation bar, para os novos controladores e verificar as views.
- Alguma alteração extra: nova migração.

(novos packages foram introduzidos automaticamente para resolução de incompatibilidades e funcionalidades)

Nome da base de dados:
`CA_RS11_P2-2_AlexandraMendes`

A interface de utilizador (UI)

Desenvolvida com:

- **Views em Razor**
 - Logotipo da aplicação incluído no index referente a cada entidade, assim como extras de estilização desenvolvidos a partir do bootstrap.css (interpretação) e site.css (extras e acrescentos).
 - Barra de navegação que identifica controlador e ação (barra de navegação já por default tendo sido estilizada com css)
 - Identificação do formando no rodapé nas Shared Views no _Layout.cshtml visto que é partilhado por todas as views.
 - Outras estilizações a gosto com css e vendo o bootstrap por defeito
- **Ações nos controladores em C#**
 - Mais validações e verificações de negócio e lógica de relação das tabelas adicionada nos controladores

Client

RSGymPT Management

[Home](#)

[Privacy](#)

[Clients](#)

[Client services](#)

[Services](#)

[Contract Types](#)

[Fidelization Types](#)

[Payments](#)



RSGYMP T

CLIENT MANAGEMENT AND PAYMENTS CONTROL

Welcome

Welcome

to the RSGymPT Management Application

solution for managing clients, services, payments, and more.

exemplo de página inicial

This intuitive tool allows to manage customer details, service types, payments, contract management, etc.
Let's get started and optimize business processes with ease!

© 2024 - [Alexandra Mendes](#) - [Privacy](#)

<mailto:alexandramendes.mm@gmail.com>

© 2024 - [Alexandra Mendes](#) - [Privacy](#)

Privacy Policy

Effective Date: 25.07.2024

Welcome to RSGymPT Comprehensive Management Application. At RSGymPT, we value your privacy and are committed to protecting your personal information. This Privacy Policy outlines how we collect, use, disclose, and safeguard your data. This policy applies to all who handle personal data in the course of managing gym operations.

1. Information We Collect

Personal Information:

- Members: We collect personal details such as name, contact information, membership, contract, and payment information.
- Employees: We collect employment details including name, contact information, job role, and payroll data.

Operational Data:

We also collect information related to gym usage, such as check-in times, class attendance, and equipment usage.

2. How We Use Your Information

For Members:

- Membership Management: To manage membership records, process payments, and provide customer support.
- Personalization: To tailor fitness programs and recommendations based on your preferences.
- Communication: To inform you about gym events, updates, and special offers.

For Employees:

- Payroll and Administration: To manage salaries, benefits, and employment records.
- Operational Efficiency: To ensure smooth gym operations and enhance the overall member experience.

3. How We Protect Your Information

We implement a range of security measures to protect your personal data, including:

- Access Controls: Restricted access to personal information based on job responsibilities.
- Data Encryption: Encryption of sensitive data during storage and transmission.
- Regular Audits: Conducting regular security audits to identify and address vulnerabilities.

4. Data Sharing and Disclosure

- Communication: To inform you about gym events, updates, and special offers.

For Employees:

- Payroll and Administration: To manage salaries, benefits, and employment records.
- Operational Efficiency: To ensure smooth gym operations and enhance the overall member experience.

3. How We Protect Your Information

We implement a range of security measures to protect your personal data, including:

- Access Controls: Restricted access to personal information based on job responsibilities.
- Data Encryption: Encryption of sensitive data during storage and transmission.
- Regular Audits: Conducting regular security audits to identify and address vulnerabilities.

4. Data Sharing and Disclosure

We do not sell or rent personal information to third parties. We may share data with:

- Service Providers: Third-party vendors who assist us with payment processing, customer support, and IT services, subject to confidentiality agreements.
- Legal Requirements: Disclosure required by law, such as to comply with a subpoena or other legal processes.

5. Your Rights

- Access and Correction: You have the right to access and correct your personal information.
- Data Retention: We retain personal data only as long as necessary for operational purposes and legal compliance.

6. Changes to This Policy

We may update this Privacy Policy from time to time. We will notify you of any significant changes by posting the updated policy on our website and, where appropriate, directly informing you.


7. Contact Us

If you have any questions or concerns about this Privacy Policy or how we handle your personal data, please contact us at:

- Email: rsgympt@hotmail.com
- Phone: +351 210 129 670
- Address: C. Colombo, Loja A, 201, Avenida Lusíada, 1500-392, Lisboa

Thank you for trusting RSGymPT with your personal information. We are committed to ensuring your privacy and providing a secure and trustworthy environment.

RSGymPT Management [Home](#) [Privacy](#) [Clients](#) [Client services](#) [Services](#) [Contract Types](#) [Fidelization Types](#) [Payments](#)




Clients

Welcome to the Client Registry Management Data

This display allows you to efficiently manage your customer database with features to view, edit, delete, add customer records, and more.

Keep the information organized and up-to-date with ease. Use this tool for customer management and enhance operational efficiency.

RSGymPT Management [Home](#) [Privacy](#) [Clients](#) [Client services](#) [Services](#) [Contract Types](#) [Fidelization Types](#) [Payments](#)




Clients and services

Welcome to the Client Service Management Data

This display allows you to efficiently track and manage the types of services the customers have chosen, with features to have a clear overview, edit, delete, add records, and more.

Keep records accurate and detailed for client service management efficiency.

RSGymPT Management [Home](#) [Privacy](#) [Clients](#) [Client services](#) [Services](#) [Contract Types](#) [Fidelization Types](#) [Payments](#)




Services

Welcome to the Service Management Data

This display allows you to efficiently manage and review the services available with features to have a clear overview, edit, delete, add records, and more.

Keep the information organized and stay informed about all service options and their costs, ensuring accurate information about service offerings. Use this for service management.

RSGymPT Management [Home](#) [Privacy](#) [Clients](#) [Client services](#) [Services](#) [Contract Types](#) [Fidelization Types](#) [Payments](#)




Contract types

Welcome to the Contract Type Management Data

This display allows you to efficiently manage and review the various types of contracts available with features to view, edit, delete, add records, and more.

Keep accurate records of all contract types for contract management and enhance efficiency.

RSGymPT Management [Home](#) [Privacy](#) [Clients](#) [Client services](#) [Services](#) [Contract Types](#) [Fidelization Types](#) [Payments](#)




Fidelization types

Welcome to the Fidelization Type Management Data

This display allows you to efficiently manage and review different types of customer fidelization programs, helping to track and understand various loyalty types, and with features to have a clear overview, edit, delete, add records, and more.

Keep records up-to-date and organized to enhance customer retention and improve loyalty strategy management.

RSGymPT Management [Home](#) [Privacy](#) [Clients](#) [Client services](#) [Services](#) [Contract Types](#) [Fidelization Types](#) [Payments](#)



Payments

Welcome to the Payment Management Data

This display allows you to efficiently manage all payment transactions with detailed insights including the payment date, method, amount, and associated client, and with features to have a clear overview, edit, delete, add records, and more.

Keep accurate records of all financial transactions for correct accounting processes and enhance overall financial management.



RSGYMP T
CLIENT MANAGEMENT AND PAYMENTS CONTROL

Payments

Welcome to the Payment Management Data

This display allows you to efficiently manage all payment transactions with detailed insights including the payment date, method, amount, and associated client, and with features to have a clear overview, edit, delete, add records, and more.

Keep accurate records of all financial transactions for correct accounting processes and enhance overall financial management.

[Create a new row for a new record](#)

| Operations in each row | Payment date | Payment method | Value | Client |
|---|--------------|----------------------|-------|------------------|
| Edit Details Delete | 30/04/2024 | Cash | 20,00 | Inês Soares |
| Edit Details Delete | 30/05/2024 | BankTransfer | 17,00 | Madalena Tavares |
| Edit Details Delete | 30/06/2024 | Cash | 17,00 | Madalena Tavares |
| Edit Details Delete | 30/07/2024 | MobilePaymentService | 17,00 | Madalena Tavares |

exemplo de página pagamentos



RSGYMP T
CLIENT MANAGEMENT AND PAYMENTS CONTROL

Clients

Welcome to the Client Registry Management Data

This display allows you to efficiently manage your customer database with features to view, edit, delete, add customer records, and more.

Keep the information organized and up-to-date with ease. Use this tool for customer management and enhance operational efficiency.

Create a new row for a new record

Create a new row for a new record



Details

| Operations in each row | First name | Last name | Street | Door and floor | City | Postal code | Fiscal code | Email | Phone | Registration date | ContractType | FidelizationType |
|---|------------|-----------|--------------|----------------|---------|-------------|-------------|--|---------------|-------------------|-------------------|------------------|
| Edit Details Delete | Madalena | Tavares | Rua D.Joao V | 5 | Lisboa | 1000-001 | 123456789 | madalena90_soares@gmail.com | +351919158988 | 18/08/2024 | Fixed monthly fee | Basic Membership |
| Edit Details Delete | Inês | Soares | Rua D.Manuel | 45 | Coimbra | 3000-001 | 192837465 | ij_soares@gmail.com | +351919158985 | 18/08/2024 | Fixed monthly fee | Basic Membership |

exemplo de página clientes



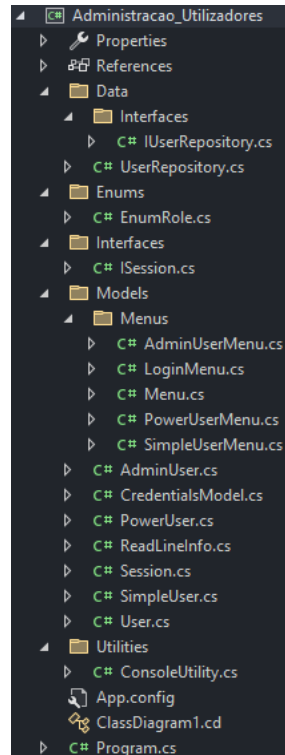
//

Módulo 'Administração de Utilizadores'

//



Estrutura



- A ter em conta inicialmente:
 - Distinção inicial de classes necessárias: users (vários), menus (vários), enum, repository e utilidades.
 - Estrutura de pastas para melhor legibilidade de código: pastas que diferenciam Data, Enums, Interfaces, Models e Utilities.
 - Organização geral do fluxo de operações e interações.

01

Princípios OOP & SOLID

Código organizado e robusto para manutenção e evolução ao longo do tempo para desenvolvimento de qualidade. Ter em conta:

SOLID: cinco princípios - responsabilidade única, aberto/fechado, substituição de Liskov, segregação de interfaces, e inversão de dependência.

OOP: quatro pilares - abstração, encapsulamento, herança e polimorfismo, que facilita a organização, reutilização e manutenção do código.

Simplicidade

02

Fácil de entender, manter e modificar. Código simples, direto e claro, evitando complexidade desnecessária.

03

Flexibilidade

Que seja possível adaptar a mudanças sem exigir grandes modificações. Aplicação flexível para incluir novos requisitos e funcionalidades com facilidade.

Agilidade

04

Respeitar os limites de cada membro e encontrar um caminho ágil na integração de todos os membros.



Interfaces e Herança

Para aumentar a flexibilidade e a robustez, permitindo reutilização de código e capacidade de definir comportamentos comuns entre classes.

Classes e Objetos

Pode representar algo visível e real ou até a representação de algo que não seja tangível, com atributos/características e métodos/comportamento.

Métodos e Construtores

- Métodos: múltiplos parâmetros de entrada possíveis e no máximo um tipo de retorno ou void. Estes podem ser variados, como por exemplo, usar o overriding de métodos.
- Construtores: pode haver múltiplos construtores explicitamente (com assinaturas diferentes) assim como usos

Modificadores de acesso usados:

private: só acedido na classe onde foi declarado - proteção

- Motivação: encapsulamento e abstração

public: acedido a partir de qualquer classe (inclusive fora do assembly onde foi definido)

- Motivação: uso de interfaces e reutilização de código

internal: acedido a partir de qualquer classe dentro do assembly (inclusive fora do assembly onde foi definido).

protected: só pode ser acedido a partir da classe onde foi declarado e a partir de classes derivadas (inclusive fora do assembly onde foi definido)

Estrutura

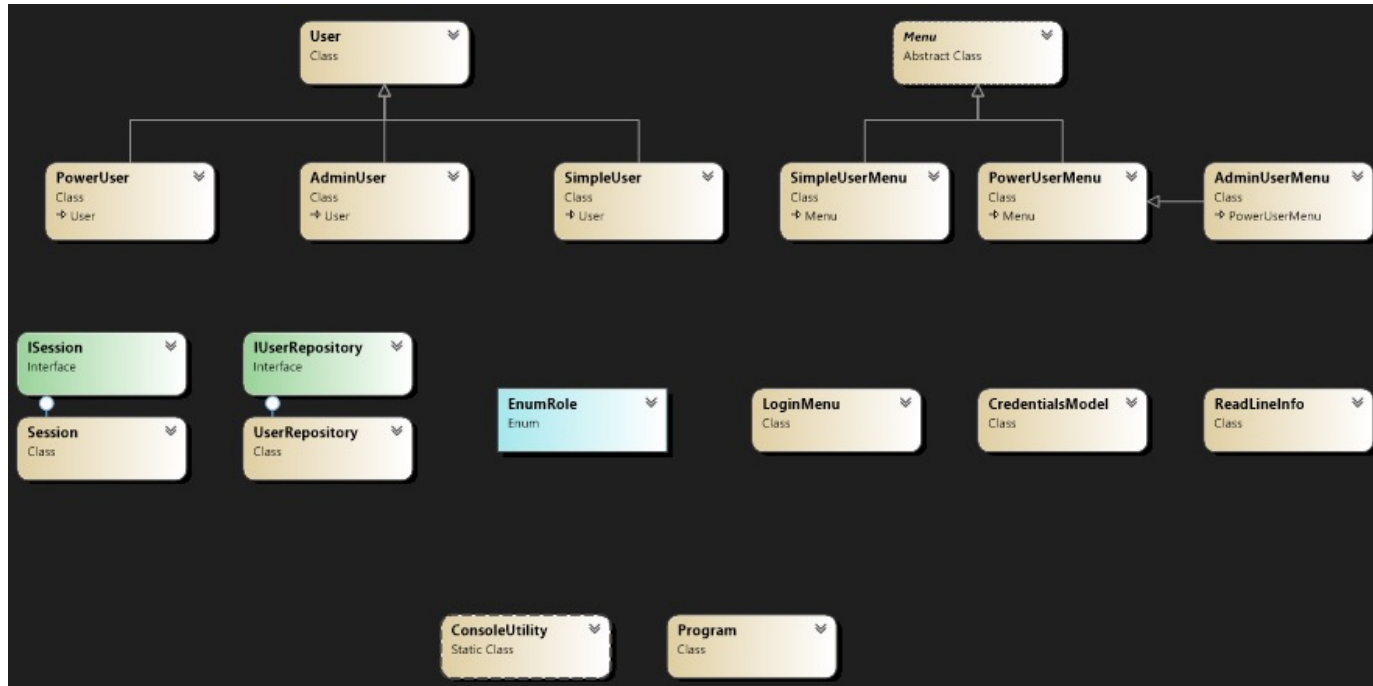


diagrama de classes

Lógica de decisão

- **Users**

- As classes de users partilham naturalmente propriedades e comportamentos iguais e, por isso, foi escolhida a herança como abordagem de implementação. Para reutilização de código, as classes Admin, PowerUser e SimpleUser descendem da classe abstrata User.
- As propriedades dos mesmos estão públicas, no entanto há o uso do modificador de acesso privado nos sets para que a definição dos valores do mesmo seja apenas permitida dentro da própria classe. Também inclui validações dentro das próprias propriedades para que se tentarem criar um objeto com dados inválidos não seja permissível lançando uma exceção.
- As propriedades têm virtual como inheritance modifier para atender ao princípio aberto-fechado, aberto para extensão e fechado para alteração (se os requisitos de validação forem alterados é possível estender e assim facilmente cumpri-los).
- O método SetId é apenas para ser possível definir o Id quando se adiciona um novo user ao repositório. Assim, há a consistência nos id sequenciais sem colocar em causa outros objetos diferentes.
- O método Update tem o propósito a alteração das propriedades da classe. Este método é necessário visto que para definir o valor das propriedades é apenas possível pelo construtor e nem sempre queremos instanciar e criar um novo objeto dessa forma a ocupar um novo espaço em memória. Assim, conseguimos reutilizar o mesmo objeto.

- **Menus**

- A criação de classes dedicadas apenas à manipulação de menus pretende atender ao princípio de responsabilidade única em que cada classe é responsável pela implementação de funcionalidades do seu contexto. Como existem algumas funcionalidades partilhadas entre os mesmos, há uso novamente do princípio aberto/fechado em que há uma classe menu abstrata (não instanciável) que permite ser herdada e tem presente métodos específicos com modificador virtual (que fazem validações e as mesmas podem evoluir ou mudar de critérios por isso permite estender). Cada classe menu é pensada de forma a transmitir funcionalidade dependendo do role do user. O uso do modificador protected pretende encapsular para que só uma classe derivada possa usar o método. Além disto, há o uso do construtor e variáveis para armazenar/aceder a informação necessária com declaração das variáveis privadas e só de leitura que são inicializadas no construtor. Assim, armazena valores da interface IUserRepository, da interface ISession, e um tuple contendo duas cores do console (background e foreground) para cumprir com os critérios do enunciado.
- O menu Login, por sua vez, é uma classe com comportamento independente diferente das restantes que apenas tem funcionalidades do seu contexto.

Lógica de decisão

- **UserRepository**

- o O objetivo da criação do user repository foi, mais uma vez, para cumprir o princípio da responsabilidade única em que a classe apenas se foca no armazenamento de dados do utilizador e seus comportamentos, ou seja, leitura e escrita de dados.
- o Esta classe usa uma interface, o que faz com que quem a use não necessite de perceber como está implementada. Desta forma, se no futuro se escolher guardar estes dados numa base de dados apenas é necessário mudar a implementação da interface.
- o Esta interface vai permitir que quem a usa atender ao princípio de inversão de dependência visto que vai ficar dependente da interface e não da implementação.
- o A classe user repository também tem lógica de encapsulamento tendo métodos públicos para poderem ser acedidos externamente mantendo a informação da própria classe privada. Os métodos que manipulam dados usam Linq.

- **CredentialsModel**

- o Esta classe foi criada no seu contexto para ler e guardar valores das credenciais, em que o construtor atribui valor às propriedades pelos seus parâmetros de entrada. Assim, pretendo encapsular a atribuição de valor, guardar os dados e manipular apenas quando o construtor é chamado (tentativa única de credenciais). Estas circunstâncias foram também pensadas com a alternativa de um tuple, porém assim há um controlo das propriedades e formas de uso mais organizado e seguro.

- **Session**

- o O objetivo da criação de session foi para cumprir o princípio da responsabilidade única em que a classe apenas se foca na lógica de gestão de sessão. Há também encapsulamento com modificadores privados das propriedades na atribuição de valor.
- o Esta classe usa uma interface, o que faz com que quem a use não necessite de perceber como está implementada. Desta forma, se no futuro existir uma lógica da funcionalidade de início de sessão diferente apenas é necessário mudar a implementação da interface.
- o Esta interface vai permitir que quem a usa atender ao princípio de inversão de dependência visto que vai ficar dependente da interface e não da implementação.

Lógica de decisão

- **ReadLineInfo**

- o Ao pensar na estrutura desta aplicação tenho em conta a experiência do utilizador. Então, foi criada a funcionalidade de permitir em qualquer momento voltar atrás em qualquer ação. Para isso foi necessário criar uma classe utilitária para conseguir registar o input do utilizador a cada tecla premida e foi usada a classe ReadLineInfo para ter o controlo do que foi premido.

- **ConsoleUtiliy**

- o Esta classe foi criada para facilitar o desenvolvimento desta aplicação.
- o Esta classe tem 3 métodos de controlo de input que usam a classe mencionada anterior para controlo do que foi premido:
 1. Um método que faz uma pergunta e recebe a resposta positiva ou negativa.
 2. Um método que regista com um ciclo cada tecla premida pelo utilizador. Se o utilizador premir a tecla escape é porque quer voltar par trás, se premir a tecla enter é porque finalizou de escrever o desejado, se premiu a tecla backspace é porque quer apagar a última tecla com caracter premida, e se só carregar em teclas com carateres estes mesmos são registados.
 3. Um método que tem um ciclo que regista cada tecla premida à semelhança do método do ponto anterior, mas este não mostra a tecla premida e substitui por uma estrela. Este método é usado para escrever passwords.
- o Esta classe tem 3 métodos informativos:
 1. Informação de erro
 2. Informação de sucesso
 3. Apenas informação
- o Esta classe tem outros 3 métodos para construir tabelas:
 1. Uma tabela para mostrar o utilizador pesquisado.
 2. Uma tabela para mostrar os utilizadores.
 3. Uma tabela em formulário para mostrar as alterações do utilizador.

Exemplos de objetivos e princípios aplicados:

Abstração: princípio de esconder os detalhes complexos de implementação e expor apenas os aspetos essenciais de um objeto, facilitando o uso e a compreensão. Por exemplo, a classe abstrata `User` é uma abstração que representa um utilizador genérico. Ela define propriedades e métodos comuns e permite que as classes concretas implementem os detalhes específicos. As interfaces `IUserRepository` e `ISession` são outra forma de abstração sem expor detalhes de implementação.

Encapsulamento: As classes encapsulam os dados e comportamentos específicos dos diferentes tipos de menus/utilizadores. Limita o acesso a variáveis e lógica interna do objeto para não serem acedidas de forma descuidada, impossibilitando também a manipulação de dados de uma classe por outra, ex. com membros internos como privados e os de destino externo como públicos. Por exemplo, usar modificadores de acesso privado nas propriedades.

Herança/Especialização: permite que uma classe derive de outra, herdando os seus atributos e métodos. Isto promove a reutilização de código e a especialização de comportamentos. Por exemplo, as classes que herdam da classe `User`, reutilizam código comum (como propriedades e métodos) permitindo a especialização de comportamentos específicos para cada tipo de utilizador.

Polimorfismo: capacidade de tratar objetos de diferentes classes de forma uniforme, desde que essas classes partilhem a mesma interface ou classe base. Isso permite que o mesmo código funcione com diferentes tipos de objetos.

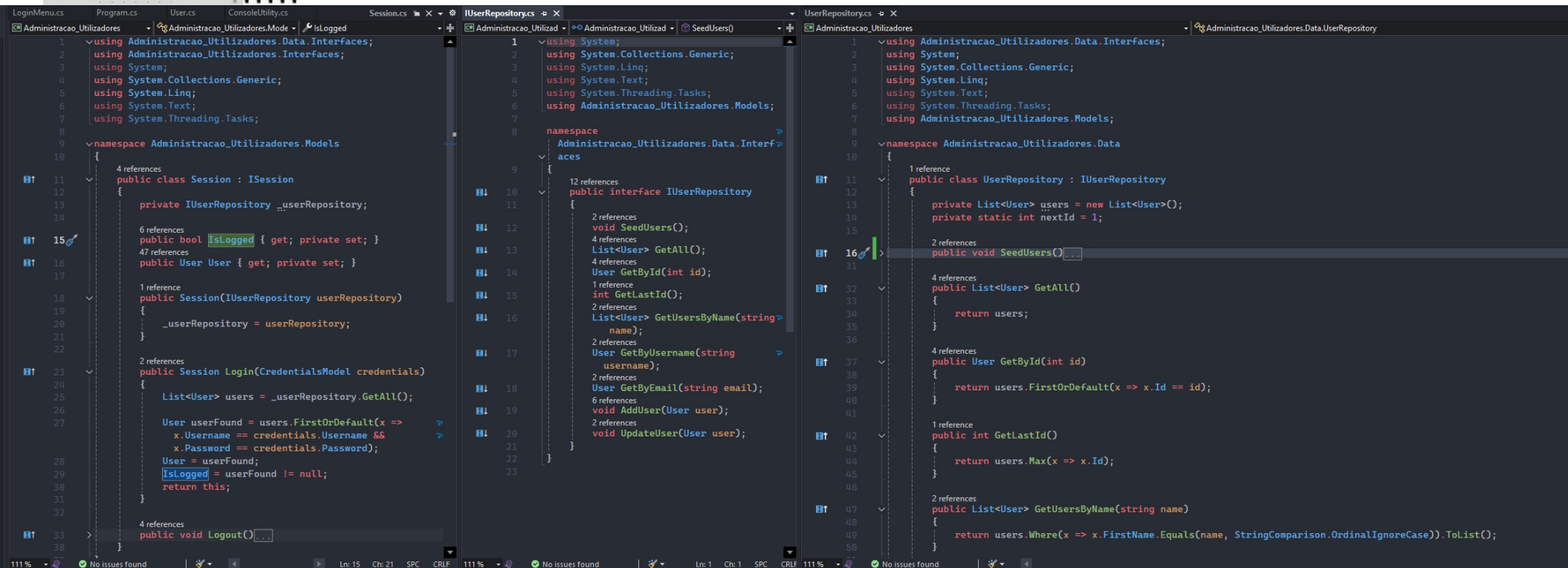
Princípio da Responsabilidade única (SRP): Cada classe tem uma única responsabilidade ou motivo para mudar. Por exemplo, a classe `UserRepository` é responsável apenas por armazenamento, leitura e escrita dos dados dos utilizadores. Não se envolve noutras responsabilidades, como a lógica de sessão ou a manipulação de menus.

Princípio do Aberto/Fechado (OCP): As classes estão abertas para extensão, mas fechadas para modificação. Por exemplo, as propriedades da classe `User` são marcadas como `virtual`, permitindo que classes derivadas possam estender ou modificar o comportamento dessas propriedades sem precisar alterar a classe base `User`.

Princípio da Segregação de Interfaces (ISP): este projeto não explora muito este princípio, devido ao contexto apresentado e lógica de decisão.

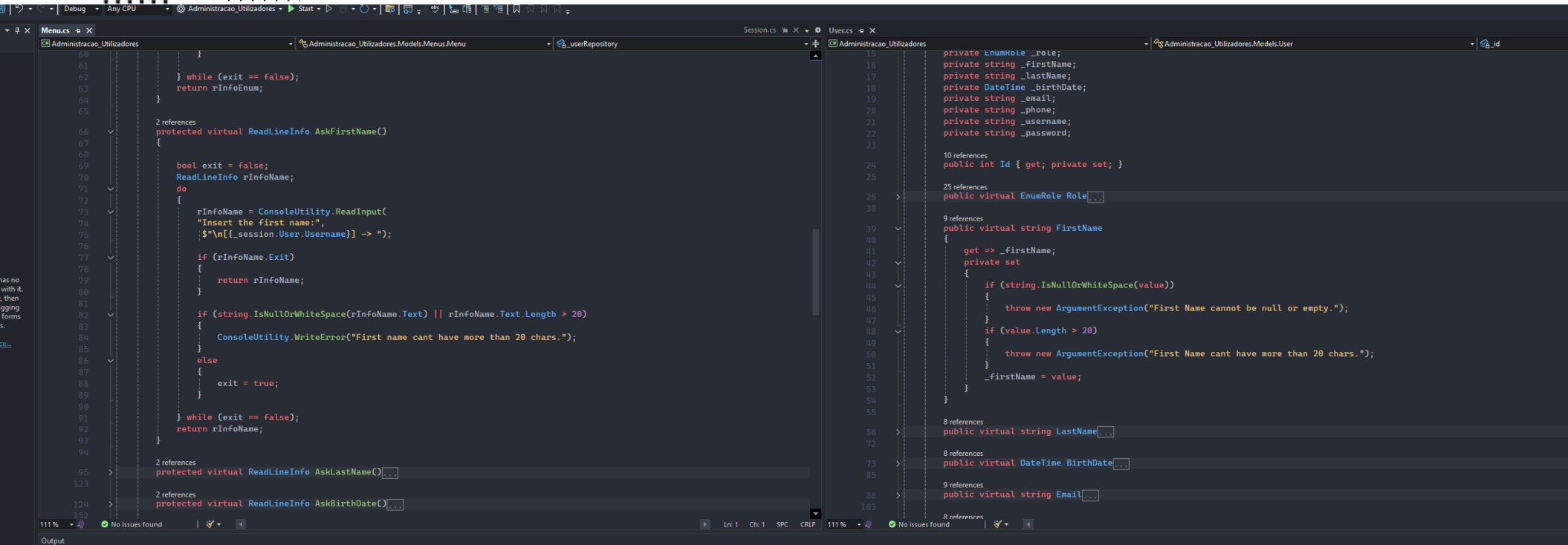
Princípio da Inversão de Dependências (DIP): As classes dependem de abstrações (interfaces) e não de implementações concretas, permite que a lógica subjacente seja alterada sem afetar o código que consome essas interfaces (exemplo: `ISession`).

Princípio de Substituição de Liskov (LSP): Os objetos do tipo de uma interface devem poder ser substituídos por objetos de classes que implementam a interface sem alterar a funcionalidade. Por exemplo, qualquer classe que implementa `IUserRepository` deve poder ser usada onde `IUserRepository` é esperado.



Esta imagem tem 3 exemplos:

- 1 – Responsabilidade única de gerir sessão e responsabilidade única de armazenamento de dados.
- 2 – Inversão de dependência - Session depende de IUserRepository em vez de UserRepository.
- 3 – Session e UserRepository estão a encapsular informação que só pode ser alterada a partir da própria classe.



Exemplos do principio aberto/ fechado: estes métodos podem ser estendidos se necessário, mesmo que de momento nesta aplicação não seja preciso permitir isso.

```
else
{
    Console.Clear();
    switch (session.User.Role)
    {
        case Enums.EnumRole.Admin:
            Menu adminUserMenu = new AdminUserMenu(userRepository, session, (ConsoleColor.Black, ConsoleColor.Magenta));
            adminUserMenu.MainMenu();
            break;

        case Enums.EnumRole.PowerUser:
            Menu powerUserMenu = new PowerUserMenu(userRepository, session, (ConsoleColor.Black, ConsoleColor.Blue));
            powerUserMenu.MainMenu();
            break;

        case Enums.EnumRole.SimpleUser:
            Menu simpleUserMenu = new SimpleUserMenu(userRepository, session, (ConsoleColor.Black, ConsoleColor.DarkGreen));
            simpleUserMenu.MainMenu();
            break;

        default:
            break;
    }
}
```

Exemplo de polimorfismo: a classe Menu tem um método chamado MainMenu que dependendo do tipo de instância tem um comportamento diferente.

Repositório GitHub

- Criar um repositório com o git ignore;
- Incluir ficheiro README;
- Adicionar o projeto dentro do repositório;
- Garantir que o commit, push e pull são feitos convenientemente às necessidades (além de branches serem usadas se necessário)

O repositório é partilhado via convite para fácil acesso e partilha do mesmo.

Conclusão e Próximos Passos

Este relatório refere aspetos da lógica de pensamento e estruturação no desenvolvimento das duas aplicações, apesar de eu ser relativamente iniciante na área, procuro constantemente aplicar as melhores práticas disponíveis e expandir os meus conhecimentos para alcançar os objetivos propostos. Tive bastante esforço para tentar aproveitar ao máximo os recursos que foram dados e para aprender mais ao longo do processo.

Idealmente, a testagem deveria ser realizada posteriormente ao desenvolvimento (visto não ter acontecido com formalidade de metodologias de teste agilmente no decorrer do projeto), permitindo assim uma validação mais rigorosa e independente das funcionalidades implementadas.

De seguida, há a demonstração das aplicações para aprovação das funcionalidades com as intenções do negócio e posterior integração das mesmas com o objetivo de gerar resultados relevantes e fornecer informações úteis ao negócio, alinhadas às suas perspetivas e características específicas.

