Relatório do trabalho da disciplina de POO

Relatório IPCAbitA - POO

António Jorge Magalhães da Rocha – a26052

Adelino Daniel da Rocha Vilaça – a16939

LESI-PL

Novembro de 2023

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Afirmo por minha honra que não recebi qualquer apoio não autorizado na realização deste trabalho prático. Afirmo igualmente que não copiei qualquer material de livro, artigo, documento web ou de qualquer outra fonte exceto onde a origem estiver expressamente citada.

António Jorge Magalhães da Rocha - 26052

Adelino Daniel da Rocha Vilaça - 16939

Índice

Lista de Tabelas

Lista de Figuras

1. Introdução

O objetivo deste trabalho é implementar um programa em C# para servir como plataforma e representação do projeto IPCAbitA, referente ao aluguer de quartos para alunos do IPCA. Irá ser desenvolvido usando Classes como Pessoa, Aluno, Senhorio, Admin, etc para melhorar a organização e acesso aos dados correspondentes. Irá também usar listas e ficheiros CSV como estruturas de dados do projeto, inicialmente para teste de armazenamento de dados, e numa outra fase SQL com interfaces em Windows Forms.

Este trabalho seguirá uma metodologia conforme apreendida em aula que inclui desde a definição de requisitos até a documentação do trabalho.

2. Classes

2.1. Classe Pessoa

Representação de um Utilizador com as propriedades base de uma Pessoa e métodos para verificação do Login da mesma.

```
1 referência
public interface Login
{
    3 referências
    bool Login(string email, string password);
}

7 referências
public class Pessoa : Login
{
    7 referências
    public string Nome { get; set; }
    4 referências
    public DateTime DataNascimento { get; set; }
    9 referências
    public string Email { get; set; }
    6 referências
    public string Password { get; set; }
    5 referências
    public bool IsBlocked { get; set; }

    3 referências
    public Pessoa(string nome, DateTime dataNascimento, string email, string password)
    {
        Nome = nome;
        DataNascimento = dataNascimento;
        Email = email;
        Password = password;
    }

    3 referências
    public virtual bool Login(string email, string password)
    {
        if (IsBlocked)
        {
            Console.WriteLine("Conta bloqueada!");
            return false;
        }
        else
        {
            return Email == email && Password == password;
        }
    }

}
```

2.2. Classe Aluno

Representação de um Estudante tendo propriedades herdadas da Classe Pessoa com propriedades em acréscimo como AlunoID, Curso, etc.

```
18 referências
public class Aluno : Pessoa
{
    List<Aluno> listaAlunos = new List<Aluno>();

    private static int proximoAlunoId = 16000;

    2 referências
    public int AlunoId { get; set; }
    1 referência
    public string Curso { get; set; }
    1 referência
    public string Instituicao { get; set; }


    2 referências
    public Aluno(string nome, DateTime dataNascimento, string email, string password, string curso, string instituicao)
        : base(nome, dataNascimento, email, password)
    {
        AlunoId = GerarAlunoId();
        Curso = curso;
        Instituicao = instituicao;
    }
```

2.2.1. Classe Aluno – *Método GerarAlunoID*

Método GerarAlunoID para gerar um ID único novo, sempre que for feito um Registo de um Aluno.

```
//gerar id incrementável a partir de 16000 até 26999
1 referência
private int GerarAlunoId()
{
    int novoAlunoId = proximoAlunoId;
    proximoAlunoId++;

    if (proximoAlunoId > 27000)
    {
        throw new Exception("Limite de IDs de aluno atingido.");
    }

    return novoAlunoId;
}
```

2.2.2. Classe Aluno – *Método RegistarNovoAluno*

Método RegistarNovoAluno para registar um novo aluno inquilino guardando num ficheiro CSV, usando o Método GuardaAluno, os dados da lista correspondente.

```
//registar novo aluno
1 referência
public static void RegistarNovoAluno(List<Aluno> listaAlunos, string nome, DateTime dataNascimento, string curso, string instituicao,
    string email, string password)
{
    Aluno novoAluno = new Aluno(nome, dataNascimento, curso, instituicao, email, password);
    listaAlunos.Add(novoAluno);
    Console.WriteLine("Aluno registado com sucesso!");

    GuardaAluno(listaAlunos, "dadosaluno.csv");

}
```

2.2.3. Classe Aluno – *Método GuardaAluno*

Método GuardaAluno para guardar dados da lista num ficheiro CSV.

```
1 referência
public static void GuardaAluno(List<Aluno> listaAlunos, string fileName)
{
    string filePath = Path.Combine(Directory.GetCurrentDirectory(), "../Dados", fileName);
    List<string> dadosAluno = new List<string>();

    foreach (var aluno in listaAlunos)
    {
        string dado = $"{aluno.AlunoId},{aluno.Nome},{aluno.DataNascimento},{aluno.Email},{aluno.Password},{aluno.IsBlocked}\n";
        dadosAluno.Add(dado);
    }

    File.AppendAllLines(fileName, dadosAluno);
}
```

2.2.4. Classe Aluno – *Método ValidarLoginAluno*

Método ValidarLoginAluno para validar Login usando email e password guardados no ficheiro CSV.

```
//validar login pelo csv

1 referência
public static bool ValidarLoginAluno(string email, string password)
{
    List<Aluno> listaAlunos = LerAlunos("dadosaluno.csv");

    foreach (var aluno in listaAlunos)
    {
        if (aluno.Email == email && aluno.Password == password)
        {
            return true;
        }
    }

    return false;
}
```

2.2.5. Classe Aluno – *Método LerAlunos*

Método LerAlunos para mostrar lista com os dados guardados do ficheiro CSV.

```
//mostrar lista
1 referência
public static List<Aluno> LerAlunos(string fileName)
{
    List<Aluno> lerAlunos = new List<Aluno>();

    string filePath = Path.Combine(Directory.GetCurrentDirectory(), "../Dados", fileName);
    string[] lines = File.ReadAllLines(filePath);

    foreach (string line in lines)
    {
        string[] fields = line.Split(',');

        Aluno aluno = new Aluno(
            fields[1],
            DateTime.Parse(fields[2]),
            fields[5],
            fields[6],
            fields[3],
            fields[4]
        );

        lerAlunos.Add(aluno);
    }

    return lerAlunos;
}
```

2.3. Classe Senhorio

Representação de um Senhorio tendo propriedades herdadas da Classe Pessoa com propriedades em acréscimo como SenhorioId.

```
16 referências
public class Senhorio : Pessoa
{
    List<Senhorio> listaSenhorios = new List<Senhorio>();

    private static int proximoSenhorioId = 1;

    1 referência
    public int SenhorioId { get; set; }


    2 referências
    public Senhorio(string nome, DateTime dataNascimento, string email, string password) : base(nome, dataNascimento, email, password)
    {
        SenhorioId = GerarSenhorioId();
    }
```

2.3.1. Classe Senhorio - *Método GerarSenhorioId*

Método GerarSenhorioId para gerar um ID único novo, sempre que for feito um Registo de um Senhorio.

```
//gerar id incrementável
1 referência
private int GerarSenhorioId()
{
    int novoSenhorioId = proximoSenhorioId;
    proximoSenhorioId++;

    return novoSenhorioId;
}
```

2.3.2. Classe Senhorio - *Método RegistarNovoSenhorio*

Método RegistarNovoSenhorio para registar um novo Senhorio guardando num ficheiro CSV, usando o Método GuardaSenhorio, a lista de dados correspondente ao mesmo.

```
//registar novo senhorio
1 referência
public static void RegistarNovoSenhorio(List<Senhorio> listaSenhorios, string nome, DateTime dataNascimento, string email, string password)
{
    Senhorio novoSenhorio = new Senhorio(nome, dataNascimento, email, password);
    listaSenhorios.Add(novoSenhorio);
    Console.WriteLine("Senhorio registado com sucesso!");

    GuardaSenhorio(listaSenhorios, "../Dados/dadossenhorio.csv");
}
```

2.3.3. Classe Senhorio - *Método GuardaSenhorio*

Método GuardaSenhorio para guardar dados da lista num ficheiro CSV.

```
1 referência
public static void GuardaSenhorio(List<Senhorio> listaSenhorios, string fileName)
{
    string[] dadosSenhorio = new string[listaSenhorios.Count];

    for (int i = 0; i < listaSenhorios.Count; i++)
    {
        dadosSenhorio[i] = listaSenhorios[i].Nome + ";" + listaSenhorios[i].DataNascimento + ";" + listaSenhorios[i].Email +
            ";" + listaSenhorios[i].Password;
    }

    File.WriteAllLines(fileName, dadosSenhorio);
}
```

2.3.4. Classe Senhorio - *Método ValidarLoginSenhorio*

Método ValidarLoginSenhorio para validar Login usando email e password guardados no ficheiro CSV.

```
//validar login do Senhorio
0 referências
public static bool ValidarLoginSenhorio(string email, string password)
{
    List<Senhorio> listaSenhorios = LerSenhorios("dadossenhorio.csv");

    foreach (var senhorio in listaSenhorios)
    {
        if (senhorio.Email == email && senhorio.Password == password)
        {
            return true;
        }
    }

    return false;
}
```

2.3.5. Classe Aluno – *Método LerSenhorios*

Método LerSenhorios para mostrar lista com os dados guardados do ficheiro CSV.

```
//mostrar lista
1 referência
public static List<Senhorio> LerSenhorios(string fileName)
{
    List<Senhorio> lerSenhorios = new List<Senhorio>();

    string filePath = Path.Combine(Directory.GetCurrentDirectory(), "../Dados/", fileName);
    string[] lines = File.ReadAllLines(filePath);

    foreach (string line in lines)
    {
        string[] fields = line.Split(',');

        Senhorio senhorio = new Senhorio(
            fields[1],
            DateTime.Parse(fields[2]),
            fields[3],
            fields[4]
        );

        lerSenhorios.Add(senhorio);
    }

    return lerSenhorios;
}
```

2.4. Classe Admin

Representação de um Admin tendo propriedades herdadas da Classe Pessoa como propriedades estáticas/predefinidas.

```
12 referências
public class Admin : Pessoa
{
    private static int proximoAdminId = 3;
    1 referência
    public int AdminId { get; private set; }


    private static List<Admin> infoadministrador = new List<Admin>
    {
        new Admin("Adelino Daniel da Rocha Vilaça", new DateTime(1998, 5, 3), "a16939@alunos.ipca.pt", "Admin1password#"),
        new Admin("António Jorge Magalhães da Rocha", new DateTime(1993, 3, 19), "a26052@alunos.ipca.pt", "Admin2password#")
        //adicionar + caso necessário
    };

    3 referências
    private Admin(string nome, DateTime dataNascimento, string email, string password) : base(nome, dataNascimento, email, password)
    {
        AdminId = GerarAdminId();
    }
```

2.4.1. Classe Admin - *Método GerarAdminId*

Método GerarAdminId para gerar um ID único novo, sempre que for feito um Registo de um Admin.

```
//gerar id incrementável
1 referência
private int GerarAdminId()
{
    int novoAdminId = proximoAdminId;
    proximoAdminId++;

    return novoAdminId;
}
```

2.4.2. Classe Admin - *Método RegistarNovoAdmin*

Método RegistarNovoAdmin para registar um novo Admin guardando num ficheiro CSV, usando o Método GuardaAdmin, a lista de dados correspondente ao mesmo.

```
//registar novo admin
0 referências
public static void RegistarNovoAdmin(List<Admin> infoadministrador, string nome, DateTime dataNascimento, string email, string password)
{
    Admin novoAdmin = new Admin(nome, dataNascimento, email, password);
    infoadministrador.Add(novoAdmin);
    Console.WriteLine("Admin registado com sucesso!");

    GuardaAdmin(infoadministrador, "../Dados/dadosadmin.csv");
}
```

2.4.3. Classe Admin - *Método GuardaAdmin*

Método GuardaAdmin para guardar dados da lista num ficheiro CSV.

```
//guardar dados do admin
1 referência
public static void GuardaAdmin(List<Admin> infoadministrador, string ficheiro)
{
    string[] dadosAdmin = new string[infoadministrador.Count];

    for (int i = 0; i < infoadministrador.Count; i++)
    {
        dadosAdmin[i] = infoadministrador[i].Nome + ";" + infoadministrador[i].DataNascimento + ";" + infoadministrador[i].Email +
            ";" + infoadministrador[i].Password;
    }

    File.WriteAllLines(ficheiro, dadosAdmin);
}
```

2.4.4. Classe Admin - *Método BlockAluno/BlockSenhorio*

Método para mudar status de conta de um Aluno/Senhorio permitindo um Admin bloquear um utilizador da plataforma.

```
//bloquear acesso Aluno
0 referências
public void BlockAluno(Aluno aluno)
{
    aluno.IsBlocked = true;
}

//bloquear acesso Senhorio
0 referências
public void BlockSenhorio(Senhorio senhorio)
{
    senhorio.IsBlocked = true;
}
```

2.5. Classe Pagamento

Representação da Classe Pagamento tendo como propriedades o PagamentoId, Valor, Descrição, etc.

```
1 referência
public class Pagamento
{
    1 referência
    public int PagamentoId { get; set; }
    1 referência
    public decimal Valor { get; set; }
    1 referência
    public DateTime DataPagamento { get; set; }
    1 referência
    public string Descricao { get; set; }

    // Constructor
    0 referências
    public Pagamento(int pagamentoId, decimal valor, DateTime dataPagamento, string descricao)
    {
        PagamentoId = pagamentoId;
        Valor = valor;
        DataPagamento = dataPagamento;
        Descricao = descricao;
    }
}
```

2.6. Classe Quarto

Representação da Classe Quarto tendo como propriedades o QuartoId, Número do Quarto e Capacidade.

```
9 referências
public class Quarto
{
    List<Quarto> listaQuartos = new List<Quarto>();

    2 referências
    public int QuartoId { get; set; }
    2 referências
    public string Numero { get; set; }
    2 referências
    public int Capacidade { get; set; }

    1 referência
    public Quarto(int quartoId, string numero, int capacidade)
    {
        QuartoId = quartoId;
        Numero = numero;
        Capacidade = capacidade;
    }
```

2.6.1. Classe Quarto - *Método RegistarNovoQuarto*

Método RegistarNovoQuarto para registar um novo Quarto guardando num ficheiro CSV, usando o Método GuardaQuarto, a lista de dados correspondente ao mesmo.

```
0 referências
public static void RegistarNovoQuarto(List<Quarto> listaQuartos, string numero, int capacidade)
{
    int quartoId = listaQuartos.Count + 1;
    Quarto quarto = new Quarto(quartoId, numero, capacidade);
    listaQuartos.Add(quarto);
}
```

2.6.2. Classe Quarto - *Método GuardaQuarto*

Método GuardaQuarto para guardar dados da lista num ficheiro CSV.

```
0 referências
public static void GuardaQuarto(List<Quarto> listaQuartos, string fileName)
{
    string[] dadosSenhorio = new string[listaQuartos.Count];

    for (int i = 0; i < listaQuartos.Count; i++)
    {
        dadosSenhorio[i] = listaQuartos[i].QuartoId + ";" + listaQuartos[i].Numero + ";" + listaQuartos[i].Capacidade;
    }

    File.WriteAllLines(fileName, dadosSenhorio);
}
```

2.7.Classe Serviço

Representação da Classe Serviço tendo como propriedades o ServicoId, Nome, Preço, etc.

```
1 referência
public class Servico
{
    1 referência
    public int ServicoId { get; set; }
    1 referência
    public string Nome { get; set; }
    1 referência
    public decimal Preco { get; set; }
    1 referência
    public string Descricao { get; set; }


    0 referências
    public Servico(int servicoId, string nome, decimal preco, string descricao)
    {
        ServicoId = servicoId;
        Nome = nome;
        Preco = preco;
        Descricao = descricao;
    }
```

3. Estruturas de Dados

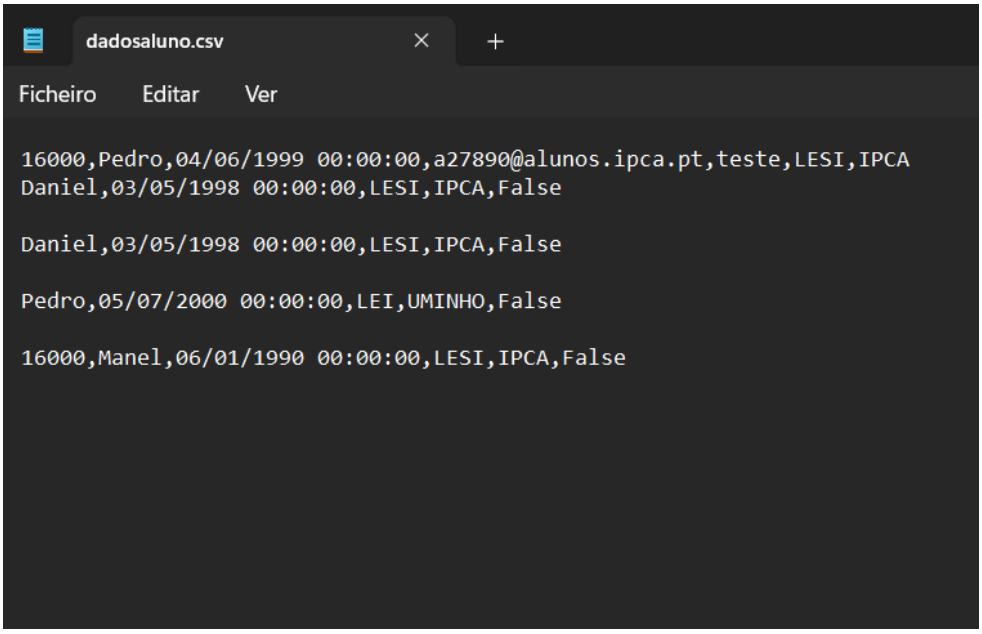
3.1. Listas

Como representado em cima nas Classes, iremos usar Listas como um dos tipos de estruturas de dados para organização e acessibilidade de dados necessários para o bom funcionamento da plataforma.

```
List<Aluno> listaAlunos = new List<Aluno>();
List<Senhorio> listaSenhorios = new List<Senhorio>();
List<Admin> listaAdmins = new List<Admin>();
List<Quarto> listaQuartos = new List<Quarto>();
```

3.2. Ficheiros CSV

Outro tipo de estrutura de dados utilizada seriam os ficheiros CSV, de momento, para testes de armazenamento e carregamento de dados até serem implementados em bases de dados como MySQL.



4. Conclusão

Em resumo, este projeto foi desenvolvido para POO mas dá continuidade aos processos de negócio e modelação de software das disciplinas de PES e AMS, tendo sido desenvolvidos além plataforma em C#, os requisitos e funcionalidades da mesma.

O projeto na sua totalidade seria composto por um menu de Logins, validação dos mesmos, apresentação do menu correspondente à entidade em causa, funcionalidades de acordo com a entidade, etc.

Na entrega da 2ª fase, é seguro que essas mesmas funcionalidades estarão finalizadas, tornando-se assim num projeto viável.