Relatório de Projeto

Programação Orientada a Objetos

Tiago Vale (27675)

Instituto Politécnico do Cávado e do Ave (IPCA)
Curso: Engenharia de Sistemas Informáticos
14 de novembro de 2024

Índice

Capítulo 1

Resumo

O projeto em questão visa aplicar os conhecimentos adquiridos nas unidades curriculares "Programação Orientada a Objetos"e "Programação Imperativa", utilizando exclusivamente a linguagem de programação C#. O objetivo principal deste projeto é resolver problemas, implementar funcionalidades e desenvolver soluções relacionadas a um sistema de gestão de alojamentos turísticos.

Neste contexto, o foco está no desenvolvimento de um sistema em Windows Forms. Utiliza-se a abordagem de herança, interfaces, e em particular o carregamento e guardamento de dados em "json", para representar dados, possibilitando uma maior flexibilidade e eficiência. Através deste projeto, espera-se não apenas aplicar os conceitos teóricos aprendidos em sala de aula, mas também desenvolver habilidades práticas na implementação desses conceitos. Além disso, o projeto oferece a oportunidade de explorar técnicas de programação em C#, incluindo boas práticas de codificação.

Ao abordar desafios relacionados à própria implementação do sistema, o projeto consegue proporcionar uma oportunidade valiosa para aprimorar a resolução de problemas e desenvolver soluções em um ambiente de programação. Este relatório detalhará as etapas do processo de desenvolvimento, as decisões tomadas e as análises realizadas durante a implementação, oferecendo uma visão abrangente do trabalho realizado e das contribuições alcançadas.

Capítulo 2

Introdução

Este projeto tem como objetivo a implementação de um sistema para a gestão de alojamentos turísticos, utilizando os princípios da "Programação Orientada a Objetos". O sistema foi definido para gerir as reservas, pagamentos, check-ins, clientes e alojamentos, deste modo capaz de proporcionar uma solução prática e eficiente.

O presente relatório descreve a definição do sistema, as funcionalidades implementadas e a análise do código desenvolvido ao longo do processo. O projeto é dividido em duas fases, sendo que na fase número um pretende-se: o desenvolvimento inicial do sistema(models e enums) e a elaboração deste relatório, que documenta as etapas, decisões e resultados alcançados.

Capítulo 3

Âmbito do Projeto

O objetivo do projeto é o desenvolvimento de um sistema de gestão de alojamentos turísticos, utilizando a programação orientada a objetos. O sistema tem como base dois tipos de utulizadores,ou seja, um determinado admistrador que contém área de gestão com determinados tipos de gestão, de reservas, de pagamentos, de check-ins, de alojamentos, tendo como base diversas funcionalidades como é o caso de adicionar , remover, atualziar, entre outras. Contudo é possível expandir eventualmente no futuro para um novo e outro usuário, um determinado cliente que tem a possibilidade de visualizar alojamentos possíveis(imagens ou/e descrição), selecionar e realizar reserva com a sucessiva confirmação e respetivo pagamento, tudo isto interligado numa interface de usuário(cliente ou admistrador) construída com Windows Forms.

A linguagem de programação C# permite um boa integração com a plataforma Windows, enquanto o armazenamento de dados é realizado utilizando o formato "JSON", ou seja , leitura e escrita dos dados. O sistema foi projetado para aplicar conceitos essenciais da programação orientada a objetos, como encapsulamento, herança, interfaces, com a capacidade de permitir futuras modificações e extensões.

Programação Orientada a Objetos e C#

A *Programação Orientada a Objetos* (POO) é uma abordagem de programação que tem como base a criação de objetos, que são unidades que combinam dados e funções relacionadas. No C#, a POO é um conceito essencial, pois ela ajuda a organizar o código, além de permitir a criação de sistemas mais modulares, ou seja, estruturados e fáceis de manter.

A seguir, destacam-se algumas características principais da programação orientada a objetos no C#:

- **Classes e Objetos**: Em C#, as classes servem como modelos para criar objetos, onde é possível armazenar dados e definir os comportamentos que esses objetos terão.
- **Herança**: A herança permite a criação de novas classes a partir de classes existentes. Ou seja, a nova classe herda as propriedades e métodos da classe base, o que facilita a reutilização do código e a adição de novas funcionalidades sem a necessidade de começar do zero.
- **Polimorfismo**: O polimorfismo permite que a mesma função ou método se comporte de formas diferentes, dependendo do tipo de objeto em que é chamado. Isso contribui para a flexibilidade do código.
- **Encapsulamento**: O encapsulamento é utilizado para esconder os detalhes internos de uma classe, definindo o que é visível e acessível externamente. Isso é feito através de modificadores de acesso como `public`, `private` e `protected`, o que ajuda a proteger os dados e evitar alterações inesperadas.

Esses conceitos tornam o C# uma linguagem de certa forma poderosa, capaz de criar desde programas simples até sistemas mais complexos. O principal benefício de usar esses conceitos é que o código se torna mais organizado, fácil de entender e facilita a reutilização e melhoria contínua no futuro.

Capítulo 4

Desenvolvimento

4.1 Requisitos

Para o desenvolvimento deste projeto, foram utilizados alguns requisitos essenciais para garantir um bom desempenho e facilitar a implementação das funcionalidades.

Primeiramente, a ferramenta principal para codificação foi o *Visual Studio*, sendo utilizadas as versões 2022 e 2010. O *Visual Studio* é um ambiente de desenvolvimento integrado (IDE) robusto, amplamente utilizado para o desenvolvimento em C# e para a criação de aplicações *Windows Forms*, que foi o framework escolhido para este projeto.

Além disso, o projeto foi desenvolvido no sistema operativo *Windows*, que oferece suporte total ao *Visual Studio*, além de ser a plataforma mais indicada para o desenvolvimento de aplicações *Windows Forms*.

Durante o desenvolvimento, foi utilizado o *GitHub* para o versionamento do código. O repositório no GitHub foi fundamental para manter o controlo das alterações, possibilitar a colaboração entre membros da equipa e facilitar a gestão do histórico do código, além de fornecer uma plataforma segura para o armazenamento e backup do projeto.

A escolha do *Windows Forms* foi motivada pela simplicidade na construção de interfaces gráficas de utilizador (UI) e pela integração com o .NET Framework. Este ambiente permite criar aplicações desktop com facilidade, sendo ideal para este tipo de sistema de gestão de alojamentos turísticos.

Por fim, para a documentação do código e a criação de relatório, foi utilizado outras ferramentas pessoais. Isto ajudou a manter o código organizado.

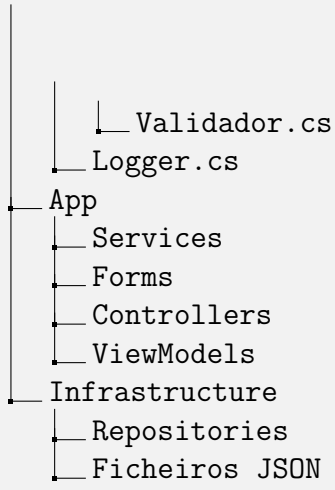
4.2 Arquitetura do Projeto

O projeto será e é estruturado de forma a separar claramente os componentes do código-fonte, dados, documentação, entre outros.

4.2.1 Estrutura do projeto

De momento, de acordo com o proceso de desenvolvimento :

```
GestaoAlojamentosApp
├── Domain
│   ├── Models
│   │   ├── Alojamento.cs
│   │   ├── CheckIn.cs
│   │   ├── Cliente.cs
│   │   ├── Pagamento.cs
│   │   ├── PessoaBase.cs
│   │   ├── Preco.cs
│   │   └── Reserva.cs
│   ├── Enums
│   │   ├── StatusAlojamento.cs
│   │   ├── StatusCheckIn.cs
│   │   ├── StatusPagamento.cs
│   │   ├── StatusReserva.cs
│   │   ├── TipoAlojamento.cs
│   │   ├── TipoPagamento.cs
│   │   └── TipoUtilizador.cs
│   └── Interfaces
│       ├── Repositories
│       │   ├── IRepositoryAlojamento.cs
│       │   ├── IRepositoryCliente.cs
│       │   ├── IRepositoryReserva.cs
│       │   ├── IRepositoryCheckIn.cs
│       │   └── IRepositoryPagamento.cs
│       └── Services
│           ├── IServiceAlojamento.cs
│           ├── IServiceCheckIn.cs
│           ├── IServiceCliente.cs
│           ├── IServicePagamento.cs
│           └── IServiceReserva.cs
└── Utils
    ├── CriarIds.cs
    └── Validacoes
```


Validador.cs
Logger.cs
App
Services
Forms
Controllers
ViewModels
Infrastructure
Repositories
Ficheiros JSON

4.2.2 Descrição da estrutura

O diretório `Domain/` contém as subpastas essenciais para a aplicação de gestão de alojamentos. Abaixo estão os detalhes de cada um:

- `Models/`: Contém as classes essenciais, como:
  - `Alojamento.cs`: Define as características de um alojamento como localização, entre outros.
  - `CheckIn.cs`: Define a classe responsável pelo processo de check-in.
  - `Cliente.cs`: Contém as informações sobre os clientes do sistema (nome, telefone etc.).
  - `Pagamento.cs`: Define a estrutura relacionada aos pagamentos dos clientes.
  - `PessoaBase.cs`: Classe abstrata que contém informações comuns entre pessoas (clientes).
  - `Reserva.cs`: Representa uma reserva feita por um cliente para um alojamento.
- `Enums/`: Contém os ficheiros que definem os valores enumerados usados no sistema para representar os estados e tipos das entidades, como:
  - `StatusAlojamento.cs`: Define os status de um alojamento (disponível, ocupado, manutenção).
  - `StatusCheckIn.cs`: Enum que representa os status de um check-in (pendente, confirmado, cancelado).
  - `StatusPagamento.cs`: Define os status dos pagamentos (pendente, confirmado, cancelado).
  - `StatusReserva.cs`: Enum que representa os status de uma reserva (confirmada, cancelada).
  - `TipoAlojamento.cs`: Enum que define os diferentes tipos de alojamentos disponíveis no sistema (hotel, hostel, resort).

- `Utilis/`: Contém classe auxiliares para a questão de criar ids, como:
  - `CriarIds.cs`: Contém função para criar identificadores únicos para diferentes entidades do sistema.
- `Validacoes/`: Contém os ficheiros responsáveis pelas validações dos dados de entrada e regras do sistema, como:
  - `Validador.cs`: Classe que contém funções para validação de dados e regras de negócios do sistema, onde é possível verificar a integridade dos dados, como é o caso de um novo cliente.
- `Services/`: Contém as interfaces e implementações dos serviços utilizados no sistema para manipulação dos dados. A pasta `Interfaces/` dentro de `Services/` contém as interfaces que definem os métodos de serviço para cada entidade.
  - `IAlojamentoService.cs`: Contém os métodos de serviço para a gestão de alojamentos.
  - `ICheckInService.cs`: Define os métodos de serviço para o processo de check-in.
  - `IClienteService.cs`: Contém os métodos de serviço para a gestão de clientes.
  - `IPagamentoService.cs`: Define os métodos de serviço para a gestão de pagamentos.
  - `IReservaService.cs`: Contém os métodos de serviço para a gestão de reservas.

4.3 Propriedades dos Modelos do Sistema

Descrição de forma tabular de todas as propriedades dos modelos, até ao momento de entrega de fase número um:

4.3.1 Tabela de Alojamento

Campo	**Tipo**	**Descrição**
Id	`int`	Identificador único do alojamento.
Nome	`string`	Nome do alojamento.
Localizacao	`string`	Localização do alojamento.
PrecoPorNoite	`decimal`	Preço por noite do alojamento.
Capacidade	`int`	Capacidade máxima de pessoas no alojamento.
NumeroDeQuartos	`int`	Número de quartos disponíveis no alojamento.
Status	`StatusAlojamento`	Status atual do alojamento.
Tipo	`TipoAlojamento`	Tipo de alojamento.
Imagens	`List<string>`	Lista caminhos para imagens do alojamento.
DataInicioDisponibilidade	`DateTime`	Data inicial de disponibilidade.
DataFimDisponibilidade	`DateTime`	Data final de disponibilidade.

4.3.2 Tabela de Reserva

Campo	Tipo	Descrição
Id	`int`	Identificador único da reserva.
ClienteId	`int`	ID do cliente associado à reserva.
Cliente	`Cliente`	Cliente associado.
AlojamentoId	`int`	ID do alojamento associado à reserva.
Alojamento	`Alojamento`	Alojamento associado.
DataInicio	`DateTime`	Data de início da reserva.
DataFim	`DateTime`	Data de fim da reserva.
Status	`StatusReserva`	Status da reserva (ex: pendente, confirmada, cancelada).
PrecoTotal	`decimal`	Preço total da reserva.
NumeroDePessoas	`int`	Número de pessoas associadas à reserva.

4.3.3 Tabela de Pagamento

Campo	Tipo	Descrição
Id	`int`	Identificador único do pagamento.
ReservaId	`int`	ID da reserva associada ao pagamento.
Reserva	`Reserva`	Reserva associada ao pagamento.
Valor	`decimal`	Valor total do pagamento.
DataPagamento	`DateTime`	Data em que o pagamento foi efetuado.
MetodoPagamento	`MetodoPagamento`	Método utilizado para o pagamento.
Status	`StatusPagamento`	Status do pagamento.

4.3.4 Tabela de CheckIn

Campo	Tipo	Descrição
Id	`int`	Identificador único do check-in.
ReservaId	`int`	ID da reserva associada ao check-in.
Reserva	`Reserva`	Reserva associada ao check-in.
DataHoraCheckIn	`DateTime`	Data e hora em que o check-in foi realizado.
NumeroDeClientesPresentes	`int`	Número de clientes presentes no check-in.
Observacoes	`string`	Observações relacionadas ao check-in.
Status	`StatusCheckIn`	Status do check-in.

4.3.5 Tabela de PessoaBase

Campo	Tipo	Descrição
Id	`int`	Identificador único da pessoa.
Nome	`string`	Nome da pessoa.
Email	`string`	Email da pessoa.
Telefone	`string`	Número de telefone da pessoa.
Idade	`int`	Idade da pessoa.
TipoUtilizador	`TipoUtilizador`	Tipo de utilizador.

4.4 Criação de enumeradores

Exemplo de enumerador criado para verificar status de checkIn.

```
namespace GestaoAlojamentosApp.Domain.Enums
{
    public enum StatusCheckIn
    {
        // Check-in ainda não foi realizado.
        Pendente,

        // Check-in foi confirmado.
        Confirmado,

        // Check-in foi cancelado.
        Cancelado
    }
}
```

4.5 Interfaces de Serviço

4.5.1 IAlojamentoService

- **CriarAlojamento**: Cria um novo alojamento com os dados fornecidos (nome, localização, preço por noite, capacidade, número de quartos, status e tipo).
- **AtualizarAlojamento**: Atualiza os dados de um alojamento existente com base no ID (nome, localização, preço por noite, capacidade, número de quartos, status e tipo).
- **RemoverAlojamento**: Remove um alojamento pelo ID.
- **ObterTodosAlojamentos**: Retorna uma lista de todos os alojamentos registados.
- **AdicionarImagem**: Adiciona uma imagem a um alojamento com base no ID.
- **RemoverImagem**: Remove uma imagem de um alojamento com base no ID.
- **ObterAlojamentoPorId**: Retorna um alojamento pelo ID.

- **ObterAlojamentoPorNome**: Retorna um alojamento pelo nome.
- **ObterAlojamentosPorLocalizacao**: Retorna uma lista de alojamentos com base na localização.
- **AlojamentosDisponiveis**: Retorna os alojamentos disponíveis.
- **AlterarStatusAlojamento**: Altera o status de um alojamento.
- **CarregarAlojamentosDeFicheiro**: Carrega os alojamentos de um ficheiro.
- **GuardarAlojamentosEmFicheiro**: Guarda os alojamentos em um ficheiro.

4.5.2 ICheckInService

- **CriarCheckIn**: Cria um novo check-in com base na reserva, na data e na hora do check-in, e no número de clientes presentes.
- **AlterarStatusCheckIn**: Altera o status de um check-in.
- **RemoverCheckIn**: Remove um check-in pelo ID.
- **ObterCheckInPorId**: Retorna um check-in com base no ID.
- **ObterCheckInsPorReserva**: Retorna a lista de check-ins associados a uma reserva.
- **ObterCheckInsPorStatus**: Retorna a lista de check-ins de acordo o status.
- **ObterTodosCheckIns**: Retorna uma lista com todos os check-ins cadastrados.
- **CarregarCheckInsDeFicheiro**: Carrega os check-ins de um ficheiro.
- **GuardarCheckInsEmFicheiro**: Guarda os check-ins em um ficheiro.

4.5.3 IClienteService

- **CriarCliente**: Cria um novo cliente com os dados fornecidos (nome, email, telefone, idade).
- **AtualizarCliente**: Atualiza as informações de um cliente existente (nome, email, telefone, idade).
- **RemoverCliente**: Remove um cliente com base no ID.

- **ObterClientePorId**: Retorna um cliente com base no ID.
- **ObterTodosClientes**: Retorna uma lista com todos os clientes cadastrados.
- **ObterClientesPorNome**: Retorna uma lista de clientes com base no nome.
- **ObterClientesPorEmail**: Retorna uma lista de clientes com base no email.
- **CarregarClientesDeFicheiro**: Carrega os clientes de um ficheiro.
- **GuardarClientesEmFicheiro**: Guarda os clientes em um ficheiro.

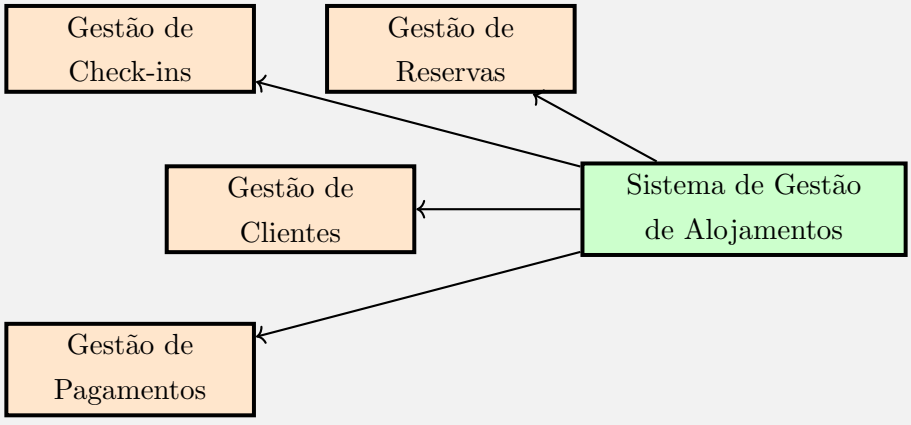
4.5.4 IPagamentoService

- **CriarPagamento**: Cria um novo pagamento associado a uma reserva, valor, método de pagamento e status.
- **AdicionarPagamento**: Adiciona um pagamento.
- **RemoverPagamentoPorId**: Remove um pagamento com base no ID.
- **AlterarStatusPagamento**: Altera o status de um pagamento.
- **ObterPagamentoPorId**: Retorna um pagamento com base no ID.
- **ObterPagamentosPorReserva**: Retorna os pagamentos associados a uma reserva.
- **ObterPagamentosPorCliente**: Retorna os pagamentos associados a um cliente.
- **ObterPagamentosPorAlojamento**: Retorna os pagamentos associados a um alojamento.
- **ObterPagamentosPorMetodo**: Retorna os pagamentos filtrados por método de pagamento.
- **ObterPagamentosPorStatus**: Retorna os pagamentos de acordo o status.
- **ObterTodosPagamentos**: Retorna todos os pagamentos registados.
- **CarregarPagamentosDeFicheiro**: Carrega os pagamentos de um ficheiro.
- **GuardarPagamentosEmFicheiro**: Guarda os pagamentos em um ficheiro.

4.5.5 IReservaService

- **CriarReserva**: Cria uma nova reserva associada a um cliente e alojamento, com base na data de início, data de fim, preço total e número de pessoas.
- **AtualizarReserva**: Atualiza os dados de uma reserva existente, incluindo cliente, alojamento, datas e preço total.
- **RemoverReserva**: Remove uma reserva com base no ID.
- **AlterarStatusReserva**: Altera o status de uma reserva.
- **ConfirmarPagamento**: Confirma o pagamento de uma reserva.
- **ObterTodasReservas**: Retorna uma lista com todas as reservas registadas.
- **ObterReservaPorId**: Retorna uma reserva com base no ID.
- **ObterReservasPorCliente**: Retorna todas as reservas feitas por um dado cliente.
- **ObterReservasPorAlojamento**: Retorna todas as reservas associadas a um dado alojamento.
- **ObterReservasDisponiveis**: Retorna as reservas disponíveis.
- **ObterReservasPorStatus**: Retorna reservas de acordo o status.
- **CarregarReservasDeFicheiro**: Carrega as reservas de um ficheiro.
- **GuardarReservasEmFicheiro**: Guarda as reservas em um ficheiro.

4.6 Diagrama Contextual



Capítulo 5

Análise e Discussão

Durante o desenvolvimento do projeto "Projeto_POO", várias etapas foram executadas e diversos desafios foram enfrentados. Neste capítulo, discutiremos em geral os resultados obtidos, os desafios encontrados e as possíveis melhoria para o trabalho realizado.

5.1 Resultados obtidos

O projeto atingiu seus principais objetivos iniciais, que incluíam a definição e implementação das classes correspondentes as entidades principais e acompanhado por enumeradores, foi também definido algumas estruturas de dados e métodos exemplares que poderão vir a ser usadas durante o desenvolvimento deste sistema.

5.2 Desafios e possíveis melhorias

Durante o desenvolvimento inical do projeto, alguns desafios foram enfrentados. Um dos principais desafios foi a implementação de uma classe abstrata que acaba por ser útil, claramente a definição do projeto e consequentemente do sistema, que acaba por levar a tomada de decisões e diferentes perspetivas. Por isso, deste modo o projeto poderá ser melhorado em alguns aspetos, bem como otimização do desempenho do código e além disso, o projeto poderia ser estendido para suportar outros tipos de dados, permitindo uma maior flexibilidade e aplicabilidade em diferentes cenários.

Capítulo 6

Conclusão

Este projeto possibilita a integração de várias técnicas de programação orientada a objetos utilizando a linguagem C#, além de aprofundar o conhecimento sobre persistência de dados em ficheiros e estruturas de dados.

A implementação demonstrou a importância da organização do código em classes e métodos bem definidos, além do uso de enumeradores para representar "status"e tipos de dados. Além disso, na segunda fase para além da elaboração dos forms e melhorias e extensões da fase número um,será tido em conta a utilização do "JSON"como formato de armazenamento de dados, deste modo capaz de permitir um sistema mais flexível e eficiente.

Capítulo 7

Bibliografia

[1] Documentação de bibliotecas em C#.

[3] Fórum StackOverflow (https://stackoverflow.com).

[4] Ambiente de desenvolvimento Visual Studio Code (https://code.visualstudio.com).

[5] Disciplina Programação Orientada a Objetos - Moodle IPCA LESIPL.

[8] Texmaker (https://www.xm1math.net/texmaker/).

[9] Overleaf (https://www.overleaf.com).

[10] CTAN (https://ctan.org).

[11] LaTeX Project (https://www.latex-project.org).