IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Gestão de Condomínios

Programação Orientada a Objetos

Docente: Ernesto Casanova

Aluno: João Filipe Rodrigues Ferreira

(a25275)

Engenharia de Sistemas Informáticos

dezembro, 2024


ESCOLA
SUPERIOR
DE TECNOLOGIA
IPCA

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Relatório de Trabalho Prático

Programação Orientada a Objetos

Docente: Ernesto Casanova

Aluno: João Filipe Rodrigues Ferreira – a25275

Engenharia de Sistemas Informáticos

dezembro, 2024

1. Resumo

Este relatório apresenta o desenvolvimento de um sistema de **Gestão de Condomínios**, com foco na aplicação prática dos princípios de *Programação Orientada a Objetos (POO)*. O projeto resultou num sistema funcional e escalável, permitindo a gestão eficiente de condomínios, condóminos, frações, pagamentos, despesas, reservas e reuniões. A aplicação foi projetada para garantir uma administração eficiente e organizada dos diversos elementos de um condomínio, utilizando conceitos como modularidade e separação de responsabilidades. Durante o desenvolvimento, foram implementadas funcionalidades essenciais, como a persistência de dados com ficheiros JSON e a implementação de validações rigorosas para garantir a integridade do sistema. O trabalho destaca-se pela sua flexibilidade e organização, estando pronto para futuras melhorias e integrações, e proporcionando uma solução prática e eficiente para a gestão de condomínios.

Índice

Índice

Lista de Figuras

2. Introdução

O presente relatório documenta o desenvolvimento de um sistema de **Gestão de Condomínios**, com o objetivo principal de criar uma aplicação funcional que permita uma gestão eficiente de condôminos, frações, pagamentos, despesas, reservas e reuniões. O sistema foi desenvolvido com base nos princípios da **Programação Orientada a Objetos (POO)**, utilizando conceitos como modularidade, encapsulamento e separação de responsabilidades para garantir uma solução organizada e escalável.

Este projeto surgiu como um desafio técnico e académico, proporcionando uma oportunidade de aplicar conhecimentos em POO para resolver problemas práticos da gestão de condomínios. O sistema integra múltiplos componentes, como os serviços de pagamento, despesa, reserva e geração de relatórios, oferecendo funcionalidades essenciais para a administração diária de um condomínio. Além disso, a utilização de arquivos JSON para persistência de dados assegura flexibilidade e integridade, permitindo a fácil manipulação e armazenamento de informações de forma segura.

O relatório está estruturado para apresentar a evolução do sistema, desde a definição das classes e serviços na Fase 1 até a implementação final de todas as funcionalidades na Fase 2. Serão também discutidas as dificuldades enfrentadas durante o desenvolvimento e as soluções adotadas, além das melhorias que poderão ser implementadas no futuro. Este trabalho representa não apenas um exercício técnico, mas uma demonstração de como boas práticas de desenvolvimento de software e tecnologias adequadas podem ser aplicadas para resolver problemas reais de gestão de condomínios.

3. Estrutura base do Código Atualizada

Na fase 2, o sistema foi aprimorado com novas implementações e atualizações significativas nas classes, serviços e interface gráfica. A organização do projeto foi dividida em várias pastas, com a seguinte estrutura: *Models*, *Services*, *Forms*, *UserControls*, e *Utilities*. Abaixo está a lista atualizada dos ficheiros e uma breve explicação das principais adições e alterações:

3.1 Models

Classes Principais:

- **Program.cs**
- **Condominio.cs**
- **Condomino.cs**
- **Pagamento.cs**
- **Despesa.cs**
- **Reserva.cs**
- **Reuniao.cs**
- **FracaoAutonoma.cs**

3.2 Services

- **CondominioService.cs** – Gerencia as operações CRUD dos condomínios e a persistência de dados em ficheiros JSON.
- **CondominoService.cs** – Responsável pela manipulação de condôminos, incluindo validações e adição, remoção e listagem de dados.
- **PagamentoService.cs** – Gerencia os pagamentos, incluindo a validação de estados e a consistência dos dados relacionados aos pagamentos dos condôminos.
- **DespesaService.cs** – Responsável pela gestão das despesas do condomínio, com operações CRUD e persistência em ficheiros JSON.
- **ReservaService.cs** – Gerencia a lógica de reservas, incluindo criação, atualização, remoção e filtros.
- **ReuniaoService.cs** – Responsável pela gestão das reuniões do condomínio, incluindo agendamento, cancelamento e listagem de reuniões futuras.
- **FinanceiroService.cs** – Realiza o cálculo de saldos e balancetes financeiros dos condomínios, calculando receitas e despesas.
- **RelatorioService.cs** – Gera relatórios financeiros, de pagamentos, despesas, reservas e atas de reuniões.
- **FileService.cs** – Gerencia a persistência de dados em ficheiros JSON para os dados do condomínio, condôminos, frações, pagamentos e reservas.

3.3 Enums

- **Tipos.cs** – contém um ou mais enums que são usados para representar listas de valores fixos, tais como tipo de condomínio, tipo de condómino, tipo de reserva ou tipo de despesa.

3.4 Interface

- **IRelatorioService.cs** – Interface com métodos para gerar relatórios.

3.5 UI Layer (Forms)

- **MenuPrincipal.cs** – Janela principal do sistema que serve como ponto de entrada para o utilizador.
- **GestaoCondominios.cs** – Formulário para a gestão de condomínios, permitindo adicionar, editar e listar condomínios.
- **GestaoCondominos.cs** – Formulário para a gestão de condóminos, com operações de adição, edição e remoção.
- **GestaoPagamentos.cs** – Formulário para gestão de pagamentos dos condóminos, permitindo o controle das quotas pagas e em atraso.
- **GestaoDespesas.cs** – Formulário para a gestão de despesas do condomínio.
- **GestaoReservas.cs** – Formulário para criação e gestão de reservas de áreas comuns no condomínio.
- **GestaoReunioesForm.cs** – Formulário para agendamento, cancelamento e visualização das reuniões do condomínio.
- **GestaoRelatorios.cs** – Formulário que permite a geração de relatórios financeiros e operacionais.

3.6 Utilities

- **CalculadoraFinanceira.dll** – Biblioteca dinâmica (DLL) com métodos auxiliares para cálculos financeiros, como a determinação do saldo de um condomínio, cálculo de despesas e receitas, e geração de relatórios financeiros.
- **Ficheiros JSON** – Persistência de dados: Condomínios, condóminos, frações, pagamentos, despesas, reservas e reuniões são armazenados em ficheiros JSON, garantindo durabilidade, integridade e flexibilidade ao acessar os dados (Data).

ESTRUTURA DO PROJETO

CLASSES		
Models/	Condómino	Principal
	Proprietario	Herança
	Inquilino	Herança
	Condominio	
	FracaoAutonoma	
	Pagamento	
	Despesa	
	Reserva	
	Reunião	
Enums/	Tipos	Tipo Condomínio
		Tipo Fração
		Tipo Reserva
		Tipo Despesa
Services/	CondominioService	Métodos
	FinanceiroService	Métodos
	PagamentoService	Métodos
	DespesaService	Métodos
	ReservaService	Métodos
	FracaoAutonomaService	Métodos
	RelatorioService	Métodos
	ReuniaoService	Métodos
	FileService	Métodos JSON
Interfaces/	IRelatorioService	
Unit Tests/	Testes unitários	
Files/	Ler e Salvar dados	ficheiros JSON
DLL/	CalculadoraFinanceira	CalculosFinanceiros
Forms/	Form Principal	Menu principal
	Form GestaoCondominos	Gestão de Condóminos
	Form GestaoCondominios	Gestão de Condominios
	Form GestaoFracoes	Gestão de Frações
	Form GestaoReservas	Gestão de Reservas
	Form GestaoReunioes	Gestão de Reuniões
	Form GestaoDespesas	Gestão de Despesas
	Form GestaoPagamentos	Gestão de Pagamentos
	Form GestaoRelatorios	Gestão de Relatórios

Figura 1 - Estrutura base do projeto

4. Implementação final das classes e serviços

4.1 Classes e subclasses

4.1.1 Program.cs

A classe Program contém o método Main, que serve como ponto de entrada para a execução do programa. O código dentro do método realiza as seguintes operações:

1. ApplicationConfiguration.Initialize(); é usado para definir configurações da aplicação, como DPI e fontes padrão.

2. Application.Run(new MainWindow()); abre o *MenuPrincipal*, que é o formulário principal e a interface inicial para o utilizador.

4.1.2 Condomino.cs

A classe Condomino.cs é a classe principal que representa um condómino dentro do sistema. Ela contém informações como nome, Nif, contato, tipo de condômino e o condomínio ao qual está associado. Ela também possui a capacidade de verificar se o condômino é um inquilino através da propriedade IsInquilino e exibir as informações do condômino. O *ID* é gerado automaticamente para garantir identificadores únicos.

Propriedades:

- Id: Identificador único do condômino.
- Nome: Nome do condomino.
- NIF: Número de Identificação Fiscal do condómino.
- Contato: Informações de contato do condómino.
- TipoCondomino: Tipo do condômino (proprietário ou inquilino).
- CondominioId: ID do condomínio ao qual o condómino está associado.
- Condominio: Associação ao objeto Condominio.
- FracaoAutonoma: Associação à fração autónoma do condomino.
- IsInquilino: Verifica se o condómino é um inquilino.

Métodos:

- ExibirInfo(): Exibe as informações do condômino.

4.1.3 Proprietario.cs

A classe *Proprietario* é uma subclasse de *Condomino* e representa um condómino proprietário. Ela herda todas as propriedades e métodos de *Condomino*, mas sobrescreve o método *ExibirInfo()* para exibir informações específicas de um proprietário.

Propriedades e Métodos:

- Herda todas as propriedades e métodos de *Condomino*.
- Sobrescreve o método *ExibirInfo()* para exibir informações específicas de um proprietário.

4.1.4 Inquilino.cs

A classe Inquilino é uma subclasse de *Condomino* e representa um condômino inquilino. Ela herda todas as propriedades e métodos de *Condomino*, mas sobrescreve o método *ExibirInfo()* para exibir informações específicas de um inquilino.

Propriedades e Métodos:

- Herda todas as propriedades e métodos de `Condomino`.
- Sobrescreve o método `ExibirInfo()` para exibir informações específicas de um inquilino.

4.1.5 Condominio.cs

A classe Condominio é a classe principal que representa um condomínio no sistema. Ela contém informações sobre o nome, endereço, tipo, orçamento anual e a lista de frações e condôminos associados.

Propriedades:

- Id: Identificador único do condomínio.
- Nome: Nome do condomínio.
- Endereco: Endereço do condomínio.
- TipoCondominio: Tipo de condomínio (residencial, comercial, etc.).
- OrcamentoAnual: Orçamento anual do condomínio.
- Fracoes: Lista de frações autónomas do condomínio.

Métodos:

- AdicionarCondomino(): Adiciona um condômino ao condomínio.
- RemoverCondomino(): Remove um condômino do condomínio.
- AdicionarFracao(): Adiciona uma fração autónoma ao condomínio.
- RemoverFracao(): Remove uma fração autónoma do condomínio.

4.1.6 FracaoAutonoma.cs

A classe FracaoAutonoma é a classe principal que representa uma fração autónoma de um condomínio, com informações como identificação, área, permilagem, quota e tipo de fração. Também mantém o vínculo com o condomínio, proprietário e inquilino.

Propriedades:

- Identificacao: Identificador único da fração.
- Area: Área da fração.
- Permilagem: Permilagem da fração.
- Quota: Quota da fração.
- TipoFracao: Tipo da fração (apartamento, loja, etc.).
- CondominioId: ID do condomínio ao qual a fração pertence.
- ProprietarioId: ID do proprietário da fração.
- InquilinoId: ID do inquilino da fração (caso haja).

Métodos:

- AssociarObjetos(): Associa a fração aos objetos relacionados (condomínio, proprietário e inquilino).

4.1.7 Reserva.cs

A classe Reserva é a classe principal que representa uma reserva feita por um condômino para o uso de um espaço dentro do condomínio, com informações sobre a descrição, data, horário e tipo de reserva.

Propriedades:

- Id: Identificador único da reserva.
- Descricao: Descrição da reserva.
- Data: Data da reserva.
- HoraInicio: Hora de início da reserva.
- HoraFim: Hora de término da reserva.
- TipoReserva: Tipo da reserva (p. ex., evento, reunião, etc.).
- Condomino: O condômino que fez a reserva.
- Condominio: O condomínio ao qual a reserva está associada.

Métodos:

- Nenhum método específico foi definido para a classe Reserva além das validações no setter das propriedades.

4.1.8 Reunião.cs

A classe Reuniao é a classe principal que representa uma reunião dentro do condomínio, com informações sobre a data e hora, local, pauta e os condôminos participantes.

Propriedades:

- DataHora: Data e hora da reunião.
- Local: Local da reunião.
- Pauta: Pauta da reunião.
- Participantes: Lista de participantes da reunião.

Métodos:

- AdicionarParticipante(): Adiciona um condômino como participante.
- RemoverParticipante(): Remove um condômino da lista de participantes.

4.1.9 Pagamento.cs

A classe Pagamento é a classe principal que representa o pagamento de uma quota dentro do condomínio, com informações sobre o valor, descrição, data e hora do pagamento.

Propriedades:

- Id: Identificador único do pagamento.
- ValorQuota: Valor da quota paga.
- Descricao: Descrição do pagamento.
- Condominio: Condomínio associado ao pagamento.
- Condomino: Condômino que efetuou o pagamento.
- DataHoraPagamento: Data e hora do pagamento.

Métodos:

- ToString(): Retorna uma string com informações sobre o pagamento.

4.1.10 Pagamento.cs

A classe Despesa é a classe principal que representa uma despesa do condomínio, com informações sobre o tipo, valor, data e hora da despesa.

Propriedades:

- Tipo: Tipo da despesa (p. ex., manutenção, segurança, etc.).
- Id: Identificador único da despesa.
- Valor: Valor da despesa.
- Condominio: Condomínio associado à despesa.
- DataHoraDespesa: Data e hora em que a despesa foi registrada.

Métodos:

- ToString(): Retorna uma string com informações sobre a despesa.

4.2 Classes de Serviços

A fim de garantir uma estrutura organizada e modular para o sistema, foram desenvolvidos serviços especializados que centralizam a lógica de negócios e as operações essenciais. Estes serviços interagem de forma direta com as classes principais e ajudam a manter o código mais limpo e facilmente escalável. A seguir, estão descritos os serviços implementados, destacando suas funcionalidades e como contribuem para a organização do sistema.

4.2.1 CondominoService

O CondominoService é responsável pela gestão de condôminos dentro do sistema, permitindo a realização de operações essenciais, como a adição, remoção, atualização e listagem de condôminos. Este serviço também é integrado com um sistema de persistência, garantindo que todas as alterações realizadas na lista de condôminos sejam armazenadas de forma segura em um arquivo JSON. Através de métodos como *Adicionar(), Remover(), Atualizar()* e *ListarTodos(),* o serviço oferece um meio eficiente e simples de manipulação dos dados, enquanto valida a integridade das informações. Com isso, o sistema é mantido sincronizado e as operações realizadas sobre os condôminos são automaticamente persistidas, garantindo a consistência e escalabilidade do sistema.

4.2.2 CondomínioService

O serviço CondominioService foi desenvolvido para gerenciar as operações relacionadas aos condomínios, facilitando a adição, remoção, busca e cálculo de propriedades dos condomínios. Esse serviço centraliza as funcionalidades essenciais, permitindo que o sistema seja gerido de forma eficiente e organizada.

- **ObterCondominios():** Retorna a lista completa de condomínios, permitindo que o sistema consulte todos os dados de uma vez. Este método é fundamental para facilitar a visualização e manipulação de todos os condomínios registrados.
- **ObterPorId(int id):** Permite buscar um condomínio específico pelo seu ID. Caso o condomínio não seja encontrado, o método retorna um valor nulo, garantindo a integridade da operação e evitando a manipulação de dados inválidos.
- **AdicionarCondominio(Condominio condominio):** Este método permite adicionar um novo condomínio à lista. Caso o condomínio fornecido seja nulo, uma exceção é lançada. Após a adição, as alterações são automaticamente salvas no arquivo JSON, mantendo os dados persistentes.
- **RemoverCondominio(Condominio condominio):** Permite a remoção de um condomínio da lista. A remoção é realizada após a validação da presença do condomínio na lista, e as alterações são salvas no arquivo de persistência. Assim como no método anterior, isso garante que o arquivo JSON seja atualizado com a alteração.
- **CalcularAreaTotalCondominio(Condominio condominio):** Este método realiza o cálculo da área total de um condomínio com base nas frações que o compõem. Ele utiliza o serviço CalculosFinanceirosService para somar as áreas das frações e calcular o total de área do condomínio.

A classe CondominioService utiliza o FileService para gerenciar a persistência dos dados em um arquivo JSON, garantindo que todas as operações de adição, remoção e atualização sejam refletidas de forma consistente e segura no armazenamento dos dados. O serviço assegura que todas as mudanças nos condomínios sejam automaticamente salvas, mantendo o sistema sempre atualizado e garantindo a persistência dos dados ao longo do tempo.

4.2.3 FracaoAutonomaService

O serviço FracaoAutonomaService foi projetado para gerenciar as frações autónomas de um condomínio, oferecendo funcionalidades para adicionar, remover, listar e atualizar frações, além de calcular e atribuir valores como permilagem e quota. Este serviço também lida com a persistência dos dados e assegura que todas as alterações realizadas nas frações sejam devidamente salvas.

- **CarregarFracoes():** Carrega as frações autónomas do arquivo JSON, associando-as corretamente aos seus objetos relacionados, como o condomínio, proprietário e inquilino. Em caso de erro, o serviço lança uma exceção, garantindo que os dados sejam carregados corretamente.
- **AdicionarFracao():** Este método permite adicionar uma nova fração autónoma ao sistema. Ele realiza várias validações, como a verificação de valores inválidos para área, permilagem e quota, e busca os objetos relacionados, como condomínio e proprietário, antes de criar e associar a fração ao sistema. Após a adição, os dados são salvos no arquivo JSON.
- **SalvarFracoes():** Responsável por salvar as frações autónomas no arquivo JSON. Este método é chamado sempre que há uma alteração nas frações, garantindo que as modificações sejam persistidas.
- **CorrigirDadosFracoes():** Após o carregamento dos dados, o serviço corrige eventuais inconsistências nos dados das frações, como a sincronização do ID do condomínio com o objeto Condominio, assegurando a integridade dos dados.
- **ListarTodasFracoes():** Retorna todas as frações autónomas, garantindo que cada fração tenha os objetos relacionados associados corretamente, como o condomínio, proprietário e inquilino.
- **ListarCondominios():** Este método retorna uma lista de todos os condomínios, tanto os que têm frações associadas quanto os que não possuem, permitindo uma visão abrangente do sistema.
- **ListarFracoesPorCondominio():** Filtra as frações com base no ID do condomínio e retorna apenas aquelas associadas ao condomínio especificado. Ele também garante que os objetos relacionados a cada fração sejam carregados corretamente.
- **AtualizarFracao():** Permite atualizar os dados de uma fração existente, incluindo áreas, permilagem, quota e os IDs dos objetos relacionados. O método garante que todos os dados sejam validados antes de serem atualizados e que as mudanças sejam salvas no arquivo JSON.
- **RemoverFracao():** Permite remover uma fração do sistema, localizando-a pelo ID do condomínio e identificação da fração. A fração é removida da lista e os dados são atualizados no arquivo JSON.
- **CalcularPermilagensParaCondominio():** Calcula a permilagem de cada fração dentro de um condomínio com base na área total do condomínio, utilizando o serviço CalculosFinanceirosService para calcular a permilagem proporcional de cada fração.
- **CalcularQuotaFracao():** Este método calcula a quota de cada fração no condomínio com base no orçamento anual do condomínio, utilizando a permilagem de cada fração. O cálculo da quota também é feito por meio do serviço CalculosFinanceirosService.
- **CalcularMultaAtrasoPagamento():** Calcula a multa por atraso de pagamento de uma fração, considerando o valor da quota, a quantidade de dias de atraso e a taxa de multa diária. O cálculo é feito utilizando o serviço CalculosFinanceirosService.
- **GerarNotificacaoPagamento():** Gera uma notificação de pagamento com base na data de vencimento, fornecendo um aviso de cobrança ao condômino.

A classe FracaoAutonomaService se integra com os serviços de condomínio e condómino para garantir que as frações sejam corretamente associadas aos seus respectivos condomínios

e proprietários, além de realizar todos os cálculos necessários relacionados à gestão das frações. A persistência dos dados é realizada através do serviço *FileService*, que manipula os arquivos JSON e assegura que todas as alterações sejam mantidas ao longo do tempo.

4.2.4 ReservaService

O serviço ReservaService foi desenvolvido para gerenciar reservas dentro de um condomínio, permitindo o registro, atualização, cancelamento e consulta de reservas. A classe assegura que as reservas sejam válidas, evitando conflitos de horário e garantindo que todas as informações relevantes sejam registradas corretamente.

- **RegistrarReserva():** Este método permite o registro de uma nova reserva, realizando diversas validações antes de adicionar a reserva à lista. A validação inclui verificar se a descrição da reserva é fornecida, se o horário de início é anterior ao horário de término, e se já existe outra reserva para a mesma área e horário. Caso todas as condições sejam atendidas, a reserva é registrada com sucesso.
- **CancelarReserva():** Permite cancelar uma reserva existente identificando-a pelo ID único. Se a reserva não for encontrada, uma exceção KeyNotFoundException é lançada. Após localizar a reserva, ela é removida da lista de reservas.
- **AtualizarReserva():** Este método é utilizado para atualizar as informações de uma reserva existente. A validação é realizada para garantir que a descrição seja fornecida e que os horários sejam válidos. A reserva é localizada pelo ID, e seus dados são atualizados de acordo com os novos valores fornecidos.
- **ListarReservas():** Retorna todas as reservas registradas no sistema. Se não houver reservas, retorna uma lista vazia.
- **ListarReservasPorArea():** Filtra e retorna as reservas de um tipo específico, com base no tipo de reserva (definido pela enumeração TipoReserva). Esse método permite consultar reservas por categoria de uso de área dentro do condomínio.
- **ListarReservasPorData():** Permite listar as reservas para uma data específica. O método verifica se a data das reservas corresponde à data fornecida, considerando apenas a data (ignorando a hora).
- **ListarReservasPorCondominio():** Este método retorna todas as reservas associadas a um condomínio específico, permitindo filtrar as reservas com base no condomínio em questão.

A classe *ReservaService* é fundamental para garantir o controle de uso de áreas comuns dentro do condomínio, permitindo que os moradores ou administradores realizem reservas de forma eficaz e sem sobreposições. As validações implementadas asseguram que as regras de agendamento sejam seguidas corretamente, evitando conflitos de horários e assegurando a integridade das informações de reserva.

4.2.5 ReuniaoService

O serviço *ReuniaoService* foi desenvolvido para gerenciar reuniões dentro de um condomínio, permitindo o agendamento, cancelamento, listagem e a geração de relatórios para cada reunião. Além disso, oferece a funcionalidade de listar reuniões futuras, ajudando na organização e no controle de eventos do condomínio.

- **AgendarReuniao():** Este método permite agendar uma nova reunião. Ele recebe um objeto do tipo Reuniao e adiciona à lista interna de reuniões. Caso o objeto fornecido seja nulo, uma exceção ArgumentNullException é lançada.
- **CancelarReuniao():** Permite cancelar uma reunião já agendada. Para isso, o método recebe o objeto Reuniao a ser cancelado e o remove da lista de reuniões. Se o objeto for nulo, uma exceção é lançada.

- **ListarReunioes():** Retorna todas as reuniões agendadas. Caso não haja reuniões agendadas, a lista retornada estará vazia.
- **GerarRelatorioReuniao():** Este método gera um relatório (ou ata) para uma reunião específica. Utiliza o serviço *IRelatorioService* para gerar o relatório com base nas informações da reunião. O relatório gerado pode ser uma ata detalhada, resumindo o que foi discutido ou decidido durante a reunião.
- **ListarReunioesFuturas():** Retorna todas as reuniões agendadas que ainda não ocorreram (ou seja, que têm a data e hora superior ao momento atual). Esse método é útil para listar apenas os compromissos futuros, ajudando na organização de eventos.

A classe *ReuniaoService* desempenha um papel crucial na organização de reuniões e eventos no contexto do condomínio. Ela garante que as reuniões sejam agendadas corretamente, permite seu cancelamento e oferece uma maneira prática de obter relatórios de cada reunião, além de facilitar o acompanhamento das próximas reuniões a serem realizadas.

4.2.6 PagamentoService

O serviço **PagamentoService** foi desenvolvido para gerenciar os pagamentos dentro de um condomínio. Ele permite adicionar, remover, buscar e listar os pagamentos realizados. Além disso, oferece métodos personalizados para filtrar os pagamentos por condomínio e outras funcionalidades como o cálculo do total recebido.

- .**AdicionarPagamento():** Este método adiciona um novo pagamento à lista interna de pagamentos. Ele recebe um objeto do tipo Pagamento e o adiciona à lista *_pagamentos.* Caso o objeto Pagamento seja nulo, o método não faz nada, mas poderia lançar uma exceção dependendo de como o sistema for modelado.
- .**RemoverPagamento():** Este método permite remover um pagamento existente. Ele recebe o *id* do pagamento a ser removido, busca o pagamento correspondente na lista de pagamentos e, caso encontrado, o remove da lista. Se o pagamento não for encontrado, ele simplesmente não faz nada.
- **BuscarPagamentoPorId():** Este método busca um pagamento específico pelo seu *id*. Ele retorna o pagamento correspondente ou *null* caso o pagamento não seja encontrado.
- **ListarPagamentos():** Este método retorna todos os pagamentos registrados na lista _pagamentos. Caso não haja nenhum pagamento, ele retornará uma lista vazia.
- **ListarPagamentosPorCondominio():** Este método permite filtrar e listar os pagamentos de um condomínio específico. Ele recebe um objeto *Condominio* como parâmetro e retorna uma lista de pagamentos associados a esse condomínio. Caso não haja pagamentos para o condomínio, ele retornará uma lista vazia.
- **Cálculo de Total Recebido (funcionalidade personalizada):** Uma possível funcionalidade que poderia ser adicionada a este serviço é o cálculo do *total recebido* de um condomínio específico. Para isso, o método percorreria a lista de pagamentos e somaria os valores dos pagamentos associados ao condomínio. Isso ajudaria na gestão financeira do condomínio, oferecendo um resumo do quanto foi recebido até o momento.

O serviço *PagamentoService* tem como objetivo fornecer uma maneira eficiente de gerenciar os pagamentos do condomínio, com funcionalidades que possibilitam consultar, adicionar ou remover pagamentos, além de incluir métodos que ajudam na organização financeira.

4.2.7 DespesaService

A *DespesaService* é responsável por gerenciar as despesas dentro de um condomínio, oferecendo funcionalidades para adicionar, remover, listar e buscar despesas. A classe também fornece métodos personalizados para filtrar as despesas de um condomínio específico.

- **AdicionarDespesa():** Este método permite adicionar uma nova despesa à lista interna de despesas. Ele recebe um objeto *Despesa* como parâmetro e o adiciona à lista *_despesas*. Caso o objeto fornecido seja nulo, o método não realiza nenhuma operação, mas poderia ser configurado para lançar uma exceção, dependendo do caso.
- **RemoverDespesa():** Permite remover uma despesa existente da lista. O método recebe um *id* de uma despesa e, ao encontrá-la na lista, a remove. Se a despesa não for encontrada, o método simplesmente não faz nada.
- **BuscarDespesaPorId():** Este método busca uma despesa específica pelo seu *id*. Ele retorna o objeto *Despesa* correspondente ou *null* caso não encontre nenhuma despesa com o *id* fornecido.
- **ListarDespesas():** Este método retorna todas as despesas registradas na lista de despesas _despesas. Se não houver nenhuma despesa cadastrada, ele retorna uma lista vazia.
- **ListarDespesasPorCondominio():** Método personalizado que permite listar todas as despesas associadas a um determinado Condomínio. Ele recebe um objeto *Condominio* como parâmetro e retorna uma lista de despesas relacionadas a esse condomínio. Caso não haja despesas para o condomínio, retorna uma lista vazia.

A *DespesaService* oferece uma gestão simplificada e eficiente das despesas de um condomínio. Com os métodos *CRUD* (Criar, Ler, Atualizar e Deletar), o serviço permite manipular as despesas e também oferece a possibilidade de filtragem por condomínio. Ele garante uma maneira organizada de controlar e consultar as despesas do condomínio, ajudando na gestão financeira e na tomada de decisões.

4.2.8 FinanceiroService

A *FinanceiroService* é responsável pela gestão financeira de um condomínio, realizando o cálculo de saldo, despesas e receitas, além de gerar relatórios financeiros. Ela utiliza os serviços de *PagamentoService* e *DespesaService* para calcular o saldo financeiro do condomínio e fornecer relatórios detalhados.

Propriedades Privadas:

- **_pagamentoService**: Serviço utilizado para acessar e calcular os pagamentos realizados no condomínio.
- **_despesaService**: Serviço utilizado para acessar e calcular as despesas do condomínio.

_calculadoraFinanceira: Um serviço externo (provavelmente da biblioteca *CalculosFinanceiros*) utilizado para realizar cálculos financeiros como somatórios e outros cálculos necessários.

Construtor:

- O *FinanceiroService* recebe como parâmetros os serviços *PagamentoService*, *DespesaService* e *CalculosFinanceirosService***.**

Se algum desses serviços for *null*, uma exceção *ArgumentNullException* será lançada.

Métodos:

- **CalcularSaldoCondominio(Condominio condominio)**: Este método calcula o saldo financeiro do condomínio, subtraindo o total de despesas do total de pagamentos. Para isso, ele utiliza os métodos *ObterTotalPagamentos* e ObterTotalDespesas. O saldo é retornado como um valor decimal.
- **ObterTotalPagamentos(Condominio condominio)**: Responsável por calcular o total de pagamentos para um determinado condomínio. Ele lista todos os pagamentos associados ao condomínio e soma os valores das *quotas* de cada pagamento.
- **ObterTotalDespesas(Condominio condominio)**: Este método calcula o total de despesas de um condomínio. Ele lista todas as despesas associadas ao condomínio e soma os valores das despesas.
- **CalcularBalancoFinanceiro(List<decimal> receitas, List<decimal> despesas)**: Esse método calcula o balanço financeiro geral, levando em consideração uma lista de receitas e despesas. Ele utiliza a *CalculosFinanceirosService* para calcular o total de receitas e despesas e retorna o balanço financeiro (a diferença entre as receitas e as despesas).
- **GerarRelatorioFinanceiro(Condominio condominio)**: Gera um relatório financeiro básico do condomínio. O relatório inclui o saldo atual do condomínio, o total de pagamentos e o total de despesas. Ele utiliza o método *CalcularSaldoCondominio* e os métodos *ObterTotalPagamentos* e *ObterTotalDespesas* para obter as informações necessárias.

O *FinanceiroService* tem como objetivo fornecer uma visão clara da saúde financeira do condomínio. Ele realiza cálculos cruciais, como saldo, receitas, despesas e balanço financeiro, além de gerar relatórios que auxiliam na gestão financeira. Ao integrar os serviços de *PagamentoService* e *DespesaService*, o serviço garante que todos os aspetos financeiros do condomínio sejam gerenciados corretamente.

4.2.9 RelatorioService

A classe *RelatorioService* é um serviço responsável pela geração de relatórios diversos no contexto de um condomínio. Ela oferece funcionalidades para gerar relatórios relacionados a pagamentos, despesas, reservas e finanças, além de criar atas de reuniões.

Propriedades Privadas:

- _**pagamentoService**: Serviço utilizado para acessar informações sobre os pagamentos realizados no condomínio.
- _**despesaService**: Serviço utilizado para acessar informações sobre as despesas do condomínio.
- _**reservaService**: Serviço utilizado para acessar informações sobre as reservas realizadas nas áreas comuns do condomínio.

Construtor:

O construtor recebe como parâmetros os serviços *PagamentoService*, *DespesaService* e *ReservaService**,*** e os inicializa para uso interno da classe. Isso garante que o *RelatorioService* tenha acesso a todas as informações necessárias para gerar os relatórios.

Métodos:

- **GerarRelatorioPagamentos()**: Este método gera um relatório sobre os pagamentos realizados no condomínio. Ele lista todas as transações de pagamento, incluindo o ID, valor, nome do condômino, nome do condomínio e a data do pagamento. O relatório é formatado como uma string e retornado.
- **GerarRelatorioDespesas()**: Similar ao relatório de pagamentos, este método gera um relatório sobre as despesas do condomínio. Ele lista as despesas com seu ID, valor, tipo de despesa, nome do condomínio e data da despesa.
- **GerarRelatorioReservas()**: Este método gera um relatório sobre as reservas feitas pelas unidades do condomínio. Ele inclui informações como ID da reserva, nome do condômino, nome do condomínio, a área reservada e a data da reserva.
- **GerarRelatorioFinanceiro()**: Este método gera um relatório financeiro do condomínio, que inclui o total de pagamentos, o total de despesas e o saldo final (calculado como a diferença entre pagamentos e despesas). O relatório é formatado como uma string e retorna o resumo financeiro do condomínio.
- **GerarAtaReuniao(Reuniao reuniao)**: Gera uma ata detalhada para uma reunião específica. O método recebe um objeto do tipo *Reuniao* e cria um relatório com informações como data e hora da reunião, local, pauta e participantes. Esse relatório pode ser utilizado para documentar o que foi discutido e decidido durante a reunião.

A classe também menciona a possibilidade de exportar os relatórios em outros formatos, como PDF ou Excel, embora essa funcionalidade não esteja implementada no código atual.

A *RelatorioService* tem como objetivo central fornecer relatórios detalhados para várias operações no contexto do condomínio. Esses relatórios incluem dados sobre pagamentos, despesas, reservas e informações financeiras gerais. Além disso, a classe também é capaz de gerar atas de reuniões, permitindo a documentação das discussões e decisões. Esse serviço facilita a transparência e o controle financeiro, além de manter um histórico organizado das atividades do condomínio.

4.2.10 FileService

A *FileService* é um serviço responsável pela gestão de arquivos JSON, permitindo salvar e carregar dados relacionados aos condomínios, condóminos e frações autónomas. Ele lida com a leitura e escrita de arquivos JSON para persistir dados do sistema de forma eficiente e segura.

- **SalvarCondominios(List<Condominio> condominios)**: Este método recebe uma lista de objetos *Condominio* e os serializa para o formato JSON. O JSON gerado é então salvo no arquivo **"***condominios.json***"**. Caso haja falhas durante o processo de serialização ou gravação, uma exceção será gerada.
- **CarregarCondominios()**: Este método tenta carregar os dados de condomínios a partir do arquivo **"***condominios.json***"**. Se o arquivo existir, ele lê o conteúdo, desserializa o JSON para uma lista de objetos *Condominio* e os retorna. Caso o arquivo não exista, retorna uma lista vazia.
- **SalvarCondominos(List<Condomino> condominos)**: Similar ao método *SalvarCondominios*, este método salva os dados dos condôminos em um arquivo JSON. Ele recebe uma lista de objetos *Condomino*, os serializa para JSON e grava no arquivo **"***condominos.json***"**.
- **CarregarCondominos()**: Este método carrega os dados dos condôminos a partir do arquivo **"***condominos.json***"**. Ele lê o conteúdo do arquivo JSON e o desserializa para uma lista de objetos *Condomino*. Se o arquivo não existir, retorna uma lista vazia.
- **SalvarFracoes(List<FracaoAutonoma> fracoes)**: Este método serializa uma lista de *FracaoAutonoma* para o formato JSON e a grava no arquivo **"***fracoes.json***"**. Caso ocorra algum erro durante o processo de serialização ou gravação, uma exceção será

gerada. A configuração de serialização também ignora valores nulos para otimizar o arquivo JSON gerado.

- **CarregarFracoes()**: Este método tenta carregar os dados das frações autônomas a partir do arquivo "*fracoes.json*"**.** Se o arquivo existir, ele lê e desserializa os dados para uma lista de objetos *FracaoAutonoma*. Caso ocorra algum erro durante a leitura ou o arquivo não exista, uma exceção será gerada ou retornará uma lista vazia.

A *FileService* é essencial para o armazenamento e carregamento de dados no sistema. Com ela, as informações de *Condominio*, *Condomino* e *FracaoAutonoma* são persistidas em arquivos JSON, facilitando o gerenciamento desses dados de forma simples e estruturada. O serviço implementa a leitura e gravação desses arquivos, garantindo que os dados sejam salvos de maneira persistente e possam ser carregados quando necessário.

Pontos-chave**:**

- Persistência dos dados em arquivos JSON.
- Uso de *JsonConvert.SerializeObject* e *JsonConvert.DeserializeObject* para conversão entre objetos e JSON.
- Criação automática do diretório de arquivos caso ele não exista.
- Tratamento de exceções em métodos de leitura e gravação para garantir a robustez.

A classe *FileService* permite que os dados sejam facilmente armazenados e recuperados, promovendo a persistência do estado do sistema entre as execuções.

5. Demonstração da aplicação

5.1 Menu Pincipal (Home)

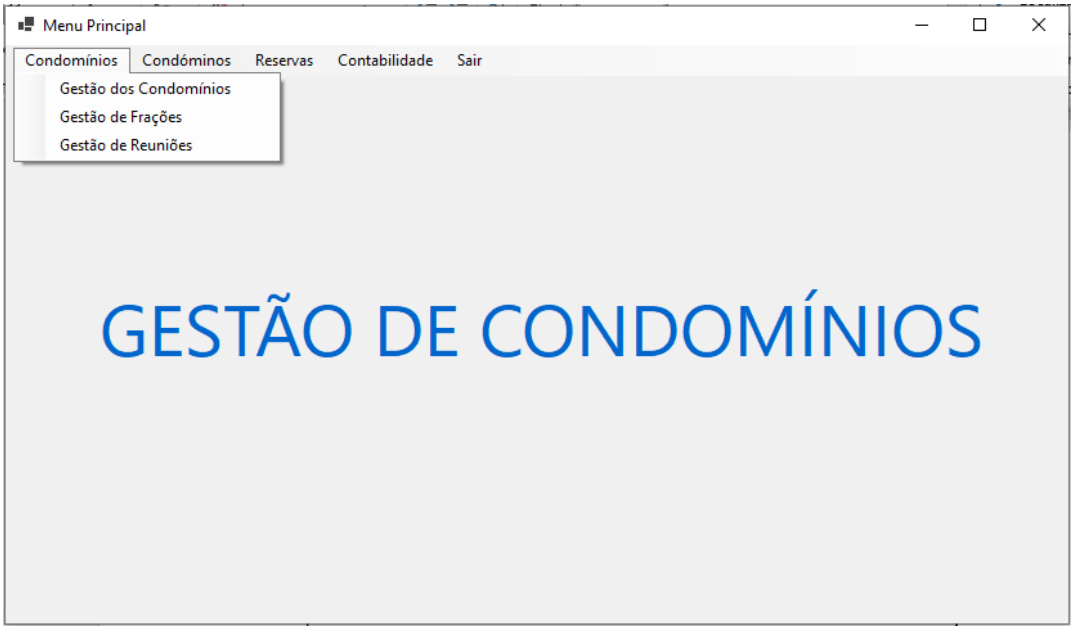

Figura 2 - Menu Principal

Este é o ponto de entrada do sistema, permitindo o acesso às principais funcionalidades através do menu para Condóminos**,** Condomínios**,** Reservas**,** Contabilidade**.** Tem como funcionalidade redirecionar o utilizar para os Formulários correspondentes onde os serviços de gestão específicos são utilizados. Navegando pelo menu irá mostrar os métodos CRUD implementadas em todas elas e incluem outras funcionalidades mais especificas.

5.2 Painéis de Gestão

O menu de gestão dos condomínios é um exemplo para todos os outros e mostra que é possível adicionar, editar, remover ou listar a categoria pretendida.

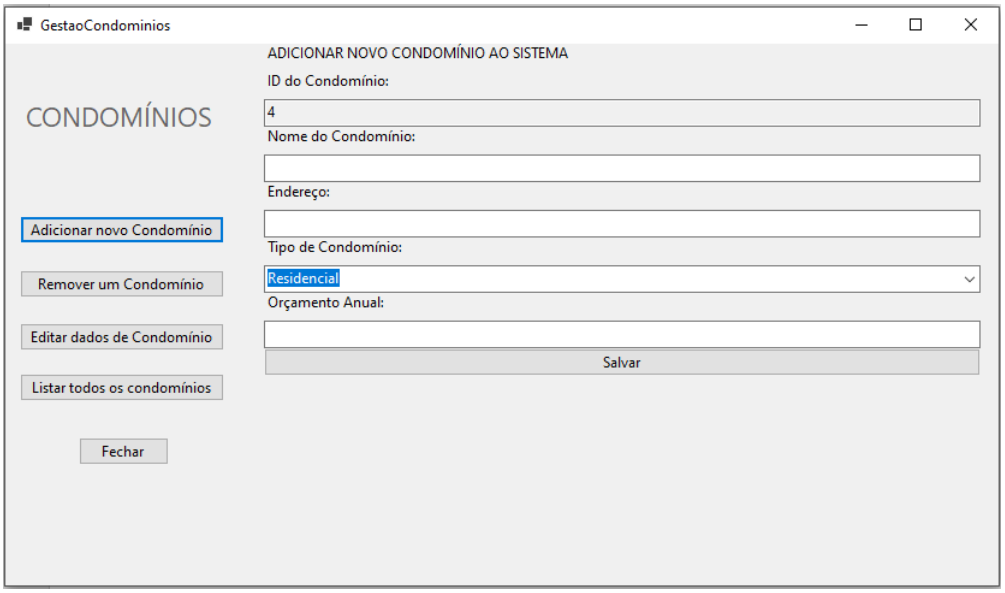

Figura 3 - Gestão de Condomínios

O Sistema utiliza um formulário único por cada bloco de gestão no sistema, e cada um permite visualizar e utilizar os botões de ação sempre disponíveis num painel fixo e os componentes e campos a preencher aparecem do lado num painel dinâmico e interativo, alternando conforme o botão escolhido.

Os Condomínios são a base central do sistema e cada um tem identificação única e as suas especificidades e podem ser de vários tipos, residencial, comercial, industrial ou misto.

5.3 Gestão de Frações Autónomas

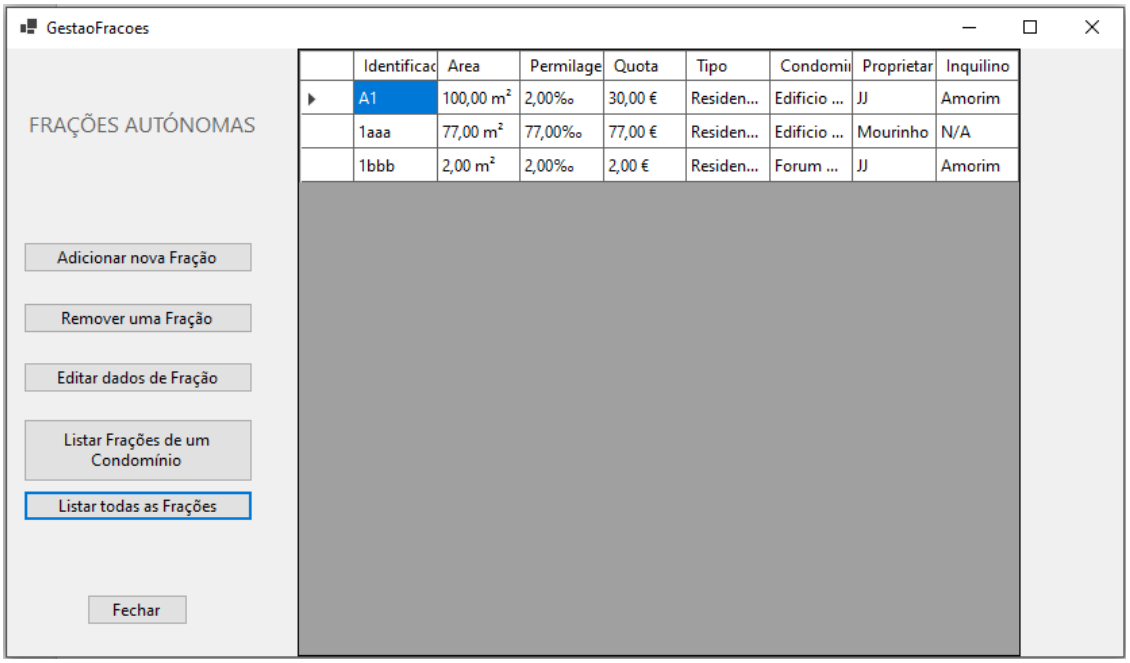

Figura 4 - Gestão de Frações Autónomas - Lista

Outro exemplo de um formulário de gestão, frações autónomas, que estão associadas a condomínios específicos e a um condómino específico também, que pode ser o próprio proprietário ou um inquilino (opcional) que vai ser o responsável pela fração e despesas do condomínio, como a quota mensal. Os valores das quotas e permilagens podem ser calculados em outras funcionalidades no sistema: A quota do condomínio resulta do que vier a ser orçamentado pela administração, e deve compreender os encargos com as partes comuns, incluindo as despesas correntes com a limpeza e vigilância das mesmas, e a comparticipação para o Fundo Comum de Reserva. (Fórmula de cálculo: Quota de condomínio = orçamento anual x Permilagem da fração 1 000).

As frações podem ainda ser de vários tipos como residência, loja, garagem, pavilhão industrial ou outros.

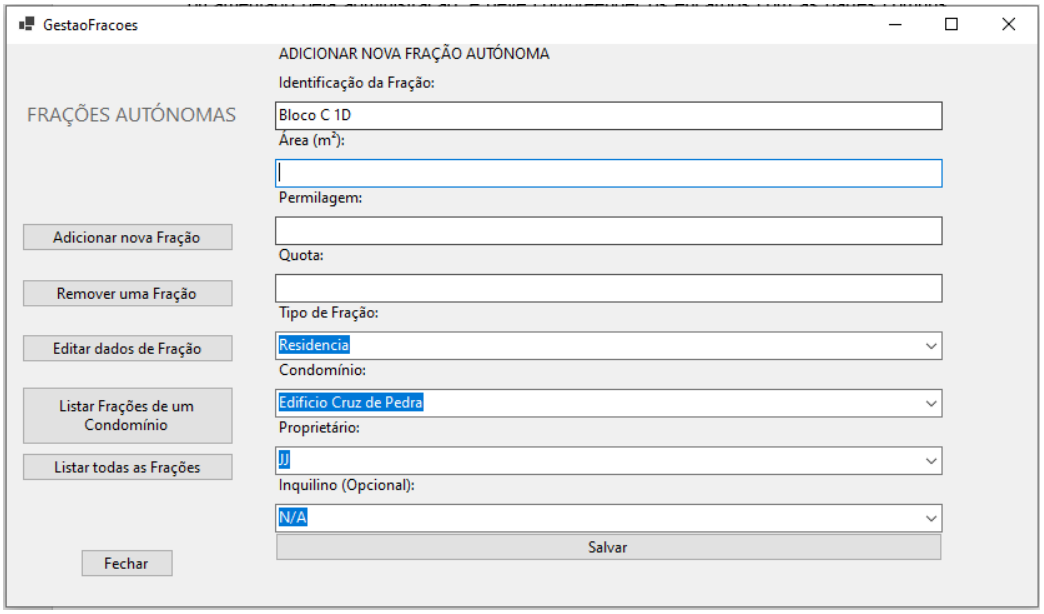

Figura 5 - Adicionar nova Fração

5.4 Gestão de Reuniões

No Formulário de Gestão de Reuniões, permite gerir as reuniões de condomínio com ações do tipo agendar, cancelar, atualizar ou ver as reuniões efetuadas ou as próximas reuniões, com as informações da reunião assim como os participantes e as atas das reuniões efetuadas.

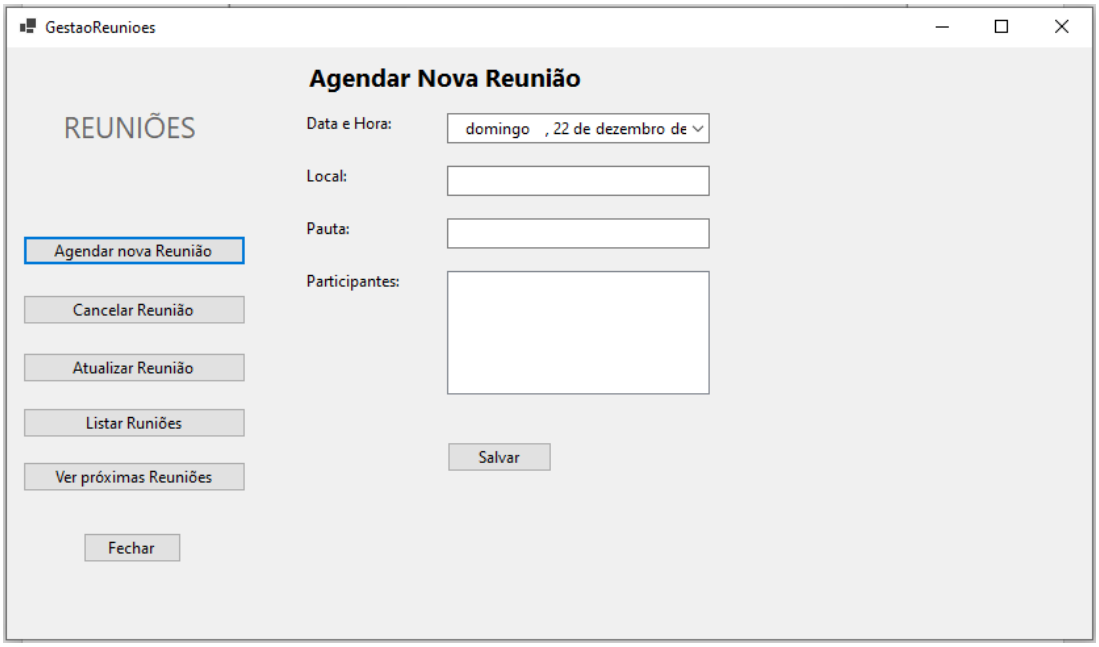

Figura 6 - Gestão das reuniões de condomínio

5.5 Gestão de Reservas

Este formulário do sistema permite fazer a gestão de reservas de áreas comuns dos condomínios destinadas a eventos ou lazer para uso dos condóminos. Essas áreas comuns podem ser um salão para festas, sala de reuniões, piscina, parque infantil, recinto desportivo ou outros, dependendo de cada condomínio.

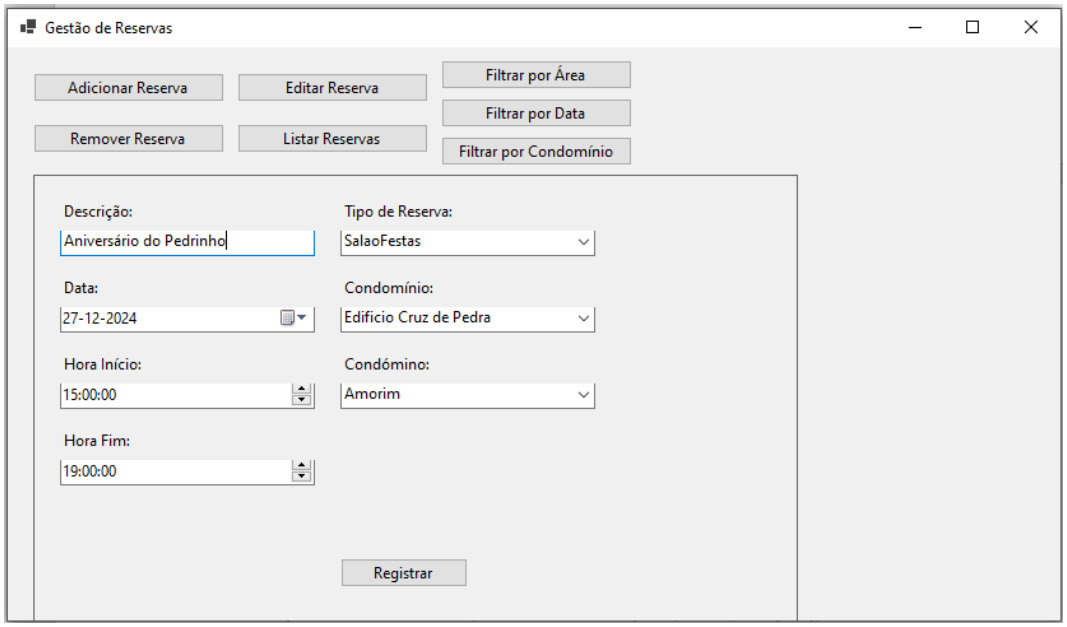

Figura 7 - Gestão de Reservas de áreas comuns

Assim, podemos fazer marcação, alteração ou cancelamento da reserva, assim como listar ou filtrar as reservas marcadas ou efetuadas em cada condomínio, com o registo do dia, hora de início e hora do fim do evento ou marcação, efetuada por um condómino pertencente ao condomínio específico identificando a área comum que pretende reservar.

5.6 Gestão Financeira (Pagamentos e Despesas)

Este é o formulário responsável pela gestão financeira de cada condomínio no sistema. As classes de serviços PagamentoService, DespesaService e FinanceiroService têm a responsabilidade de encapsular a lógica de negócios relacionada às finanças do condomínio. Cada serviço é responsável por um aspeto específico das finanças, enquanto

FinanceiroService unifica os dados para relatórios e análises estratégicas. Isso resulta num sistema eficiente, transparente e confiável. O Formulário é responsável pela organização e rastreamento de todos os pagamentos e despesas realizados no sistema, com atualização automática do saldo pendente.

5.7 Gestão de Relatórios

Este formulário faz a gestão de relatórios financeiros precisos para administração e condóminos. Identifica as tendências financeiras e possíveis déficits, dá suporte para decisões estratégicas, como aumento de taxas de condomínio ou redução de custos. Usa dados de PagamentoService e DespesaService para gerar relatórios financeiros e calcular o saldo do condomínio.

A Gestão de Relatórios Financeiros compila dados de pagamentos e despesas num relatório detalhado que fornece:

- Receita total arrecadada.
- Total de despesas (pagas, pendentes, atrasadas).
- Saldo final do período.

A interface IRelatoriosService tem um papel fundamental nesta gestão e define um contrato para a geração de relatórios dentro do sistema de gestão de condomínios. O seu objetivo principal é padronizar e centralizar a lógica de gerar relatórios financeiros, de pagamentos e de despesas. Essa abordagem traz diversos benefícios e organiza as responsabilidades no sistema.

A interface adiciona uma camada de organização e modularidade ao sistema, permitindo a criação de relatórios claros, exportáveis e confiáveis para a administração do condomínio. Ela ajuda a consolidar informações financeiras importantes e a fornecer insights úteis para a gestão e os condóminos.

O uso da DLL de Cálculos Financeiros também é importante na centralização da Lógica Financeira, pois toda a lógica de cálculo fica isolada, facilitando a manutenção e atualização. A DLL de cálculos financeiros é fundamental para garantir que todas as operações financeiras sejam feitas com consistência e precisão. Ela melhora a escalabilidade do sistema, reduz erros manuais e facilita o crescimento do projeto ao encapsular toda a lógica financeira num único lugar.

6. Desafios e melhorias do projeto

Durante o desenvolvimento deste projeto, enfrentaram-se diversos desafios que exigiram soluções criativas e técnicas para garantir a funcionalidade e a integridade do projeto. O principal desafio foi sem dúvida a interação com o Windows Forms, pois nunca antes tinha experimentado. Outro dos desafios foi a manipulação de ficheiros JSON, que demandou a implementação de tratamentos de exceção e validações adicionais para assegurar a consistência dos dados, mesmo em casos de falhas ou operações interrompidas. Outro ponto crítico foi a validação de dados, tratamento de exceções e verificações operacionais. A integração entre os serviços e a interface gráfica também se destacou como um desafio significativo, pois era necessário evitar a mistura de lógica de negócios com o código dos formulários, o que foi superado através da utilização de eventos e métodos que permitiram a separação clara de responsabilidades por camadas.

Apesar de o sistema estar funcional e cumprir a maioria dos objetivos estabelecidos, existem algumas melhorias futuras que poderiam ser implementadas para aumentar a eficiência e a escalabilidade do projeto. Uma melhoria no aspeto visual, deixando o sistema mais profissional, seria certamente a primeira, assim como ajustes a nível funcional, nomeadamente o uso de testes unitários pois até ao momento ainda não foi possível a implementação. A substituição dos ficheiros JSON por uma base de dados relacional, como SQL Server, seria um passo importante para suportar múltiplos utilizadores simultâneos e melhorar a gestão dos dados. Além disso, a implementação de relatórios automáticos em formatos como PDF ou Excel poderia oferecer análises financeiras e operacionais de forma prática e acessível. Por fim, a integração com serviços de envio de notificações, como emails para confirmação de reservas ou alertas de pagamento, envio de recibos, tratamento de faturas, etc, traria maior interatividade e eficiência ao sistema, tornando-o ainda mais útil e profissional para o seu público-alvo.

7. Conclusão

O desenvolvimento do sistema de Gestão de Condomínios representou uma oportunidade significativa para aplicar os princípios de Programação Orientada a Objetos (POO) e criar uma solução funcional e escalável. Ao longo do projeto, foram implementadas funcionalidades essenciais como a gestão de condomínios, condóminos, reservas, reuniões, pagamentos, despesas e relatórios, todas integradas numa interface gráfica intuitiva e persistidas em ficheiros JSON.

O sistema demonstrou ser eficiente e coeso, com uma arquitetura modular baseada em serviços dedicados que centralizam a lógica de negócios e facilitam a manutenção. As funcionalidades desenvolvidas foram acompanhadas por validações rigorosas, sempre que possível, garantindo a consistência e integridade dos dados, enquanto a interface gráfica proporciona ao utilizador uma experiência fluida e prática.

Em resumo, o projeto cumpriu os objetivos propostos e demonstrou a importância de uma abordagem estruturada e orientada a objetos para o desenvolvimento de software. Ele representa não apenas um produto funcional, mas também um marco de aprendizado e crescimento técnico. O código-fonte e detalhes adicionais estão disponíveis no repositório do GitHub para análise e futuras expansões, reafirmando a flexibilidade e o potencial do sistema.

8. Anexos e recursos adicionais

O código-fonte e a documentação completos do sistema de gestão de um alojamento turístico desenvolvido como parte deste trabalho acadêmico estão disponíveis no seguinte repositório do GitHub:

https://github.com/JoaoFilipe-RSI/TP-POO.

9. Bibliografia

1. GitHub do Professor - Projeto POO
   IPCA-Content. Disponível em: https://github.com/IPCA-Content/POO-LESI-PL-202425
2. Windows Forms Documentation
   Microsoft. Disponível em: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/
3. C# Programming Documentation
   Microsoft. Disponível em: https://learn.microsoft.com/en-us/dotnet/csharp/
4. C# Stack Overflow. Disponível em: https://pt.stackoverflow.com/
5. JSON for Modern C# Development
   Microsoft. Disponível em: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-overview