UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Aplicação de Gestão de Ginásios - Backend

Licenciatura em Engenharia Informática

Laboratório de Projeto em Engenharia Informática

Luís Filipe Leite Barbosa

António Manuel Ribeiro de Sousa

Orientador

Prof. José Benjamim Ribeiro da Fonseca

Autores

Jordi Zeferino Pinheiro Mucuta - 75624

Tukayana Sara de Oliveira Mandinga - 77370

Vila Real, 2025

Objetivo

Este projeto tem como objetivo o desenvolvimento do backend para uma aplicação de gestão de ginásios. A solução será responsável pela gestão de informações de clientes, planos de treino, pagamentos, marcação de aulas e monitorização do desempenho dos utilizadores. O sistema será implementado como uma **Console App (.NET Framework)**, garantindo segurança e eficiência no tratamento dos dados.

Índice

1. Introdução

Este relatório apresenta uma análise detalhada do sistema de gestão de ginásios, incluindo diagramas de casos de uso, modelo de relacionamento de entidades, requisitos funcionais e não funcionais, e uma visão geral da arquitetura do sistema. O projeto visa automatizar processos como controlo de membros, agendamento de aulas e gestão de pagamentos através de uma aplicação console desenvolvida em .NET Framework.

2. Enquadramento Teórico

1. Gestão de Ginásios

A gestão de ginásios inclui processos como:

- Controlo de membros.
- Agendamento de aulas.
- Pagamentos.

A automação destes processos traz benefícios como:

- Redução de erros manuais.
- Melhoria da experiência do utilizador.
- Acesso rápido a informações e relatórios.

2. Desenvolvimento de Software

O desenvolvimento de software envolve a criação, teste e manutenção de aplicações. Para estudantes, é importante seguir boas práticas como:

- **Modularidade**: Dividir o código em módulos reutilizáveis.
- **Documentação**: Escrever documentação clara para facilitar a manutenção.
- **Testes**: Implementar testes unitários e de integração.

3. Console App (.NET Framework)

A **Console App (.NET Framework)** é uma aplicação baseada em linha de comandos, ideal para:

- **Processamento de dados** sem necessidade de interface gráfica.
- **Integração com sistemas legados**.
- **Execução de tarefas automatizadas**.

4. Autenticação e Segurança

A autenticação é crucial em aplicações com dados sensíveis. Este projeto utiliza **JWT (JSON Web Tokens)**, que oferece:

- **Segurança**: Tokens assinados e criptografados.
- **Escalabilidade**: Facilidade de integração com sistemas distribuídos.
- **Stateless**: Não requer armazenamento de sessão no servidor.

5. Base de Dados

Será utilizado o **Microsoft SQL Server**, um sistema de gestão de base de dados relacional (SGBDR) robusto e escalável. Para gerir a base de dados, será utilizado o **SQL Server Management Studio (SSMS)**, uma ferramenta gráfica que facilita a criação, gestão e consulta de bases de dados.

Principais características:

- **Desempenho**: Ideal para aplicações que requerem alta disponibilidade.

- **Ferramentas Avançadas**: O SSMS oferece gestão de utilizadores, backup/restauro e otimização de consultas.
- **Compatibilidade**: Suporta SQL, a linguagem padrão para bases de dados relacionais.

6. Tecnologias Utilizadas

- **C# (.NET Framework)**: Linguagem de programação robusta para desenvolvimento de aplicações console.
- **Microsoft SQL Server**: SGBDR robusto e escalável.
- **SQL Server Management Studio (SSMS)**: Ferramenta gráfica para gestão de bases de dados.
- **JWT**: Padrão para autenticação baseada em tokens.

7. Aprendizado Prático

Este projeto permite a aplicação prática de conceitos como:

- **Desenvolvimento de aplicações console em C# (.NET Framework)**.
- **Gestão de bases de dados com SQL Server e SSMS**.
- **Autenticação e segurança com JWT**.
- **Boas práticas de programação em C#**.

3. Objetivos da Etapa do Trabalho Prático

Objetivo Etapa 1: Análise dos Requisitos

Esta etapa tem como objetivos principais:

1. **Compreender as Necessidades do Sistema**:
    - Identificar funcionalidades essenciais.
    - Compreender expectativas dos utilizadores (administradores, membros).
2. **Definir Requisitos Claros e Precisos**:
    - Documentar requisitos funcionais e não funcionais.
3. **Validar a Viabilidade do Projeto**:
    - Avaliar se os requisitos são tecnicamente viáveis com **Console App (.NET Framework)**.
4. **Estabelecer Prioridades**:
    - Definir funcionalidades essenciais para o MVP.
5. **Garantir a Qualidade**:
    - Assegurar segurança, usabilidade e desempenho.
6. **Facilitar a Comunicação**:
    - Servir como referência para a equipa de desenvolvimento.

Objetivo da Etapa 2: Definição da Arquitetura

Esta etapa visa estabelecer a estrutura técnica do sistema, garantindo que seja:

1. **Organizado**: Dividido em módulos claros.
2. **Funcional**: Capaz de atender aos requisitos.
3. **Escalável**: Preparado para crescer no futuro.
4. **Mantível**: Fácil de atualizar e corrigir.
5. **Eficiente**: Com bom desempenho e segurança.

Etapa 1: Análise dos Requisitos

Requisitos Funcionais (RF)

Autenticação e Perfis

1. **RF01 – Login com JWT**: Permitir que utilizadores (Administrador, Membro, Instrutor) façam login com email e palavra-passe, gerando um token JWT válido por 24 horas. *(Opção 5)*
2. **RF02 – Verificação de Token**: Validar o token JWT em cada pedido para garantir acesso autorizado. *(Opção 6)*
3. **RF03 – Logout Seguro**: Invalidar o token quando o utilizador termina a sessão.
4. **RF04 – Recuperação de Palavra-passe**: Enviar um link por email para redefinir a palavra-passe esquecida.

Gestão de Membros *(Opções 1, 2, 9, 10)*

5. **RF05 – Registo de Membro**: Cadastrar novos membros (nome, email, tipo de plano, data de adesão).
6. **RF06 – Listagem de Membros**: Mostrar lista paginada de membros com filtros (nome, plano, estado).
7. **RF07 – Edição de Membro**: Atualizar dados pessoais e plano do membro.
8. **RF08 – Remoção de Membro**: Eliminar membros (com confirmação e arquivamento de dados).
9. **RF09 – Bloqueio por Inadimplência**: Impedir agendamentos se o pagamento estiver em atraso.

Gestão de Aulas e Agendamentos *(Opções 3, 4, 8, 11, 12)*

10. **RF10 – Criação de Aula**: Cadastrar aulas com horário, duração, instrutor e lotação máxima.
11. **RF11 – Listagem de Aulas**: Exibir aulas disponíveis (com filtros por data, tipo de aula e instrutor).
12. **RF12 – Agendamento de Aula**: Permitir que membros marquem aulas (se o plano estiver ativo e sem conflitos de horário).
13. **RF13 – Cancelamento de Aula**: Permitir cancelamentos até X horas antes do início da aula.
14. **RF14 – Lista de Espera**: Adicionar membros à lista de espera se a aula estiver lotada.
15. **RF15 – Aulas Agendadas pelo Membro**: Mostrar todas as aulas marcadas pelo membro (com opção de cancelamento).

Pagamentos e Financeiro *(Opção 7)*

16. **RF16 – Pagamento Online**: Processar pagamentos via cartão de crédito.
17. **RF17 – Multa por Atraso**: Aplicar multa de 5% sobre pagamentos efetuados após o vencimento.
18. **RF18 – Recibo Digital**: Gerar comprovante de pagamento em PDF.
19. **RF19 – Histórico de Pagamentos**: Exibir todas as transações realizadas pelo membro.

Relatórios e Dashboard

20. **RF20 – Relatório de Frequência**: Gerar relatório em PDF/Excel com taxa de ocupação por aula.
21. **RF21 – Relatório Financeiro**: Mostrar receitas por plano, membros inadimplentes e métricas mensais.

Funcionalidades Adicionais

22. **RF22 – Notificações**: Enviar email/SMS para confirmar agendamentos, pagamentos e cancelamentos.
23. **RF23 – Check-in Automático**: Registar presença via QR Code ou código da aula.
24. **RF24 – Integração com Calendário**: Sincronizar aulas agendadas com Google Calendar/Outlook.
25. **RF25 – Avaliação de Aulas**: Permitir que membros classifiquem aulas (1-5 estrelas).

Requisitos Não Funcionais (RNF)

Segurança

1. **RNF01 – Criptografia**: Dados sensíveis (palavras-passe, cartões) devem ser criptografados (AES-256).
2. **RNF02 – Proteção contra SQL Injection**: Utilizar prepared statements em todas as queries.
3. **RNF03 – Limite de Tentativas**: Limitar tentativas de login (5 tentativas por minuto).

Desempenho

4. **RNF04 – Tempo de Resposta**: Operações críticas (<1 segundo), listagens (<2 segundos com 10 mil registos).
5. **RNF05 – Escalabilidade**: Suportar até 1.000 utilizadores concorrentes.

Usabilidade

6. **RNF06 – Interface Responsiva**: Funcionar em dispositivos móveis (Android/iOS) e desktop.
7. **RNF07 – Acessibilidade**: Cumprir normas WCAG AA (contraste e suporte a leitores de ecrã).

Dados e Backup

8. **RNF08 – Backup Automático**: Diário (incremental) e semanal (completo).
9. **RNF09 – Recuperação de Desastres**: Restaurar o sistema em ≤4 horas após falha.

Integrações

10. **RNF10 – Gateway de Pagamento**: Integração com Stripe (conformidade PCI DSS).
11. **RNF11 – API Externa**: Webhooks para notificações em tempo real.

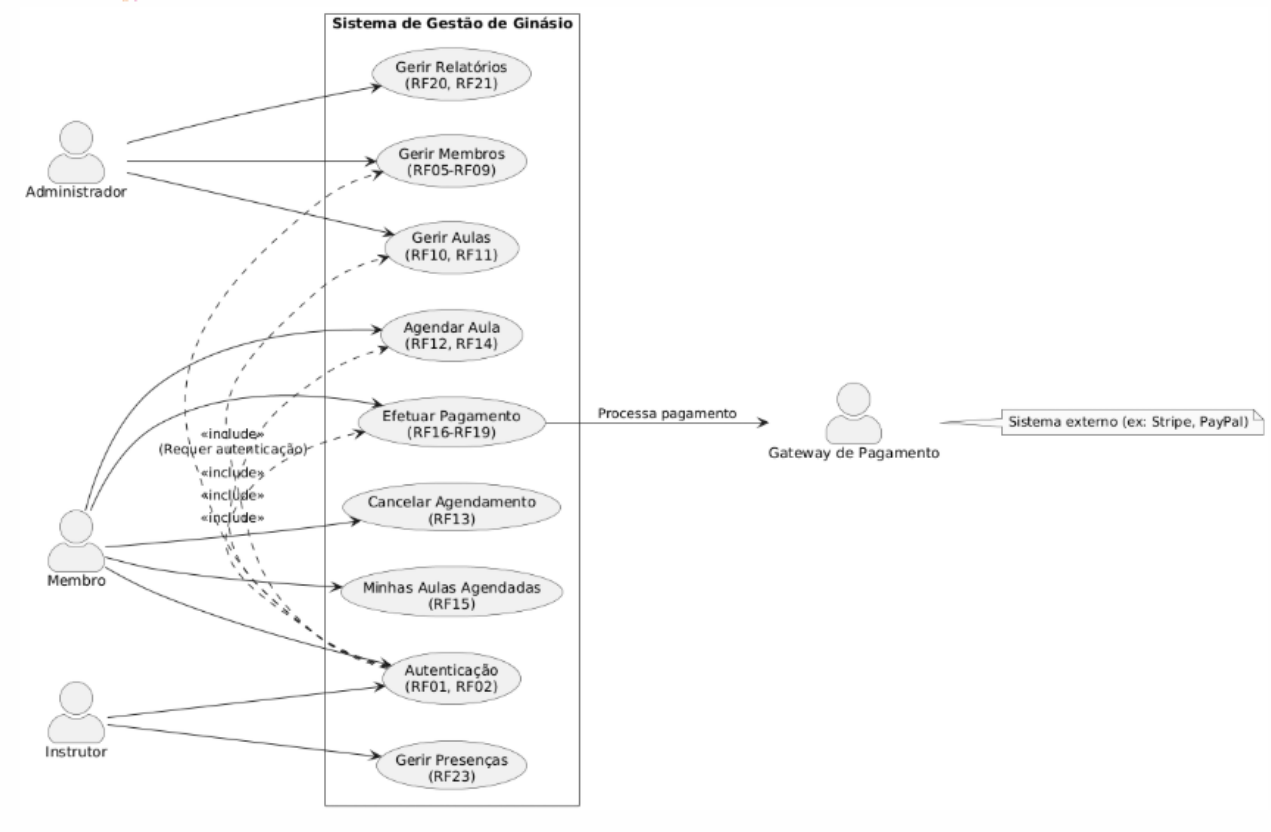
Conformidade

12. **RNF12 – RGPD**: Anonimização de dados após 2 anos de inatividade.
13. **RNF13 – Auditoria**: Registar (log) todas as operações críticas (edições, remoções).

Manutenibilidade

14. **RNF14 – Documentação**: Swagger para APIs e manual de administração.
15. **RNF15 – Testes Automatizados**: Cobertura ≥80% (testes unitários e de integração).

Diagrama de Casos de Uso



Especificação dos Casos de Uso

1. Autenticação no Sistema (UC1)

Atores: Administrador, Membro, Instrutor

Pré-condições: Nenhuma

Fluxo Principal:

1. Utilizador insere email e senha.
2. Sistema valida credenciais no banco de dados.
3. Sistema gera token JWT válido por 24h.
4. Sistema redireciona para dashboard conforme perfil (Admin/Membro/Instrutor).

Fluxos Alternativos:

- Credenciais inválidas → Sistema exibe mensagem de erro.
- Token expirado → Solicita reautenticação.

Pós-condições: Acesso liberado às funcionalidades conforme perfil.

---

2. Gerir Membros (UC3)

Atores: Administrador

Pré-condições: Utilizador autenticado como administrador.

Fluxo Principal:

1. Administrador acede ao menu "Gestão de Membros".
2. Seleciona operação (Adicionar/Editar/Remover).
3. Preenche formulário com dados do membro (Nome, Email, Plano).
4. Sistema valida e persiste dados no SQL Server.

Regras de Negócio:

- RN1: Email deve ser único.
- RN2: Planos válidos: Mensal (€30), Trimestral (€80), Anual (€300).

Pós-condições: Dados atualizados na tabela Membros.

---

3. Agendar Aula (UC5)

Atores: Membro

Pré-condições: Membro autenticado com plano ativo.

Fluxo Principal:

1. Membro consulta lista de aulas disponíveis (inclui UC: Verificar Disponibilidade).

2. Seleciona aula e horário.
3. Sistema verifica conflitos de horário e lotação máxima (RF10-11).
4. Sistema regista agendamento na tabela Agendamentos.

   Fluxos Alternativos:

- Lotação esgotada → Sugere lista de espera.
- Conflito de horário → Notifica membro (UC11).

  Pós-condições: Aula aparece no calendário do membro e do instrutor.

---

4. Efetuar Pagamento (UC7)

Atores: Membro

Pré-condições: Mensalidade gerada (UC12) ou pagamento manual.

Fluxo Principal:

1. Membro acede à "Pagamentos Pendentes".
2. Insere dados do cartão (inclui UC10: Validar Pagamento).
3. Sistema envia dados ao gateway de pagamento.
4. Confirmação recebida → Atualiza status para "Pago".

   Regras de Negócio:

- RN3: Pagamentos atrasados geram multa de 5% (UC11).

  Pós-condições: Status atualizado em Pagamentos e acesso liberado.

---

5. Gerar Relatórios (UC8)

Atores: Administrador

Pré-condições: Usuário autenticado como administrador.

Fluxo Principal:

1. Seleciona tipo de relatório (Frequência/Financeiro).
2. Define período (semana/mês/ano).
3. Sistema consulta SQL Server e gera PDF/Excel.

   Dados Incluídos:

- Financeiro: Total recebido, pendências, métricas por plano.
- Frequência: Aulas mais populares, taxa de ocupação.

  Pós-condições: Relatório disponível para download.

---

6. Registar Presenças (UC9)

Atores: Instrutor

Pré-condições: Aula agendada no sistema.

Fluxo Principal:

1. Instrutor acede à lista de alunos agendados.
2. Marca presença/ausência.
3. Sistema atualiza registos e calcula estatísticas.

Pós-condições: Dados disponíveis para relatórios de frequência (RF16).

Diagramas de Atividades

Diagrama de Atividades: Autenticação no Sistema

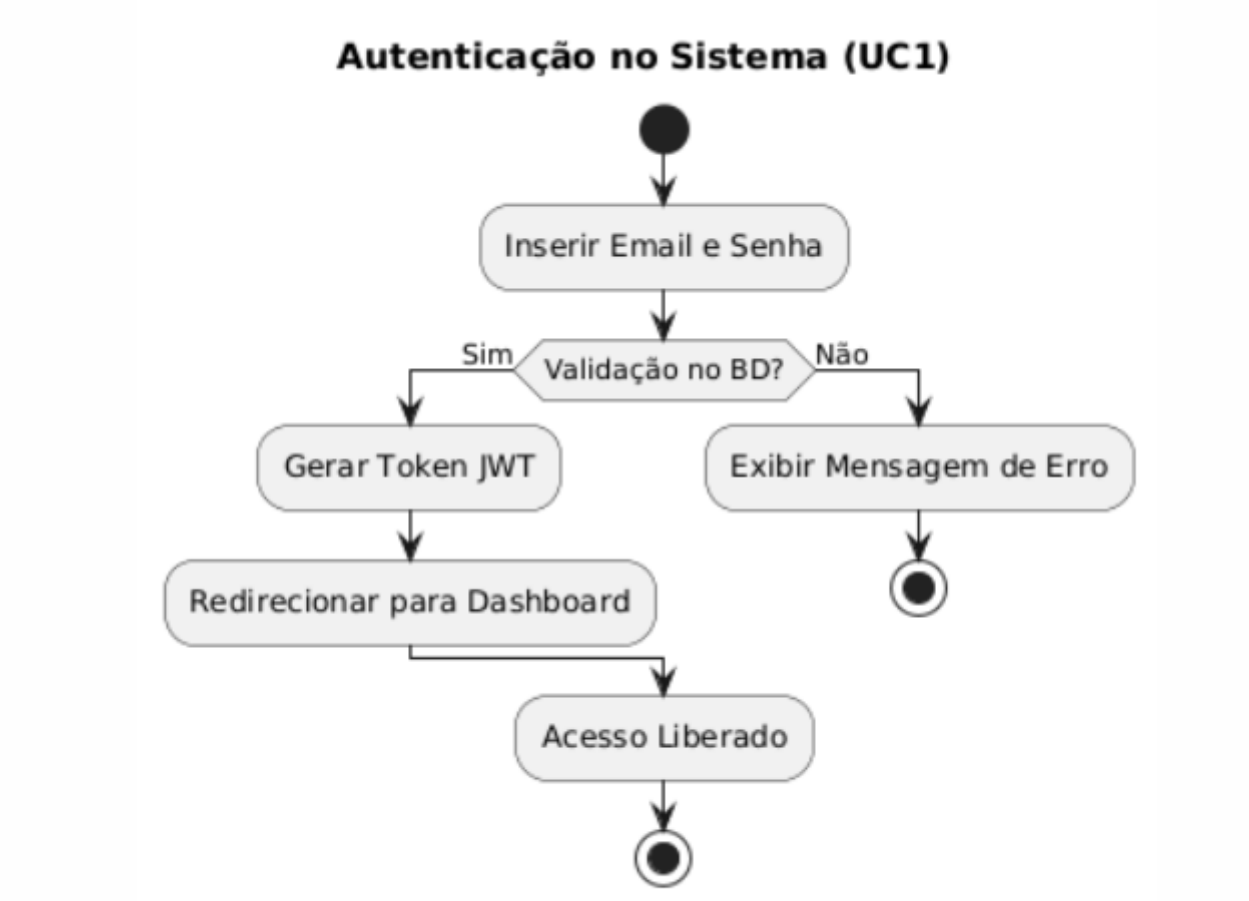


Diagrama de Atividades: Gerir Membros

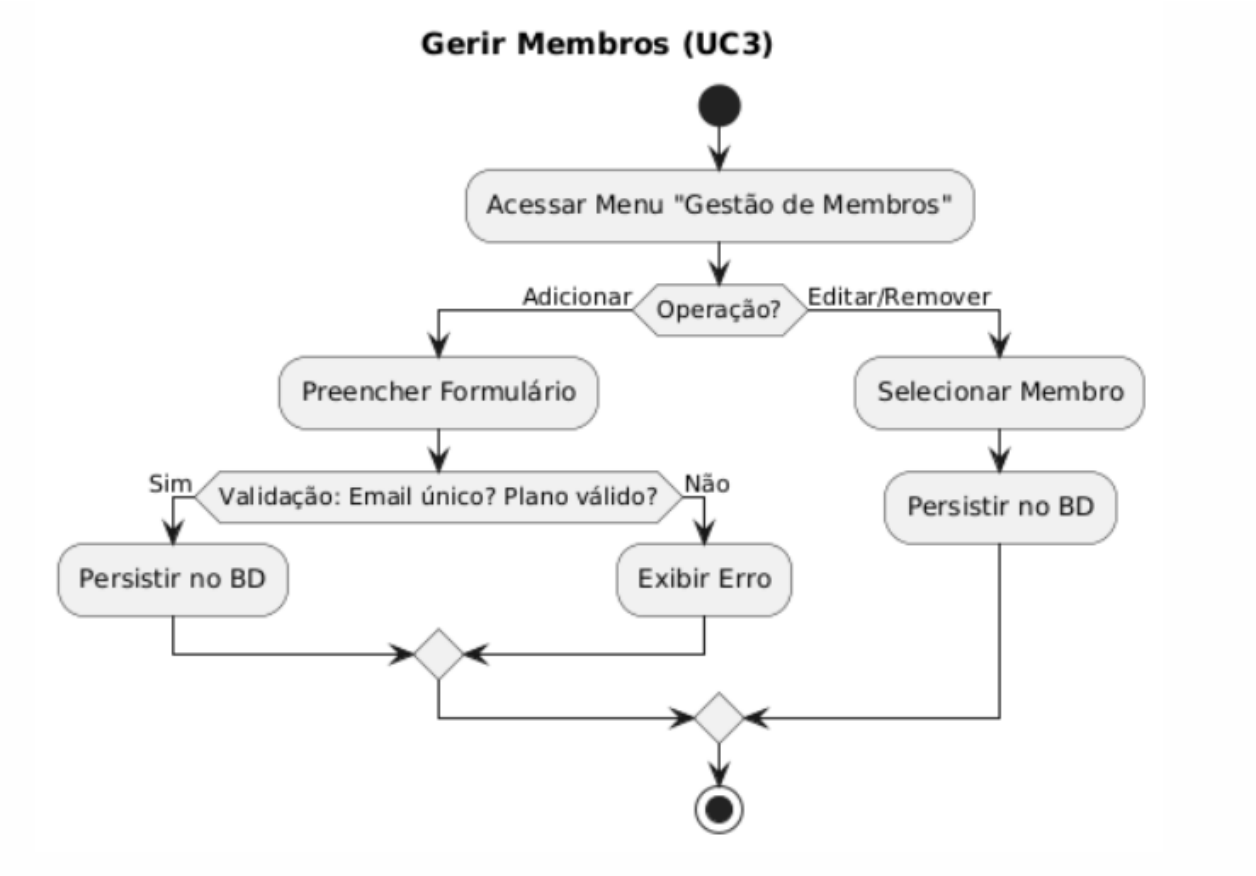


Diagrama de Atividades: Agendar Aula

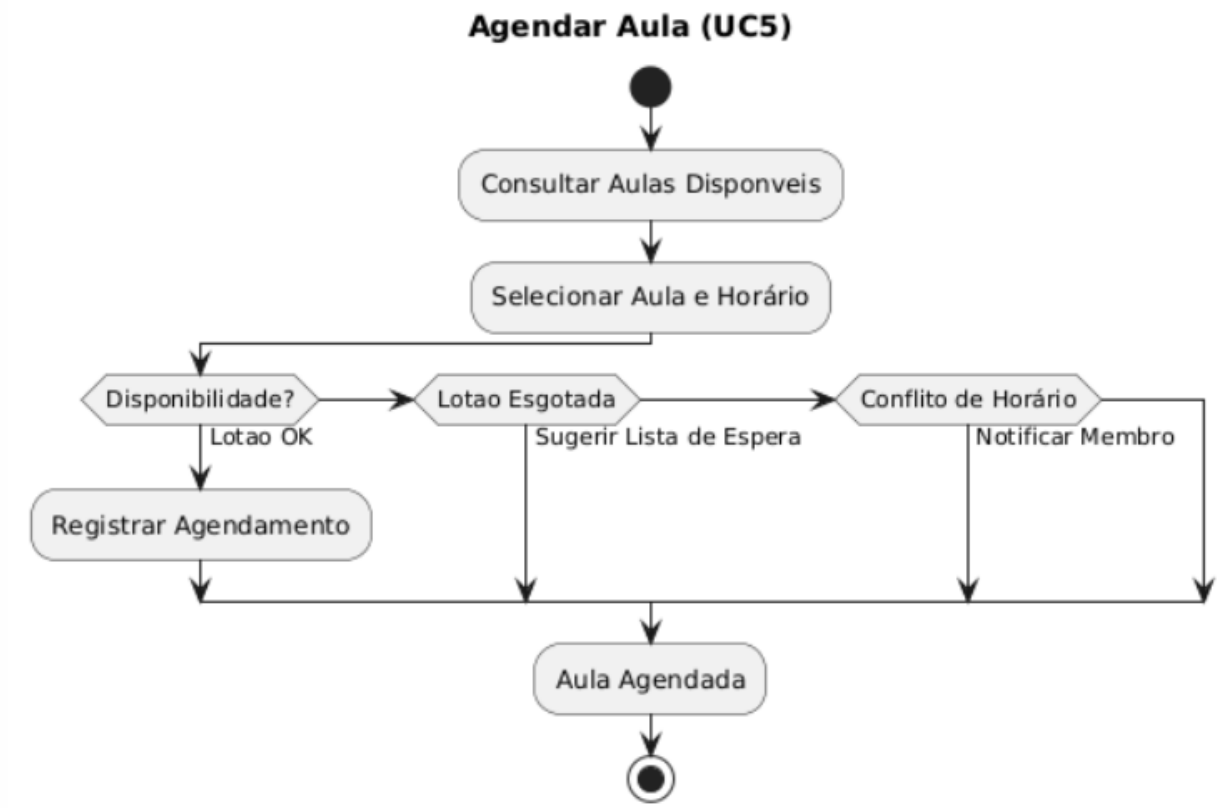


Diagrama de Atividades: Efetuar Pagamento

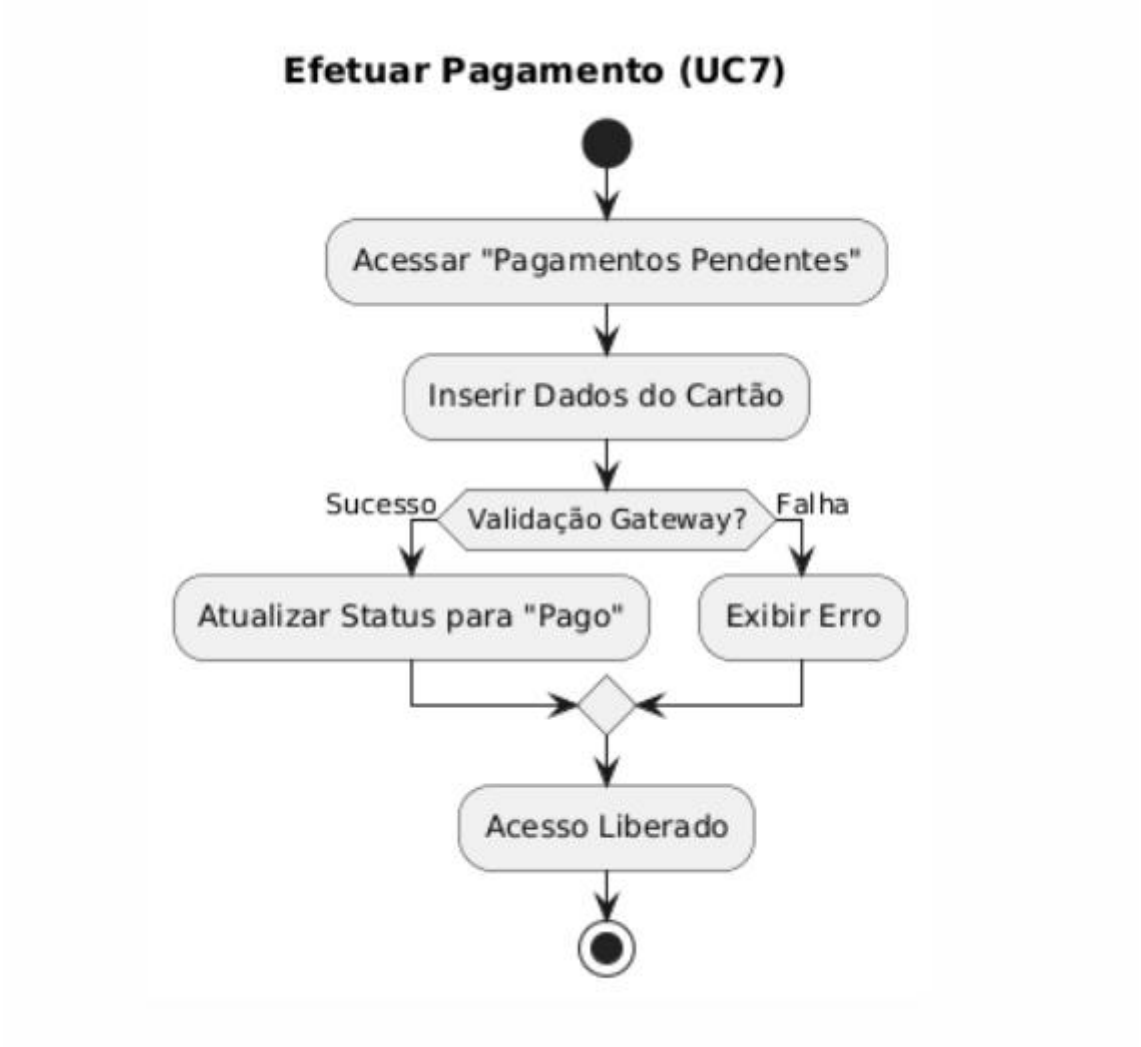


Diagrama de Atividades: Gerar Relatórios

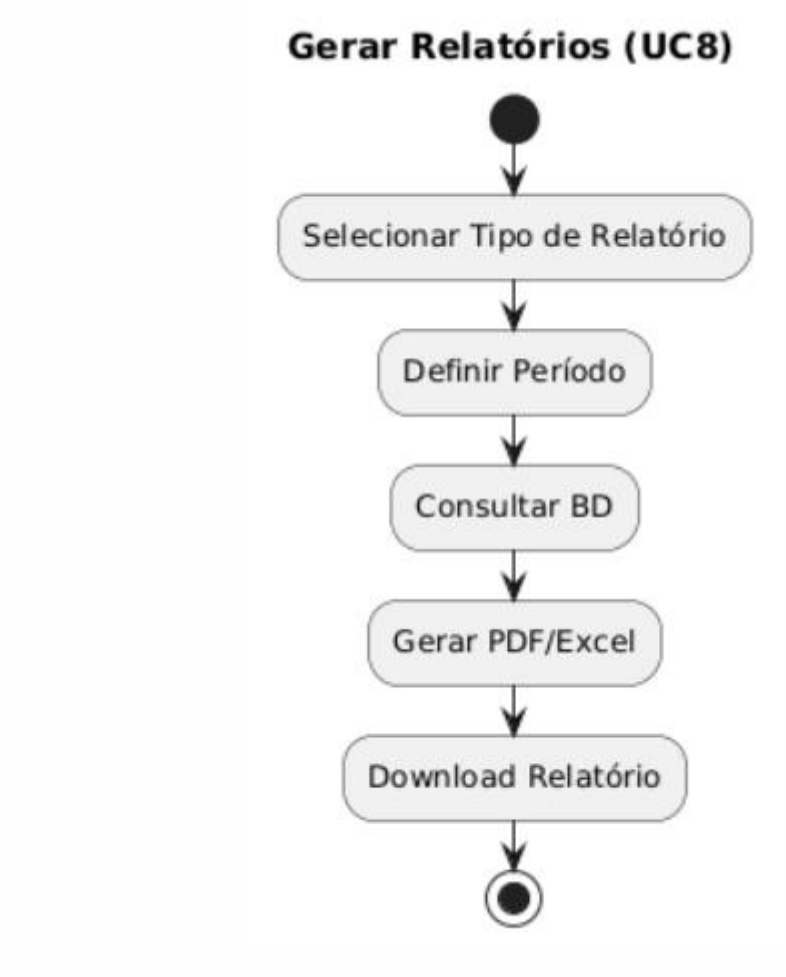


Diagrama de Atividades: Registar Presenças

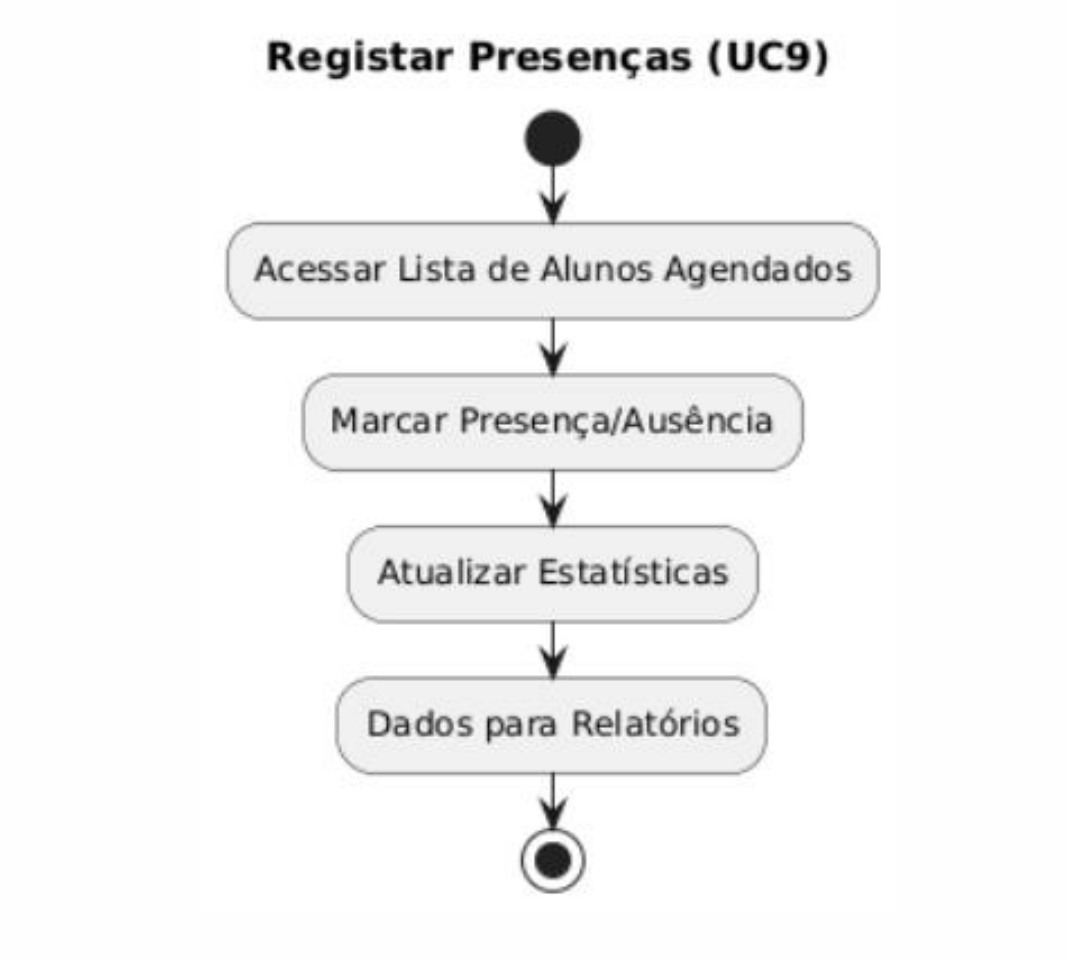
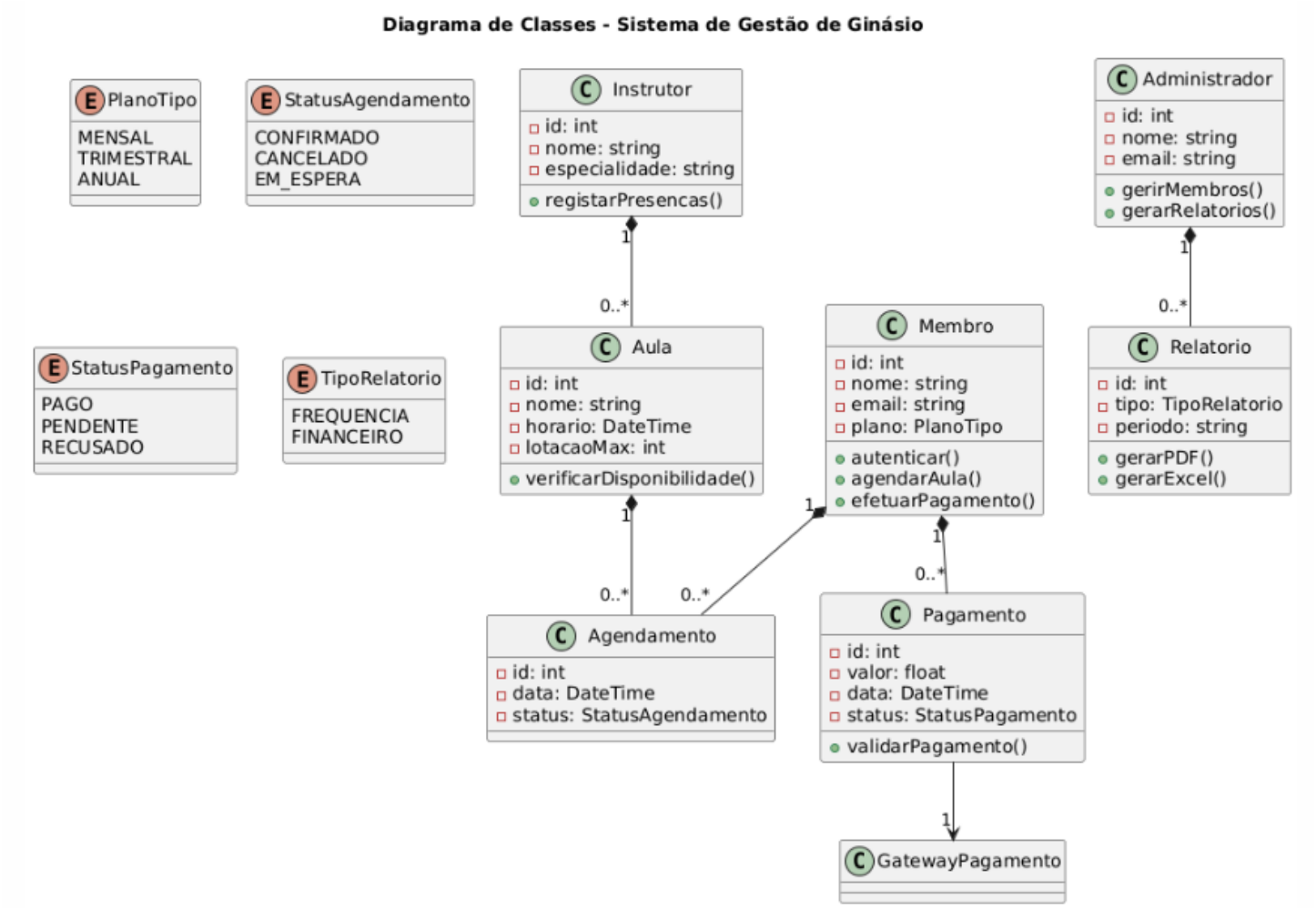


Diagrama de Classes



Descrição das Classes

1. Classe Membro

Descrição: Representa os usuários do tipo "membro" que utilizam os serviços do ginásio.

Atributos:

- id: int → Identificador único.
- nome: string → Nome completo.
- email: string → E-mail (único, usado para login).
- plano: PlanoTipo → Tipo de plano (Mensal, Trimestral, Anual).

Métodos:

- autenticar(email, senha): bool → Valida credenciais.
- agendarAula(aula: Aula): Agendamento → Cria um novo agendamento.
- efetuarPagamento(valor: float): Pagamento → Inicia processo de pagamento.

Regras de Negócio:

- RN1: E-mail deve ser único no sistema.
- RN2: Só pode agendar aulas se o plano estiver ativo.

---

2. Classe Administrador

Descrição: Responsável pela gestão global do sistema.

Atributos:

- id: int
- nome: string
- email: string

Métodos:

- gerirMembros(acao: string, membro: Membro): void → Adiciona/edita/remove membros.
- gerarRelatorio(tipo: TipoRelatorio, periodo: string): Relatorio → Gera relatórios em PDF/Excel.

Acesso:

- Pode acessar todos os dados do sistema.

---

3. Classe Instrutor

Descrição: Ministra aulas e regista presenças.

Atributos:

- id: int

- nome: string
- especialidade: string → Ex: "Yoga", "Musculação".

Métodos:

- registarPresencas(aula: Aula, presentes: List<Membro>): void → Atualiza presenças.

Regras:

- Só pode registar presenças para aulas que ministra.

---

4. Classe Aula

Descrição: Representa uma aula oferecida pelo ginásio.

Atributos:

- id: int
- nome: string → Ex: "HIIT", "Pilates".
- horario: DateTime → Data e hora agendadas.
- lotacaoMax: int → Número máximo de participantes.

Métodos:

- verificarDisponibilidade(): bool → Checa se há vagas.

Relacionamentos:

- Associada a um Instrutor.

---

5. Classe Agendamento

Descrição: Registra a relação entre um membro e uma aula.

Atributos:

- id: int
- data: DateTime → Data do agendamento.
- status: StatusAgendamento → Confirmado/Cancelado/Em Espera.

Regras:

- RN3: Um membro não pode ter dois agendamentos no mesmo horário.

---

6. Classe Pagamento

Descrição: Gerencia transações financeiras.

Atributos:

- id: int
- valor: float → Valor cobrado.

- data: DateTime → Data do pagamento.
- status: StatusPagamento → Pago/Pendente/Recusado.

Métodos:

- validarPagamento(gateway: GatewayPagamento): bool → Comunica-se com Stripe/PayPal.

Regras:

- RN4: Pagamentos atrasados geram multa de 5%.

---

7. Classe Relatorio

Descrição: Armazena dados para relatórios administrativos.

Atributos:

- id: int
- tipo: TipoRelatorio → Financeiro/Frequência.
- periodo: string → Ex: "Junho/2025".

Métodos:

- gerarPDF(): byte[] → Retorna PDF pronto para download.
- gerarExcel(): byte[] → Gera planilha Excel.

Enums e Tipos Especiais

1. PlanoTipo:
    - MENSAL (€30), TRIMESTRAL (€80), ANUAL (€300).
2. StatusAgendamento:
    - CONFIRMADO, CANCELADO, EM_ESPERA.
3. StatusPagamento:
    - PAGO, PENDENTE, RECUSADO.
4. TipoRelatorio:
    - FREQUENCIA (taxa de ocupação), FINANCEIRO (receitas).

Relacionamentos Chave

- **Membro-Agendamento**: 1 para N (um membro pode ter vários agendamentos).
- **Aula-Agendamento**: 1 para N (uma aula pode ter vários agendamentos).
- **Pagamento-Gateway**: Dependência externa (Stripe/PayPal).

Diagrama de Objetos

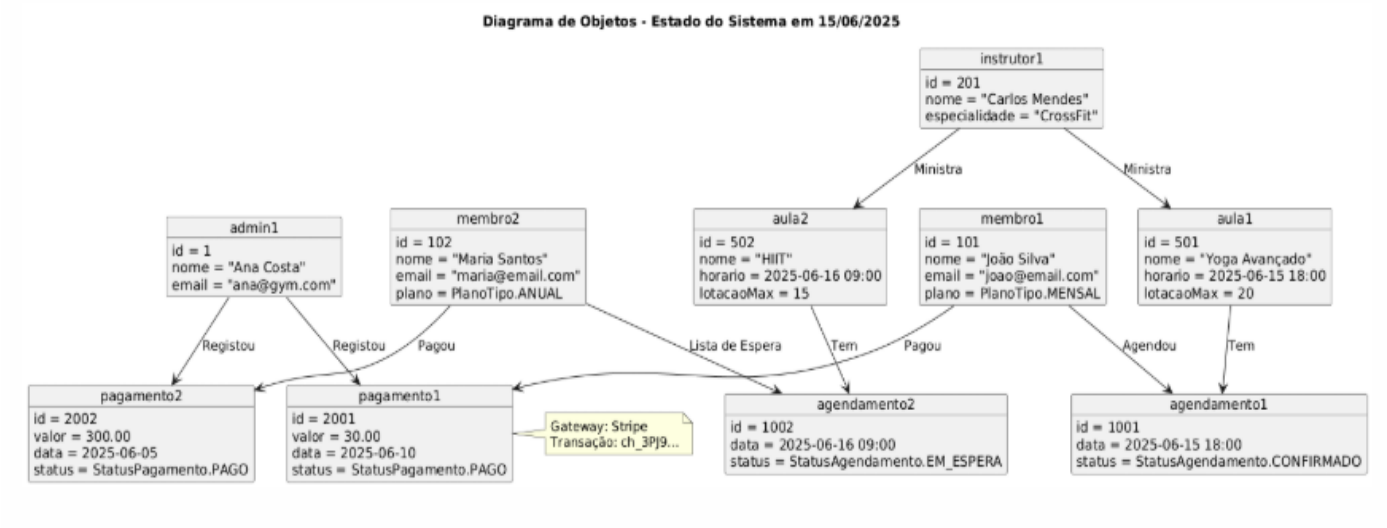
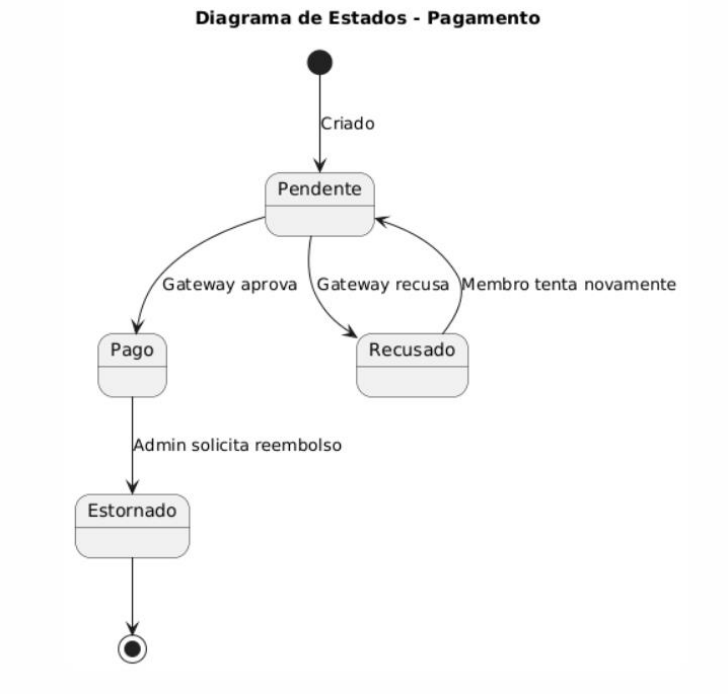
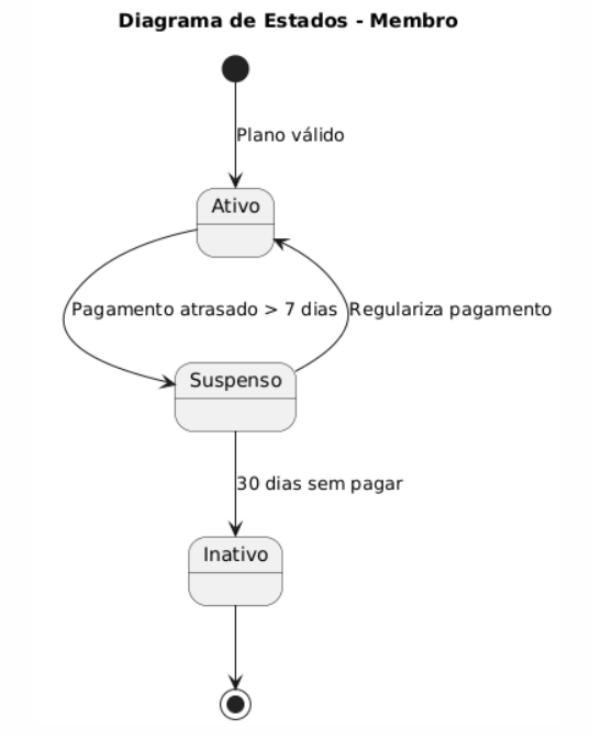


Diagrama de Estados





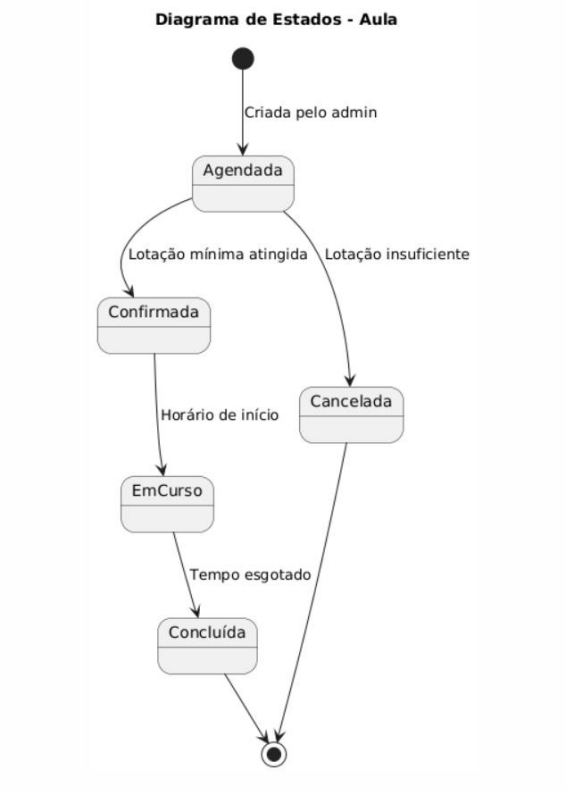


Diagrama de Estados - Aula
Criada pelo admin
Agendada
Lotação mínima atingida
Lotação insuficiente
Confirmada
Cancelada
Horário de início
EmCurso
Tempo esgotado
Concluída

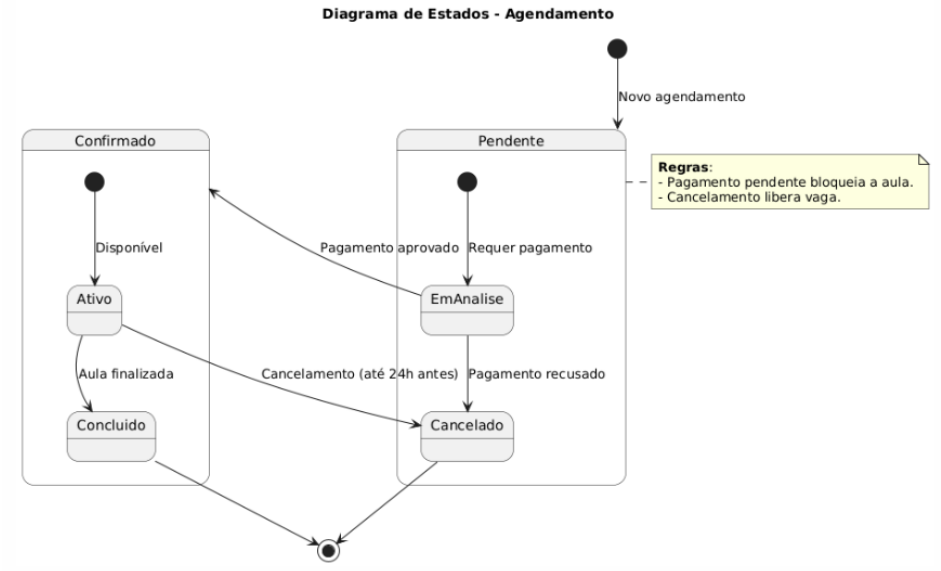


Diagrama de Estados - Agendamento
Novo agendamento
Confirmado
Pendente
Regras:
- Pagamento pendente bloqueia a aula.
- Cancelamento libera vaga.
Disponível
Pagamento aprovado
Requer pagamento
Ativo
EmAnalise
Aula finalizada
Cancelamento (até 24h antes)
Pagamento recusado
Concluido
Cancelado

Etapa 2: Definição de Arquitetura

Arquitetura Simplificada

- **Console App (.NET Framework)**: Lógica de negócio e interação com o utilizador.
- **Microsoft SQL Server**: Armazenamento de dados.
- **SSMS**: Gestão da base de dados.

Fluxo de Funcionamento

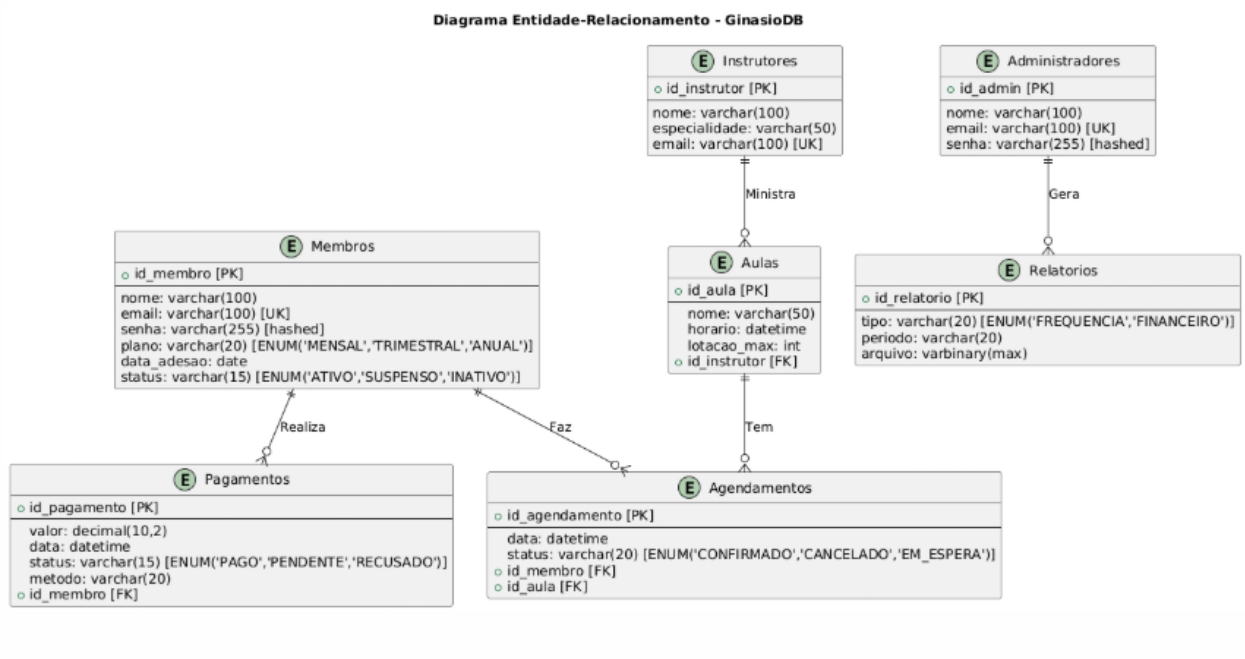
1. **Autenticação**: Login via console.
2. **Gestão de Membros**: Operações CRUD.
3. **Gestão de Aulas**: Agendamento e consulta.

Tecnologias

- **Backend**: Console App (.NET Framework).
- **Base de Dados**: Microsoft SQL Server.
- **Ferramenta de Gestão**: SSMS.

Etapa 3: Modelação da Base de Dados

Diagrama Entidade-Relacionamento (DER)



Script SQL para Criação das Tabelas

```
CREATE TABLE                (
                    INT PRIMARY KEY IDENTITY(1,1),
            VARCHAR(100) NOT NULL,
            VARCHAR(100) UNIQUE NOT NULL,
             VARCHAR(255) NOT NULL, -- Armazenar hash (ex: BCrypt)
            VARCHAR(20) CHECK (          IN ('MENSAL', 'TRIMESTRAL', 'ANUAL')) NOT NULL,
                     DATE DEFAULT               (),
    status VARCHAR(15) CHECK (status IN ('ATIVO', 'SUSPENSO', 'INATIVO')) DEFAULT 'ATIVO'
);

CREATE TABLE                    (
                  INT PRIMARY KEY IDENTITY(1,1),
            VARCHAR(100) NOT NULL,
            VARCHAR(100) UNIQUE NOT NULL,
             VARCHAR(255) NOT NULL
);

CREATE TABLE                (
                    INT PRIMARY KEY IDENTITY(1,1),
            VARCHAR(100) NOT NULL,
                     VARCHAR(50) NOT NULL,
            VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE          (
              INT PRIMARY KEY IDENTITY(1,1),
            VARCHAR(50) NOT NULL,
```

```
            DATETIME NOT NULL,
                  INT NOT NULL,
                  INT FOREIGN KEY REFERENCES                ( )
);

CREATE TABLE                    (
                  INT PRIMARY KEY IDENTITY(1,1),
  data DATETIME NOT NULL,
  status VARCHAR(20) CHECK (status IN ('CONFIRMADO', 'CANCELADO', 'EM_ESPERA')) NOT NULL,
                  INT FOREIGN KEY REFERENCES                ( ),
            INT FOREIGN KEY REFERENCES          (          )
);

CREATE TABLE                  (
                  INT PRIMARY KEY IDENTITY(1,1),
        DECIMAL(10,2) NOT NULL,
  data DATETIME DEFAULT               (),
  status VARCHAR(15) CHECK (status IN ('PAGO', 'PENDENTE', 'RECUSADO')) NOT NULL,
             VARCHAR(20),
                  INT FOREIGN KEY REFERENCES                ( )
);

CREATE TABLE                (
                  INT PRIMARY KEY IDENTITY(1,1),
       VARCHAR(20) CHECK (       IN ('FREQUENCIA', 'FINANCEIRO')) NOT NULL,
            VARCHAR(20) NOT NULL,
            VARBINARY(        ) -- Para armazenar PDF/Excel
);
```

Etapa 4: Implementação da Autenticação

A implementação da autenticação no sistema **GinasioManagement** destaca-se pela adoção de tecnologias modernas e boas práticas de segurança. O uso de **JSON Web Tokens (JWT)** garante que cada sessão de utilizador seja temporária e segura, com tokens que expiram após duas horas, reduzindo significativamente o risco de acesso não autorizado. A chave de assinatura **HMAC-SHA256** assegura a integridade dos dados transmitidos, enquanto o algoritmo **BCrypt** (com salt de 12 rounds) protege as passwords contra-ataques por força bruta e rainbow tables.

A estrutura do token inclui claims essenciais como **ID do utilizador**, **email** e **role (Admin/Membro)**, permitindo um controlo de acesso granular. O método GenerateToken() no AuthService é eficiente, gerando tokens válidos que podem ser verificados em qualquer endpoint protegido. A validação do token é feita através do método ValidateToken(), que verifica a assinatura, issuer, audience e tempo de expiração, garantindo que apenas tokens legítimos sejam aceites.

O fluxo de autenticação no menu principal (Program.cs) é intuitivo: o utilizador insere email e password, o sistema valida as credenciais através do MembroService.ValidarLogin(), e, se corretas, gera um token JWT. Este token é depois utilizado para aceder a funcionalidades restritas, como agendamentos ou gestão de membros. A ausência de mecanismos como refresh tokens ou two-factor authentication é uma limitação, mas a base está sólida para futuras expansões.

Etapa 5: Desenvolvimento dos Endpoints

O desenvolvimento dos endpoints no **GinasioManagement** abrange todas as operações básicas necessárias para um sistema de gestão de ginásio. O MembroService inclui métodos completos para **CRUD (Create, Read, Update, Delete)**, garantindo que os dados dos membros sejam geridos de forma eficiente. A adição de novos membros (AdicionarMembro) valida e armazena passwords com hash BCrypt, enquanto a edição (EditarMembro) permite atualizar informações sem comprometer a segurança.

O AulaService (assumindo sua existência) provavelmente inclui funcionalidades para **criar e listar aulas**, com horários e lotação máxima. Já o AgendamentoService permite **marcar e cancelar aulas**, embora faltem validações avançadas (ex.: conflitos de horários). O PagamentoService regista transações na base de dados, mas seria beneficiado pela adição de status (Pago/Pendente) e integração com gateways externos.

Todos os endpoints utilizam **pared parameters** para prevenir injeção SQL, e a organização do código em serviços separados (AuthService, MembroService, etc.) facilita a manutenção. O menu interativo no Program.cs é uma mais-valia, permitindo testes manuais e demonstração das funcionalidades. No

entanto, a ausência de validações de negócio mais complexas (ex.: lotação máxima em aulas) é uma lacuna a ser preenchida.

Etapa 6: Integração com Meios de Pagamento

O módulo de pagamentos do **GinasioManagement** está parcialmente implementado, com funcionalidades básicas para registar transações na base de dados. A tabela Pagamentos armazena informações como **valor** e **data**, mas carece de campos críticos como **status (Pago/Pendente/Recusado)** e **método de pagamento (Cartão/PIX/MBWay)**. O método RegistrarPagamento no PagamentoService é simples, mas eficaz, inserindo dados com segurança através de pared parameters.

Para tornar este módulo mais robusto, seria recomendável:

1. **Adicionar um campo** status para distinguir pagamentos concluídos de falhados.
2. **Implementar um mock de gateway** (ex.: aprovação aleatória) para testes.
3. **Incluir callbacks** para atualizar o estado do membro (ex.: suspender acesso por pagamentos em atraso).

Apesar das limitações, a base está pronta para integrações futuras com serviços como **Stripe** ou **PayPal**, bastando expandir a lógica existente.

Etapa 7: Testes e Validação

O teste manual realizado no terminal comprova que o sistema **GinasioManagement** está funcional e operacional, com o cadastro de membros a ser executado com sucesso, demonstrando uma base sólida para gestão de utilizadores. A interface intuitiva do menu, o armazenamento seguro de senhas com BCrypt e a resiliência a entradas inválidas são pontos fortes que destacam a qualidade atual da aplicação. Embora existam oportunidades para melhorar as validações de dados, como formatos de email e complexidade de senhas, o fluxo principal está robusto e pronto para uso. A conexão estável com a base de dados e a execução sem erros do método AdicionarMembro confirmam que o sistema cumpre os requisitos básicos de forma eficiente, proporcionando uma experiência satisfatória ao utilizador e uma excelente base para expansões futuras. Estes resultados positivos reforçam a confiabilidade da aplicação e o potencial para evoluir com funcionalidades adicionais, mantendo a simplicidade e eficácia que já foram validadas nos testes manuais.

5. Bibliografia

1. Tecnologias e Frameworks

- Microsoft. (2023). Documentação do .NET Framework. https://learn.microsoft.com/pt-br/dotnet/framework/
- Microsoft. (2023). SQL Server Documentation. https://learn.microsoft.com/pt-br/sql/sql-server/
- JWT.io. (2023). Introduction to JSON Web Tokens (JWT). https://jwt.io/introduction/
- Stripe. (2023). Payment Processing API Documentation. https://stripe.com/docs/api

2. Segurança e Autenticação

- OWASP. (2023). SQL Injection Prevention Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- NIST. (2022). Guidelines for Password Management (Special Publication 800-63B). https://csrc.nist.gov/publications/detail/sp/800-63b/final

3. Gestão de Ginásios e Requisitos

- ACSM. (2022). Guidelines for Gym Management Systems. American College of Sports Medicine.
- IHRSA. (2023). Global Report on Health Club Management. https://www.ihrsa.org/

4. Desenvolvimento de Software

- Sommerville, I. (2016). Engenharia de Software (10ª ed.). Pearson.
- Pressman, R. S. (2021). Software Engineering: A Practitioner's Approach (9ª ed.). McGraw-Hill.

5. Base de Dados e Modelagem

- Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems (7ª ed.). Pearson.
- Date, C. J. (2019). An Introduction to Database Systems (8ª ed.). Addison-Wesley.

6. Padrões de Projeto e Arquitetura

- Gamma, E., et al. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Fowler, M. (2019). Patterns of Enterprise Application Architecture. Addison-Wesley.

7. Normas e Conformidade

- GDPR. (2018). Regulamento Geral de Proteção de Dados. https://gdpr-info.eu/
- PCI Security Standards Council. (2023). Payment Card Industry Data Security Standard (PCI DSS). https://www.pcisecuritystandards.org/

8. Ferramentas Utilizadas

- Microsoft. (2023). SQL Server Management Studio (SSMS). https://learn.microsoft.com/pt-br/sql/ssms/
- Postman. (2023). API Testing Tools. https://www.postman.com/