Especificação de um sistema backend do tipo REST API para o gerenciamento de um **Restaurante Universitário (RU)**.

A especificação contempla **visão geral, atores, casos de uso, entidades, regras de negócio, fluxos, estrutura de dados**, e respeita princípios de **Clean Architecture**, **DDD** e foco no **desenvolvimento com boas práticas**.

Nome do Projeto

Sistema de Gerenciamento do Restaurante Universitário (RU-API)

Visão Geral

Trata-se de uma **REST API backend** para suportar as operações do restaurante universitário de uma instituição pública. A API oferece serviços para:

- Gerenciamento de agendamentos e comparecimentos a refeições (almoço e jantar).
- Controle de créditos de refeição por semestre letivo.
- Registro de avaliações pelos usuários.
- Administração de opções de cardápio e votações.
- Autenticação e autorização de usuários com perfis distintos.

A API será consumida por:

- Um **aplicativo móvel** (usuários finais).
- Um **painel web administrativo** (equipe do RU).

Atores do Sistema

Ator	Descrição
Usuário Comum (Estudante ou Servidor)	Realiza agendamentos, votações, avaliações e consultas.

Ator	Descrição
Administrador	Gera relatórios, publica cardápios, acompanha agendamentos, penalidades e estatísticas.
Sistema	Automatiza a liberação de créditos, aplicação de penalidades, encerramento de votações.

Casos de Uso Principais

1. Autenticar Usuário

- Login e geração de token JWT.
- Controle de acesso por perfis (`USER` , `ADMIN`).

2. Agendar Refeição

- Permitir que o usuário agende almoço e/ou jantar em uma data futura.
- O sistema valida se o usuário possui créditos disponíveis.

3. Registrar Presença

- No dia da refeição, o usuário registra sua presença (via token, QR Code ou login).
- A presença valida o consumo do crédito agendado.

4. Aplicar Penalidade por Ausência

- Se o usuário agendou e não registrou presença, é penalizado automaticamente.
- Penalidades podem bloquear agendamentos futuros ou reduzir créditos.

5. Publicar Opções de Cardápio

- O administrador publica opções para almoço e jantar com antecedência.

6. Votar no Cardápio

- O usuário escolhe entre as opções disponíveis para a data futura.
- O sistema encerra as votações automaticamente antes da data da refeição.

7. Avaliar Refeição e Serviço

- Após a presença, o usuário pode avaliar a refeição do turno consumido.

8. Gerenciar Créditos

- A cada semestre, usuários recebem uma cota de refeições.
- Créditos são debitados apenas após comparecimento ou penalizados por ausência.

9. Consultar Relatórios

- O administrador pode consultar relatórios de:
  - Comparecimento diário
  - Agendamentos por turno
  - Médias de avaliação
  - Penalidades aplicadas

Modelagem de Domínio (Entidades)

`User`

- `id: UUID`
- `name: string`
- `email: string`
- `passwordHash: string`
- `role: 'USER' | 'ADMIN'`
- `enrollment: string` (matrícula)
- `creditBalance: number`
- `penaltyCount: number`

`Meal`

- `id: UUID`
- `date: Date`
- `shift: 'LUNCH' | 'DINNER'`
- `menu: Menu`

Schedule

- id: UUID
- userId: UUID
- mealId: UUID
- status: 'SCHEDULED' | 'CANCELLED' | 'PRESENT' | 'ABSENT'
- registeredAt: Date | null

Menu

- id: UUID
- mealId: UUID
- description: string
- finalized: boolean

MenuOption

- id: UUID
- menuId: UUID
- description: string
- voteCount: number

MenuVote

- id: UUID
- userId: UUID
- menuOptionId: UUID

Evaluation

- id: UUID
- userId: UUID
- mealId: UUID
- ratingFood: number (1–5)
- ratingService: number (1–5)
- comments: string | null

`Penalty`

- `id: UUID`
- `userId: UUID`
- `scheduleId: UUID`
- `reason: 'NO_SHOW' | 'OTHER'`
- `appliedAt: Date`

Regras de Negócio

1. O usuário só pode agendar uma refeição por turno por dia.
2. O agendamento deve ser feito com até `X` horas de antecedência.
3. O agendamento é descontado apenas se houver presença ou ausência injustificada.
4. Penalidades são aplicadas por ausência não justificada e podem:
    - Impedir novos agendamentos por `Y` dias.
    - Reduzir a cota de refeições futuras.
5. Avaliações só podem ser feitas por quem **compareceu** à refeição.
6. Votações no cardápio encerram-se `N` horas antes da data da refeição.

Endpoints REST (exemplos)

Auth

- `POST /auth/signup`
- `POST /auth/signin`

Agendamentos

- `GET /meals/:date`
- `POST /schedules`
- `PATCH /schedules/:id/cancel`
- `PATCH /schedules/:id/confirm`

Presença

- `POST /presences/:scheduleId`

Penalidades

- `GET /users/:id/penalties`

Cardápios

- `POST /menus`
- `POST /menus/:id/options`
- `POST /menus/:id/vote`

Avaliações

- `POST /evaluations`
- `GET /evaluations/summary`

Créditos

- `GET /users/:id/credits`
- `POST /credits/reload`

Relatórios (admin)

- `GET /reports/attendance`
- `GET /reports/evaluations`
- `GET /reports/menu-results`

Arquitetura e Camadas

1. Domínio (`domain/`)

- Entidades, Value Objects, Interfaces de Repositório
- Validações de invariantes
- Lógica de regras puras

2. Aplicação (`application/`)

- Casos de uso (ex: `CreateSchedule` , `VoteInMenu`)
- DTOs, orchestradores, adaptação de regras de negócio

3. Infraestrutura (`infrastructure/`)

- Persistência com ORM (Prisma, TypeORM)
- Implementações dos repositórios
- Serviços externos (notificações, autenticação)

4. Interfaces (`interfaces/`)

- Controllers REST
- Middlewares de autenticação e autorização
- Validações e serializações

Testabilidade com TDD

- Casos de uso testados com **Jest** e **mocks injetáveis**
- Separação clara entre regras puras (unit tests) e adaptadores (integration tests)
- Testes de regressão e de falhas para penalidades e agendamentos