

FLUXO DEFINITIVO DO YRKE (COM TERMO E ENVIO AO GESTOR)

1. A cria o pedido

Acessa o calendário → seleciona plantão próprio.

Preenche:

- Data e turno
- Destinatário B
- Observação (opcional)

Estado: Pendente (Aguardando B)

2. B recebe e analisa

B vê:

- Plantão de A
- Observação
- Informações completas

Ao clicar em Aceitar, B deve informar:

- Data + turno do plantão que B oferece em troca

O sistema então gera automaticamente o Termo de Permuta, preenchido com:

- Nome A
- Nome B
- Função
- Matrículas (ou IDs)
- Plantão de A
- Plantão de B
- Motivo/Observação
- Regras formais da permuta

- Data atual
-

3. B visualiza o termo e aceita

B lê o termo completo e então clica:

- "Concordo com o Termo e Aceito a Troca"

Estado: Aguardando Confirmação de A

4. A revisa e confirma

A recebe a resposta de B, e vê:

- Seu plantão
- Plantão de B
- Termo de Permuta completo e preenchido

A então escolhe:

- "Confirmo o Termo e a Troca"
 "Cancelar"

Se confirmar:

Estado: Aguardando envio ao gestor

5. Coleta do contato do gestor

O sistema solicita:

- E-mail ou telefone WhatsApp do gestor

(Um gestor por usuário — sempre o mesmo no MVP.)

6. Envio do termo ao gestor

Depois de A confirmar e informar o contato:

O sistema:

- Constrói automaticamente o Termo em texto formatado

- Abre WhatsApp com o termo colado
ou
- Abre E-mail com o termo no corpo

Gestor recebe o documento completo.

Estado: Concluída

7. Histórico

Após a conclusão, o sistema salva:

- Dados da troca
- Conteúdo completo do Termo
- Momento em que B aceitou
- Momento em que A confirmou

Histórico:

- Imutável
 - Exibe trocas solicitadas e recebidas
-

8. Pesquisa automática

A cada duas trocas concluídas, o sistema apresenta pesquisa:

- Dificuldade da troca
- Se indicaria o sistema
- Nota de 1 a 5

Resultados são armazenados.

🔥 Fluxo final: resumo simples

- 1 A solicita
- 2 B aceita e informa plantão → termo gerado
- 3 B aceita termo
- 4 A confirma termo

- 5** A informa contato do gestor
- 6** Sistema envia termo via WhatsApp/E-mail
- 7** Histórico salva tudo
- 8** Pesquisa a cada 2 trocas

◆ 1. OBJETIVO DO DOCUMENTO

Definir todos os requisitos necessários para desenvolver o **MVP do sistema Yrke**, com clareza e precisão, garantindo alinhamento total entre fluxo, domínio e arquitetura.

◆ 2. ESCOPOS

- Incluído no MVP
 - Não incluído (futuras versões)
 - Regras de Negócio
 - Critérios de Aceite
-

✳ 3. REQUISITOS FUNCIONAIS (RF) – MVP

RF-01 – Cadastro de Usuário

O sistema deve permitir que o usuário crie uma conta informando:

- Nome completo
 - E-mail
 - Telefone
 - Senha
 - Função (selecção a partir de lista fixa)
 - Tipo de escala (ex.: 12x36, 6x1, etc.)
-

RF-02 – Login

O sistema deve permitir login via:

- E-mail + senha
- Conta Google (opcional)

RF-03 – Recuperação de Senha

O sistema deve permitir redefinir senha via e-mail.

RF-04 – Cadastro Automático do Calendário

Após o cadastro do tipo de escala, o sistema deve gerar automaticamente o calendário do **mês atual** com base no padrão informado.

Usuário poderá visualizar apenas seus plantões.

RF-05 – Visualização do Calendário

O usuário deve visualizar:

- Plantões carregados automaticamente
 - Plantões futuros
 - Identificação clara de dias e turnos
-

RF-06 – Criar Pedido de Troca

O usuário A deve poder:

- Selecionar seu plantão
- Selecionar outro usuário B da mesma função
- Inserir observação (opcional)
- Enviar solicitação

Estado inicial: **Pendente**

RF-07 – Aceitar Pedido de Troca (Usuário B)

B deve visualizar:

- Plantão de A
- Observação (se houver)

Se aceitar:

- Deve informar o **plantão dele** oferecido em troca

O sistema deve então gerar automaticamente o **Termo de Permuta**.

RF-08 – Exibir Termo para B

O sistema deve gerar e exibir o **Termo de Permuta**, com:

- Nome A e B
- Função
- IDs/matrículas
- Plantões envolvidos
- Observações
- Data atual
- Cláusulas formais

B deve clicar em:

✓ "Concordo com o termo e aceito a troca"

ou

✗ "Cancelar"

RF-09 – Exibir Termo para A

Após B aceitar, A deve visualizar:

- Plantões envolvidos
- Termo de Permuta preenchido

A deve confirmar:

✓ "Confirmo o termo e a troca"

ou

✗ "Cancelar"

RF-10 – Coleta do Gestor

Após confirmação de A, o sistema deve solicitar:

- E-mail OU WhatsApp do gestor
-

RF-11 – Envio do Termo ao Gestor

O sistema deve:

- Gerar o termo completo em **texto formatado**
 - Abrir:
 - WhatsApp com o termo colado
 - ou e-mail com o termo no corpo
 - Não usar API de WhatsApp (envio manual)
-

RF-12 – Registro do Histórico

O sistema deve registrar trocas com:

- Plantão de A
- Plantão de B
- Datas e turnos
- Aceite de B
- Confirmação de A
- Termo completo
- Timestamp de tudo
- Status final (Concluído)

Histórico deve ser:

- Imutável
 - Exibido tanto para A quanto para B
-

RF-13 – Pesquisa Pós-Troca

A cada duas trocas concluídas, o sistema deve exibir:

- Dificuldade da troca
- Se indicaria o sistema
- Nota de 1 a 5

Respostas devem ser armazenadas.

4. REGRAS DE NEGÓCIO (RN)

RN-01 – Troca apenas entre mesma função

O usuário só pode solicitar troca com outro que tenha **mesma função**.

RN-02 – Histórico imutável

Nenhuma troca pode ser excluída ou editada após concluir.

RN-03 – Termo obrigatório

A troca só é válida após:

1. B aceitar o termo
 2. A confirmar o termo
-

RN-04 – Envio ao gestor obrigatório

A troca só é concluída após A enviar o termo ao gestor.

RN-05 – Um gestor por usuário

Cada usuário deve ter apenas **um gestor** (informado no momento da troca).

RN-06 – Observação não obrigatória

Observações são opcionais.

RN-07 – Registro da dupla permuta

Trocas devem ser sempre bilaterais:

A dá o plantão X

B dá o plantão Y

RN-08 – Pesquisa a cada 2 trocas

Pesquisa é acionada automaticamente.

5. REQUISITOS NÃO FUNCIONAIS (RNF)

RNF-01 – Segurança

- Armazenar senhas criptografadas
 - Uso de HTTPS
 - Evitar dados sensíveis desnecessários
 - LGPD: permitir visualizar, exportar, mas não excluir histórico
-

RNF-02 – Desempenho

- Carregar calendário em < 2 segundos
 - Solicitação de troca em operações assíncronas
-

RNF-03 – Escalabilidade

- Banco relacional (SQL Server)
 - EF Core
 - Estrutura preparada para futuros módulos
-

RNF-04 – Usabilidade

- Layout mobile-first
 - Interfaces simples (usuário intermediário)
-

RNF-05 – Disponibilidade

- MVP hospedado em ambiente simples (GitHub Pages + backend hospedado em serviço cloud econômico)
 - Futuro: Azure
-

6. CRITÉRIOS DE ACEITE (CA)

Para uma troca ser concluída:

- ✓ Pedido criado
 - ✓ B aceitou e informou plantão
 - ✓ Termo exibido e aceito por B
 - ✓ Termo confirmado por A
 - ✓ Gestor recebeu o termo (via WhatsApp/e-mail)
 - ✓ Transferência salva no histórico
 - ✓ Pesquisa acionada conforme regra
-

7. ESCOPOS FUTUROS (V2, V3...)

V2

- Geração de PDF do Termo
- Dashboard básico
- Painel do gestor
- Histórico filtrável
- Anexar justificativas/fotos

V3

- App mobile nativo
- Assinatura digital avançada
- Integração WhatsApp API oficial
- Controle de banco de horas
- Múltiplos gestores
- Chat interno

ARQUITETURA TÉCNICA DO SISTEMA YRKE (ASP.NET MVC + EF Core)

Versão Sênior, completa e justificável

A arquitetura está desenhada para:

- Simplicidade e velocidade no MVP
- Segurança e robustez
- Crescimento futuro (Clean Architecture-ready)
- Baixo acoplamento + alta coesão

- Testabilidade
-

◆ 1. Visão Geral da Arquitetura

A solução adotará uma arquitetura em **camadas**, com elementos fortemente inspirados em **Clean Architecture**, porém com simplicidade de MVC para o MVP.

Camadas Principais

1. **Presentation (ASP.NET MVC)**
 2. **Application Services (Services)**
 3. **Domain (Entidades + Regras)**
 4. **Infrastructure (Repository + EF Core + SQL Server)**
-

◆ 2. Organização do Projeto (Pastas)

/src

/Yrke.Web -> ASP.NET MVC (Controllers, Views, ViewModels)

/Yrke.Application -> Services, DTOs, Mappers

/Yrke.Domain -> Entidades, Regras, Interfaces

/Yrke.Infrastructure -> EF Core, Repositórios, Config, Migrations

/Yrke.Tests -> Testes futuros

◆ 3. Responsabilidades de Cada Camada

3.1 Presentation — ASP.NET MVC

- Controllers
- ViewModels (somente dados de exibição e input)
- Autenticação/Autorização
- Geração do texto do Termo
- Redirecionamento para WhatsApp/e-mail

Não contém regra de negócio.

3.2 Application — Services

- Contém toda a lógica de aplicação
 - Orquestra casos de uso:
 - Criar pedido
 - Aceitar pedido
 - Gerar termo
 - Confirmar termo
 - Registrar histórico
 - Usa DTOs → retorna dados prontos para a controller
 - Contém validações funcionais simples (RF)
-

3.3 Domain

- Entidades inteligentes:
 - User
 - Shift (Plantão)
 - SwapRequest (Pedido de troca)
 - SwapHistory
 - SwapTerm
 - ScheduleType (12x36, 6x1, etc.)
- Regras de negócio (RN)
 - Mesma função
 - Termo obrigatório
 - Histórico imutável
 - Fluxo de estados
- Enumerações:
 - SwapStatus:
 - Pending
 - B_Accepted

- A_Confirmed
 - SentToManager
 - Completed
 - Cancelled
-

3.4 Infrastructure

- EF Core Context
 - Mapeamento das entidades
 - Repositórios (Repository Pattern)
 - Unit of Work
 - Migrations
 - Configuração SQL Server
 - Logging (Serilog ou ILogger)
 - Cache (Futuro)
-

◆ 4. Padrões Utilizados

- **Repository Pattern** → abstrai acesso a dados
 - **Unit of Work** → salva alterações atomicamente
 - **DTO + ViewModel** → separa domínio da interface
 - **AutoMapper** (opcional) para mapeamentos
 - **SOLID**
 - SRP → cada camada tem uma responsabilidade
 - OCP → fácil de evoluir
 - DIP → services dependem de interfaces
-

◆ 5. Fluxo Arquitetural de um Caso de Uso (Ex.: A pede troca)

Controller → Service → Domain → Repository → EF → DB

Detalhando:

1. **Controller recebe input**
2. **Valida ModelState**
3. **Chama Service: CreateSwapRequest**
4. **Service valida regras funcionais**
5. **Domain valida regras de negócio**
6. **Repository cria registro**
7. **Data salva**
8. **Service retorna DTO**
9. **Controller redireciona View**

Nenhuma regra de negócio fica na Controller ou na View.

◆ 6. Modelagem dos Principais Serviços

UserService

- CreateUser
 - AuthenticateUser
 - GetUserByFunction
-

ShiftService

- GenerateMonthlyCalendar
 - GetUserShifts
 - GetShiftByDate
-

SwapService

- CreateSwapRequest ($A \rightarrow B$)
- AcceptSwapRequest ($B \rightarrow A$)
- ConfirmSwapRequest ($A \rightarrow \text{termo}$)
- SendToManager
- FinalizeSwap

- SaveHistory
-

TermService

- BuildTermText (gerar o termo em texto)
 - FillTermWithData
 - GetTermForSwap
-

SurveyService

- TriggerSurvey
 - SaveSurveyResults
-

◆ 7. Tecnologias Definidas

Backend

- ASP.NET MVC
- C#
- EF Core 8
- AutoMapper (opcional)
- Identity (login local + Google)

Frontend

- Razor Views
- Bootstrap 5
- JQuery mínimo (somente para UI leve)

Infraestrutura

- SQL Server
 - GitHub para repositório
 - Deploy inicial:
 - Railway / Render / Azure (básico)
-

◆ 8. Banco de Dados (Visão Inicial)

Entidades principais:

User

- Id
- Nome
- Email
- Telefone
- Função
- Escala
- Tipo de Login (Google / Local)

Shift

- Id
- UserId
- Data
- Turno
- Tipo (Gerado automático)

SwapRequest

- Id
- A.UserId
- B.UserId
- A.ShiftId
- B.ShiftId
- Status
- Observação
- DataSolicitação
- DataAceiteB
- DataConfirmaA
- DataEnvioGestor

- TermoText

Survey

- Id
 - UserId
 - Nota
 - Dificuldade
 - Indicaria
 - Data
-

◆ 9. Justificativas Arquiteturais

1. **ASP.NET MVC** → rápido para MVP, boa organização, ideal para web mobile-friendly.
 2. **Arquitetura em camadas** → separa responsabilidades, mais sustentável no futuro.
 3. **Domain isolado** → garante que as regras de negócio não se espalhem.
 4. **EF Core** → agilidade na persistência e facilidade nos relacionamentos.
 5. **Repository** → abstração ideal para facilitar testes e evolução.
 6. **Fluxo com Services** → controllers únicas e limpas.
 7. **Termo gerado no backend** → padronização e rastreabilidade.
-

🚀 Arquitetura concluída.

Agora vamos para a próxima etapa:

📦 MODELAGEM DO DOMÍNIO

Será entregue com:

- Entidades
- Relacionamentos
- UML (Classes)
- Diagrama ER

- Regras dentro de cada entidade

Perfeito — agora iniciaremos uma das partes mais importantes do projeto:

MODELAGEM DO DOMÍNIO – YRKE

(Modelo completo, profissional e pronto para implementação)

Organizei em:

1. Entidades do Domínio
 2. Atributos por entidade
 3. Regras de negócio por entidade
 4. Relacionamentos
 5. Diagrama de Classes (UML)
 6. Diagrama ER (Banco de Dados)
-

◆ 1. ENTIDADES DO DOMÍNIO — VISÃO GERAL

O domínio do Yrke gira em torno de **usuários**, **plantões** e **trocas formais**.

As entidades principais são:

1. **User** – Representa o funcionário.
 2. **Shift** – Representa um plantão específico.
 3. **SwapRequest** – Pedido formal de troca entre A e B.
 4. **SwapTerm** – Texto formal do termo da permuta.
 5. **SwapHistory** – Registro imutável da troca finalizada.
 6. **Survey** – Pesquisa de satisfação após 2 trocas.
-

◆ 2. ENTIDADES + ATRIBUTOS

2.1 User

Representa o funcionário que usa o sistema.

Atributo	Tipo	Observação
Id	Guid	Identificador

Atributo	Tipo	Observação
Nome	string	Obrigatório
Email	string	Obrigatório
Telefone	string	Obrigatório
Funcao	string	Mesmo grupo obrigatório para troca
TipoEscala	enum	12x36, 6x1 etc
LoginGoogle	bool	se fez login via Google
DataCadastro	DateTime	—

2.2 Shift (Plantão)

Representa um plantão na agenda do usuário.

Atributo	Tipo	Observação
Id	Guid	—
UserId	Guid	FK User
Data	DateTime	—
Turno	enum	Manhã / Tarde / Noite
Tipo	enum	Gerado / Alterado
CriadoEm	DateTime	—

Regra importante:

- Um usuário não pode ter *dois plantões no mesmo dia e turno*.
-

2.3 SwapRequest (Pedido de troca)

O núcleo do sistema.

Atributo	Tipo	Observação
Id	Guid	—

Atributo	Tipo	Observação
A_UserId	Guid	Quem pediu
B_UserId	Guid	Quem recebeu
A_ShiftId	Guid	Plantão de A
B_ShiftId	Guid?	Plantão de B (preenchido após aceite)
Observacao	string	Opcional
Status	enum	Fluxo de estados
TermoId	Guid?	FK Termo
DataSolicitacao	DateTime	—
DataAceiteB	DateTime?	—
DataConfirmacaoA	DateTime?	—
DataEnvioGestor	DateTime?	—

📌 2.4 SwapTerm (Termo)

O documento formal da permuta.

Atributo	Tipo	Observação
Id	Guid	—
SwapId	Guid	FK
Texto	string	Termo formatado
CriadoEm	DateTime	—

📌 2.5 SwapHistory (Histórico)

Registro imutável.

Atributo	Tipo	Observação
Id	Guid	—

Atributo	Tipo	Observação
SwapId	Guid	FK
A_UserId	Guid	—
B_UserId	Guid	—
A_ShiftId	Guid	—
B_ShiftId	Guid	—
TermoTexto	string	—
ConcluidoEm	DateTime	—

✍ 2.6 Survey (Pesquisa)

Disparada a cada 2 trocas concluídas.

Atributo	Tipo	Observação
Id	Guid	—
UserId	Guid	FK
Nota	int	1 a 5
Dificuldade	string	pergunta qualitativa
Indicaria	bool	Sim/Não
CriadoEm	DateTime	—

◆ 3. REGRAS DE NEGÓCIO POR ENTIDADE

User

- Deve ter exatamente **uma função**.
- Login simples ou via Google.

Shift

- Pertence a **um único usuário**.
- Cada usuário só pode ter **um plantão por dia/turno**.

- Gerado automaticamente a partir da escala.

SwapRequest

- Estado inicial: **Pendente**
- Só pode ocorrer entre usuários da **mesma função** (regra crítica).
- Possui fluxo obrigatório de estados:
 1. Pending
 2. B_Accepted
 3. A_Confirmed
 4. SentToManager
 5. Completed
- Só conclui quando:
 - Termo assinado por A e B
 - Termo enviado ao gestor

SwapTerm

- Deve ser **imutável** após gerado.
- Cada troca possui exatamente **um termo**.

SwapHistory

- Registros são imutáveis.
- Contém todos os dados finais da troca.

Survey

- Disparada automaticamente **a cada 2 trocas** do usuário.
- Somente leitura posterior.

◆ 4. RELACIONAMENTOS ENTRE ENTIDADES

Usuário 1 — N Plantões

Um usuário possui muitos plantões.

Usuário A — N Solicitações de troca iniciadas

Usuário B — N Solicitações recebidas

Troca 1 — 1 Termo

Sempre um termo por troca.

Troca 1 — 1 Histórico (após concluída)

Histórico sempre vinculado a uma troca.

Usuário 1 — N Pesquisas

Uma pesquisa para cada grupo de 2 trocas.

◆ 5. DIAGRAMA UML (CLASSES)

Representado textualmente (estrutura perfeita para converter depois em ferramenta UML):

```
+-----+
|   User   |
+-----+
| Id      |
| Nome    |
| Email   |
| Telefone |
| Funcao  |
| TipoEscala |
| LoginGoogle |
| DataCadastro |
+-----+
| GetPlantões() |
| PodeTrocarCom()|
+-----+
| Shift   |
```

```
+-----+
```

Id
UserId
Data
Turno
Tipo
CriadoEm

```
+-----+
```

```
+-----+
```

SwapRequest

```
+-----+
```

Id
A_UserId
B_UserId
A_ShiftId
B_ShiftId
Observacao
Status
Termold
Datas do Fluxo...

```
+-----+
```

GerarTermo()
AceitarPorB()
ConfirmarPorA()
EnviarGestor()

```
+-----+
```

```
+-----+
```

```
| SwapTerm |
```

```
+-----+
```

```
| Id |
```

```
| SwapId |
```

```
| Texto |
```

```
| CriadoEm |
```

```
+-----+
```

```
+-----+
```

```
| SwapHistory |
```

```
+-----+
```

```
| Id |
```

```
| SwapId |
```

```
| A_UserId |
```

```
| B_UserId |
```

```
| A_ShiftId |
```

```
| B_ShiftId |
```

```
| TermoTexto |
```

```
| ConcluidoEm |
```

```
+-----+
```

```
+-----+
```

```
| Survey |
```

```
+-----+
```

```
| Id |
```

```
| UserId |
```

```
| Nota |
```

| Dificuldade |

| Indicaria |

| CriadoEm |

+-----+

◆ 6. DIAGRAMA ER (BANCO DE DADOS)

Representado textualmente:

User (1) ----- (N) Shift

User (1) ----- (N) SwapRequest (como A_UserId)

User (1) ----- (N) SwapRequest (como B_UserId)

SwapRequest (1) --- (1) SwapTerm

SwapRequest (1) --- (1) SwapHistory

User (1) ----- (N) Survey

■ MODELAGEM DE DADOS (SQL) – YRKE

(versão profissional e pronta para código)

Organizei em:

1. Tabelas
2. FKs e relacionamentos
3. Tipos SQL recomendados
4. Índices recomendados
5. Restrições de integridade
6. Esquema completo em SQL

◆ 1. TABELAS PRINCIPAIS

As tabelas foram criadas com base nas entidades:

1. **Users**
 2. **Shifts**
 3. **SwapRequests**
 4. **SwapTerms**
 5. **SwapHistory**
 6. **Surveys**
-

◆ 2. DEFINIÇÃO DAS TABELAS

📌 2.1 Users

Representa os funcionários.

Users

Id	UNIQUEIDENTIFIER (PK)
Nome	NVARCHAR(120) NOT NULL
Email	NVARCHAR(120) NOT NULL UNIQUE
Telefone	NVARCHAR(30) NOT NULL
Funcao	NVARCHAR(50) NOT NULL
TipoEscala	NVARCHAR(20) NOT NULL
LoginGoogle	BIT NOT NULL DEFAULT 0
DataCadastro	DATETIME2 NOT NULL DEFAULT SYSDATETIME()

📌 2.2 Shifts

Plantões já alocados.

Shifts

Id	UNIQUEIDENTIFIER (PK)
UserId	UNIQUEIDENTIFIER (FK Users.Id)

```
Data      DATE      NOT NULL
Turno     TINYINT    NOT NULL -- 0=Manhã,1=Tarde,2=Noite
Tipo      TINYINT    NOT NULL -- 0=Gerado,1=Manual
CriadoEm  DATETIME2 NOT NULL DEFAULT SYSDATETIME()
UNIQUE(UserId, Data, Turno)
```

📌 2.3 SwapRequests

Pedidos de troca.

SwapRequests

```
Id        UNIQUEIDENTIFIER (PK)
A_UserId  UNIQUEIDENTIFIER (FK Users.Id)
B_UserId  UNIQUEIDENTIFIER (FK Users.Id)
A_ShiftId UNIQUEIDENTIFIER (FK Shifts.Id)
B_ShiftId UNIQUEIDENTIFIER NULL (FK Shifts.Id)
Observacao NVARCHAR(500) NULL
Status     TINYINT    NOT NULL -- Enum
Termoid    UNIQUEIDENTIFIER NULL (FK SwapTerms.Id)
DataSolicitacao DATETIME2 NOT NULL DEFAULT SYSDATETIME()
DataAceiteB   DATETIME2 NULL
DataConfirmacaoA DATETIME2 NULL
DataEnvioGestor DATETIME2 NULL
```

📌 2.4 SwapTerms

Termos de permuta.

SwapTerms

Id UNIQUEIDENTIFIER (PK)
SwapId UNIQUEIDENTIFIER (FK SwapRequests.Id)
Texto NVARCHAR(MAX) NOT NULL
CriadoEm DATETIME2 NOT NULL DEFAULT SYSDATETIME()

2.5 SwapHistory

Registro imutável das trocas concluídas.

SwapHistory

Id UNIQUEIDENTIFIER (PK)
SwapId UNIQUEIDENTIFIER (FK SwapRequests.Id)
A_UserId UNIQUEIDENTIFIER (FK Users.Id)
B_UserId UNIQUEIDENTIFIER (FK Users.Id)
A_ShiftId UNIQUEIDENTIFIER (FK Shifts.Id)
B_ShiftId UNIQUEIDENTIFIER (FK Shifts.Id)
TermoTexto NVARCHAR(MAX) NOT NULL
ConcluidoEm DATETIME2 NOT NULL

2.6 Surveys

Pesquisas automáticas.

Surveys

Id UNIQUEIDENTIFIER (PK)
UserId UNIQUEIDENTIFIER (FK Users.Id)
Nota TINYINT NOT NULL -- 1 a 5
Dificuldade NVARCHAR(200)
Indicaria BIT NOT NULL

CriadoEm DATETIME2 NOT NULL DEFAULT SYSDATETIME()

◆ 3. RELACIONAMENTOS (FKs)

Origem	Destino	Tipo
Shifts.UserId	Users.Id	N:1
SwapRequests.A_UserId	Users.Id	N:1
SwapRequests.B_UserId	Users.Id	N:1
SwapRequests.A_ShiftId	Shifts.Id	N:1
SwapRequests.B_ShiftId	Shifts.Id	N:1
SwapTerms.SwapId	SwapRequests.Id	1:1
SwapHistory.SwapId	SwapRequests.Id	1:1
Surveys.UserId	Users.Id	N:1

◆ 4. ÍNDICES RECOMENDADOS (performance)

Users

CREATE UNIQUE INDEX IX_Users_Email ON Users (Email);

Shifts

CREATE UNIQUE INDEX IX_Shifts_UserDateTurno ON Shifts (UserId, Data, Turno);

SwapRequests

CREATE INDEX IX_SwapRequests_Status ON SwapRequests (Status);

CREATE INDEX IX_SwapRequests_AUser ON SwapRequests (A_UserId);

CREATE INDEX IX_SwapRequests_BUser ON SwapRequests (B_UserId);

SwapHistory

CREATE INDEX IX_SwapHistory_UserA ON SwapHistory (A_UserId);

CREATE INDEX IX_SwapHistory_UserB ON SwapHistory (B_UserId);

◆ 5. RESTRIÇÕES DE INTEGRIDADE

✓ Plantões não duplicados por usuário

UNIQUE(UserId, Data, Turno)

✓ Trocas só entre usuários existentes

FK + validação no domínio.

✓ Termo sempre atrelado a uma troca

FK 1:1.

✓ Histórico imutável

Nenhuma ação de Update no histórico.

✓ Troca com fluxo consistente

Controlado no domínio + status enum.

◆ 6. SCRIPT COMPLETO – MODELO BASE SQL SERVER

Abaixo um modelo simplificado (você poderá usar EF Core para gerar as migrations, mas o SQL está aqui para referência):

```
CREATE TABLE Users (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    Nome NVARCHAR(120) NOT NULL,
    Email NVARCHAR(120) NOT NULL UNIQUE,
    Telefone NVARCHAR(30) NOT NULL,
    Funcao NVARCHAR(50) NOT NULL,
    TipoEscala NVARCHAR(20) NOT NULL,
    LoginGoogle BIT NOT NULL DEFAULT 0,
    DataCadastro DATETIME2 NOT NULL DEFAULT SYSDATETIME()
);
```

```
CREATE TABLE Shifts (
```

```
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    UserId UNIQUEIDENTIFIER NOT NULL,
```

```
        Data DATE NOT NULL,  
        Turno TINYINT NOT NULL,  
        Tipo TINYINT NOT NULL,  
        CriadoEm DATETIME2 NOT NULL DEFAULT SYSDATETIME(),  
        FOREIGN KEY (UserId) REFERENCES Users(Id),  
        UNIQUE(UserId, Data, Turno)  
    );
```

```
CREATE TABLE SwapRequests (  
    Id UNIQUEIDENTIFIER PRIMARY KEY,  
    A_UserId UNIQUEIDENTIFIER NOT NULL,  
    B_UserId UNIQUEIDENTIFIER NOT NULL,  
    A_ShiftId UNIQUEIDENTIFIER NOT NULL,  
    B_ShiftId UNIQUEIDENTIFIER NULL,  
    Observacao NVARCHAR(500),  
    Status TINYINT NOT NULL,  
    Termoid UNIQUEIDENTIFIER NULL,  
    DataSolicitacao DATETIME2 NOT NULL DEFAULT SYSDATETIME(),  
    DataAceiteB DATETIME2 NULL,  
    DataConfirmacaoA DATETIME2 NULL,  
    DataEnvioGestor DATETIME2 NULL,  
    FOREIGN KEY (A_UserId) REFERENCES Users(Id),  
    FOREIGN KEY (B_UserId) REFERENCES Users(Id),  
    FOREIGN KEY (A_ShiftId) REFERENCES Shifts(Id),  
    FOREIGN KEY (B_ShiftId) REFERENCES Shifts(Id)  
);
```

```
CREATE TABLE SwapTerms (
```

```
Id UNIQUEIDENTIFIER PRIMARY KEY,  
SwapId UNIQUEIDENTIFIER NOT NULL,  
Texto NVARCHAR(MAX) NOT NULL,  
CriadoEm DATETIME2 NOT NULL DEFAULT SYSDATETIME(),  
FOREIGN KEY (SwapId) REFERENCES SwapRequests(Id)  
);
```

```
CREATE TABLE SwapHistory (  
Id UNIQUEIDENTIFIER PRIMARY KEY,  
SwapId UNIQUEIDENTIFIER NOT NULL,  
A_UserId UNIQUEIDENTIFIER NOT NULL,  
B_UserId UNIQUEIDENTIFIER NOT NULL,  
A_ShiftId UNIQUEIDENTIFIER NOT NULL,  
B_ShiftId UNIQUEIDENTIFIER NOT NULL,  
TermoTexto NVARCHAR(MAX) NOT NULL,  
ConcluidoEm DATETIME2 NOT NULL,  
FOREIGN KEY (SwapId) REFERENCES SwapRequests(Id),  
FOREIGN KEY (A_UserId) REFERENCES Users(Id),  
FOREIGN KEY (B_UserId) REFERENCES Users(Id)  
);
```

```
CREATE TABLE Surveys (  
Id UNIQUEIDENTIFIER PRIMARY KEY,  
UserId UNIQUEIDENTIFIER NOT NULL,  
Nota TINYINT NOT NULL,  
Dificuldade NVARCHAR(200),  
Indicaria BIT NOT NULL,  
CriadoEm DATETIME2 NOT NULL DEFAULT SYSDATETIME(),
```

```
FOREIGN KEY (UserId) REFERENCES Users(Id)  
);
```

BACKLOG DO MVP – YRKE

Organizado por **épicos** → **features** → **user stories** com **prioridade**.

ÉPICO 1 — Autenticação & Cadastro

Feature 1.1 — Criar Conta

User Story – US001

Como **usuário**, quero **criar uma conta** para acessar o sistema de trocas.

Critérios de Aceite:

- Deve permitir cadastro com: nome, email, telefone, função, tipo de escala.
 - Deve validar email único.
 - Deve gerar shifts automaticamente com base no tipo de escala.
 - Após cadastro, usuário é redirecionado para o painel.
-

Feature 1.2 — Login (Google ou Próprio)

User Story – US002

Como **usuário**, quero **entrar usando Google ou e-mail/senha**.

Critérios de Aceite:

- Login Google (OAuth) opcional.
 - Login local com email + senha.
 - Recuperação de senha disponível.
-

ÉPICO 2 — Gerenciamento de Plantões (Shifts)

Feature 2.1 — Visualizar Plantões

User Story – US003

Como **usuário**, quero **ver meus plantões do mês** para entender quando estou escalado.

Critérios de Aceite:

- Exibir calendário mensal.

- Plantões gerados automaticamente conforme tipo de escala.
 - Não permitir editar plantões automáticos.
-

Feature 2.2 — Adicionar Plantão Extra (opcional)

User Story – US004

Como usuário, quero **registrar manualmente um plantão**, caso o gestor tenha me adicionado em cima da hora.

Critérios de Aceite:

- Deve validar duplicidade (não pode dois plantões no mesmo turno/dia).
 - Deve marcar como “manual”.
-

■ ÉPICO 3 — Troca de Plantão (Core)

Processo completo A → B → A → Gestor

Feature 3.1 — Solicitar Troca

User Story – US005

Como **usuário A**, quero **selecionar um colega B** e enviar um pedido de troca.

Critérios de Aceite:

- Usuário escolhe o colega a partir de uma lista filtrada por **mesma função**.
 - Usuário seleciona o plantão dele (A) e o plantão desejado do outro (B).
 - Permitir adicionar observação.
 - Status inicial: **Aguardando Aceite de B**.
-

Feature 3.2 — Aceitar ou Recusar Troca (Usuário B)

User Story – US006

Como **usuário B**, quero **aceitar ou recusar** a troca para ajudar (ou porque não posso trocar).

Critérios de Aceite:

- Ao aceitar: registrar horário de aceite.
- Ao recusar: troca é marcada como **Recusada** e notificar A.

- Ao aceitar: gerar automaticamente o **Termo de Permuta** preenchido.
-

Feature 3.3 — Confirmação Final (Usuário A)

User Story – US007

Como usuário A, quero **confirmar o aceite do B**, antes de enviar ao gestor.

Critérios de Aceite:

- Mostrar o termo completo para revisão.
 - Mostrar o email/whatsapp do gestor para envio.
 - Ao confirmar: status muda para **Pendente Envio Gestor**.
-

Feature 3.4 — Envio automático do Termo ao Gestor

User Story – US008

Como usuário A, quero **enviar o termo automaticamente para o gestor**, garantindo que a troca fique formalizada.

Critérios de Aceite:

- Enviar o PDF via email ou WhatsApp.
 - Registrar DataEnvioGestor no banco.
 - Mudar status para **Concluído**.
 - Copiar registro para **SwapHistory**.
-

ÉPICO 4 — Termos, Histórico e Auditoria

Feature 4.1 — Gerar Termo automaticamente

User Story – US009

Como sistema, quero **gerar um termo padronizado**, baseado no texto fornecido e dados da troca.

Critérios de Aceite:

- Termo deve incluir: nomes, matrícula, datas, motivos, responsabilidades etc.
- Deve ser gerado como texto e PDF.
- Deve ser anexado à troca via SwapTerm.

Feature 4.2 — Histórico de trocas

User Story – US010

Como usuário, quero **ver minhas trocas anteriores**, para controle pessoal.

Critérios de Aceite:

- Listar histórico completo.
 - Mostrar PDF do termo.
 - Mostrar situação da troca.
-

■ ÉPICO 5 — Pesquisa de Satisfação

Feature 5.1 — Solicitar feedback

User Story – US011

Como sistema, quero **solicitar uma pesquisa a cada 2 trocas**, para avaliar usabilidade.

Critérios de Aceite:

- Enviar pesquisa somente após 2 trocas concluídas.
 - Perguntas: dificuldade, recomendaria, nota 1–5.
 - Registrar no banco.
-

■ ÉPICO 6 — Notificações

Feature 6.1 — Notificações internas

User Story – US012

Como usuário, quero **receber alertas** quando houver atualizações em minha troca.

Critérios de Aceite:

- Notificação ao receber pedido.
- Notificação quando B aceitar.
- Notificação quando A confirmar.
- Notificação de conclusão.

ÉPICO 7 — Segurança e Acesso

Feature 7.1 — Perfis e Permissões

User Story – US013

Como sistema, quero permitir que apenas **o gestor veja os termos enviados**, garantindo privacidade.

Critérios de Aceite:

- Gestor acessa somente termos enviados.
 - Funcionários não veem termos alheios.
-

PRIORIDADE DO MVP

Prioridade	User Story
Urgente	US001, US002, US003, US005, US006, US007, US008, US009
Alta	US010, US012
Média	US004
Baixa / Pós-MVP	US011, US013

SPRINT PLANNING (SUGESTÃO)

Sprint 1 — Fundamentos

- US001 Cadastro
 - US002 Login
 - Arquitetura MVC + Identity
 - Base do EF Core
 - Tabela Users criada
 - Tela inicial
-

Sprint 2 — Plantões

- US003 Visualizar plantões

- Gerador automático do calendário
 - Tabela Shifts
 - Integração no dashboard
-

Sprint 3 — Troca (1/3)

- US005 Solicitar troca
 - US006 Aceitar/Recusar troca
 - Notificações básicas
-

Sprint 4 — Troca (2/3)

- US009 Geração do termo
 - US007 Confirmação A
 - Visualização do Termo
-

Sprint 5 — Troca (3/3)

- US008 Envio ao gestor
 - SwapHistory
 - Conclusão do fluxo de troca
-

Sprint 6 — Pós-MVP

- US010 Histórico
 - US011 Pesquisa
 - US012 Notificações completas
 - Ajustes finais
-

BACKLOG PRONTO!

Agora o Yrke já tem:

- ✓ Lista completa de User Stories
- ✓ Critérios de Aceite bem definidos
- ✓ Priorização
- ✓ Sprints organizadas

1. ROADMAP DE DESENVOLVIMENTO

Organizado em:

- **MVP (Primeiros 60 dias)**
 - **V1 (90–120 dias)**
 - **V2 (6–12 meses)**
-

MVP — Primeiros 60 dias

Sprint 1 – Fundamentos (Semana 1–2)

- Setup do projeto ASP.NET MVC
 - Setup EF Core + SQL Server
 - Configurar Identity + Login Google
 - Cadastro de usuário (US001)
 - Login (US002)
 - Dashboard inicial
-

Sprint 2 – Plantões (Semana 3–4)

- Modelagem dos Shifts
 - Geração automática de plantões pela escala
 - Visualização no calendário (US003)
-

Sprint 3 – Troca (parte 1 — Solicitação) (Semana 5)

- Envio de pedido de troca A → B (US005)
- Aceite/Recusa de B (US006)
- Validações de regra (mesma função, data válida etc.)

█ Sprint 4 – Troca (parte 2 — Termo + Confirmação) (Semana 6)

- Gerar termo automaticamente (US009)
 - Confirmação A (US007)
 - Tela para revisar termo
-

█ Sprint 5 – Troca (parte 3 — Envio ao Gestor) (Semana 7–8)

- Geração do PDF
 - Envio via e-mail / WhatsApp (US008)
 - Registro em SwapHistory
 - Auditoria e logs
-

█ Sprint 6 – Closures do MVP (Semana 9–10)

- Histórico de trocas (US010)
 - Notificações internas básicas (US012)
-

★ V1 — Após validação com usuários reais (90–120 dias)

- Pesquisa de satisfação (US011)
 - Painel do Gestor
 - Estatísticas e dashboards (trocas por período, faltas evitadas, etc.)
 - Melhorias de UX mobile
 - Notificações push (via PWA ou Firebase)
 - Relatórios completos em PDF
-

★ V2 — Expansão (6–12 meses)

- Aplicativo mobile nativo (Flutter ou MAUI)
- Multi-gestores (por setor/hospital)
- Auditoria avançada (logs + trilha legal)

- Integração com sistemas hospitalares (API REST)
 - Escalas automáticas mais inteligentes
 - Controle de faltas integrado ao RH
-

🚀 2. ESTRATÉGIA GITFLOW — RECOMENDADA

GitFlow é o fluxo ideal para sistemas corporativos.

O Yrke deve seguir essa estrutura:

```
main
|
└ develop
    |
    ├── feature/US001-cadastro
    ├── feature/US005-solicitar-troca
    ├── feature/geracao-termo
    └ ...
```

◆ Branch principal

main

- Código estável
 - Versões publicadas
 - Cada deploy gera uma tag
-

◆ Branch de desenvolvimento

develop

- Onde tudo é integrado
 - Recebe PRs das features
-

◆ Branches de feature

Padrão:

feature/USxxx-descricao-curta

Exemplos:

- feature/US005-solicitar-troca
 - feature/identity-login-google
 - feature/termo-pdf
-

◆ Branches de release

release/1.0.0

- Para preparar versão
 - Ajustes de última hora
 - Após finalizar: merge → main + develop
-

◆ Branches de hotfix

hotfix/erro-envio-gestor

Para corrigir bugs em produção rapidamente.



3. PIPELINE CI/CD — GitHub Actions

Aqui está o pipeline ideal para o Yrke:



CI (Continuous Integration)

Aciona a cada pull request para o develop.

Pipeline: .github/workflows/ci.yml

Passos:

1. Checkout do código
2. Restore do .NET
3. Build
4. Rodar testes unitários

5. Rodar análise estática (SonarCloud opcional)
 6. Gerar artefato para o CD
-

CD (Continuous Deployment)

Aciona quando algo é enviado para main.

Ambiente alvo:

- **Azure App Service ou Railway ou Render** (escolher depois)

Pipeline:

1. Baixar artefato do CI
 2. Deploy automatizado
 3. Rodar migrações EF Core
 4. Notificar em Slack/WhatsApp (opcional)
-

Exemplo simplificado do workflow:

```
name: CI
```

```
on:
```

```
  pull_request:
```

```
    branches: [ "develop" ]
```

```
jobs:
```

```
  build:
```

```
    runs-on: ubuntu-latest
```

```
    steps:
```

```
      - uses: actions/checkout@v3
```

```
      - name: Setup .NET
```

```
uses: actions/setup-dotnet@v3
```

with:

```
dotnet-version: '8.0.x'
```

```
- name: Restore
```

```
run: dotnet restore
```

```
- name: Build
```

```
run: dotnet build --no-restore
```

```
- name: Test
```

```
run: dotnet test --no-build --verbosity normal
```

4. PADRÕES DE CODIFICAÇÃO (Clean Architecture + MVC)

Camadas recomendadas

/Yrke.Web (MVC Controllers, Views)

/Yrke.Application (Serviços, DTOs)

/Yrke.Domain (Entidades, regras)

/Yrke.Infrastructure (EF Core, Repositórios)

Regras principais

Entidades do domínio não conhecem banco de dados

Nada de DbContext dentro das entidades.

Controllers não têm regra de negócio

Somente chamam serviços de aplicação.

Services orquestram fluxo

Exemplo: SwapService gerencia todo o processo de troca (A → B → A → Gestor).

✓ Repositórios encapsulam EF Core

Exemplo:

```
public interface ISwapRepository
{
    Task<SwapRequest> GetByIdAsync(Guid id);
    Task AddAsync(SwapRequest swap);
    Task<IEnumerable<SwapRequest>> GetPendingForUser(Guid userId);
}
```

✓ DTOs sempre entre Controller ↔ Service

Entidades **nunca** vão direto pra View.

✓ Automapper para conversões

Evita poluição dos controllers.

✓ Logs com Serilog

Armazena histórico de erros e trilhas.

✓ Validação com FluentValidation

Exemplo:

```
public class CreateSwapValidator : AbstractValidator<CreateSwapDto>
{
    public CreateSwapValidator()
    {
        RuleFor(x => x.AUserId).NotEmpty();
        RuleFor(x => x.BUserId).NotEmpty();
    }
}
```

```
        RuleFor(x => x.AShiftId).NotEmpty();  
    }  
}
```

ENTREGA CONCLUÍDA

Você agora tem:

- ✓ Roadmap completo
- ✓ Estratégia GitFlow
- ✓ Pipeline CI/CD recomendado
- ✓ Padrões de desenvolvimento
- ✓ Estrutura profissional de projeto

DOCUMENTO FINAL DE ARQUITETURA – YRKE

1 Visão Geral

O Yrke é um sistema web para **gerenciamento e permuta de plantões**, com foco em:

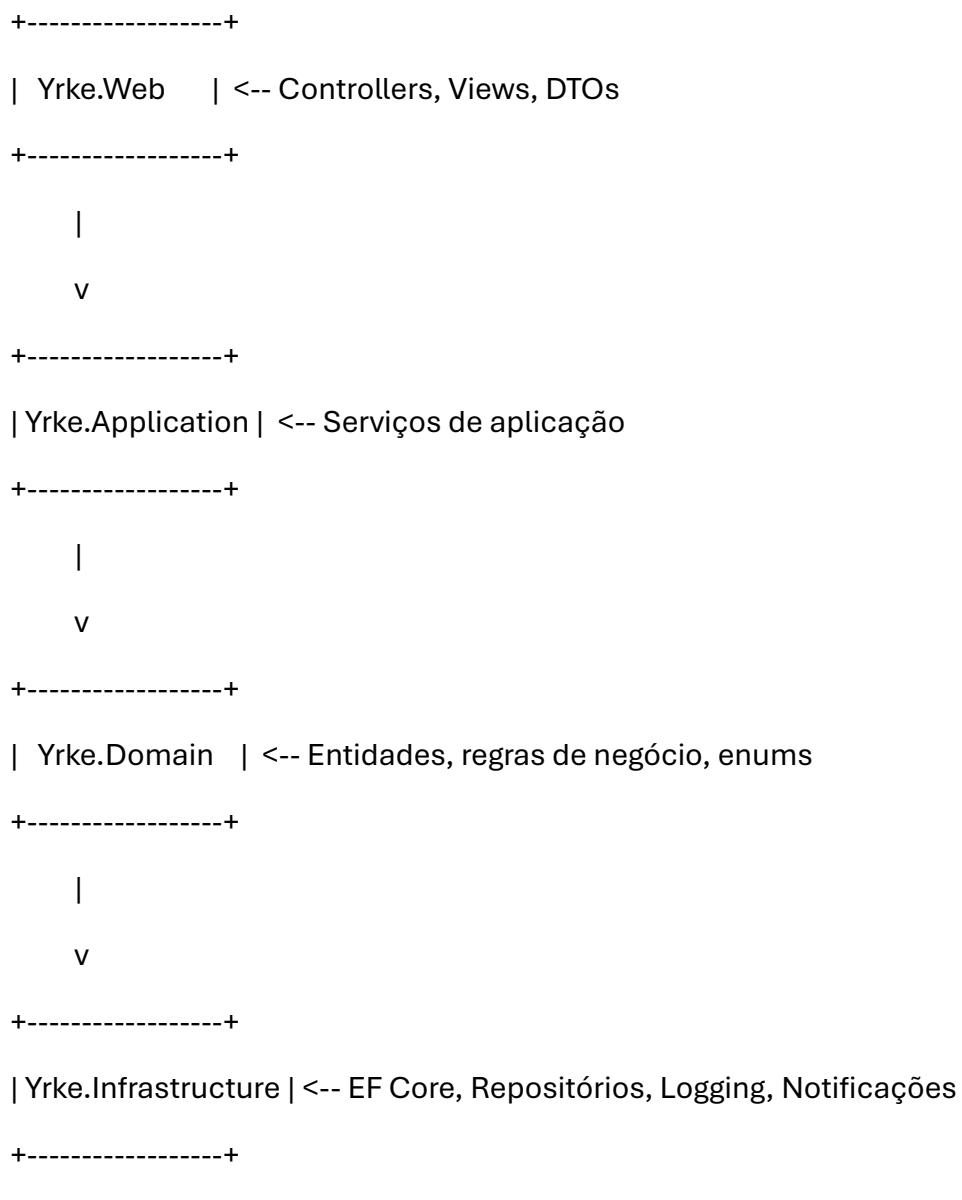
- Segurança, rastreabilidade e histórico de trocas.
- Interface web responsiva (desktop + mobile) e futura expansão para app mobile.
- Geração automática de **termos de permuta** e envio para gestor.
- Pesquisa de satisfação e auditoria para melhoria contínua.

Tecnologias base:

- **C# .NET 8 / ASP.NET MVC**
 - **EF Core 8**
 - **SQL Server**
 - **Identity + OAuth (Google)**
 - **Serilog** para logging
 - **FluentValidation** para validações
 - **Automapper** para DTOs
-

2 Arquitetura em Camadas

2.1 Diagrama de Camadas (Clean Architecture)



2.2 Responsabilidades das Camadas

Camada	Responsabilidade
Web	Receber requisições HTTP, renderizar views, chamar Application Services
Application	Orquestra fluxos de negócio (ex: SwapService), valida regras complexas, envia notificações
Domain	Entidades e regras de negócio puras, enums, eventos de domínio

Camada	Responsabilidade
--------	------------------

Infrastructure	Persistência (EF Core), envio de e-mails/WhatsApp, logging, cache
-----------------------	---

3 Diagrama de Classes (Resumo)

User

Id, Nome, Email, Telefone, Funcao, TipoEscala

Shift

Id, UserId, Data, Turno, Tipo

SwapRequest

Id, A_UserId, B_UserId, A_ShiftId, B_ShiftId, Status, Observacao, TermoId

SwapTerm

Id, SwapId, Texto, CriadoEm

SwapHistory

Id, SwapId, A_UserId, B_UserId, TermoTexto, ConcluidoEm

Survey

Id, UserId, Nota, Dificuldade, Indicaria, CriadoEm

4 Componentes Técnicos

- **Controllers:**

- UserController → Cadastro, Login
- ShiftController → Plantões, calendário
- SwapController → Solicitação de trocas
- SurveyController → Feedback pós-troca

- **Services:**

- SwapService → Processa fluxo completo de A → B → A → Gestor
- UserService → Cadastro e login
- ShiftService → Geração de plantões e validações
- SurveyService → Cria e registra pesquisas

- **Repositories:**

- IUserRepository, IShiftRepository, ISwapRepository,
ISwapTermRepository, ISwapHistoryRepository

- **DTOs e Automapper:**

- Todos os Controllers recebem e retornam DTOs.
- Entidades do Domain **nunca** vão direto para a View.

- **Logging e Auditoria:**

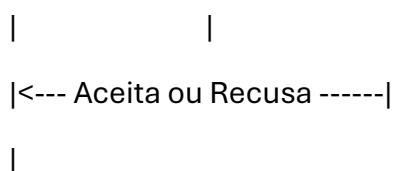
- Serilog registra todos os eventos críticos: trocas, envios de termo, falhas de notificação.

- **Validações:**

- FluentValidation para todas as entradas (ex: troca de plantão, cadastro).

5 Diagrama de Fluxo – Troca de Plantão

Usuário A ---> Solicita troca ---> Usuário B



|--- Confirma termo e envia para Gestor ---> PDF + Histórico

- O **termo de permuta** é gerado automaticamente e enviado **somente ao gestor**.
 - Histórico é imutável em SwapHistory.
-

6 Fluxos Importantes

6.1 Fluxo de Autenticação

1. Login com Google ou email/senha.
2. Criar conta se necessário.
3. Redireciona para dashboard de plantões.

6.2 Fluxo de Troca

1. Usuário A seleciona plantão e coliga B.
2. B aceita/recusa.
3. A confirma termo gerado automaticamente.
4. Termo é enviado ao gestor (PDF ou WhatsApp).
5. SwapHistory registra permanentemente.

6.3 Fluxo de Plantões

- Plantões gerados automaticamente via tipo de escala (12/36, 6x1 etc.)
 - Visualização em calendário.
 - Registro manual opcional com validação de duplicidade.
-

7 Banco de Dados (Resumo)

- **Tabelas:** Users, Shifts, SwapRequests, SwapTerms, SwapHistory, Surveys
- **Relacionamentos:**
 - 1:N Users → Shifts
 - 1:N Users → SwapRequests (A e B)
 - 1:1 SwapRequests → SwapTerms
 - SwapHistory imutável

- **Índices críticos:**

- Shifts (UserId + Data + Turno)
 - SwapRequests Status e Users
-

8 Padrões e Boas Práticas

- **SOLID e Clean Architecture**
 - **DTOs entre Controllers e Services**
 - **Repositórios isolam EF Core**
 - **Validações com FluentValidation**
 - **Logging com Serilog**
 - **Automapper para conversões**
 - **Branch GitFlow + CI/CD (GitHub Actions)**
-

9 Exemplo de Service (C#)

```
public class SwapService : ISwapService
{
    private readonly ISwapRepository _swapRepo;
    private readonly IUserRepository _userRepo;
    private readonly IShiftRepository _shiftRepo;
    private readonly INotificationService _notificationService;

    public async Task<SwapRequest> CreateSwapAsync(Guid aUserId, Guid
bUserId, Guid aShiftId, Guid bShiftId)
    {
        // Validações básicas
        var aUser = await _userRepo.GetByIdAsync(aUserId);
        var bUser = await _userRepo.GetByIdAsync(bUserId);
```

```
// Cria SwapRequest

var swap = new SwapRequest { A_UserId = aUserId, B_UserId = bUserId,
A_ShiftId = aShiftId, Status = SwapStatus.Pending };

await _swapRepo.AddAsync(swap);

// Notificação

await _notificationService.NotifyUserAsync(bUserId, "Você recebeu um
pedido de troca.");

return swap;

}

}
```