

Projeto de Portfólio: "AssetFlow - Gestão de Ativos Pessoais"

Descrição: Um sistema de gestão de portfólio pessoal onde o usuário pode registrar, acompanhar e movimentar ativos (ações, criptomoedas, itens de colecionador) entre diferentes carteiras. A aplicação exige alta segurança (JWT) e integridade de dados (Transações).

1. Estrutura de Dados (Modelagem Prisma)

O projeto deve ter três modelos principais interligados:

Modelo	Relações	Propriedades Chave
User	(Base de Login/Autenticação)	id, name, email, password (hash)
Wallet (Carteira)	User (1:N)	id, name, type (Investimento, Reserva, Coleção), userId
Asset (Ativo)	Wallet (1:N)	id, name, quantity (quantidade), purchasePrice, walletId
Transaction (Histórico)	Wallet (1:N)	id, assetId, type (Compra/Venda/Transferência), date, quantity, value

2. Implementações Chave (O Destaque do Portfólio)

Funcionalidade	Conceito Aplicado	Detalhe da Implementação
Autenticação	JWT	Proteger todas as rotas de Wallet, Asset e Transaction com seu authMiddleware.
Transferência de Ativo	**PRISMA. \$TRANSACTION**	A operação mais crítica: Mover X unidades de um Asset da Wallet A para a Wallet B. Deve ser atômica: Diminuir o saldo em A E Aumentar o saldo em B . Se uma falhar, a outra reverte.

Validação de Inputs	ZOD (com Refatoração)	Refactor: Usar <code>.transform()</code> para normalizar dados (Ex: <code>email.toLowerCase()</code> no <code>User</code> , formatar valores de moeda). Avançado: Usar <code>z.refine()</code> para garantir que <code>purchasePrice</code> seja um número positivo.
Histórico de Consultas	Paginação (Offset)	A rota <code>GET /transactions</code> deve ser paginada por número de página (<code>page</code> , <code>pageSize</code>) para evitar sobrecarga.
Busca de Ativos	Paginação (Cursor)	A rota <code>GET /assets</code> deve usar Paginação por Cursor (<code>cursor</code>) para otimizar a experiência de <i>infinite scroll</i> (busca rápida) se você for implementar no Front-end.
Filtros Avançados	WHERE (AND, OR, NOT)	Filtro complexo: Criar um endpoint que use <code>OR</code> e <code>NOT</code> para buscar: "Ativos que estão na <code>Wallet</code> 'Reserva' E NÃO são criptomoedas OU foram comprados por mais de R\$ 1000."

3. Cenários de Código Crítico

Você deve criar um **Service** específico para o fluxo de transferência, assim como fizemos com `transferTask`:

Assinatura do Service (Transferência Atômica):

TypeScript

// Exemplo de Assinatura para o seu AssetService:

```
async transferAsset(assetId: number, sourceWalletId: number, targetWalletId: number, quantity: number,
userId: number) {
  // 1. Checar autoria (userId é dono da sourceWalletId)
  // 2. Checar saldo (sourceWallet.quantity >= quantity)

  await prisma.$transaction(async (tx) => {
    // 1. Decrementa o saldo da carteira de origem (sourceWalletId)
    await tx.asset.update({ /* ... decrementa aqui ... */ });

    // 2. Incrementa o saldo da carteira de destino (targetWalletId)
    await tx.asset.update({ /* ... incrementa aqui ... */ });
  });
}
```

1. Modelo Asset (O Estado Atual)

O modelo Asset (Ativo) representa a **posição atual** do usuário em relação a um item específico dentro de uma carteira. Ele armazena o saldo consolidado.

Propriedade	Tipo	Propósito
id	Int	Identificador único do ativo na carteira (ex: Bitcoin na Carteira A).
name	String	Nome/Símbolo do ativo (ex: "Bitcoin", "Apple Stock", "Item Raro X").
quantity	Float	CRÍTICO: Saldo atual de unidades desse ativo.
purchasePrice	Float	Preço de compra (pode ser o preço da última compra ou o preço médio, dependendo da regra de negócio).
walletId	Int	Chave estrangeira que vincula o ativo à sua Wallet (Carteira).

Lógica de Negócio:

- Este modelo **só deve ser alterado através de uma Transação**. Quando o usuário compra ou vende algo, a **quantity** é ajustada (**increment** ou **decrement**) dentro do bloco **\$transaction**.
- A **quantity** do Asset é a prova final do que o usuário possui **neste momento**.

2. Modelo Transaction (O Histórico Imutável)

O modelo Transaction (Transação) é um **registro de log imutável** de cada evento que altera o portfólio. É o registro histórico que justifica a **quantity** que está no modelo Asset.

Propriedade	Tipo	Propósito
id	Int	Identificador único do registro de transação.
assetId	Int	Chave estrangeira que aponta para o Asset que foi modificado.
type	String	Tipo da operação ('COMPRA', 'VENDA', 'TRANSFERENCIA'). Excelente para usar z.enum() no Zod.
date	DateTime	Data e hora exatas em que a operação ocorreu.
quantity	Float	A quantidade <i>exata</i> de unidades envolvidas nesta operação específica.
value	Float	O valor financeiro total ou unitário da transação (ex: Preço da compra no momento).

Lógica de Negócio (A Transação Atômica):

Este modelo é crucial para o seu estudo de \$transaction:

Sempre que um usuário faz uma **Compra**, **Venda** ou **Transferência**, duas coisas críticas devem acontecer **de forma atômica**:

1. **Atualização do Asset:** A **quantity** no modelo Asset deve ser atualizada.
2. **Registro do Transaction:** Um novo registro deve ser criado no modelo Transaction.

Cronograma Otimizado (2-3 Horas/Dia)

Fase	Dias de Trabalho	Foco Principal do Dia	Ferramentas Chave
FASE 1: ESTRUTURA E BASE	Dia 1	Setup inicial, <code>package.json</code> , <code>tsconfig.json</code> , <code>schema.prisma</code> (Modelos <code>User</code> e <code>Wallet</code>).	TypeScript, Prisma Init.
	Dia 2	Repositórios e Serviços CRUD básicos para <code>User</code> .	CRUD Base, Repositório.
	Dia 3	Repositórios e Serviços CRUD básicos para <code>Wallet</code> .	CRUD Base, Repositório.
	Dia 4	Modelos <code>Asset</code> e <code>Transaction</code> (base) e Rotas de Acesso.	Modelagem, Migrações.
FASE 2: SEGURANÇA E ZOD	Dia 5	Implementação completa do <code>JWT</code> (<code>authUtil</code> , <code>authMiddleware</code>).	JWT, Middleware.
	Dia 6	Rotas de <code>Login/Registro</code> e Validação <code>Zod</code> completa para <code>User</code> .	Zod, Autenticação.
	Dia 7	Aplicação do <code>authMiddleware</code> nas rotas de <code>Wallet</code> , <code>Asset</code> e <code>Transaction</code> .	Proteção de Rotas.
FASE 3: CONSULTAS AVANÇADAS	Dia 8	Implementação da Paginação por <code>Offset</code> (<code>skip/take</code>) no <code>AssetService</code> .	Paginação Básica.
	Dia 9	Implementação da Paginação por <code>Cursor</code> e Lógica de <code>skip: 1</code> .	Paginação Avançada.

	Dia 10	Implementação de Ordenação (<code>orderBy</code>) e Filtros Simples (<code>status</code> , <code>type</code>).	<code>orderBy</code> , <code>where</code> .
	Dia 11	Implementação de Filtros Complexos (<code>AND</code> , <code>OR</code> , <code>NOT</code>).	<code>where: { OR: [...] }</code> .
FASE 4: INTEGRIDADE (TRANSAÇÕES)	Dia 12	Lógica de Pré-Transação : Checagem de Saldo e Obtenção dos dados do <code>Asset</code> .	Lógica de Negócio.
	Dia 13	Implementação da Transação : Criar o <code>\$transaction</code> para diminuir o saldo (<code>Asset.quantity</code>) na carteira de origem.	<code>\$transaction</code> , <code>Rollback</code> .
	Dia 14	Conclusão da Atomicidade : Implementar o incremento na carteira de destino E o registro na tabela Transaction .	Integridade de Dados.
FASE 5: POLIMENTO E PORTFÓLIO	Dia 15	Refatoração Zod Avançada (<code>.transform()</code>): Limpeza de emails para minúsculas, formatação de valores.	Zod Transform, Type Safety.
	Dia 16	Revisão e Limpeza : Remover <code>as any</code> e <code>type assertions</code> desnecessárias. Testes manuais do fluxo de transferência.	Polimento de Código.
	Dia 17	Documentação : Escrever o README do projeto para portfólio (explicando as Transações, JWT e Paginação).	README, GitHub.