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 .transform() para normalizar dados (Ex: email.toLowerCase() no User, formatar valores de moeda). **Avançado:** Usar z.refine() para garantir que purchasePrice seja um número positivo.
Histórico de Consultas	**Paginação (Offset)**	A rota GET /transactions deve ser paginada por número de página (page, pageSize) para evitar sobrecarga.
Busca de Ativos	**Paginação (Cursor)**	A rota GET /assets deve usar Paginação por Cursor (cursor) para otimizar a experiência de *infinite scroll* (busca rápida) se você for implementar no Front-end.
Filtros Avançados	**WHERE (AND, OR, NOT)**	**Filtro complexo:** Criar um endpoint que use OR e NOT para buscar: "Ativos que estão na Wallet '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 *exata* 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, package.json, tsconfig.json, schema.prisma (Modelos User e Wallet).	TypeScript, Prisma Init.
	Dia 2	Repositórios e Serviços CRUD básicos para **User**.	CRUD Base, Repositório.
	Dia 3	Repositórios e Serviços CRUD básicos para **Wallet**.	CRUD Base, Repositório.
	Dia 4	Modelos Asset e Transaction (base) e Rotas de Acesso.	Modelagem, Migrações.
FASE 2: SEGURANÇA E ZOD	**Dia 5**	Implementação completa do **JWT** (authUtil, authMiddleware).	JWT, Middleware.
	Dia 6	Rotas de **Login/Registro** e Validação **Zod** completa para User.	Zod, Autenticação.
	Dia 7	Aplicação do authMiddleware nas rotas de Wallet, Asset e Transaction.	Proteção de Rotas.
FASE 3: CONSULTAS AVANÇADAS	**Dia 8**	Implementação da Paginação por **Offset** (skip/take) no AssetService.	Paginação Básica.
	Dia 9	Implementação da Paginação por **Cursor** e Lógica de skip: 1.	Paginação Avançada.

	Dia 10	Implementação de **Ordenação** (orderBy) e **Filtros Simples** (status, type).	orderBy, where.
	Dia 11	Implementação de **Filtros Complexos** (AND, OR, NOT).	where: { OR: [...] }.
FASE 4: INTEGRIDADE (TRANSAÇÕES)	**Dia 12**	Lógica de **Pré-Transação**: Checagem de Saldo e Obtenção dos dados do Asset.	Lógica de Negócio.
	Dia 13	**Implementação da Transação:** Criar o $transaction para **diminuir** o saldo (Asset.quantity) na carteira de origem.	$transaction, Rollback.
	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** (.transform()): Limpeza de emails para minúsculas, formatação de valores.	Zod Transform, Type Safety.
	Dia 16	**Revisão e Limpeza:** Remover as any e *type assertions* 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.

Casos de Uso e Regras de Negócio do Projeto AssetFlow

1. Gestão do Usuário (`User`)

Caso de Uso	Regra de Negócio Chave (Onde a Lógica Entra)
Registro (Criação de Conta)	1. **Email Único:** Deve-se checar se o email já existe no banco de dados, retornando um `ConflictError` caso positivo. 2. **Validação:** Nome, Email e Senha devem seguir as regras Zod (Email válido, Senha min. 8 caracteres). 3. **Integridade (Refatoração Zod):** O email deve ser normalizado (ex: para minúsculas) antes de ser salvo no banco, garantindo que "Usuario@mail.com" e "usuario@mail.com" sejam o mesmo usuário. 4. **Segurança:** A senha deve ser *hasheada* (com `bcrypt`) antes de ser persistida.
Login e Autenticação	1. **Credenciais:** A senha fornecida pelo usuário deve ser comparada com o hash salvo no banco (`comparePassword`). 2. **Autorização:** Se as credenciais forem válidas, um token **JWT** deve ser gerado e enviado ao cliente para ser usado nas requisições futuras.
Proteção de Rotas	1. **`authMiddleware`:** Todas as rotas de `Wallet`, `Asset` e `Transaction` devem ser protegidas. O middleware deve checar a validade do JWT e extrair o `userId` do token para que os Services possam garantir a autoria.

2. Gestão de Carteiras (`Wallet`)

Caso de Uso	Regra de Negócio Chave (Onde a Lógica Entra)
Criação de Carteira	1. **Autoria:** O `userId` deve ser associado automaticamente à carteira recém-criada, estabelecendo a relação 1:N com o `User`. 2. **Validação:** O campo `type` da carteira (ex: Investimento, Reserva, Coleção) deve ser validado, idealmente com `z.enum()`.
Operação em Carteira	1. **Propriedade:** Qualquer operação (visualizar, atualizar, deletar) em uma `Wallet` deve ser precedida pela checagem se o `userId` logado é o proprietário daquela carteira (Regra de Autoria).

3. Transferência de Ativo (O Ponto mais Crítico do Portfólio)

Este caso de uso exige a **Atomicidade de Dados** e é o principal destaque do seu projeto (uso do `prisma.$transaction`).

Caso de Uso	Regra de Negócio Crítica (Exige `$transaction`)
Transferir X unidades de um Ativo (ex: Mover 1 Bitcoin da **Carteira A** para a **Carteira B**)	1. **Pré-Transação:** Checar se o `Asset` de origem existe e se o `userId` logado é o proprietário da **Carteira A**. 2. **Checagem de Saldo:** O valor `quantity` a ser transferido não pode ser maior do que o `Asset.quantity` consolidado na **Carteira A** (saldo). 3. **Abertura da Transação Atômica:** Todos os passos a seguir devem estar dentro do bloco `prisma.$transaction(async (tx) => { ... })`:
Passos Atômicos	* **Decremento:** Usar `tx.asset.update` para **diminuir** o campo `quantity` na **Carteira A**. * **Incremento:** Usar `tx.asset.update` para **aumentar** o campo `quantity` na **Carteira B**. * **Registro (Log Imutável):** Criar um novo registro na tabela `Transaction` com o `type: 'TRANSFERENCIA'` para justificar a movimentação.

Lógica do Rollback: Se qualquer um desses três passos falhar (por exemplo, erro de conexão ou regra de negócio), o `$transaction` garante que **todos os outros passos serão desfeitos** (rollback), preservando a integridade dos dados.

4. Consultas e Relatórios

Caso de Uso	Regra de Negócio Chave (Performance/Experiência)
Visualizar Ativos	1. **Paginação (Cursor):** A busca principal de `/assets` deve usar Paginação por Cursor (`cursor`) para ser mais eficiente em cenários de "infinite scroll" e busca rápida (Dia 9 do cronograma).
Visualizar Histórico	1. **Paginação (Offset):** A busca de `/transactions` deve usar Paginação por Offset (`page`, `pageSize`) para evitar sobrecarga (Dia 8 do cronograma). 2. **Filtros Complexos:** Deve ser possível combinar filtros usando operadores lógicos (`AND`, `OR`, `NOT`) na *query* do Prisma.