ChatGPT

Diagnóstico do Projeto PrevisionMarket

Resumo Geral

O repositório **PrevisionMarket** contém um back-end em TypeScript/Express e um front-end em React que integra mercados de previsão próprios e mercados espelhados da Polymarket. A aplicação permite que usuários depositem BRL ou USDC, participem de mercados e acompanhem preços. Durante a análise foram identificadas três áreas de melhoria importantes:

1. **Arquivos e rotas antigas não utilizadas** – há código legado que não é mais referenciado após as refatorações, ocupando espaço e atrapalhando a manutenção.
2. **Bug no depósito manual** – quando o administrador aprova um depósito, os tokens correspondentes devem ser "mintados" na carteira do administrador. Atualmente esse mint só é disparado em depósitos em BRL (`BRL3`) e mesmo assim depende de variáveis de ambiente que podem não estar configuradas; o processo falha silenciosamente.
3. **Gráfico da eleição de 2026** – o componente `PriceChart` deveria consumir o endpoint `/api/polymarket/history/:slug?range=...` para mostrar a variação de preço espelhada da Polymarket. Devido à construção da URL na função `queryFn` do `queryClient` e à ausência do parâmetro `range` na chamada, a API recebe caminhos inválidos como `/api/polymarket/history/brazil-election-2026-lula/1W`, retornando `[]` e consequentemente o gráfico não aparece [1] . [2]

Arquivos/Pastas Não Utilizados

A seguir estão arquivos e diretórios que não são mais referenciados e podem ser removidos (ou ao menos movidos para um diretório `legacy/`) para reduzir o tamanho do repositório e simplificar a base de código. Recomenda-se confirmar com o time se realmente não há dependências ocultas antes de excluir.

Arquivo/Dir	Motivo da não utilização
`server/routes.ts` – **endpoints** `/api/wallet/deposit` **e** `/api/wallet/withdraw`	Esses endpoints foram deixados como "MOCKED" e duplicam a lógica de depósito/saque manual introduzida posteriormente. A documentação e as telas do administrador não usam mais essas rotas; os depósitos e saques são tratados por `/api/deposits/request`, `/api/deposits/:id/approve` e `/api/deposits/:id/reject`.
`server/polymarket-cron.ts` e flag `ENABLE_POLYMARKET_CRON`	Antigo job de sincronização de mercados da Polymarket. Desde que foi implementado um worker externo (mirror) para armazenar snapshots, esse cron não é mais chamado. Está referenciado condicionalmente apenas no início de `server/routes.ts`, mas a flag nunca é habilitada em produção. Pode ser removido para evitar confusão.

Arquivo/Dir	Motivo da não utilização
`client/src/components/market-card.tsx`	Versão antiga do cartão de mercado. O front-end usa `ModernMarketCard` e `PolymarketMarketCard` para mostrar mercados. `market-card.tsx` não é importado em nenhum arquivo e pode ser deletado.
`client/src/components/polymarket-market-card.tsx` (antiga)	Há duas versões: uma simples e uma "modern". A versão usada nas páginas é a `ModernPolymarketCard`. A versão antiga (com slug e percentuais) não é mais importada.
`server/polymarket-sync.ts` ou scripts similares que estavam associados ao cron	Se ainda existirem scripts de sincronização antigos, também não são executados.

Bug do Depósito (minting)

Problema identificado

No fluxo de depósito manual, o usuário envia um comprovante via `/api/deposits/request`. O administrador aprova o depósito através de `/api/deposits/:id/approve` que credita o saldo do usuário e deve mintar tokens na carteira do admin. O mint é feito pela função `notifyMintToBRL3` em `server/brl3-client.ts`. Contudo:

- A rota de aprovação só chama `notifyMintToBRL3` quando a moeda do depósito é `BRL` [3] . Depósitos em USDC nunca mintam tokens, o que desbalanceia o suprimento.
- `notifyMintToBRL3` executa o mint apenas se todas as variáveis de ambiente relacionadas à Polygon (`ADMIN_PRIVATE_KEY`, `TOKEN_CONTRACT_ADDRESS`, etc.) estiverem configuradas; caso contrário ele apenas escreve um log e retorna, sem avisar o administrador [4] .
- A função registra a operação na tabela `onchainOperations` vinculada ao **ID do usuário que realizou o depósito**, mas a mintagem é feita na carteira do admin. Isso dificulta rastrear quais operações pertencem ao admin.
- Não há tratamento de exceções no mint; qualquer falha da RPC ou do contrato passa despercebida, deixando o depositante com saldo creditado mas sem tokens criados.

Proposta de correção

1. **Unificar o mint para todas as moedas** – mover a chamada `notifyMintToBRL3` para fora do `if (deposit.currency === 'BRL')` na rota `/api/deposits/:id/approve`, de modo que qualquer depósito aprovado gere um mint para o admin. Se desejar políticas diferentes por moeda, criar uma função `notifyMintForDeposit` que decida entre mintar `BRL3` ou outra stablecoin e chamá-la sempre que `status` passar para `approved`.

2. **Associar a operação ao admin** – alterar `notifyMintToBRL3(userId, amount)` para receber explicitamente `adminId` ou utilizar `getAdminUserId()` para a coluna `userId` da tabela `onchainOperations`, pois o mint é feito para a carteira do admin. Registrar o `depositUserId` em um campo separado (ex: `referenceUserId`) para auditoria.

3. **Adicionar tratamento de erros** – envolver a chamada ao `polygonClient.mintTo` com `try/catch`. Em caso de erro, atualizar a operação para `failed` e enviar notificação ou log de erro para o admin. Não deve simplesmente continuar o fluxo como se nada tivesse acontecido.

4. **Validar variáveis de ambiente** – na inicialização do servidor, verificar se `POLYGON_RPC_URL`, `ADMIN_PRIVATE_KEY`, `TOKEN_CONTRACT_ADDRESS` e `TOKEN_DECIMALS` estão presentes. Caso contrário, desabilitar o botão de aprovação de depósito ou exibir alerta ao admin. Atualmente a ausência desses valores causa logs silenciosos e pode induzir a pensar que o mint ocorreu.

5. **Remover endpoints de depósito antigos** – deletar `/api/wallet/deposit` e `/api/wallet/withdraw` de `routes.ts` para evitar que usuários chamem rotas que não mintam tokens. Todos os depósitos devem passar pelo fluxo manual (upload de comprovante).

Bug do Gráfico de Preço (Eleições 2026)

Problema identificado

O componente `PriceChart` utiliza `react-query` para buscar dados históricos de cada outcome na Polymarket. Ele define a `queryKey` como `['/api/polymarket/history', slug, range]`, e o `queryClient` possui um `queryFn` padrão que une os elementos do `queryKey` com `/`. Dessa forma, a requisição gerada para `slug=brazil-election-2026` e `range=1W` torna-se `/api/polymarket/history/brazil-election-2026/1W`, que não corresponde à rota implementada no servidor. A rota correta é `/api/polymarket/history/:slug?range=1W` [1] [2]. Como resultado, o endpoint retorna uma lista vazia e o gráfico aparece em branco.

Proposta de correção

1. **Construir a URL manualmente** – no componente `PriceChart`, substituir o uso de `useQueries` com `queryFn` implícita por chamadas explícitas a `fetch` ou `axios`, por exemplo:

```
// price-chart.tsx
const historyQuery = useQuery(['polymarket-history', slug, timeRange], async ()
=> {
  const res = await fetch(`/api/polymarket/history/${slug}?range=${timeRange}`);
  if (!res.ok) throw new Error('Erro ao carregar histórico');
  return res.json();
});
```

Isso garante que o parâmetro `range` seja enviado como query string em vez de ser confundido com parte do caminho.

1. **Ajustar** `queryClient` **ou** `queryKey` – se a equipe preferir manter o `queryFn` genérico, alterar o `queryKey` do `useQueries` para incluir o `range` como parte da chave em forma de objeto em vez de string. Por exemplo:

```
useQuery({
  queryKey: ['/api/polymarket/history', { slug, range: timeRange }],
  queryFn: async ({ queryKey }) => {
    const [_base, params] = queryKey;
    const url = `${_base}/${params.slug}?range=${params.range}`;
    const res = await fetch(url);
    return res.json();
  },
});
```

1. **Tratar ausência de dados** – exibir mensagem de “Dados indisponíveis” quando a API retornar lista vazia (situação comum se `ENABLE_POLYMARKET` estiver `false`).

2. **Documentar intervalos aceitos** – o front envia valores como `1H`, `6H`, `1D`, `1W`, `1M`, `ALL`. Certifique-se de que o servidor interpreta estes valores e retorne snapshots conforme esperado.

Prompt Detalhado para o Replit

Abaixo está um prompt detalhado (em português) para orientar a implementação das mudanças no Replit. **Copie o texto exatamente** ao iniciar o deploy, certificando-se de substituir `[YOUR_EMAIL]` pela conta do administrador e conferir as variáveis de ambiente.

```
**Contexto**
Você está trabalhando no repositório PrevisionMarket. O objetivo é remover
código legado, corrigir o fluxo de depósito para mintar tokens corretamente e
ajustar o gráfico de preço espelhado da Polymarket.

### 1. Limpeza de Código

- Abra o arquivo `server/routes.ts` e remova completamente as rotas `/api/
wallet/deposit` e `/api/wallet/withdraw`. Elas foram marcadas como “MOCKED” e
não são mais usadas.
- Procure por `polymarket-cron.ts` e a variável `ENABLE_POLYMARKET_CRON`. Delete
o arquivo `server/polymarket-cron.ts` e elimine todas as referências à flag no
código (inclusive no carregamento das rotas).
- Identifique e exclua componentes React antigos não utilizados: `client/src/
components/market-card.tsx` e a versão antiga de `client/src/components/
polymarket-market-card.tsx` (deixe apenas a versão moderna). Se houver scripts
de sincronização antigos como `server/polymarket-sync.ts`, remova-os.
- Após remover, execute `npm run lint` e `npm run build` para garantir que não
há imports quebrados.

### 2. Conserto do Depósito

1. No arquivo `server/routes.ts`, localize a rota `router.patch('/api/
```

````
deposits/:id/approve', ...)`. Atualmente há um trecho:

```ts
if (deposit.currency === 'BRL') {
  await notifyMintToBRL3(deposit.userId, deposit.amount);
}
````

Mova a chamada para **fora** desse `if` ou crie uma função genérica `notifyMintForDeposit(deposit: Deposit)` que sempre mintará tokens para a carteira do administrador, independentemente da moeda.

1. Abra `server/brl3-client.ts` e altere a assinatura de `notifyMintToBRL3` para receber também o `adminId` ou, alternativamente, obtenha o ID do admin com `getAdminUserId()`. Quando registrar a operação em `onchainOperations`, use `userId: adminId` e armazene `referenceUserId: deposit.userId` para rastrear quem fez o depósito.

2. Envolva a chamada a `polygonClient.mintTo(adminAddress, mintAmount)` em um bloco `try/catch`. Se ocorrer erro, atualize a coluna `status` da operação para `'failed'` e lance uma exceção para que a rota retorne erro ao admin. Dessa forma, o depósito não será tratado como concluído sem que os tokens sejam emitidos.

3. Na inicialização do servidor (ex.: `server/index.ts`), verifique se as variáveis `POLYGON_RPC_URL`, `ADMIN_PRIVATE_KEY`, `TOKEN_CONTRACT_ADDRESS` e `TOKEN_DECIMALS` estão definidas. Caso contrário, logar erro e impedir a aprovação de depósitos, exibindo um banner de configuração no painel do admin.

4. Teste o fluxo de depósito manual em um ambiente de testes. Simule um depósito em BRL e outro em USDC, aprove cada um e verifique no Polygonscan se a função `mint()` foi chamada para a carteira do admin.

3. Correção do Gráfico Polymarket

1. No componente `client/src/components/price-chart.tsx`, substitua o uso de `useQueries` que constrói a URL errada. Defina explicitamente o fetch para:

```
const { data, isLoading } = useQuery([
  'polymarket-history', slug, timeRange
], async () => {
  const res = await fetch(`/api/polymarket/history/${slug}?range=${timeRange}`);
  if (!res.ok) throw new Error('Erro ao carregar dados do gráfico');
  return res.json();
});
```

1. Elimine ou modifique a função genérica de `queryFn` no `queryClient.ts`. Se quiser reaproveitar, aceite um objeto como segundo parâmetro para permitir `queryString` (ex.: `{ slug, range }`).
```

2. Ajuste a UI do gráfico para exibir uma mensagem de “Dados indisponíveis” quando `data.length === 0`.

3. Certifique-se de que `ENABLE_POLYMARKET` esteja `true` nas variáveis de ambiente; caso contrário, habilite-o ou mostre uma mensagem informando que a Polymarket está desativada.

4. Verifique se a `slug` usada nos cartões (ex.: `brazil-presidential-election-2026`) corresponde exatamente às slugs armazenadas em `polymarketSnapshots` no banco de dados. Se necessário, atualize os cartões para usar as slugs corretas.

4. Testes e Deploy

- Após aplicar as mudanças, rode `npm run dev` e confirme:
- Ao aprovar um depósito, o saldo do usuário aumenta **e** é criado um `onchainOperation` para o admin; a transação aparece na Polygon.
- O gráfico da eleição de 2026 exibe a variação de preço igual ao Polymarket quando `range` é alterado para 1D/1W/1M/ALL.
- Commita as alterações e faça o deploy para produção. Anote qualquer mensagem de erro e ajuste conforme necessário.

Importante: garanta que suas modificações não afetem negativamente outros fluxos (compra/venda de shares, login, etc.). Faça testes manuais abrangentes antes de entregar. ```

---

[1] raw.githubusercontent.com
https://raw.githubusercontent.com/TheGuima1/PrevisionMarket/main/client/src/components/price-chart.tsx

[2] [3] raw.githubusercontent.com
https://raw.githubusercontent.com/TheGuima1/PrevisionMarket/main/server/routes.ts

[4] raw.githubusercontent.com
https://raw.githubusercontent.com/TheGuima1/PrevisionMarket/main/server/brl3-client.ts