🎰 Documentação de Integração - Casino Platform + Casino API

Visão Geral

Este documento descreve a integração entre o **casino-platform** (frontend Next.js) e o **casino-api** (backend NestJS).

Stack Tecnológico

Frontend (casino-platform)

- **Framework**: Next.js 14+ com TypeScript
- **HTTP Client**: Axios
- **State**: React Context
- **Styling**: Tailwind CSS

Backend (casino-api)

- **Framework**: NestJS
- **ORM**: Prisma
- **Database**: PostgreSQL
- **Auth**: JWT (JSON Web Tokens)
- **Multi-tenant**: Suportado via header `x-tenant-domain`

Configuração

Variáveis de Ambiente (Frontend)

```
# .env.local
NEXT_PUBLIC_API_URL=https://n8n-casino-api.hzkzun.easypanel.host
```

Para desenvolvimento local com backend rodando localmente:

```
NEXT_PUBLIC_API_URL=http://localhost:3001
```

Serviços Disponíveis

1. API Service (`api.ts`)

Configuração base do Axios com interceptors automáticos.

```
import { api } from '@/services/api';

// Exemplo de uso
const response = await api.get('/endpoint');
```

Features:
- Token JWT automático no header `Authorization`
- Header `x-tenant-domain` para multi-tenancy
- Tratamento global de erros (401, 403, 500)
- Timeout de 30s

2. Auth Service (`auth.ts`)

```
import { authService } from '@/services/auth';

// Login (salva token)
await authService.login(token);

// Obter perfil
const user = await authService.getProfile();

// Atualizar perfil
await authService.updateProfile({ name: 'Novo Nome' });

// Atualizar senha
await authService.updatePassword({ currentPassword: '123', newPassword: '456' });

// Logout
authService.logout();
```

3. Wallet Service (`wallet.ts`)

```
import { walletService } from '@/services/wallet';

// Obter saldo
const balance = await walletService.getBalance();
console.log(balance.balanceCents); // Saldo em centavos

// Criar depósito
const deposit = await walletService.createDeposit(5000, 'pix'); // R$50

// Criar saque
const withdraw = await walletService.createWithdraw(5000, 'pix', '12345678900');

// Helpers
walletService.formatBalance(5000); // "R$ 50,00"
walletService.toCents(50); // 5000
walletService.fromCents(5000); // 50
```

4. Payments Service (`payments.ts`)

```typescript
import { paymentsService } from '@/services/payments';

// Depósito PIX (retorna QR Code)
const deposit = await paymentsService.createDeposit({
  amount: 5000, // centavos
  currency: 'BRL',
  provider: 'PIX'
});
// deposit.pixCode - Código copia e cola
// deposit.pixQrCode - URL/Base64 do QR Code

// Saque
const withdraw = await paymentsService.createWithdraw({
  amount: 5000,
  pixKey: '12345678900'
});
```

5. Benefits Service (`benefits.ts`)

```typescript
import { benefitsService } from '@/services/benefits';

// Status dos benefícios
const status = await benefitsService.getBenefitsStatus();
// status.daily - Cashback diário
// status.weekly - Bônus semanal
// status.rakeback - Rakeback acumulado

// Resgatar benefício
await benefitsService.claimBenefit('DAILY_CASHBACK');
```

6. Games Service (`games.ts`)

```typescript
import { gamesService } from '@/services/games';

// Jogos em destaque
const games = await gamesService.getTopGames();
```

7. Tenants Service (`tenants.ts`)

```typescript
import { tenantsService } from '@/services/tenants';

// Config pública do tenant
const config = await tenantsService.getPublicConfig('localhost');
// config.theme, config.colors, config.pwa

// Stats (autenticado)
const stats = await tenantsService.getStats();
```

Endpoints do Backend

Método	Endpoint	Descrição	Auth
POST	`/auth/signup`	Criar tenant + admin	❌
POST	`/auth/register`	Registrar usuário	❌
POST	`/auth/login`	Login	❌
GET	`/auth/me`	Perfil atual	✅
PATCH	`/auth/me`	Atualizar perfil	✅
PUT	`/auth/me/password`	Atualizar senha	✅
POST	`/wallets/me`	Criar/obter carteira	✅
GET	`/wallets/me/balance`	Saldo da carteira	✅
POST	`/transactions/deposit`	Criar depósito	✅
POST	`/transactions/withdraw`	Criar saque	✅
POST	`/payments/deposit`	Depósito PIX	✅
POST	`/payments/webhook/:provider`	Webhook de pagamento	❌
GET	`/benefits/status`	Status dos benefícios	✅
POST	`/benefits/claim/:type`	Resgatar benefício	✅
GET	`/games/top`	Jogos em destaque	❌
GET	`/tenants/stats`	Estatísticas	✅
GET	`/public/tenants/:domain/config`	Config do tenant	❌

Modelos de Dados

User

```typescript
interface User {
  id: string;
  email: string;
  name?: string;
  phone?: string;
  avatar?: string;
  tenantId: string;
  createdAt: string;
  updatedAt: string;
}
```

Wallet Balance

```typescript
interface WalletBalance {
  walletId: string;
  balanceCents: number; // Sempre em centavos!
  currency: string; // 'BRL'
}
```

Transaction

```typescript
interface Transaction {
  id: string;
  type: 'DEPOSIT' | 'WITHDRAW';
  status: 'PENDING' | 'PAID' | 'PROCESSING' | 'COMPLETED' | 'FAILED' | 'CANCELED';
  amountCents: number;
  currency: string;
  provider: string;
  pixCode?: string;
  createdAt: string;
}
```

Fluxo de Autenticação

```
1. Usuário faz login/registro
2. Backend retorna { access_token, user }
3. Frontend salva token em localStorage
4. Axios interceptor adiciona token em todas as requests
5. Token expira → 401 → Redireciona para login
```

Multi-Tenancy

O sistema suporta múltiplos tenants (cassinos).

- Cada tenant tem seu próprio domínio
- O header `x-tenant-domain` é enviado automaticamente
- Usuários pertencem a um tenant específico
- Configurações visuais são por tenant

Tratamento de Erros

```
try {
  await api.post('/endpoint', data);
} catch (error) {
  if (error.response?.status === 401) {
    // Token expirado - redirecionar para login
  } else if (error.response?.status === 400) {
    // Dados inválidos - mostrar mensagem
    const message = error.response.data.message;
  } else {
    // Erro genérico
  }
}
```

Próximos Passos

1. **Histórico de Transações**: Adicionar endpoint `GET /wallets/me/transactions`
2. **Notificações**: Implementar sistema de notificações em tempo real
3. **KYC**: Verificação de identidade para saques maiores
4. **VIP System**: Sistema de níveis VIP baseado em apostas