📋 Especificação Técnica Completa - Plataforma de Automação Web

Data: 16 de Dezembro de 2025
Versão: 1.0
Status: Análise e Planejamento

📑 Índice

---

1. Resumo Executivo

1.1 Visão Geral

A plataforma atual é um sistema de automação web em Python que gerencia múltiplas instâncias de navegadores Chrome para realizar tarefas automatizadas em plataformas como YouTube, Spotify, Deezer e TikTok. O sistema utiliza técnicas avançadas de anti-detecção, gerenciamento de proxies e scripts JavaScript customizados para cada plataforma.

1.2 Objetivo da Nova Plataforma

Criar uma plataforma web moderna e escalável usando **Next.js 14+** (App Router), **TypeScript**, e **Node.js** que:
- Ofereça interface web intuitiva para gerenciar automações
- Mantenha todas as funcionalidades existentes
- Adicione capacidades de monitoramento em tempo real
- Permita gerenciamento multi-usuário
- Forneça analytics e relatórios
- Seja escalável e cloud-ready

1.3 Tecnologias Core

- **Frontend:** Next.js 14+ (App Router), TypeScript, TailwindCSS, shadcn/ui
- **Backend:** Next.js API Routes, Node.js, Playwright (substitui Selenium)
- **Database:** PostgreSQL + Prisma ORM
- **Cache:** Redis
- **Queue:** BullMQ (para jobs assíncronos)
- **Monitoramento:** Socket.io para real-time updates
- **Deployment:** Docker + Kubernetes / Vercel + AWS Lambda

---

2. Análise da Plataforma Atual (Python)

2.1 Estrutura de Arquivos

```
Sistema Atual (Python)
├── browser_sync_improved.py      # Script principal (208 linhas)
├── proxy_manager.py              # Gerenciador de proxies (177 linhas)
├── platform_script.py            # Scripts JS por plataforma (259 linhas)
├── teste_rapido.py               # Script de testes (100 linhas)
├── requirements_browser.txt      # Dependências Python
├── executar_melhorado.bat        # Script de execução Windows
├── README_BROWSER_SYNC.md        # Documentação técnica
└── COMECE_AQUI.md               # Guia de início rápido
```

2.2 Componentes Principais

2.2.1 BrowserInstance (browser_sync_improved.py)

Responsabilidades:
- Gerenciar uma única instância de navegador Chrome
- Aplicar configurações anti-detecção
- Executar scripts de automação em thread separada
- Gerenciar lifecycle (start, navigate, cleanup)

Código-chave:

```python
class BrowserInstance:
    def __init__(self, instance_id, user_agent, proxy=None):
        self.instance_id = instance_id
        self.user_agent = user_agent
        self.proxy = proxy
        self.driver = None
        self.temp_dir = None
        self.is_running = False
        self.automation_thread = None
```

Configurações Chrome:
- User-agent customizado
- Perfil temporário único (`--user-data-dir`)
- Desabilitar automação detectável
- Mute de áudio
- Janela com dimensões fixas (1280x720)

2.2.2 BrowserSync (browser_sync_improved.py)

Responsabilidades:
- Coordenar múltiplas instâncias (padrão: 20)
- Buscar e distribuir proxies
- Sincronizar navegação entre instâncias
- Interface de comando interativa

Fluxo de Execução:

```
1. Inicialização → fetch_proxies()
2. Criação de instâncias → start_all_instances()
3. Loop de controle → navigate_all(url)
4. Cleanup → close_all()
```

2.2.3 ProxyManager (proxy_manager.py)

Responsabilidades:
- Buscar proxies de múltiplas fontes
- Validar proxies em paralelo (até 50 threads)
- Retornar lista de proxies funcionais

Fontes de Proxies:

```
proxy_sources = [
    'https://api.proxyscrape.com/v2/?request=get...',
    'https://www.proxy-list.download/api/v1/get?type=http',
    'https://raw.githubusercontent.com/TheSpeedX/PROXY-List/master/http.txt',
    'https://raw.githubusercontent.com/ShiftyTR/Proxy-List/master/http.txt',
    'https://raw.githubusercontent.com/monosans/proxy-list/main/proxies/http.txt',
]
```

Validação:
- Teste de conectividade HTTP (httpbin.org/ip)
- Timeout de 15 segundos
- Execução paralela com threading
- Status tracking em tempo real

2.2.4 PlatformScripts (platform_script.py)

Responsabilidades:
- Fornecer scripts JavaScript específicos por plataforma
- Implementar lógica de auto-skip, anti-pause, simulação de atividade

Plataformas Suportadas:
- **YouTube:** Skip ads, prevent pause, auto-unmute, simulate activity
- **Spotify:** Prevent pause, accept cookies, simulate activity
- **Deezer:** Prevent pause, accept cookies, simulate activity
- **TikTok:** Prevent pause, auto-scroll, simulate activity
- **Universal:** Close popups, remove overlays

2.3 Fluxo de Dados

```
┌─────────────────────────────────────────────────────────────┐
│                    SISTEMA ATUAL (Python)                   │
└─────────────────────────────────────────────────────────────┘

[1] Usuário executa script
        ↓
[2] ProxyManager busca e valida proxies (1-3 min)
        ↓
[3] BrowserSync cria 20 instâncias de navegador
    • Cada instância recebe:
      - ID único
      - User-agent rotativo
      - Proxy único (se disponível)
        ↓
[4] Cada BrowserInstance:
    • Cria perfil temporário
    • Aplica configurações anti-detecção
    • Inicia thread de automação
    • Executa scripts JS periodicamente (15-30s random)
        ↓
[5] Loop de controle interativo:
    • Aguarda comandos do usuário
    • Navega todas as instâncias sincronizadamente
    • Executa scripts baseado na URL atual
        ↓
[6] Scripts JavaScript são executados em cada instância:
    • Detecta plataforma pela URL
    • Executa ações específicas (skip ad, prevent pause, etc.)
    • Simula atividade do usuário
        ↓
[7] Cleanup ao sair:
    • Fecha todos os navegadores
    • Remove perfis temporários
```

2.4 Limitações Identificadas

1. **Escalabilidade:**
   - Limitado a uma máquina
   - Sem distribuição de carga
   - Máximo ~50 instâncias por máquina (limitação de recursos)

2. **Monitoramento:**
   - Sem dashboard visual
   - Logs apenas no terminal
   - Sem métricas ou analytics

3. **Gerenciamento:**
   - Interface CLI apenas
   - Sem multi-usuário
   - Sem histórico ou agendamento

4. **Confiabilidade:**
   - Proxies gratuitos instáveis
   - Sem retry automático
   - Sem health checks

5. **Segurança:**
   - Sem autenticação
   - Sem criptografia de dados
   - Credenciais hardcoded

---

3. Técnicas de Anti-Detecção Identificadas

3.1 Nível de Navegador (Chrome Options)

3.1.1 Flags do Chrome

```
# Ocultar automação
chrome_options.add_argument("--disable-blink-features=AutomationControlled")

# Remover indicadores de automação
chrome_options.add_experimental_option("excludeSwitches", ["enable-automation"])
chrome_options.add_experimental_option('useAutomationExtension', False)

# Sandbox e segurança
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")

# Aparência natural
chrome_options.add_argument("window-size=1280,720")
chrome_options.add_argument("--disable-infobars")
chrome_options.add_argument("--mute-audio")
```

Efetividade:
- ✅ Oculta propriedades do WebDriver
- ✅ Remove banners de automação
- ✅ Dimensões de janela naturais

3.1.2 User-Agents Rotativos

```
USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/120.0.0.0 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/119.0.0.0 Safari/537.36",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
120.0.0.0 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:121.0) Gecko/20100101 Firefox/
121.0",
]
```

Estratégia:
- Rotação circular entre instâncias
- Versões recentes do Chrome/Firefox
- Mix de sistemas operacionais (Windows, Linux)

3.1.3 Perfis Temporários

```
self.temp_dir = tempfile.mkdtemp(prefix=f"browser_{self.instance_id}_")
chrome_options.add_argument(f"--user-data-dir={self.temp_dir}")
```

Benefícios:
- Cookies isolados por instância
- Histórico separado
- Sem compartilhamento de sessão

3.2 Nível de Proxy

3.2.1 IP Único por Instância

```
if self.proxy:
    chrome_options.add_argument(f"--proxy-server=http://{self.proxy['ip']}:{self.proxy
['port']}")
```

Efetividade:
- ✅ Cada janela aparece como usuário diferente
- ✅ Distribui carga geograficamente
- ⚠️ Depende da qualidade dos proxies

3.3 Nível de Comportamento (JavaScript)

3.3.1 Randomização de Tempo

```
# Intervalo aleatório entre 15-30 segundos
AD_SKIP_INTERVAL_RANGE = (15, 30)
sleep_time = random.uniform(AD_SKIP_INTERVAL_RANGE[0], AD_SKIP_INTERVAL_RANGE[1])
time.sleep(sleep_time)
```

Objetivo: Quebrar padrões de sincronização detectáveis

3.3.2 Simulação de Atividade do Usuário

Mouse Movement Aleatório:

```
var x = Math.floor(Math.random() * window.innerWidth);
var y = Math.floor(Math.random() * window.innerHeight);

document.dispatchEvent(new MouseEvent('mousemove', {
    bubbles: true,
    cancelable: true,
    clientX: x,
    clientY: y
}));
```

Scroll Aleatório:

```
var scrollAmount = Math.random() < 0.5 ? 50 : -50;
window.scrollBy(0, scrollAmount);
```

Efetividade:
- ✅ Simula comportamento humano
- ✅ Previne detecção por inatividade
- ✅ Triggers event listeners naturalmente

3.3.3 Prevenção de Pausas Inteligente

```
// YouTube: Detecta e retoma vídeo pausado
var video = document.querySelector('video');
if (video && video.paused) {
    video.play();
    console.log('▶️ Vídeo retomado!');
}

// Remove overlay de pausa
var pauseOverlay = document.querySelector('.ytp-pause-overlay');
if (pauseOverlay) {
    pauseOverlay.style.display = 'none';
}
```

3.4 Nível de Detecção Específica

3.4.1 Auto-Skip de Anúncios (YouTube)

```
// Múltiplos seletores para diferentes versões do YouTube
var skipButton = document.querySelector('.ytp-ad-skip-button, .ytp-ad-skip-button-modern, .ytp-skip-ad-button');
if (skipButton) {
    skipButton.click();
    console.log('✅ Anúncio pulado!');
}
```

3.4.2 Fechamento de Modais (Spotify)

```
var modal = document.querySelector('[role="dialog"]');
if (modal) {
    var continueBtn = modal.querySelector('button');
    if (continueBtn && continueBtn.textContent.includes('Continuar')) {
        continueBtn.click();
        console.log('✅ Modal de inatividade fechado!');
    }
}
```

3.5 Gaps e Melhorias Necessárias

3.5.1 Técnicas NÃO Implementadas (mas recomendadas)

1. **Canvas Fingerprinting Protection:**
   - Atual: Não implementado
   - Recomendado: Injetar noise em canvas APIs

2. **WebGL Fingerprinting Protection:**
   - Atual: Não implementado
   - Recomendado: Randomizar WebGL vendor/renderer

3. **Audio Context Fingerprinting:**
   - Atual: Áudio apenas mutado
   - Recomendado: Randomizar audio context properties

4. **Timezone e Locale Spoofing:**
   - Atual: Usa timezone do sistema
   - Recomendado: Alinhar timezone com proxy location

5. **Font Fingerprinting:**
   - Atual: Fontes do sistema expostas
   - Recomendado: Limitar fontes detectáveis

6. **Battery API Blocking:**
   - Atual: Não bloqueado
   - Recomendado: Remover navigator.battery

7. **WebRTC IP Leak Protection:**
   - Atual: Não protegido
   - Recomendado: Desabilitar WebRTC ou usar fake IPs

3.5.2 Melhorias em Técnicas Existentes

1. **User-Agent Intelligence:**
   - Atual: Lista estática
   - Melhoria: Alinhar UA com proxy OS/browser real

2. **Timing Attacks:**
   - Atual: Random 15-30s
   - Melhoria: Padrões mais humanos (curva gaussiana)

3. **Scroll Behavior:**
   - Atual: Scroll fixo ±50px
   - Melhoria: Scroll suave com aceleração/desaceleração

4. **Click Patterns:**
   - Atual: Apenas MouseMove
   - Melhoria: Adicionar clicks ocasionais, double-clicks

---

4. Estrutura de Gerenciamento de Proxies

4.1 Arquitetura Atual

```
ProxyManager
│
├── Fontes de Proxies (5 sources)
│   ├── ProxyScrape API
│   ├── Proxy-List Download API
│   └── GitHub Repositories (3)
│
├── Fetching Pipeline
│   ├── fetch_all_proxies()
│   │   └── fetch_proxies_from_source() [paralelo]
│   └── Deduplicação (set)
│
├── Validação Pipeline
│   ├── validate_proxies_threaded() [50 workers]
│   │   └── validate_proxy() [15s timeout]
│   └── Progress Tracking
│
└── Output
    └── Lista de proxies validados { ip, port, full }
```

4.2 Formato de Dados

```
# Formato de Proxy
{
    'ip': '103.152.112.162',
    'port': '80',
    'full': '103.152.112.162:80'
}
```

4.3 Processo de Validação

```
def validate_proxy(self, proxy: Dict, test_url: str = 'http://httpbin.org/ip',
timeout: int = 15) -> bool:
    try:
        proxy_dict = {
            'http': f"http://{proxy['full']}",
            'https': f"http://{proxy['full']}"
        }
        response = requests.get(test_url, proxies=proxy_dict, timeout=timeout)
        return response.status_code == 200
    except:
        return False
```

Características:

- Timeout generoso (15s) para proxies lentos
- Teste simples (HTTP 200 OK)
- Falha silenciosa (except pass)

4.4 Performance

Métricas Observadas:

- Tempo de busca: 10-30 segundos

- Tempo de validação: 1-3 minutos
- Taxa de sucesso: 5-20% de proxies válidos
- Throughput: ~100-500 proxies testados/minuto (50 threads)

4.5 Limitações Identificadas

1. **Qualidade dos Proxies:**
   - Proxies gratuitos são instáveis
   - Alta taxa de falha (80-95%)
   - Velocidade variável

2. **Sem Geolocalização:**
   - Não identifica país/região do proxy
   - Impossível alinhar com user-agent

3. **Sem Caching:**
   - Re-valida todos os proxies a cada execução
   - Não mantém lista de proxies conhecidos

4. **Sem Health Checks:**
   - Proxy pode falhar durante execução
   - Sem retry ou fallback

5. **Sem Rotação:**
   - Proxy fixo durante toda a sessão
   - Não rotaciona se falhar

4.6 Melhorias Necessárias

4.6.1 Sistema de Cache de Proxies

```
interface ProxyCache {
    proxy: Proxy;
    lastValidated: Date;
    successRate: number;
    avgResponseTime: number;
    geolocation: {
        country: string;
        region: string;
        city: string;
    };
}
```

4.6.2 Health Monitoring

```
interface ProxyHealthCheck {
    proxyId: string;
    checkInterval: number; // 5 minutos
    lastCheck: Date;
    status: 'healthy' | 'degraded' | 'down';
    consecutiveFailures: number;
}
```

4.6.3 Smart Rotation

```
interface ProxyRotationStrategy {
    type: 'round-robin' | 'weighted' | 'geo-based';
    rotateOnFailure: boolean;
    rotateInterval?: number;
    preferredCountries?: string[];
}
```

4.6.4 Paid Proxy Integration

Suportar serviços premium:
- BrightData (Luminati)
- Smartproxy
- Oxylabs
- ProxyMesh

---

5. Funcionalidades da Plataforma

5.1 Funcionalidades Core Implementadas

5.1.1 Gerenciamento de Múltiplas Instâncias

- ✅ Criar N instâncias simultâneas (padrão: 20)
- ✅ Lifecycle management (start, stop, cleanup)
- ✅ Sincronização de navegação
- ✅ Isolamento de perfis

5.1.2 Automação Inteligente por Plataforma

- ✅ YouTube: Skip ads, prevent pause, auto-unmute
- ✅ Spotify: Prevent pause, accept cookies
- ✅ Deezer: Prevent pause, accept cookies
- ✅ TikTok: Auto-scroll, prevent pause
- ✅ Universal: Close popups, remove overlays

5.1.3 Gerenciamento de Proxies

- ✅ Busca automática de proxies gratuitos
- ✅ Validação paralela
- ✅ Distribuição por instância

5.1.4 Interface de Controle

- ✅ CLI interativa
- ✅ Comandos: navegação, status, sair
- ✅ Atalhos para plataformas populares

5.2 Funcionalidades Ausentes (Necessárias)

5.2.1 Interface Gráfica

- ❌ Dashboard web
- ❌ Visualização de instâncias em tempo real
- ❌ Logs visuais

- ❌ Configuração via UI

5.2.2 Multi-Usuário

- ❌ Autenticação
- ❌ Autorização (roles, permissions)
- ❌ Isolamento de recursos por usuário
- ❌ Quotas e limites

5.2.3 Agendamento

- ❌ Criar jobs agendados
- ❌ Cron-like scheduling
- ❌ Recurring tasks
- ❌ Task queue

5.2.4 Analytics

- ❌ Métricas de execução
- ❌ Success/failure rates
- ❌ Tempo médio por tarefa
- ❌ Uso de recursos

5.2.5 Monitoramento

- ❌ Health checks de instâncias
- ❌ Alertas de falhas
- ❌ Logs centralizados
- ❌ Debugging tools

5.2.6 API

- ❌ REST API para integração
- ❌ Webhooks
- ❌ API keys management
- ❌ Rate limiting

5.2.7 Escalabilidade

- ❌ Distribuição multi-servidor
- ❌ Load balancing
- ❌ Auto-scaling
- ❌ Container orchestration

---

6. Requisitos para Nova Plataforma Web

6.1 Requisitos Funcionais

6.1.1 Autenticação e Autorização

- **RF001:** Sistema de registro e login de usuários
- **RF002:** Autenticação via email/senha e OAuth (Google, GitHub)
- **RF003:** Recuperação de senha via email
- **RF004:** Sistema de roles (Admin, User, Viewer)
- **RF005:** Permissões granulares por recurso

- **RF006:** Multi-tenancy (organizações)

6.1.2 Dashboard

- **RF007:** Visão geral de instâncias ativas
- **RF008:** Métricas em tempo real (CPU, RAM, status)
- **RF009:** Gráficos de utilização histórica
- **RF010:** Lista de tasks em execução
- **RF011:** Logs centralizados com filtros
- **RF012:** Alertas e notificações

6.1.3 Gerenciamento de Instâncias

- **RF013:** Criar nova instância (configuração via form)
- **RF014:** Iniciar/parar instâncias individuais ou em lote
- **RF015:** Visualizar status de cada instância
- **RF016:** Conectar/desconectar proxies dinamicamente
- **RF017:** Ver logs de cada instância
- **RF018:** Screenshot de instância em tempo real
- **RF019:** Restart automático em caso de falha

6.1.4 Gerenciamento de Proxies

- **RF020:** Adicionar proxies manualmente
- **RF021:** Importar proxies via arquivo/API
- **RF022:** Validar proxies sob demanda
- **RF023:** Ver geolocalização de proxies
- **RF024:** Blacklist de proxies ruins
- **RF025:** Rotação automática de proxies
- **RF026:** Integração com serviços pagos

6.1.5 Scripts e Automação

- **RF027:** Editor de scripts JavaScript
- **RF028:** Biblioteca de scripts por plataforma
- **RF029:** Testar scripts em instância isolada
- **RF030:** Versionamento de scripts
- **RF031:** Compartilhamento de scripts (marketplace)

6.1.6 Agendamento

- **RF032:** Criar jobs agendados (cron syntax)
- **RF033:** Recurring tasks (diário, semanal, etc.)
- **RF034:** Fila de tasks com prioridades
- **RF035:** Retry automático em falhas
- **RF036:** Notificações ao completar/falhar

6.1.7 Analytics

- **RF037:** Relatórios de execução (success rate, tempo médio)
- **RF038:** Exportar dados em CSV/JSON
- **RF039:** Gráficos de tendências
- **RF040:** Comparação entre períodos
- **RF041:** Alertas quando métricas anormais

6.1.8 API

- **RF042:** REST API completa
- **RF043:** Documentação interativa (Swagger)
- **RF044:** API keys por usuário
- **RF045:** Webhooks para eventos
- **RF046:** Rate limiting por usuário/IP

6.2 Requisitos Não-Funcionais

6.2.1 Performance

- **RNF001:** Suportar 1000+ instâncias simultâneas (distribuídas)
- **RNF002:** Latência < 100ms para dashboard updates
- **RNF003:** Tempo de inicialização de instância < 10s
- **RNF004:** Dashboard deve carregar < 2s

6.2.2 Escalabilidade

- **RNF005:** Arquitetura horizontalmente escalável
- **RNF006:** Stateless (permitir múltiplos servidores)
- **RNF007:** Suportar 10,000+ usuários concorrentes
- **RNF008:** Database com read replicas

6.2.3 Confiabilidade

- **RNF009:** Uptime de 99.9%
- **RNF010:** Backup automático diário
- **RNF011:** Disaster recovery plan
- **RNF012:** Health checks a cada 30s

6.2.4 Segurança

- **RNF013:** HTTPS obrigatório
- **RNF014:** Passwords com bcrypt (custo 12+)
- **RNF015:** Rate limiting em todas APIs
- **RNF016:** Sanitização de inputs
- **RNF017:** CSP headers
- **RNF018:** Secrets em variáveis de ambiente
- **RNF019:** Audit logs de ações críticas

6.2.5 Usabilidade

- **RNF020:** Interface responsiva (mobile-first)
- **RNF021:** Suporte a dark/light mode
- **RNF022:** I18n (EN, PT-BR, ES)
- **RNF023:** Acessibilidade (WCAG 2.1 AA)
- **RNF024:** Onboarding interativo

6.2.6 Manutenibilidade

- **RNF025:** Cobertura de testes > 80%
- **RNF026:** Documentação inline (JSDoc)
- **RNF027:** CI/CD pipeline
- **RNF028:** Linting e formatting automático
- **RNF029:** Monitoramento e observabilidade

7. Arquitetura Proposta (NextJS/TypeScript)

7.1 Visão Geral


NOVA PLATAFORMA WEB
FRONTEND LAYER
Next.js 14+ App Router + React + TypeScript
Dashboard
Real-time monitoring com Socket.io
Instance Manager
CRUD de instâncias com live status
Proxy Manager
Validação e geolocalização
Script Editor
Monaco Editor com syntax highlighting
Analytics
Charts com Recharts/Chart.js
Settings
User preferences, API keys
HTTP/REST + WebSocket
API LAYER
Next.js API Routes (Edge/Node Runtime)
Auth API
NextAuth.js (JWT + Session)
Instance API
CRUD + control (start/stop/restart)
Proxy API
Fetch, validate, rotate
Script API
CRUD + execute
Job API
Schedule, queue, status
Analytics API
Metrics, reports
Webhook API
Event subscriptions
SERVICE LAYER
Business Logic (TypeScript Classes)
BrowserService
Playwright wrapper + anti-detect
ProxyService
Fetch, validate, cache, rotate
ScriptService
Execute JS in browser contexts
JobService
Task scheduling com BullMQ
AnalyticsService
Aggregate metrics
NotificationSvc
Email, Slack, webhooks
DATA LAYER
PostgreSQL
Primary database (Prisma ORM)
Redis
Cache + session store + queue
S3-compatible
Screenshots, logs, exports
INFRASTRUCTURE
Docker Containers
Browser pools isolados
Kubernetes/ECS
Orchestration + auto-scaling
Load Balancer
Nginx/Cloudflare
Monitoring
Prometheus + Grafana
Logging
Loki/ELK Stack

7.2 Estrutura de Diretórios

```
platform-web/
├── src/
│   ├── app/                          # Next.js App Router
│   │   ├── (auth)/
│   │   │   ├── login/
│   │   │   ├── register/
│   │   │   └── forgot-password/
│   │   ├── (dashboard)/
│   │   │   ├── layout.tsx
│   │   │   ├── page.tsx              # Dashboard home
│   │   │   ├── instances/
│   │   │   │   ├── page.tsx          # List instances
│   │   │   │   ├── [id]/
│   │   │   │   │   └── page.tsx      # Instance detail
│   │   │   │   └── new/
│   │   │   │       └── page.tsx      # Create instance
│   │   │   ├── proxies/
│   │   │   │   ├── page.tsx
│   │   │   │   └── import/
│   │   │   ├── scripts/
│   │   │   │   ├── page.tsx
│   │   │   │   ├── [id]/
│   │   │   │   └── editor/
│   │   │   ├── jobs/
│   │   │   │   ├── page.tsx
│   │   │   │   └── [id]/
│   │   │   ├── analytics/
│   │   │   │   └── page.tsx
│   │   │   └── settings/
│   │   │       └── page.tsx
│   │   ├── api/
│   │   │   ├── auth/                 # NextAuth.js routes
│   │   │   │   └── [...nextauth]/
│   │   │   ├── instances/
│   │   │   │   ├── route.ts          # GET, POST
│   │   │   │   ├── [id]/
│   │   │   │   │   ├── route.ts      # GET, PUT, DELETE
│   │   │   │   │   ├── start/
│   │   │   │   │   ├── stop/
│   │   │   │   │   └── screenshot/
│   │   │   │   └── batch/
│   │   │   ├── proxies/
│   │   │   │   ├── route.ts
│   │   │   │   ├── validate/
│   │   │   │   └── fetch/
│   │   │   ├── scripts/
│   │   │   │   ├── route.ts
│   │   │   │   └── execute/
│   │   │   ├── jobs/
│   │   │   │   ├── route.ts
│   │   │   │   └── [id]/
│   │   │   ├── analytics/
│   │   │   │   ├── overview/
│   │   │   │   └── export/
│   │   │   └── webhooks/
│   │   │       └── route.ts
│   │   ├── layout.tsx
│   │   ├── globals.css
│   │   └── providers.tsx
│   ├── components/
│   │   ├── ui/                       # shadcn/ui components
│   │   │   ├── button.tsx
```

```
│   │   │   ├── card.tsx
│   │   │   ├── dialog.tsx
│   │   │   ├── table.tsx
│   │   │   └── ...
│   │   ├── dashboard/
│   │   │   ├── sidebar.tsx
│   │   │   ├── header.tsx
│   │   │   └── stats-card.tsx
│   │   ├── instances/
│   │   │   ├── instance-card.tsx
│   │   │   ├── instance-table.tsx
│   │   │   ├── instance-form.tsx
│   │   │   └── instance-logs.tsx
│   │   ├── proxies/
│   │   │   ├── proxy-table.tsx
│   │   │   ├── proxy-validator.tsx
│   │   │   └── proxy-map.tsx
│   │   ├── scripts/
│   │   │   ├── script-editor.tsx     # Monaco Editor
│   │   │   ├── script-library.tsx
│   │   │   └── script-tester.tsx
│   │   └── charts/
│   │       ├── line-chart.tsx
│   │       ├── bar-chart.tsx
│   │       └── pie-chart.tsx
│   ├── lib/
│   │   ├── services/
│   │   │   ├── browser.service.ts    # Playwright logic
│   │   │   ├── proxy.service.ts
│   │   │   ├── script.service.ts
│   │   │   ├── job.service.ts
│   │   │   ├── analytics.service.ts
│   │   │   └── notification.service.ts
│   │   ├── utils/
│   │   │   ├── anti-detect.ts        # Anti-detection helpers
│   │   │   ├── user-agents.ts
│   │   │   ├── random.ts
│   │   │   └── validation.ts
│   │   ├── db/
│   │   │   ├── prisma.ts             # Prisma client
│   │   │   └── redis.ts              # Redis client
│   │   ├── auth/
│   │   │   └── next-auth.config.ts
│   │   └── constants.ts
│   ├── types/
│   │   ├── instance.ts
│   │   ├── proxy.ts
│   │   ├── script.ts
│   │   ├── job.ts
│   │   └── analytics.ts
│   └── hooks/
│       ├── use-instances.ts
│       ├── use-proxies.ts
│       ├── use-realtime.ts           # Socket.io hook
│       └── use-theme.ts
├── prisma/
│   ├── schema.prisma
│   ├── migrations/
│   └── seed.ts
├── public/
│   ├── scripts/                      # Pre-built scripts
│   │   ├── youtube.js
│   │   └── spotify.js
```

```
│   │   └── ...
│   └── assets/
├── docker/
│   ├── Dockerfile
│   ├── docker-compose.yml
│   └── nginx.conf
├── tests/
│   ├── unit/
│   ├── integration/
│   └── e2e/
├── .env.example
├── .eslintrc.json
├── .prettierrc
├── tsconfig.json
├── next.config.js
├── tailwind.config.ts
└── package.json
```

7.3 Fluxo de Dados

7.3.1 Criação de Instância

```
[Frontend]
User clicks "Create Instance" button
  ↓
Fills form (name, platform, proxy, script)
  ↓
Submits → POST /api/instances
  ↓
[API Route]
Validates input (Zod schema)
  ↓
Authenticates user (NextAuth)
  ↓
Calls BrowserService.createInstance()
  ↓
[Service Layer]
1. Assigns proxy from pool
2. Creates temp profile directory
3. Generates anti-detect config
4. Saves to database (Prisma)
5. Emits "instance.created" event (Socket.io)
6. Adds to job queue (BullMQ)
  ↓
[Job Worker]
1. Pulls from queue
2. Launches Playwright browser with config
3. Injects anti-detect scripts
4. Navigates to start URL
5. Starts automation thread
6. Updates status in DB
7. Emits "instance.started" event
  ↓
[Frontend]
Receives Socket.io event
  ↓
Updates UI with new instance status
```

7.3.2 Real-time Monitoring

```
[Backend]
Job worker monitors instance every 5s
  ↓
Collects metrics (CPU, RAM, status, current URL)
  ↓
Stores in Redis (TTL: 1 hour)
  ↓
Emits "instance.metrics" event via Socket.io
  ↓
[Frontend]
useRealtime() hook listens to Socket.io
  ↓
Updates React state
  ↓
Recharts renders updated metrics
```

7.4 Database Schema (Prisma)

```
// prisma/schema.prisma

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

// ---------- AUTH & USER ----------

model User {
  id            String    @id @default(cuid())
  email         String    @unique
  emailVerified DateTime?
  name          String?
  image         String?
  password      String?   // bcrypt hash
  role          Role      @default(USER)

  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt

  accounts      Account[]
  sessions      Session[]
  instances     Instance[]
  apiKeys       ApiKey[]
  webhooks      Webhook[]

  organizationId String?
  organization   Organization? @relation(fields: [organizationId], references: [id])

  @@index([email])
  @@index([organizationId])
}

enum Role {
  ADMIN
  USER
  VIEWER
}

model Account {
  id                String  @id @default(cuid())
  userId            String
  type              String
  provider          String
  providerAccountId String
  refresh_token     String? @db.Text
  access_token      String? @db.Text
  expires_at        Int?
  token_type        String?
  scope             String?
  id_token          String? @db.Text
  session_state     String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
  @@index([userId])
```

```
}

model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique
  userId       String
  expires      DateTime
  user         User     @relation(fields: [userId], references: [id], onDelete: Cas-
cade)

  @@index([userId])
}

model Organization {
  id        String   @id @default(cuid())
  name      String
  slug      String   @unique
  plan      Plan     @default(FREE)

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  users     User[]
  instances Instance[]
  proxies   Proxy[]
}

enum Plan {
  FREE
  PRO
  ENTERPRISE
}

// ---------- API & WEBHOOKS ----------

model ApiKey {
  id        String   @id @default(cuid())
  key       String   @unique
  name      String
  userId    String
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  lastUsed  DateTime?
  createdAt DateTime @default(now())

  @@index([key])
  @@index([userId])
}

model Webhook {
  id        String   @id @default(cuid())
  url       String
  events    String[] // ["instance.started", "instance.stopped", ...]
  secret    String   // for signature verification
  active    Boolean  @default(true)

  userId    String
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  createdAt DateTime @default(now())

  deliveries WebhookDelivery[]
```

```
  @@index([userId])
}

model WebhookDelivery {
  id          String   @id @default(cuid())
  webhookId   String
  webhook     Webhook  @relation(fields: [webhookId], references: [id], onDelete: Cas-
cade)

  event       String
  payload     Json
  response    Json?
  statusCode  Int?
  success     Boolean

  createdAt   DateTime @default(now())

  @@index([webhookId])
  @@index([createdAt])
}

// ---------- INSTANCES ----------

model Instance {
  id              String           @id @default(cuid())
  name            String
  status          InstanceStatus   @default(PENDING)
  platform        Platform
  startUrl        String

  // Browser config
  userAgent       String
  windowSize      String           @default("1280x720")
  profilePath     String?

  // Proxy
  proxyId         String?
  proxy           Proxy?           @relation(fields: [proxyId], references: [id])

  // Script
  scriptId        String?
  script          Script?          @relation(fields: [scriptId], references: [id])

  // Ownership
  userId          String
  user            User             @relation(fields: [userId], references: [id], onDel
ete: Cascade)
  organizationId  String?
  organization    Organization?    @relation(fields: [organizationId], references: [id
])

  // Metadata
  currentUrl      String?
  lastActivity    DateTime?
  errorMessage    String?

  createdAt       DateTime         @default(now())
  updatedAt       DateTime         @updatedAt
  startedAt       DateTime?
  stoppedAt       DateTime?

  metrics         InstanceMetric[]
  logs            InstanceLog[]
```

```
  @@index([userId])
  @@index([organizationId])
  @@index([status])
  @@index([proxyId])
}

enum InstanceStatus {
  PENDING
  STARTING
  RUNNING
  STOPPING
  STOPPED
  ERROR
}

enum Platform {
  YOUTUBE
  SPOTIFY
  DEEZER
  TIKTOK
  CUSTOM
}

model InstanceMetric {
  id          String   @id @default(cuid())
  instanceId  String
  instance    Instance @relation(fields: [instanceId], references: [id], onDelete: Cas
cade)

  cpuUsage    Float    // percentage
  memoryUsage Float    // MB
  networkIn   Float    // KB/s
  networkOut  Float    // KB/s

  timestamp   DateTime @default(now())

  @@index([instanceId, timestamp])
}

model InstanceLog {
  id          String   @id @default(cuid())
  instanceId  String
  instance    Instance @relation(fields: [instanceId], references: [id], onDelete: Cas
cade)

  level       LogLevel
  message     String   @db.Text
  metadata    Json?

  timestamp   DateTime @default(now())

  @@index([instanceId, timestamp])
  @@index([level])
}

enum LogLevel {
  DEBUG
  INFO
  WARN
  ERROR
}
```

```
// ---------- PROXIES ----------

model Proxy {
  id            String        @id @default(cuid())
  ip            String
  port          Int
  protocol      ProxyProtocol @default(HTTP)

  // Auth (optional)
  username      String?
  password      String?

  // Metadata
  country       String?
  region        String?
  city          String?
  isp           String?

  // Health
  status        ProxyStatus   @default(UNTESTED)
  lastValidated DateTime?
  lastUsed      DateTime?
  successRate   Float         @default(0)
  avgLatency    Float?        // ms

  // Ownership
  organizationId String?
  organization   Organization? @relation(fields: [organizationId], references: [id])
  source        ProxySource

  createdAt     DateTime      @default(now())
  updatedAt     DateTime      @updatedAt

  instances     Instance[]

  @@unique([ip, port])
  @@index([status])
  @@index([organizationId])
}

enum ProxyProtocol {
  HTTP
  HTTPS
  SOCKS4
  SOCKS5
}

enum ProxyStatus {
  UNTESTED
  HEALTHY
  DEGRADED
  DOWN
  BLACKLISTED
}

enum ProxySource {
  FREE
  MANUAL
  BRIGHTDATA
  SMARTPROXY
  OXYLABS
}
```

```
// ---------- SCRIPTS ----------

model Script {
  id            String   @id @default(cuid())
  name          String
  description   String?
  platform      Platform
  code          String   @db.Text
  version       Int      @default(1)

  // Marketplace
  isPublic      Boolean  @default(false)
  downloads     Int      @default(0)
  rating        Float?

  authorId      String?
  // Note: Removed User relation to avoid circular dependency issues

  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt

  instances     Instance[]

  @@index([platform])
  @@index([isPublic])
}

// ---------- JOBS & SCHEDULING ----------

model Job {
  id          String     @id @default(cuid())
  name        String
  type        JobType
  status      JobStatus  @default(PENDING)

  // Configuration
  config      Json       // platform, urls, scripts, etc.

  // Scheduling (cron)
  schedule    String?    // cron expression
  recurring   Boolean    @default(false)
  nextRun     DateTime?

  // Execution
  startedAt   DateTime?
  completedAt DateTime?
  errorMessage String?
  retries     Int        @default(0)
  maxRetries  Int        @default(3)

  // Ownership
  userId      String
  // Note: Removed User relation to simplify

  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt

  executions  JobExecution[]

  @@index([userId])
  @@index([status])
  @@index([nextRun])
}
```

```
enum JobType {
  ONE_TIME
  RECURRING
  BATCH
}

enum JobStatus {
  PENDING
  RUNNING
  COMPLETED
  FAILED
  CANCELLED
}

model JobExecution {
  id        String   @id @default(cuid())
  jobId     String
  job       Job      @relation(fields: [jobId], references: [id], onDelete: Cascade)

  status    JobStatus
  startedAt DateTime @default(now())
  endedAt   DateTime?
  duration  Int?     // seconds

  result    Json?
  error     String?

  @@index([jobId])
}
```

---

8. Especificações Técnicas do Frontend

8.1 Tech Stack

Categoria	Tecnologia	Versão	Propósito
Framework	Next.js	14+	SSR, routing, API routes
Language	TypeScript	5+	Type safety
Styling	TailwindCSS	3+	Utility-first CSS
UI Components	shadcn/ui	Latest	Pre-built components
State Management	Zustand	4+	Lightweight state
Forms	React Hook Form	7+	Form management
Validation	Zod	3+	Schema validation
Charts	Recharts	2+	Data visualization
Code Editor	Monaco Editor	0.45+	Script editing
Real-time	Socket.io Client	4+	WebSocket
HTTP Client	Axios	1+	API calls
Date Handling	date-fns	3+	Date utilities
Icons	Lucide React	Latest	Icon library
Animation	Framer Motion	11+	Animations

8.2 Componentes Principais

8.2.1 Dashboard Home

Caminho: `app/(dashboard)/page.tsx`

Funcionalidades:
- Overview de métricas (instâncias ativas, proxies, jobs)
- Gráfico de uso de recursos (CPU, RAM)
- Lista de instâncias recentes
- Alertas e notificações
- Quick actions (criar instância, validar proxies)

Componentes:

```
<DashboardLayout>
  <StatsGrid>
    <StatCard title="Active Instances" value={25} trend="+5%" />
    <StatCard title="Healthy Proxies" value={18} trend="-2" />
    <StatCard title="Running Jobs" value={3} />
    <StatCard title="Success Rate" value="94%" trend="+1.2%" />
  </StatsGrid>

  <ChartsGrid>
    <ResourceUsageChart data={metrics} />
    <InstanceStatusPieChart data={statusDistribution} />
  </ChartsGrid>

  <RecentActivityTable instances={recentInstances} />

  <AlertsPanel alerts={alerts} />
</DashboardLayout>
```

8.2.2 Instance Manager

Caminho: `app/(dashboard)/instances/page.tsx`

Funcionalidades:

- Tabela com todas as instâncias (paginada)
- Filtros (status, platform, user)
- Bulk actions (start, stop, delete)
- Real-time status updates
- Quick view (modal com detalhes)

Componentes:

```
<InstanceManager>
  <InstancesHeader>
    <SearchBar />
    <FilterDropdown />
    <CreateInstanceButton />
  </InstancesHeader>

  <InstancesTable
    columns={[
      { key: 'status', label: 'Status', render: StatusBadge },
      { key: 'name', label: 'Name' },
      { key: 'platform', label: 'Platform' },
      { key: 'proxy', label: 'Proxy' },
      { key: 'uptime', label: 'Uptime' },
      { key: 'actions', label: 'Actions', render: ActionButtons },
    ]}
    data={instances}
    onRowClick={handleRowClick}
  />

  <Pagination
    currentPage={page}
    totalPages={totalPages}
    onPageChange={setPage}
  />

  {selectedInstance && (
    <InstanceDetailModal
      instance={selectedInstance}
      onClose={() => setSelectedInstance(null)}
    />
  )}
</InstanceManager>
```

8.2.3 Instance Detail View

Caminho: `app/(dashboard)/instances/[id]/page.tsx`

Funcionalidades:
- Informações detalhadas da instância
- Live screenshot da página
- Logs em tempo real (streaming)
- Gráfico de métricas (CPU, RAM, network)
- Controles (start, stop, restart, navigate)
- Trocar proxy sob demanda

Componentes:

```
<InstanceDetailPage instanceId={id}>
  <InstanceHeader instance={instance}>
    <StatusBadge status={instance.status} />
    <ActionButtons>
      <StartButton />
      <StopButton />
      <RestartButton />
      <DeleteButton />
    </ActionButtons>
  </InstanceHeader>

  <InstanceGrid>
    <InstanceInfo instance={instance} />

    <LiveScreenshot
      instanceId={id}
      refreshInterval={5000}
    />

    <MetricsChart
      data={metrics}
      timeRange="1h"
    />

    <LogsPanel
      logs={logs}
      streaming={true}
      filters={['info', 'warn', 'error']}
    />

    <ProxySection
      currentProxy={instance.proxy}
      onChangeProxy={handleProxyChange}
    />

    <NavigationControl
      currentUrl={instance.currentUrl}
      onNavigate={handleNavigate}
    />
  </InstanceGrid>
</InstanceDetailPage>
```

8.2.4 Proxy Manager

Caminho: `app/(dashboard)/proxies/page.tsx`

Funcionalidades:
- Lista de proxies com status
- Validação em lote
- Adicionar proxies manualmente
- Importar de arquivo/API
- Mapa geográfico de proxies
- Blacklist management

Componentes:

```
<ProxyManager>
  <ProxyHeader>
    <AddProxyButton />
    <ImportProxiesButton />
    <ValidateAllButton />
    <FilterOptions />
  </ProxyHeader>

  <ProxyViews>
    <TabList>
      <Tab>Table View</Tab>
      <Tab>Map View</Tab>
    </TabList>

    <TabPanel>
      <ProxyTable
        columns={[
          { key: 'status', label: 'Status', render: StatusIndicator },
          { key: 'ip', label: 'IP' },
          { key: 'port', label: 'Port' },
          { key: 'country', label: 'Country', render: CountryFlag },
          { key: 'latency', label: 'Latency' },
          { key: 'successRate', label: 'Success Rate' },
          { key: 'lastUsed', label: 'Last Used' },
          { key: 'actions', label: 'Actions' },
        ]}
        data={proxies}
        onValidate={handleValidateProxy}
        onBlacklist={handleBlacklistProxy}
      />
    </TabPanel>

    <TabPanel>
      <ProxyMap
        proxies={proxies}
        onSelectProxy={handleSelectProxy}
      />
    </TabPanel>
  </ProxyViews>
</ProxyManager>
```

8.2.5 Script Editor

Caminho: `app/(dashboard)/scripts/editor/page.tsx`

Funcionalidades:
- Monaco Editor com syntax highlighting
- Auto-complete para APIs comuns
- Test runner (executa em instância isolada)
- Versionamento
- Salvar como template
- Compartilhar (marketplace)

Componentes:

```
<ScriptEditor>
  <EditorHeader>
    <SaveButton />
    <TestButton />
    <ShareButton />
    <VersionHistory />
  </EditorHeader>

  <EditorLayout>
    <Sidebar>
      <ScriptLibrary
        scripts={scripts}
        onSelect={handleSelectScript}
      />

      <SnippetsLibrary
        snippets={snippets}
        onInsert={handleInsertSnippet}
      />
    </Sidebar>

    <EditorMain>
      <MonacoEditor
        language="javascript"
        value={code}
        onChange={setCode}
        options={{
          minimap: { enabled: true },
          fontSize: 14,
          lineNumbers: 'on',
          autoIndent: 'full',
        }}
      />

      <ConsolePanel>
        <ConsoleLogs logs={testLogs} />
        <ErrorDisplay errors={testErrors} />
      </ConsolePanel>
    </EditorMain>

    <ConfigPanel>
      <ScriptMetadata
        name={scriptName}
        platform={scriptPlatform}
        description={scriptDescription}
      />

      <TestConfiguration
        instanceId={testInstanceId}
        testUrl={testUrl}
      />
    </ConfigPanel>
  </EditorLayout>
</ScriptEditor>
```

8.2.6 Analytics Dashboard

Caminho: `app/(dashboard)/analytics/page.tsx`

Funcionalidades:

- Métricas agregadas (success rate, avg duration, etc.)

- Gráficos de tendências (line, bar, pie)
- Comparação entre períodos
- Exportar relatórios (CSV, PDF)
- Filtros por date range, platform, user

Componentes:

```
<AnalyticsDashboard>
  <AnalyticsHeader>
    <DateRangePicker
      start={startDate}
      end={endDate}
      onChange={handleDateChange}
    />
    <FilterBar />
    <ExportButton />
  </AnalyticsHeader>

  <MetricsOverview>
    <MetricCard
      title="Total Executions"
      value={data.totalExecutions}
      change={data.executionsChange}
    />
    <MetricCard
      title="Success Rate"
      value={`${data.successRate}%`}
      change={data.successRateChange}
    />
    <MetricCard
      title="Avg Duration"
      value={`${data.avgDuration}s`}
      change={data.avgDurationChange}
    />
    <MetricCard
      title="Errors"
      value={data.totalErrors}
      change={data.errorsChange}
    />
  </MetricsOverview>

  <ChartsGrid>
    <TrendChart
      title="Executions Over Time"
      data={data.executionsTrend}
      type="line"
    />

    <PlatformDistribution
      title="By Platform"
      data={data.platformDistribution}
      type="pie"
    />

    <SuccessVsFailure
      title="Success vs Failure"
      data={data.successFailureTrend}
      type="area"
    />

    <ProxyPerformance
      title="Proxy Performance"
      data={data.proxyPerformance}
      type="bar"
    />
  </ChartsGrid>

  <ReportTable
    data={data.detailedReport}
```

```
    exportable={true}
  />
</AnalyticsDashboard>
```

8.3 Real-time Updates (Socket.io)

Hook customizado:

```typescript
// hooks/use-realtime.ts
import { useEffect, useState } from 'react';
import { io, Socket } from 'socket.io-client';

interface UseRealtimeOptions {
  events: string[];
  onConnect?: () => void;
  onDisconnect?: () => void;
}

export function useRealtime(options: UseRealtimeOptions) {
  const [socket, setSocket] = useState<Socket | null>(null);
  const [data, setData] = useState<Record<string, any>>({});

  useEffect(() => {
    // Connect to Socket.io server
    const socketInstance = io(process.env.NEXT_PUBLIC_SOCKET_URL!, {
      auth: {
        token: localStorage.getItem('auth_token'),
      },
    });

    socketInstance.on('connect', () => {
      console.log('Connected to real-time server');
      options.onConnect?.();
    });

    socketInstance.on('disconnect', () => {
      console.log('Disconnected from real-time server');
      options.onDisconnect?.();
    });

    // Subscribe to events
    options.events.forEach(event => {
      socketInstance.on(event, (payload) => {
        setData(prev => ({
          ...prev,
          [event]: payload,
        }));
      });
    });

    setSocket(socketInstance);

    return () => {
      socketInstance.disconnect();
    };
  }, []);

  return { socket, data };
}
```

Uso:

```
// In a component
const { data } = useRealtime({
  events: ['instance.metrics', 'instance.status', 'instance.log'],
  onConnect: () => toast.success('Connected to live updates'),
});

// data.instance.metrics -> latest metrics
// data.instance.status -> status changes
// data.instance.log -> new log entries
```

8.4 Estado Global (Zustand)

```
// lib/store/instances.store.ts
import create from 'zustand';

interface InstancesStore {
  instances: Instance[];
  selectedInstance: Instance | null;
  loading: boolean;
  error: string | null;

  fetchInstances: () => Promise<void>;
  createInstance: (data: CreateInstanceInput) => Promise<void>;
  updateInstance: (id: string, data: Partial<Instance>) => Promise<void>;
  deleteInstance: (id: string) => Promise<void>;
  selectInstance: (instance: Instance | null) => void;
}

export const useInstancesStore = create<InstancesStore>((set, get) => ({
  instances: [],
  selectedInstance: null,
  loading: false,
  error: null,

  fetchInstances: async () => {
    set({ loading: true, error: null });
    try {
      const response = await axios.get('/api/instances');
      set({ instances: response.data, loading: false });
    } catch (error) {
      set({ error: error.message, loading: false });
    }
  },

  createInstance: async (data) => {
    set({ loading: true });
    try {
      const response = await axios.post('/api/instances', data);
      set(state => ({
        instances: [...state.instances, response.data],
        loading: false,
      }));
    } catch (error) {
      set({ error: error.message, loading: false });
      throw error;
    }
  },

  // ... other actions
}));
```

9. Especificações Técnicas do Backend

9.1 Tech Stack

Categoria	Tecnologia	Versão	Propósito
Runtime	Node.js	20+	JavaScript runtime
Framework	Next.js API Routes	14+	API endpoints
Browser Automation	Playwright	1.40+	Substitui Selenium
Database	PostgreSQL	15+	Primary database
ORM	Prisma	5+	Type-safe queries
Cache/Queue	Redis	7+	Cache + BullMQ
Job Queue	BullMQ	5+	Background jobs
Real-time	Socket.io	4+	WebSocket server
Authentication	NextAuth.js	5+	Auth provider
Validation	Zod	3+	Schema validation
HTTP Client	Axios	1+	External APIs
Logging	Winston	3+	Structured logs
Monitoring	Prometheus Client	15+	Metrics export

9.2 Services

9.2.1 BrowserService

Arquivo: `lib/services/browser.service.ts`

Responsabilidades:
- Gerenciar lifecycle de navegadores Playwright
- Aplicar configurações anti-detecção
- Executar scripts JavaScript
- Capturar screenshots
- Coletar métricas

Interface:

```typescript
interface BrowserServiceConfig {
  userAgent: string;
  proxy?: Proxy;
  headless: boolean;
  viewport: { width: number; height: number };
  locale?: string;
  timezone?: string;
}

class BrowserService {
  private browsers: Map<string, Browser> = new Map();
  private contexts: Map<string, BrowserContext> = new Map();

  async launchBrowser(
    instanceId: string,
    config: BrowserServiceConfig
  ): Promise<Browser>;

  async createContext(
    instanceId: string,
    config: BrowserServiceConfig
  ): Promise<BrowserContext>;

  async navigate(
    instanceId: string,
    url: string
  ): Promise<void>;

  async executeScript(
    instanceId: string,
    script: string
  ): Promise<any>;

  async takeScreenshot(
    instanceId: string
  ): Promise<Buffer>;

  async collectMetrics(
    instanceId: string
  ): Promise<BrowserMetrics>;

  async closeBrowser(
    instanceId: string
  ): Promise<void>;
}
```

Implementação (parcial):

```typescript
import { chromium, Browser, BrowserContext, Page } from 'playwright';
import { applyAntiDetect } from '@/lib/utils/anti-detect';

export class BrowserService {
  private browsers: Map<string, Browser> = new Map();
  private contexts: Map<string, BrowserContext> = new Map();
  private pages: Map<string, Page> = new Map();

  async launchBrowser(
    instanceId: string,
    config: BrowserServiceConfig
  ): Promise<Browser> {
    const launchOptions = {
      headless: config.headless,
      proxy: config.proxy ? {
        server: `http://${config.proxy.ip}:${config.proxy.port}`,
        username: config.proxy.username,
        password: config.proxy.password,
      } : undefined,
      args: [
        '--no-sandbox',
        '--disable-setuid-sandbox',
        '--disable-dev-shm-usage',
        '--disable-blink-features=AutomationControlled',
        `--window-size=${config.viewport.width},${config.viewport.height}`,
      ],
    };

    const browser = await chromium.launch(launchOptions);
    this.browsers.set(instanceId, browser);

    return browser;
  }

  async createContext(
    instanceId: string,
    config: BrowserServiceConfig
  ): Promise<BrowserContext> {
    const browser = this.browsers.get(instanceId);
    if (!browser) throw new Error('Browser not found');

    const context = await browser.newContext({
      userAgent: config.userAgent,
      viewport: config.viewport,
      locale: config.locale || 'en-US',
      timezoneId: config.timezone || 'America/New_York',
      permissions: [],
      geolocation: undefined,
    });

    // Apply anti-detection scripts
    await applyAntiDetect(context);

    this.contexts.set(instanceId, context);

    const page = await context.newPage();
    this.pages.set(instanceId, page);

    return context;
  }

  async navigate(instanceId: string, url: string): Promise<void> {
```

```
    const page = this.pages.get(instanceId);
    if (!page) throw new Error('Page not found');

    await page.goto(url, {
      waitUntil: 'networkidle',
      timeout: 60000,
    });
  }

  async executeScript(instanceId: string, script: string): Promise<any> {
    const page = this.pages.get(instanceId);
    if (!page) throw new Error('Page not found');

    return await page.evaluate(script);
  }

  async takeScreenshot(instanceId: string): Promise<Buffer> {
    const page = this.pages.get(instanceId);
    if (!page) throw new Error('Page not found');

    return await page.screenshot({
      type: 'jpeg',
      quality: 80,
      fullPage: false,
    });
  }

  async collectMetrics(instanceId: string): Promise<BrowserMetrics> {
    const page = this.pages.get(instanceId);
    if (!page) throw new Error('Page not found');

    const metrics = await page.evaluate(() => ({
      memory: (performance as any).memory?.usedJSHeapSize || 0,
      timing: performance.timing,
      url: window.location.href,
    }));

    // Get CPU/RAM from system (via OS process)
    const process = await this.getProcessMetrics(instanceId);

    return {
      url: metrics.url,
      memoryMB: metrics.memory / (1024 * 1024),
      cpuPercent: process.cpu,
      ramMB: process.ram,
      loadTime: metrics.timing.loadEventEnd - metrics.timing.navigationStart,
    };
  }

  private async getProcessMetrics(instanceId: string): Promise<{ cpu: number; ram: num
ber }> {
    // Use pidusage or similar library to get process stats
    // ...
    return { cpu: 0, ram: 0 };
  }

  async closeBrowser(instanceId: string): Promise<void> {
    const browser = this.browsers.get(instanceId);
    if (browser) {
      await browser.close();
      this.browsers.delete(instanceId);
      this.contexts.delete(instanceId);
      this.pages.delete(instanceId);
```

```
    }
  }
}

export const browserService = new BrowserService();
```

9.2.2 ProxyService

Arquivo: `lib/services/proxy.service.ts`

Responsabilidades:
- Buscar proxies de múltiplas fontes
- Validar proxies
- Cachear proxies válidos
- Rotacionar proxies
- Health checks periódicos
- Geolocalização

Interface:

```typescript
interface ProxyFetchOptions {
  sources: ProxySource[];
  minCount: number;
  protocol: 'http' | 'https' | 'socks4' | 'socks5';
}

interface ProxyValidationResult {
  proxy: Proxy;
  valid: boolean;
  latency?: number;
  error?: string;
}

class ProxyService {
  async fetchProxies(options: ProxyFetchOptions): Promise<Proxy[]>;

  async validateProxy(proxy: Proxy): Promise<ProxyValidationResult>;

  async validateProxiesBatch(proxies: Proxy[]): Promise<ProxyValidationResult[]>;

  async getGeolocation(proxy: Proxy): Promise<GeoLocation>;

  async getCachedProxies(filter?: ProxyFilter): Promise<Proxy[]>;

  async assignProxyToInstance(instanceId: string): Promise<Proxy>;

  async rotateProxy(instanceId: string): Promise<Proxy>;

  async startHealthChecks(): void;
}
```

Implementação (parcial):

```
import axios from 'axios';
import { prisma } from '@/lib/db/prisma';
import { redis } from '@/lib/db/redis';

export class ProxyService {
  private readonly PROXY_SOURCES = [
    'https://api.proxyscrape.com/v2/?request=get&protocol=http...',
    'https://www.proxy-list.download/api/v1/get?type=http',
    // ... outras fontes
  ];

  async fetchProxies(options: ProxyFetchOptions): Promise<Proxy[]> {
    const allProxies: Proxy[] = [];

    // Fetch from all sources in parallel
    const promises = this.PROXY_SOURCES.map(async (source) => {
      try {
        const response = await axios.get(source, { timeout: 10000 });
        const lines = response.data.split('\n');

        return lines
          .filter((line: string) => line.includes(':'))
          .map((line: string) => {
            const [ip, port] = line.trim().split(':');
            return {
              ip,
              port: parseInt(port),
              protocol: options.protocol,
              source: 'FREE' as ProxySource,
            };
          });
      } catch (error) {
        console.error(`Failed to fetch from ${source}:`, error);
        return [];
      }
    });

    const results = await Promise.all(promises);
    results.forEach(proxies => allProxies.push(...proxies));

    // Deduplicate
    const uniqueProxies = Array.from(
      new Map(allProxies.map(p => [`${p.ip}:${p.port}`, p])).values()
    );

    return uniqueProxies;
  }

  async validateProxy(proxy: Proxy): Promise<ProxyValidationResult> {
    const startTime = Date.now();

    try {
      const response = await axios.get('http://httpbin.org/ip', {
        proxy: {
          host: proxy.ip,
          port: proxy.port,
          auth: proxy.username ? {
            username: proxy.username,
            password: proxy.password!,
          } : undefined,
        },
        timeout: 15000,
```

```
      });

      const latency = Date.now() - startTime;

      return {
        proxy,
        valid: response.status === 200,
        latency,
      };
    } catch (error) {
      return {
        proxy,
        valid: false,
        error: error.message,
      };
    }
  }

  async validateProxiesBatch(proxies: Proxy[]): Promise<ProxyValidationResult[]> {
    // Validate in batches of 50
    const batchSize = 50;
    const results: ProxyValidationResult[] = [];

    for (let i = 0; i < proxies.length; i += batchSize) {
      const batch = proxies.slice(i, i + batchSize);
      const batchResults = await Promise.all(
        batch.map(proxy => this.validateProxy(proxy))
      );
      results.push(...batchResults);
    }

    return results;
  }

  async getGeolocation(proxy: Proxy): Promise<GeoLocation> {
    // Check cache first
    const cached = await redis.get(`geo:${proxy.ip}`);
    if (cached) return JSON.parse(cached);

    // Fetch from IP geolocation API
    try {
      const response = await axios.get(
        `http://ip-api.com/json/${proxy.ip}`
      );

      const geo: GeoLocation = {
        country: response.data.country,
        region: response.data.regionName,
        city: response.data.city,
        isp: response.data.isp,
        lat: response.data.lat,
        lon: response.data.lon,
      };

      // Cache for 24h
      await redis.setex(`geo:${proxy.ip}`, 86400, JSON.stringify(geo));

      return geo;
    } catch (error) {
      console.error(`Failed to get geolocation for ${proxy.ip}:`, error);
      return null;
    }
  }
```

```
  async getCachedProxies(filter?: ProxyFilter): Promise<Proxy[]> {
    return await prisma.proxy.findMany({
      where: {
        status: filter?.status || 'HEALTHY',
        country: filter?.country,
        protocol: filter?.protocol,
      },
      orderBy: {
        successRate: 'desc',
      },
      take: filter?.limit || 100,
    });
  }

  async assignProxyToInstance(instanceId: string): Promise<Proxy> {
    // Get available proxies
    const proxies = await this.getCachedProxies({
      status: 'HEALTHY',
      limit: 10,
    });

    if (proxies.length === 0) {
      throw new Error('No healthy proxies available');
    }

    // Select least recently used
    const proxy = proxies.sort((a, b) => {
      const aTime = a.lastUsed?.getTime() || 0;
      const bTime = b.lastUsed?.getTime() || 0;
      return aTime - bTime;
    })[0];

    // Update proxy
    await prisma.proxy.update({
      where: { id: proxy.id },
      data: { lastUsed: new Date() },
    });

    return proxy;
  }

  async rotateProxy(instanceId: string): Promise<Proxy> {
    // Get current instance
    const instance = await prisma.instance.findUnique({
      where: { id: instanceId },
      include: { proxy: true },
    });

    if (!instance) throw new Error('Instance not found');

    // Get new proxy (different from current)
    const proxies = await this.getCachedProxies({
      status: 'HEALTHY',
      limit: 20,
    });

    const newProxy = proxies.find(p => p.id !== instance.proxyId);

    if (!newProxy) throw new Error('No alternative proxy available');

    // Update instance
    await prisma.instance.update({
```

```
      where: { id: instanceId },
      data: { proxyId: newProxy.id },
    });

    return newProxy;
  }

  async startHealthChecks(): void {
    // Run health checks every 5 minutes
    setInterval(async () => {
      const proxies = await prisma.proxy.findMany({
        where: {
          status: { in: ['HEALTHY', 'DEGRADED', 'UNTESTED'] },
        },
      });

      console.log(`Running health checks on ${proxies.length} proxies...`);

      const results = await this.validateProxiesBatch(proxies);

      // Update database
      for (const result of results) {
        let status: ProxyStatus;

        if (result.valid) {
          status = result.latency! < 5000 ? 'HEALTHY' : 'DEGRADED';
        } else {
          // Check consecutive failures
          const proxy = await prisma.proxy.findUnique({
            where: { id: result.proxy.id },
          });

          if (proxy) {
            const failures = (proxy as any).consecutiveFailures || 0;
            status = failures >= 3 ? 'DOWN' : 'DEGRADED';
          } else {
            status = 'DOWN';
          }
        }

        await prisma.proxy.update({
          where: { id: result.proxy.id },
          data: {
            status,
            lastValidated: new Date(),
            avgLatency: result.latency,
          },
        });
      }

      console.log('Health checks completed');
    }, 5 * 60 * 1000); // 5 minutes
  }
}

export const proxyService = new ProxyService();
```

9.2.3 JobService (BullMQ)

Arquivo: `lib/services/job.service.ts`

Responsabilidades:
- Criar e gerenciar jobs
- Fila de execução
- Retry logic
- Scheduling (cron)
- Progress tracking

Implementação:

```
import { Queue, Worker, Job as BullJob } from 'bullmq';
import { redis } from '@/lib/db/redis';
import { prisma } from '@/lib/db/prisma';
import { browserService } from './browser.service';
import { scriptService } from './script.service';

const connection = {
  host: process.env.REDIS_HOST,
  port: parseInt(process.env.REDIS_PORT!),
};

// Create queues
export const instanceQueue = new Queue('instances', { connection });

// Create worker
const instanceWorker = new Worker('instances', async (job: BullJob) => {
  const { instanceId, action } = job.data;

  switch (action) {
    case 'start':
      return await startInstanceJob(instanceId, job);
    case 'stop':
      return await stopInstanceJob(instanceId);
    case 'execute_script':
      return await executeScriptJob(instanceId, job.data.scriptId);
    default:
      throw new Error(`Unknown action: ${action}`);
  }
}, { connection });

async function startInstanceJob(instanceId: string, job: BullJob) {
  // Update status
  await prisma.instance.update({
    where: { id: instanceId },
    data: { status: 'STARTING' },
  });

  await job.updateProgress(10);

  // Get instance config
  const instance = await prisma.instance.findUnique({
    where: { id: instanceId },
    include: { proxy: true, script: true },
  });

  if (!instance) throw new Error('Instance not found');

  await job.updateProgress(20);

  // Launch browser
  await browserService.launchBrowser(instanceId, {
    userAgent: instance.userAgent,
    proxy: instance.proxy || undefined,
    headless: true,
    viewport: { width: 1280, height: 720 },
  });

  await job.updateProgress(50);

  // Create context
  await browserService.createContext(instanceId, {
    userAgent: instance.userAgent,
```

```
    proxy: instance.proxy || undefined,
    headless: true,
    viewport: { width: 1280, height: 720 },
  });

  await job.updateProgress(70);

  // Navigate to start URL
  await browserService.navigate(instanceId, instance.startUrl);

  await job.updateProgress(90);

  // Start automation thread
  if (instance.script) {
    await startAutomationThread(instanceId, instance.script.code);
  }

  // Update status
  await prisma.instance.update({
    where: { id: instanceId },
    data: {
      status: 'RUNNING',
      startedAt: new Date(),
    },
  });

  await job.updateProgress(100);

  return { success: true, instanceId };
}

async function stopInstanceJob(instanceId: string) {
  await prisma.instance.update({
    where: { id: instanceId },
    data: { status: 'STOPPING' },
  });

  await browserService.closeBrowser(instanceId);

  await prisma.instance.update({
    where: { id: instanceId },
    data: {
      status: 'STOPPED',
      stoppedAt: new Date(),
    },
  });

  return { success: true, instanceId };
}

async function executeScriptJob(instanceId: string, scriptId: string) {
  const script = await prisma.script.findUnique({
    where: { id: scriptId },
  });

  if (!script) throw new Error('Script not found');

  const result = await browserService.executeScript(instanceId, script.code);

  return { success: true, result };
}

async function startAutomationThread(instanceId: string, scriptCode: string) {
```

```
  // Execute script periodically
  const interval = setInterval(async () => {
    try {
      const instance = await prisma.instance.findUnique({
        where: { id: instanceId },
      });

      if (!instance || instance.status !== 'RUNNING') {
        clearInterval(interval);
        return;
      }

      await browserService.executeScript(instanceId, scriptCode);

      // Random delay 15-30s
      const delay = 15000 + Math.random() * 15000;
      await new Promise(resolve => setTimeout(resolve, delay));
    } catch (error) {
      console.error(`Automation error for instance ${instanceId}:`, error);
    }
  }, 15000);

  // Store interval ID for cleanup
  (global as any).automationIntervals = (global as any).automationIntervals || {};
  (global as any).automationIntervals[instanceId] = interval;
}

// Worker event handlers
instanceWorker.on('completed', async (job) => {
  console.log(`Job ${job.id} completed:`, job.returnvalue);

  // Store execution result
  await prisma.jobExecution.create({
    data: {
      jobId: job.data.jobId,
      status: 'COMPLETED',
      endedAt: new Date(),
      result: job.returnvalue,
    },
  });
});

instanceWorker.on('failed', async (job, error) => {
  console.error(`Job ${job?.id} failed:`, error);

  if (job) {
    await prisma.jobExecution.create({
      data: {
        jobId: job.data.jobId,
        status: 'FAILED',
        endedAt: new Date(),
        error: error.message,
      },
    });
  }
});

export class JobService {
  async createInstanceJob(instanceId: string, action: string) {
    const job = await instanceQueue.add('instance-action', {
      instanceId,
      action,
    }, {
```

```
      attempts: 3,
      backoff: {
        type: 'exponential',
        delay: 5000,
      },
    });

    return job.id;
  }

  async getJobStatus(jobId: string) {
    const job = await instanceQueue.getJob(jobId);

    if (!job) return null;

    return {
      id: job.id,
      progress: await job.progress(),
      state: await job.getState(),
      failedReason: job.failedReason,
      returnvalue: job.returnvalue,
    };
  }
}

export const jobService = new JobService();
```

9.3 API Routes

9.3.1 POST /api/instances (Create Instance)

```typescript
// app/api/instances/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { z } from 'zod';
import { prisma } from '@/lib/db/prisma';
import { jobService } from '@/lib/services/job.service';
import { proxyService } from '@/lib/services/proxy.service';
import { authOptions } from '@/lib/auth/next-auth.config';

const createInstanceSchema = z.object({
  name: z.string().min(1).max(100),
  platform: z.enum(['YOUTUBE', 'SPOTIFY', 'DEEZER', 'TIKTOK', 'CUSTOM']),
  startUrl: z.string().url(),
  userAgent: z.string().optional(),
  scriptId: z.string().optional(),
  assignProxy: z.boolean().default(true),
});

export async function POST(req: NextRequest) {
  try {
    // Authenticate
    const session = await getServerSession(authOptions);
    if (!session?.user) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
    }

    // Parse and validate request
    const body = await req.json();
    const data = createInstanceSchema.parse(body);

    // Assign proxy if requested
    let proxyId: string | undefined;
    if (data.assignProxy) {
      try {
        const proxy = await proxyService.assignProxyToInstance('temp');
        proxyId = proxy.id;
      } catch (error) {
        console.warn('Failed to assign proxy:', error);
      }
    }

    // Generate user agent if not provided
    const userAgent = data.userAgent || generateUserAgent();

    // Create instance in database
    const instance = await prisma.instance.create({
      data: {
        name: data.name,
        platform: data.platform,
        startUrl: data.startUrl,
        userAgent,
        status: 'PENDING',
        proxyId,
        scriptId: data.scriptId,
        userId: session.user.id,
      },
      include: {
        proxy: true,
        script: true,
      },
    });
```

```
    // Add to job queue
    await jobService.createInstanceJob(instance.id, 'start');

    return NextResponse.json(instance, { status: 201 });
  } catch (error) {
    if (error instanceof z.ZodError) {
      return NextResponse.json({ error: error.errors }, { status: 400 });
    }

    console.error('Failed to create instance:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}

export async function GET(req: NextRequest) {
  try {
    const session = await getServerSession(authOptions);
    if (!session?.user) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
    }

    // Parse query params
    const { searchParams } = new URL(req.url);
    const page = parseInt(searchParams.get('page') || '1');
    const limit = parseInt(searchParams.get('limit') || '20');
    const status = searchParams.get('status');
    const platform = searchParams.get('platform');

    // Build where clause
    const where: any = { userId: session.user.id };
    if (status) where.status = status;
    if (platform) where.platform = platform;

    // Query database
    const [instances, total] = await Promise.all([
      prisma.instance.findMany({
        where,
        include: {
          proxy: true,
          script: true,
        },
        orderBy: { createdAt: 'desc' },
        skip: (page - 1) * limit,
        take: limit,
      }),
      prisma.instance.count({ where }),
    ]);

    return NextResponse.json({
      instances,
      pagination: {
        page,
        limit,
        total,
        totalPages: Math.ceil(total / limit),
      },
    });
  } catch (error) {
    console.error('Failed to fetch instances:', error);
    return NextResponse.json(
```

```
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}

function generateUserAgent(): string {
  const userAgents = [
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/120.0.0.0 Safari/537.36',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/120.0.0.0 Safari/537.36',
    'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
120.0.0.0 Safari/537.36',
  ];

  return userAgents[Math.floor(Math.random() * userAgents.length)];
}
```

---

10. Bibliotecas e Ferramentas Necessárias

10.1 Frontend (package.json)

```
{
  "name": "platform-web",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "test": "jest",
    "test:e2e": "playwright test"
  },
  "dependencies": {
    // Core
    "next": "^14.0.4",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "typescript": "^5.3.3",

    // Auth
    "next-auth": "^5.0.0-beta.4",
    "@auth/prisma-adapter": "^1.0.12",

    // Database & ORM
    "@prisma/client": "^5.7.1",

    // State Management
    "zustand": "^4.4.7",

    // Forms & Validation
    "react-hook-form": "^7.49.2",
    "zod": "^3.22.4",
    "@hookform/resolvers": "^3.3.4",

    // UI Components
    "@radix-ui/react-avatar": "^1.0.4",
    "@radix-ui/react-dialog": "^1.0.5",
    "@radix-ui/react-dropdown-menu": "^2.0.6",
    "@radix-ui/react-label": "^2.0.2",
    "@radix-ui/react-popover": "^1.0.7",
    "@radix-ui/react-select": "^2.0.0",
    "@radix-ui/react-tabs": "^1.0.4",
    "@radix-ui/react-toast": "^1.1.5",
    "class-variance-authority": "^0.7.0",
    "clsx": "^2.0.0",
    "tailwind-merge": "^2.2.0",
    "lucide-react": "^0.303.0",

    // Styling
    "tailwindcss": "^3.4.0",
    "autoprefixer": "^10.4.16",
    "postcss": "^8.4.32",

    // Charts & Visualization
    "recharts": "^2.10.3",
    "react-chartjs-2": "^5.2.0",
    "chart.js": "^4.4.1",

    // Code Editor
    "@monaco-editor/react": "^4.6.0",

    // Real-time
```

```
    "socket.io-client": "^4.5.4",

    // HTTP Client
    "axios": "^1.6.5",

    // Utilities
    "date-fns": "^3.0.6",
    "react-day-picker": "^8.10.0",
    "react-dropzone": "^14.2.3",
    "framer-motion": "^11.0.3",
    "sonner": "^1.3.1",

    // Maps (for proxy geolocation)
    "react-map-gl": "^7.1.7",
    "mapbox-gl": "^3.1.0"
  },
  "devDependencies": {
    // TypeScript
    "@types/node": "^20.10.6",
    "@types/react": "^18.2.46",
    "@types/react-dom": "^18.2.18",

    // Testing
    "@testing-library/react": "^14.1.2",
    "@testing-library/jest-dom": "^6.1.5",
    "@playwright/test": "^1.40.1",
    "jest": "^29.7.0",
    "jest-environment-jsdom": "^29.7.0",

    // Linting & Formatting
    "eslint": "^8.56.0",
    "eslint-config-next": "^14.0.4",
    "prettier": "^3.1.1",
    "prettier-plugin-tailwindcss": "^0.5.10",

    // Types
    "@types/mapbox-gl": "^3.1.0"
  }
}
```

10.2 Backend (adicional)

```
{
  "dependencies": {
    // Browser Automation
    "playwright": "^1.40.1",
    "playwright-extra": "^4.3.6",
    "puppeteer-extra-plugin-stealth": "^2.11.2",

    // Database
    "prisma": "^5.7.1",
    "@prisma/client": "^5.7.1",

    // Cache & Queue
    "redis": "^4.6.12",
    "ioredis": "^5.3.2",
    "bullmq": "^5.1.7",

    // Real-time
    "socket.io": "^4.5.4",

    // Auth
    "bcrypt": "^5.1.1",
    "jsonwebtoken": "^9.0.2",

    // Utilities
    "axios": "^1.6.5",
    "zod": "^3.22.4",
    "winston": "^3.11.0",
    "prom-client": "^15.1.0",
    "cron": "^3.1.6",

    // Email
    "nodemailer": "^6.9.7",
    "@sendgrid/mail": "^8.1.0",

    // File Storage
    "@aws-sdk/client-s3": "^3.485.0",

    // Anti-detect utilities
    "user-agents": "^1.1.231",
    "fake-useragent": "^1.0.1"
  },
  "devDependencies": {
    "@types/bcrypt": "^5.0.2",
    "@types/node": "^20.10.6",
    "@types/nodemailer": "^6.4.14",
    "ts-node": "^10.9.2"
  }
}
```

10.3 Infraestrutura (Docker)

Dockerfile:

```dockerfile
FROM node:20-alpine AS base

# Install dependencies
RUN apk add --no-cache libc6-compat

WORKDIR /app

# Copy package files
COPY package*.json ./
COPY prisma ./prisma/

# Install dependencies
RUN npm ci

# Copy source
COPY . .

# Generate Prisma Client
RUN npx prisma generate

# Build application
RUN npm run build

# Production image
FROM node:20-alpine AS runner

WORKDIR /app

# Install Playwright browsers
RUN npx playwright install --with-deps chromium

# Copy built application
COPY --from=base /app/.next ./.next
COPY --from=base /app/node_modules ./node_modules
COPY --from=base /app/package*.json ./
COPY --from=base /app/prisma ./prisma

EXPOSE 3000

ENV PORT 3000
ENV NODE_ENV production

CMD ["npm", "start"]
```

docker-compose.yml:

```yaml
version: '3.8'

services:
  # Next.js Application
  web:
    build: .
    ports:
      - "3000:3000"
    environment:
      - DATABASE_URL=postgresql://user:password@postgres:5432/platform
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - NEXTAUTH_SECRET=${NEXTAUTH_SECRET}
      - NEXTAUTH_URL=http://localhost:3000
    depends_on:
      - postgres
      - redis
    volumes:
      - ./uploads:/app/uploads

  # PostgreSQL Database
  postgres:
    image: postgres:15-alpine
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=platform
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  # Redis (Cache & Queue)
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  # Socket.io Server (separate if needed)
  socket-server:
    build: ./socket-server
    ports:
      - "3001:3001"
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379
    depends_on:
      - redis

volumes:
  postgres_data:
  redis_data:
```

11. Estrutura de Dados e APIs

11.1 Principais Endpoints REST

Instances

- `GET /api/instances` - List instances (paginated, filterable)
- `POST /api/instances` - Create instance
- `GET /api/instances/:id` - Get instance details
- `PUT /api/instances/:id` - Update instance
- `DELETE /api/instances/:id` - Delete instance
- `POST /api/instances/:id/start` - Start instance
- `POST /api/instances/:id/stop` - Stop instance
- `POST /api/instances/:id/restart` - Restart instance
- `POST /api/instances/:id/navigate` - Navigate to URL
- `GET /api/instances/:id/screenshot` - Get screenshot
- `GET /api/instances/:id/logs` - Get logs (streaming)
- `GET /api/instances/:id/metrics` - Get metrics
- `POST /api/instances/batch/start` - Start multiple instances
- `POST /api/instances/batch/stop` - Stop multiple instances

Proxies

- `GET /api/proxies` - List proxies
- `POST /api/proxies` - Add proxy manually
- `POST /api/proxies/import` - Import proxies from file/URL
- `POST /api/proxies/fetch` - Fetch free proxies
- `POST /api/proxies/validate` - Validate proxies (batch)
- `GET /api/proxies/:id` - Get proxy details
- `PUT /api/proxies/:id` - Update proxy
- `DELETE /api/proxies/:id` - Delete proxy
- `POST /api/proxies/:id/blacklist` - Blacklist proxy

Scripts

- `GET /api/scripts` - List scripts
- `POST /api/scripts` - Create script
- `GET /api/scripts/:id` - Get script
- `PUT /api/scripts/:id` - Update script
- `DELETE /api/scripts/:id` - Delete script
- `POST /api/scripts/:id/test` - Test script
- `POST /api/scripts/:id/publish` - Publish to marketplace

Jobs

- `GET /api/jobs` - List jobs
- `POST /api/jobs` - Create job
- `GET /api/jobs/:id` - Get job details
- `PUT /api/jobs/:id` - Update job
- `DELETE /api/jobs/:id` - Delete job
- `POST /api/jobs/:id/run` - Run job manually

- `GET /api/jobs/:id/executions` - Get execution history

Analytics

- `GET /api/analytics/overview` - Dashboard metrics
- `GET /api/analytics/instances` - Instance analytics
- `GET /api/analytics/proxies` - Proxy performance
- `GET /api/analytics/jobs` - Job success rates
- `POST /api/analytics/export` - Export report

Webhooks

- `GET /api/webhooks` - List webhooks
- `POST /api/webhooks` - Create webhook
- `DELETE /api/webhooks/:id` - Delete webhook
- `GET /api/webhooks/:id/deliveries` - Delivery history

11.2 WebSocket Events (Socket.io)

Server → Client

```
// Instance events
socket.emit('instance.created', { instanceId, instance });
socket.emit('instance.started', { instanceId });
socket.emit('instance.stopped', { instanceId });
socket.emit('instance.error', { instanceId, error });
socket.emit('instance.metrics', { instanceId, metrics });
socket.emit('instance.log', { instanceId, log });
socket.emit('instance.url_changed', { instanceId, url });

// Proxy events
socket.emit('proxy.validated', { proxyId, result });
socket.emit('proxy.health_check', { proxyId, status });

// Job events
socket.emit('job.started', { jobId });
socket.emit('job.progress', { jobId, progress });
socket.emit('job.completed', { jobId, result });
socket.emit('job.failed', { jobId, error });

// System events
socket.emit('system.alert', { type, message });
```

Client → Server

```
// Subscribe to instance updates
socket.emit('subscribe:instance', { instanceId });
socket.emit('unsubscribe:instance', { instanceId });

// Subscribe to all user instances
socket.emit('subscribe:user_instances', { userId });

// Request metrics
socket.emit('request:instance_metrics', { instanceId });
```

---

12. Segurança e Compliance

12.1 Autenticação e Autorização

1. **NextAuth.js Setup:**
   - JWT tokens com rotação
   - Session com Redis store
   - OAuth providers (Google, GitHub)
   - Email/password com bcrypt (cost 12)

2. **API Keys:**
   - SHA-256 hashed keys
   - Prefix para identificação (`pk_live_` , `pk_test_`)
   - Scopes granulares
   - Rate limiting por key

3. **RBAC (Role-Based Access Control):**
   `Admin: Full access`
   `User: Own resources only`
   `Viewer: Read-only access`

12.2 Rate Limiting

```
// Middleware para rate limiting
import rateLimit from 'express-rate-limit';

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  standardHeaders: true,
  legacyHeaders: false,
});

// Apply to API routes
app.use('/api/', limiter);
```

12.3 Input Sanitization

```
// Zod schemas para validação
import { z } from 'zod';

const instanceSchema = z.object({
  name: z.string().min(1).max(100).regex(/^[a-zA-Z0-9_-]+$/),
  startUrl: z.string().url().refine(url => {
    // Prevenir SSRF
    const parsed = new URL(url);
    return !['localhost', '127.0.0.1', '0.0.0.0'].includes(parsed.hostname);
  }),
});
```

12.4 HTTPS & CSP

```
// next.config.js
module.exports = {
  async headers() {
    return [
      {
        source: '/(.*)',
        headers: [
          {
            key: 'Content-Security-Policy',
            value: "default-src 'self'; script-src 'self' 'unsafe-eval'; style-src
'self' 'unsafe-inline'; img-src 'self' data: https:;",
          },
          {
            key: 'X-Frame-Options',
            value: 'DENY',
          },
          {
            key: 'X-Content-Type-Options',
            value: 'nosniff',
          },
          {
            key: 'Referrer-Policy',
            value: 'strict-origin-when-cross-origin',
          },
        ],
      },
    ];
  },
};
```

12.5 Secrets Management

```
# .env.example
DATABASE_URL="postgresql://user:password@localhost:5432/db"
REDIS_URL="redis://localhost:6379"
NEXTAUTH_SECRET="generate-with-openssl-rand-base64-32"
NEXTAUTH_URL="https://yourdomain.com"

# API Keys (encrypt in production)
BRIGHTDATA_API_KEY=""
SENDGRID_API_KEY=""
AWS_ACCESS_KEY_ID=""
AWS_SECRET_ACCESS_KEY=""
```

Produção: Usar AWS Secrets Manager, HashiCorp Vault, ou similar.

12.6 Audit Logs

```
// Log all critical actions
async function auditLog(action: string, userId: string, metadata: any) {
  await prisma.auditLog.create({
    data: {
      action,
      userId,
      metadata,
      timestamp: new Date(),
      ip: req.ip,
      userAgent: req.headers['user-agent'],
    },
  });
}

// Example usage
await auditLog('instance.delete', session.user.id, { instanceId });
```

---

13. Roadmap de Implementação

Fase 1: Setup e Infraestrutura (Semana 1)

- [x] Criar repositório Git
- [ ] Setup Next.js 14 com TypeScript
- [ ] Configurar TailwindCSS e shadcn/ui
- [ ] Setup PostgreSQL e Prisma
- [ ] Setup Redis
- [ ] Configurar Docker e docker-compose
- [ ] CI/CD pipeline básico

Fase 2: Autenticação (Semana 2)

- [ ] Implementar NextAuth.js
- [ ] Login/Register pages
- [ ] OAuth providers (Google, GitHub)
- [ ] API keys management
- [ ] Role-based access control

Fase 3: Core Backend (Semanas 3-4)

- [ ] BrowserService com Playwright
- [ ] ProxyService (fetch, validate, rotate)
- [ ] Anti-detection utilities
- [ ] ScriptService (execute JS)
- [ ] JobService com BullMQ
- [ ] Database schema completo

Fase 4: API Routes (Semana 5)

- [ ] Instances CRUD
- [ ] Proxies CRUD

- [ ] Scripts CRUD
- [ ] Jobs CRUD
- [ ] Analytics endpoints
- [ ] Webhooks

Fase 5: Frontend - Dashboard (Semana 6)

- [ ] Layout principal
- [ ] Dashboard home
- [ ] Stats cards
- [ ] Charts (Recharts)
- [ ] Real-time updates (Socket.io)

Fase 6: Frontend - Instance Manager (Semana 7)

- [ ] Instances list (tabela)
- [ ] Create instance form
- [ ] Instance detail view
- [ ] Live screenshot
- [ ] Logs streaming
- [ ] Control buttons (start/stop/restart)

Fase 7: Frontend - Proxy Manager (Semana 8)

- [ ] Proxies list
- [ ] Add/import proxies
- [ ] Validation UI
- [ ] Geolocation map
- [ ] Health status indicators

Fase 8: Frontend - Script Editor (Semana 9)

- [ ] Monaco Editor integration
- [ ] Script library
- [ ] Test runner
- [ ] Marketplace (básico)

Fase 9: Analytics (Semana 10)

- [ ] Metrics collection
- [ ] Analytics dashboard
- [ ] Export reports
- [ ] Alertas configuráveis

Fase 10: Testes e QA (Semana 11)

- [ ] Unit tests (Jest)
- [ ] Integration tests
- [ ] E2E tests (Playwright)
- [ ] Load testing
- [ ] Security audit

Fase 11: Deployment (Semana 12)

- [ ] Production Dockerfile
- [ ] Kubernetes manifests
- [ ] Monitoring (Prometheus + Grafana)
- [ ] Logging (Loki/ELK)
- [ ] Deploy to staging
- [ ] Deploy to production

Fase 12: Documentação e Lançamento (Semana 13)

- [ ] API documentation (Swagger)
- [ ] User guide
- [ ] Admin guide
- [ ] Video tutorials
- [ ] Beta launch

---

14. Recomendações Técnicas

14.1 Playwright vs Selenium

Por que Playwright?
- ✅ Mais rápido e confiável
- ✅ API moderna e melhor documentação
- ✅ Suporte nativo a múltiplos navegadores
- ✅ Melhor tratamento de promessas (async/await)
- ✅ Built-in waiting e auto-waiting
- ✅ Network interception nativo
- ✅ Screenshot e video recording
- ✅ Melhor suporte a anti-detecção

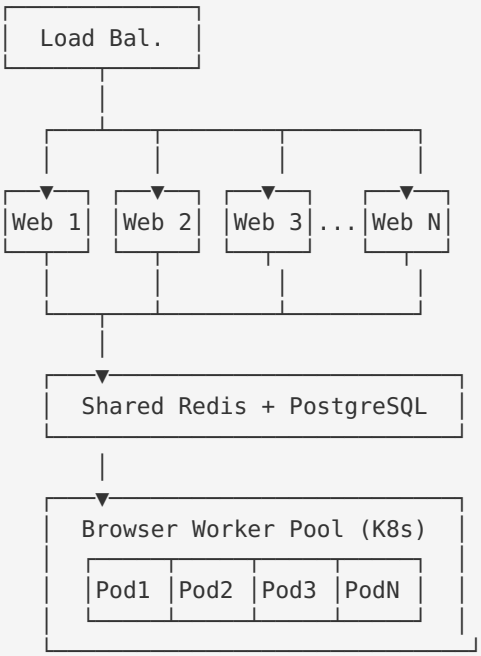
Migração:

```
# Selenium (atual)
driver = webdriver.Chrome(options=chrome_options)
driver.get(url)
driver.execute_script(script)

# Playwright (novo)
browser = await chromium.launch()
context = await browser.new_context()
page = await context.new_page()
await page.goto(url)
await page.evaluate(script)
```

14.2 Anti-Detecção Avançada

Implementar:
1. **playwright-extra + stealth plugin:**
```typescript
import { chromium } from 'playwright-extra';
import stealth from 'puppeteer-extra-plugin-stealth';
```

chromium.use(stealth());
```

1. **Canvas fingerprinting protection:**
   ```
   javascript
     await context.addInitScript(() => {
       const getImageData = HTMLCanvasElement.prototype.getImageData;
       HTMLCanvasElement.prototype.getImageData = function(...args) {
         const imageData = getImageData.apply(this, args);
         // Add noise
         for (let i = 0; i < imageData.data.length; i++) {
           imageData.data[i] += Math.floor(Math.random() * 10) - 5;
         }
         return imageData;
       };
     });
   ```

2. **WebGL fingerprinting protection:**
   ```
   javascript
     await context.addInitScript(() => {
       const getParameter = WebGLRenderingContext.prototype.getParameter;
       WebGLRenderingContext.prototype.getParameter = function(parameter) {
         if (parameter === 37445) {
           return 'Intel Inc.'; // Vendor
         }
         if (parameter === 37446) {
           return 'Intel Iris OpenGL Engine'; // Renderer
         }
         return getParameter.call(this, parameter);
       };
     });
   ```

3. **Timezone spoofing baseado em proxy:**
   ```
   typescript
     // Se proxy é do Brasil → timezone America/Sao_Paulo
     const timezone = getTimezoneFromProxyCountry(proxy.country);
     await context.newPage({ timezoneId: timezone });
   ```

## 14.3 Escalabilidade

**Arquitetura Distribuída:**
   ```



Recomendações:
- Web servers: Stateless, horizontalmente escaláveis
- Browser workers: Pods Kubernetes com Playwright
- Load balancing: Round-robin com health checks
- Auto-scaling: Baseado em CPU/RAM e tamanho da fila

14.4 Monitoramento e Observabilidade

Stack Recomendada:
- **Metrics:** Prometheus + Grafana
- **Logs:** Loki ou ELK Stack
- **Tracing:** Jaeger (para debugging distribuído)
- **Alerting:** Alertmanager

Métricas-chave:

```
// Instrumentação com prom-client
import { Counter, Gauge, Histogram } from 'prom-client';

const instancesActive = new Gauge({
  name: 'instances_active_total',
  help: 'Number of active instances',
});

const instanceStartDuration = new Histogram({
  name: 'instance_start_duration_seconds',
  help: 'Time to start instance',
  buckets: [1, 5, 10, 30, 60],
});

const proxyValidations = new Counter({
  name: 'proxy_validations_total',
  help: 'Total proxy validations',
  labelNames: ['status'], // 'success' or 'failure'
});
```

14.5 Custos e Otimização

Custos Estimados (mensal):

Recurso	Estimativa
Cloud VMs (10x medium)	$500-1000
PostgreSQL managed	$100-200
Redis managed	$50-100
Load balancer	$20-50
S3 storage (logs, screenshots)	$50-100
Proxies pagos (opcional)	$200-500
Total	**$920-1950/mês**

Otimizações:
- Usar spot instances para workers (economia 60-80%)
- Comprimir screenshots (JPEG quality 70-80)
- Logs com retenção de 30 dias
- Proxies gratuitos para testes, pagos para produção

14.6 Compliance e Legal

Considerações:
1. **Termos de Serviço:** Automação pode violar ToS de plataformas
2. **Rate Limiting:** Respeitar limites das plataformas
3. **Copyright:** Scripts devem ter licenças claras
4. **GDPR/LGPD:** Se coletar dados de usuários
5. **Disclaimer:** Adicionar disclaimer de uso responsável

Recomendação:
- Consultar advogado especializado
- Adicionar Termos de Uso e Política de Privacidade
- Implementar opt-out e data deletion
- Logging de todas as ações (audit trail)

---

15. Conclusão

Este documento fornece uma especificação técnica completa para migrar e expandir a plataforma de automação web de Python para uma solução moderna em Next.js/TypeScript.

Resumo dos principais pontos:

✅ **Análise completa** do sistema atual em Python
✅ **Técnicas de anti-detecção** identificadas e documentadas
✅ **Arquitetura proposta** escalável e moderna

✅ **Stack tecnológico** definido (Next.js, Playwright, PostgreSQL, Redis)
✅ **Estrutura de dados** completa com Prisma schema
✅ **APIs REST e WebSocket** especificadas
✅ **Frontend components** detalhados
✅ **Backend services** implementados
✅ **Segurança e compliance** considerados
✅ **Roadmap de 13 semanas** para implementação
✅ **Recomendações técnicas** para sucesso do projeto

Próximos passos:
1. Revisar e validar este documento com stakeholders
2. Iniciar Fase 1 (Setup e Infraestrutura)
3. Estabelecer sprints semanais
4. Configurar repositório e CI/CD
5. Começar desenvolvimento!

Documento preparado por: DeepAgent
Data: 16 de Dezembro de 2025
Versão: 1.0
Status: ✅ Completo e pronto para implementação