



Guia de Segurança - FlowMind AI

Este documento descreve as medidas de segurança implementadas no sistema e as melhores práticas que você deve seguir.



Proteções Implementadas

1. Autenticação e Autorização



O que já está protegido:

- **NextAuth.js:** Sistema robusto de autenticação com suporte a Google OAuth e credenciais
- **JWT Sessions:** Tokens seguros para manter usuários logados
- **Password Hashing:** Senhas armazenadas com bcrypt (nunca em texto plano)
- **Session Validation:** Todas as rotas API verificam sessão ativa



Verificações de Segurança:

```
// Exemplo de proteção em API routes
import { verifyAuth } from '@lib/security';

export async function GET(request: Request) {
  const { error, user } = await verifyAuth();
  if (error) return error;

  // Usuário autenticado, continuar...
}
```

2. Proteção de Dados Sensíveis



Implementado:

- **Criptografia de API Keys:** Credenciais de terceiros são criptografadas
- **Environment Variables:** Secrets nunca no código-fonte
- **SQL Injection Protection:** Prisma ORM previne ataques SQL
- **Input Sanitization:** Limpeza de dados de entrada para prevenir XSS



Armazenamento Seguro de Credenciais:

```
import { storeApiCredential, getApiCredential } from '@lib/api-credentials';

// Armazenar credencial do usuário (criptografada)
await storeApiCredential(userId, 'telegram', 'My Bot', {
  bot_token: 'xxx',
  api_key: 'yyy'
});

// Recuperar credencial (descriptografada apenas quando necessário)
const creds = await getApiCredential(userId, 'telegram');
```

3. Rate Limiting (Limite de Taxa)

Protege contra ataques de força bruta e abuso de API:

```
import { rateLimit } from '@lib/security';

export async function POST(request: Request) {
  const ip = request.headers.get('x-forwarded-for') || 'unknown';
  const { allowed, remaining } = await rateLimit(ip, 100, 60000);

  if (!allowed) {
    return NextResponse.json(
      { error: 'Too many requests' },
      { status: 429 }
    );
  }

  // Continuar com a requisição...
}
```

Limites Recomendados:

- **Login:** 5 tentativas por minuto
- **API Pública:** 100 requisições por minuto
- **Upload de Arquivos:** 10 uploads por hora
- **Criação de Recursos:** 50 por hora

4. Validação de Entrada

Todas as entradas são validadas:

```
import { sanitizeInput, isValidEmail } from '@lib/security';

// Validar email
if (!isValidEmail(email)) {
  return NextResponse.json({ error: 'Invalid email' }, { status: 400 });
}

// Sanitizar texto
const cleanTitle = sanitizeInput(title);
```

Protege contra:

- **XSS (Cross-Site Scripting):** Remove HTML e JavaScript maliciosos
- **SQL Injection:** Prisma ORM protege automaticamente
- **NoSQL Injection:** Validação de tipos e estruturas
- **Path Traversal:** Validação de caminhos de arquivo

5. Controle de Acesso (RBAC)

Níveis de Permissão:

1. Usuário Regular:

- Ver/criar suas próprias análises
- Ver/criar seus próprios pedidos
- Ver templates públicos
- Upload de arquivos (limite de tamanho)

2. Administrador:

- Tudo do usuário regular
- Criar/editar/deletar templates

- Ver/gerenciar todos os pedidos
- Acesso ao painel admin

```
// Proteger rota admin
const { error, user } = await verifyAuth({ requireAdmin: true });
if (error) return error;
```

6. Proteção de Arquivos Upload

```
import { validateFileUpload } from '@lib/security';

const validation = validateFileUpload(file, {
  maxSize: 10 * 1024 * 1024, // 10MB
  allowedTypes: ['application/json', 'image/png', 'image/jpeg']
});

if (!validation.valid) {
  return NextResponse.json({ error: validation.error }, { status: 400 });
}
```

7. CORS e Headers de Segurança

Configurar no `next.config.js` :

```
const securityHeaders = [
  {
    key: 'X-Frame-Options',
    value: 'DENY' // Previne clickjacking
  },
  {
    key: 'X-Content-Type-Options',
    value: 'nosniff' // Previne MIME sniffing
  },
  {
    key: 'Referrer-Policy',
    value: 'strict-origin-when-cross-origin'
  },
  {
    key: 'Permissions-Policy',
    value: 'camera=(), microphone=(), geolocation=()'
  }
];
```



Melhores Práticas para Desenvolvedores

1. Variáveis de Ambiente

✓ FAZER:

```
# .env (nunca commitar!)
DATABASE_URL="postgresql://..."
NEXTAUTH_SECRET="random-secret-here"
ENCRYPTION_KEY="another-random-key"
```

✗ NÃO FAZER:

```
// Nunca hardcode secrets!
const apiKey = "sk-abc123xyz"; // ❌ ERRADO
```

2. Validação de Dados

✅ **SEMPRE** validar entrada do usuário:

```
// Validar antes de usar
if (!title || title.length > 200) {
  return NextResponse.json({ error: 'Invalid title' }, { status: 400 });
}

// Sanitizar
const cleanTitle = sanitizeInput(title);
```

3. Logs de Segurança

```
import { logSecurityEvent } from '@lib/security';

// Logar eventos importantes
await logSecurityEvent('failed_login', null, { email, ip });
await logSecurityEvent('admin_access', userId, { action: 'delete_user' });
```

4. Verificar Propriedade de Recursos

```
import { verifyResourceOwnership } from '@lib/security';

// Antes de deletar/editar, verificar se o usuário é dono
const isOwner = await verifyResourceOwnership('analysis', analysisId, userId);
if (!isOwner) {
  return NextResponse.json({ error: 'Forbidden' }, { status: 403 });
}
```



Checklist de Segurança

Antes de fazer deploy em produção:

- [] **Variáveis de Ambiente:** Todas as secrets estão em `.env` (não no código)
- [] **HTTPS:** SSL/TLS configurado (obrigatório em produção)
- [] **Rate Limiting:** Implementado em todas as rotas críticas
- [] **Input Validation:** Todas as entradas são validadas e sanitizadas
- [] **Authentication:** Todas as rotas API verificam autenticação
- [] **Authorization:** Usuários só acessam seus próprios recursos
- [] **Encryption:** API keys e credenciais são criptografadas
- [] **File Upload:** Validação de tipo, tamanho e conteúdo
- [] **Error Messages:** Não expõem informações sensíveis
- [] **Logging:** Eventos de segurança são logados
- [] **Dependencies:** Todas as dependências estão atualizadas (`yarn audit`)
- [] **CORS:** Configurado apenas para origens confiáveis
- [] **Security Headers:** X-Frame-Options, CSP, etc.

- [] **Database:** Backups automáticos configurados
- [] **Session:** Timeout configurado (ex: 7 dias)

Comandos Úteis

```
# Verificar vulnerabilidades nas dependências
yarn audit

# Corrigir vulnerabilidades automáticas
yarn audit fix

# Verificar secrets expostos (use git-secrets ou similar)
git secrets --scan

# Backup do banco de dados
pg_dump $DATABASE_URL > backup.sql
```

Recursos Adicionais

Ferramentas Recomendadas:

- **OWASP ZAP:** Teste de penetração automatizado
- **Snyk:** Monitoramento de vulnerabilidades
- **SonarQube:** Análise de código
- **Git-Secrets:** Previne commit de secrets

Links Úteis:

- **OWASP Top 10** (<https://owasp.org/www-project-top-ten/>)
- **Next.js Security** (<https://nextjs.org/docs/advanced-features/security-headers>)
- **Prisma Security** (<https://www.prisma.io/docs/concepts/components/prisma-client/deployment#security>)

Em Caso de Incidente de Segurança

1. **Isolar:** Desabilite a área afetada imediatamente
2. **Avaliar:** Identifique o escopo do problema
3. **Notificar:** Informe usuários afetados (se necessário)
4. **Corrigir:** Implemente patch de segurança
5. **Revisar:** Faça post-mortem e melhore processos
6. **Documentar:** Registre o incidente e as ações tomadas


Contato

Para reportar vulnerabilidades de segurança:

- **Email:** security@flowmind.ai (criar email dedicado)
- **Processo:** Responsible disclosure - 90 dias antes de publicar

Última atualização: Outubro 2025

Versão: 1.0.0

 **IMPORTANTE:** Este é um documento vivo. Atualize sempre que adicionar novas features ou medidas de segurança.