





Prisma Best Practices - Prevenção de Erros

Proteções Implementadas

1. Server-Only Protection

Todos os arquivos que usam Prisma agora têm `import 'server-only'` no topo:

-  `lib/db.ts` - Instância do Prisma Client
-  `lib/auth.ts` - Autenticação NextAuth
-  `lib/email.ts` - Envio de emails
-  Todas as rotas em `app/api/**/*.ts`

2. Validação de Ambiente

O arquivo `lib/db.ts` agora valida se `DATABASE_URL` existe antes de inicializar o Prisma.

3. Singleton Pattern

Previne múltiplas instâncias do Prisma Client em desenvolvimento (hot reload).

4. Cleanup de Conexões

Fecha automaticamente as conexões do Prisma quando o processo termina.

5. Tratamento de Erros

Novo helper em `lib/prisma-helpers.ts` com:

- Conversão de erros do Prisma em mensagens amigáveis
- Retry automático para erros de conexão
- Validação de ambiente servidor

Regras de Uso do Prisma

PERMITIDO

1. Server Components (App Router)

```
// app/dashboard/page.tsx
import { prisma } from '@lib/db'

export default async function DashboardPage() {
  const users = await prisma.user.findMany()
  return <div>{/* ... */}</div>
}
```

2. API Routes

```
// app/api/users/route.ts
import 'server-only'
import { prisma } from '@lib/db'

export async function GET() {
  const users = await prisma.user.findMany()
  return Response.json(users)
}
```

3. Server Actions (se implementados)

```
// app/actions.ts
'use server'
import { prisma } from '@lib/db'

export async function createUser(data: any) {
  return await prisma.user.create({ data })
}
```

✗ PROIBIDO

1. Client Components

```
// ✗ NUNCA FAÇA ISSO
'use client'
import { prisma } from '@lib/db' // ERRO!

export function MyComponent() {
  // ...
}
```

2. Hooks ou Effects no Cliente

```
// ✗ NUNCA FAÇA ISSO
'use client'
useEffect(() => {
  const data = await prisma.user.findMany() // ERRO!
}, [])
```

3. Importações Indiretas

```
// ✗ CUIDADO COM IMPORTAÇÕES INDIRETAS
// lib/my-utils.ts (sem 'server-only')
import { prisma } from './db' // Pode vazar para o cliente!
```

Como Usar o Helper Seguro

```
import { safeQuery, handlePrismaError } from '@lib/prisma-helpers'
import { prisma } from '@lib/db'

// Exemplo 1: Com retry automático
export async function GET() {
  const result = await safeQuery(
    () => prisma.user.findMany()
  )

  if (result.error) {
    return Response.json(
      { error: result.error },
      { status: result.statusCode }
    )
  }

  return Response.json(result.data)
}

// Exemplo 2: Tratamento manual de erros
export async function POST() {
  try {
    const user = await prisma.user.create({ data: {...} })
    return Response.json(user)
  } catch (error) {
    const errorInfo = handlePrismaError(error)
    return Response.json(
      { error: errorInfo.message },
      { status: errorInfo.statusCode }
    )
  }
}
```



Erros Comuns e Soluções

Erro: “PrismaClient is unable to run in this browser environment”

Causa: Prisma sendo usado em componente cliente

Solução:

1. Mova a lógica para um Server Component ou API Route
2. Use fetch no cliente para chamar a API Route

Erro: “Environment variable not found: DATABASE_URL”

Causa: `.env` não carregado ou variável não definida

Solução:

1. Verifique se o arquivo `.env` existe
2. Reinicie o servidor de desenvolvimento
3. No Vercel, configure a variável em Settings > Environment Variables

Erro: “PrismaClient is not configured to run in this environment”

Causa: Prisma Client não foi gerado ou está desatualizado

Solução:

```
yarn prisma generate
```

Erro: “P2002: Unique constraint failed”

Causa: Tentativa de criar registro com valor único duplicado

Solução: Use `handlePrismaError()` para mostrar mensagem amigável



Estrutura de Arquivos

```
lib/
├─ db.ts           # ✓ Prisma Client (com server-only)
├─ auth.ts         # ✓ NextAuth config (com server-only)
├─ email.ts        # ✓ Email service (com server-only)
├─ prisma-helpers.ts # ✓ Helpers seguros (com server-only)
├─ types.ts        # ✓ Types (sem Prisma, seguro)
└─ utils.ts        # ✓ Utilities (sem Prisma, seguro)

app/
├─ api/           # ✓ Todas com server-only
│   └─ users/route.ts
│   └─ ...
├─ dashboard/     # ✓ Server Components (safe)
│   └─ page.tsx
└─ components/    # ✗ Client Components (SEM Prisma)
    └─ ...
```



Checklist de Segurança

Antes de fazer deploy, verifique:

- [] Todos os arquivos que importam `prisma` têm `import 'server-only'`
- [] Nenhum componente com `'use client'` importa `@/lib/db` ou `@/lib/auth`
- [] Variável `DATABASE_URL` está configurada no Vercel
- [] Build local passa sem erros: `yarn build`
- [] Não há warnings sobre server-only no console



Comando de Verificação

Use este comando para verificar se há importações problemáticas:

```
# Procura client components que importam Prisma
grep -r "'use client'" components/ app/ --include="*.tsx" --include="*.ts" -l | \
  xargs grep -l "from '@/lib/db'\|from '@/lib/auth'"

# Se retornar vazio = ✓ SEGURO
# Se retornar arquivos = ✗ CORRIGIR
```

Recursos

- [Next.js Server Components](https://nextjs.org/docs/app/building-your-application/rendering/server-components) (<https://nextjs.org/docs/app/building-your-application/rendering/server-components>)
 - [Prisma Best Practices](https://www.prisma.io/docs/guides/performance-and-optimization/connection-management) (<https://www.prisma.io/docs/guides/performance-and-optimization/connection-management>)
 - [server-only package](https://www.npmjs.com/package/server-only) (<https://www.npmjs.com/package/server-only>)
-

Última atualização: 30/10/2025

Versão: 2.0 - Proteção Completa