Auditoria no iagro

Módulo de Auditoria

O que é Auditoria

Auditoria é o **registro automático de todas as ações** realizadas no sistema. É como um "histórico detalhado" que registra:

- **Quem** fez a ação (usuário)
- **O que** foi modificado (dados anteriores vs novos)
- **Quando** aconteceu (timestamp)
- **Onde** foi feita (IP, endpoint)
- **Como** foi o resultado (sucesso ou erro)

Por que é Importante

A auditoria é essencial para:

- **Segurança**: Rastrear alterações não autorizadas
- **Compliance**: Atender requisitos legais e normativos
- **Debugging**: Investigar problemas e bugs
- **Transparência**: Mostrar histórico de mudanças aos usuários

Como Funciona Nosso Módulo

Auditoria Automática por Método HTTP

O sistema audita automaticamente **TODOS os métodos** exceto GET:

- **POST (CREATE)**: Auditoria automática para toda aplicação
- **PUT/PATCH/DELETE**: Requer decorator `@CapturePreviousData` para incluir dados anteriores
- **GET**: Não auditado por padrão (pode ser habilitado via env: `AUDIT_INCLUDE_READS=true`)

> Nota: a variável `AUDIT_INCLUDE_READS` hoje não está listada no `.env.example` do projeto — precisa ser adicionada manualmente ao `.env` de cada ambiente para ser habilitada.

Arquitetura Simples

```
Request → Interceptor Específico → Controller → Service → Interceptor de Auditoria
```

```
2          (só UPDATE/DELETE)                    (registra TODOS os métodos)
```

> **Nota de implementação**: o Interceptor de Auditoria central registra o log de todos os métodos, mas a captura do `previousData` em si é feita por um interceptor específico **por módulo** (registrado como provider no respectivo `*.module.ts`), acionado apenas quando o controller usa o decorator `@CapturePreviousData`. Ou seja, a cobertura real de `previousData` depende de cada módulo ter esse interceptor corretamente registrado — ver "Cobertura atual por módulo" abaixo.

Fluxo de Execução

Para CREATE (POST):

1. **Request POST** chega (ex: POST /branches)
2. **Controller** executa normalmente
3. **Interceptor de auditoria** registra log automaticamente

Para UPDATE/DELETE:

1. **Request chega** (ex: PUT /branches/123)
2. **Interceptor específico** captura dados atuais (se tiver `@CapturePreviousData`)
3. **Controller** executa normalmente
4. **Service** realiza a modificação
5. **Interceptor de auditoria** registra log completo com dados anteriores

Quando Usar o Decorator

```
// ✅ PRECISA do decorator (para capturar dados anteriores):
@CapturePreviousData({
  resourceType: 'branches',
  resourceIdParam: 'id',
})
@Put(':id')
@Delete(':id')

// ❌ NÃO PRECISA do decorator (auditoria automática):
@Post()
@Get()
```

Cobertura atual por módulo

Módulos com captura de `previousData` funcionando (decorator aplicado no controller + interceptor de captura registrado no módulo): `branches`, `defensives`, `locations`,

`operations`, `responsible-parties`, `smart-caldas`, `smart-calda-maintenance`, `stock`, `suppliers`, `tenants`, `trucks`, `user`.

Controllers com PUT/PATCH/DELETE que hoje não usam o decorator (auditados normalmente, mas sem `previousData`): `work-order`, `work-order-recipe-plc-sync`, `truck-route`, `supervisory`, `supervisory-component-image`, `supervisory-layout`, `branch-capabilities`, `driver`, `realm-role`, `user-profile`, `client-role`, `defensive-label`, `cost-center`, `access-roles`.

Casos de atenção conhecidos (decorator e interceptor não estão alinhados no mesmo módulo):

- `farms`: usa o decorator no controller, mas o módulo não registra o interceptor de captura — `previousData` não é gerado apesar do decorator estar presente.
- `defensive-classes` e `invoice-imports`: usam o decorator, mas não há interceptor de captura correspondente registrado nesses módulos.
- `driver`: o módulo registra o interceptor de captura, mas o controller não usa o decorator — o interceptor nunca encontra dados para capturar.

O que o Sistema Faz Automaticamente

Para TODOS os métodos (POST, PUT, PATCH, DELETE):

- Identifica o usuário da requisição
- Registra dados enviados na requisição
- Inclui metadados (IP, timestamp, status HTTP)
- Salva tudo em log estruturado

Adicional para UPDATE/DELETE (com decorator, e módulo com interceptor de captura registrado):

- Captura dados antes da modificação
- Inclui dados anteriores no log

Exemplo de Log Gerado

```
{
  "id": "3f2a1c9e-7b4d-4e21-9c3a-5d8e1f0a2b6c",
  "action": "UPDATE",
  "resourceType": "branches",
  "resourceId": "branch_456",
  "tenantId": "tenant_abc",
  "previousData": { "name": "Nome Antigo", "status": "ACTIVE" },
  "requestData": { "name": "Nome Novo" },
  "userId": "user_789",
  "userEmail": "usuario@empresa.com",
  "ipAddress": "192.168.1.10",
  "userAgent": "Mozilla/5.0 ...",
```

```
  "endpoint": "PUT /branches/456",
  "success": true,
  "statusCode": 200,
  "errorMessage": null,
  "timestamp": "2024-01-15T10:30:00Z"
}
```

> Nota: o `id` real é um UUID gerado pelo banco (`gen_random_uuid()`), não uma string simples como `audit_123` . Os campos `tenantId` , `userEmail` , `userAgent` e `errorMessage` também fazem parte do registro e foram incluídos aqui para refletir a estrutura completa da tabela `audit_logs` .

Vantagens da Nossa Implementação

- **Automática**: CREATE é auditado sem configuração adicional
- **Flexível**: UPDATE/DELETE podem incluir dados anteriores com decorator simples
- **Completa**: Captura usuário + dados + metadados automaticamente
- **Consistente**: Mesmo padrão para todos os módulos
- **Limpa**: Services focam apenas na lógica de negócio
- **Performática**: Operações GET não são auditadas por padrão

Status

- Em produção desde set/2025, com evolução contínua desde então (ex.: suporte dedicado ao módulo de estoque foi adicionado em abr/2026). Deixou de ser tratada como uma PoC isolada e passou a fazer parte do ciclo normal de desenvolvimento — novos módulos precisam registrar seu próprio interceptor de captura para ganhar cobertura de `previousData` (ver "Cobertura atual por módulo" acima).