Auditoria em novos módulos

IMPORTANTE: Métodos CREATE (POST) já são auditados automaticamente para todos os módulos.

Esta implementação é apenas para **capturar dados anteriores** em UPDATE e DELETE. Se você só precisa de auditoria básica, não precisa fazer nada!

Quando Implementar

Implemente apenas se você precisar de **dados anteriores** nos logs de UPDATE/DELETE.

Passo 1: Adicionar o Decorator no Controller

Adicione o decorator `@CapturePreviousData` **APENAS nos métodos de UPDATE e DELETE**:

```
import { CapturePreviousData } from '@shared/audit/decorators/capture-p

@Controller('suppliers')
export class SuppliersController {
  @CapturePreviousData({
    resourceType: 'suppliers',
    resourceIdParam: 'id',
  })
  @Put(':id')
  async update(@Param('id') id: string, @Body() data: UpdateSupplierReq
    return this.updateUseCase.execute({ id, data });
  }

  @CapturePreviousData({
    resourceType: 'suppliers',
    resourceIdParam: 'id',
  })
  @Delete(':id')
  async delete(@Param('id') id: string) {
    return this.deleteUseCase.execute({ id });
  }

  // ✅ CREATE não precisa de decorator - auditoria automática
  @Post()
  async create(@Body() data: CreateSupplierRequest) {
    return this.createUseCase.execute(data);
  }
}

```

Passo 2: Criar o Service de Captura

Crie um service específico para capturar os dados anteriores do seu recurso:

```
// src/modules/suppliers/services/supplier-previous-data.service.ts
import { Injectable } from '@nestjs/common';
import { PreviousDataService } from '@shared/audit/services/previous-da
import { SupplierRepository } from '../repositories/supplier.repository

@Injectable()
export class SupplierPreviousDataService {
```

```
  constructor(
    private readonly repository: SupplierRepository,
    private readonly previousDataService: PreviousDataService,
  ) {}

  async captureSupplierData(
    supplierId: string,
    requestId: string,
  ): Promise<void> {
    const existing = await this.repository.findById(supplierId);

    if (existing) {
      const previousData = {
        id: existing.id,
        name: existing.name,
        email: existing.email,
        // Mapeie apenas os campos que você quer auditar
      };

      this.previousDataService.store(requestId, previousData);
    }
  }
}

```

Passo 3: Criar o Interceptor Específico

Crie um interceptor que usa o service de captura:

```
// src/modules/suppliers/interceptors/supplier-previous-data.intercepto
import {
  Injectable,
  NestInterceptor,
  ExecutionContext,
  CallHandler,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { Observable } from 'rxjs';
import { SupplierPreviousDataService } from '../services/supplier-previ
import { CAPTURE_PREVIOUS_DATA_KEY } from '@shared/audit/decorators/cap

@Injectable()
export class SupplierPreviousDataInterceptor implements NestInterceptor
  constructor(
    private readonly supplierPreviousDataService: SupplierPreviousDataS
    private readonly reflector: Reflector,
  ) {}

  async intercept(
    context: ExecutionContext,
    next: CallHandler,
  ): Promise<Observable<any>> {
    const captureConfig = this.reflector.get(
      CAPTURE_PREVIOUS_DATA_KEY,
      context.getHandler(),
    );

    if (captureConfig?.resourceType === 'suppliers') {
      const request = context.switchToHttp().getRequest();
      const resourceId = request.params[captureConfig.resourceIdParam];
      const requestId = request.id;

      if (resourceId) {
        await this.supplierPreviousDataService.captureSupplierData(
          resourceId,
```

```
          requestId,
        );
      }
    }

    return next.handle();
  }
}

```

Passo 4: Registrar no Módulo

Adicione o service e o interceptor no seu módulo:

```
// src/modules/suppliers/suppliers.module.ts
import { Module } from '@nestjs/common';
import { APP_INTERCEPTOR } from '@nestjs/core';
import { SupplierPreviousDataService } from './services/supplier-previo
import { SupplierPreviousDataInterceptor } from './interceptors/supplie

@Module({
  controllers: [SuppliersController],
  providers: [
    // Seus services existentes...
    SupplierRepository,
    UpdateSupplierUseCase,
    DeleteSupplierUseCase,

    // Adicione estes:
    SupplierPreviousDataService,
    {
      provide: APP_INTERCEPTOR,
      useClass: SupplierPreviousDataInterceptor,
    },
  ],
})
export class SuppliersModule {}

```

Passo 5: Remover Auditoria Manual dos Services

Se seus services têm chamadas manuais para auditoria, remova-as:

```
// ANTES (remover isso):
export class UpdateSupplierService {
  async execute({ id, data }: UpdateSupplierRequest) {
    const existing = await this.repository.findById(id);
    const updated = await this.repository.update(id, data);

    // ❌ REMOVA ISSO:
    await this.auditService.logWithPreviousData({
      action: 'UPDATE',
      resourceType: 'suppliers',
      resourceId: id,
      previousData: existing,
      requestData: data,
    });

    return updated;
  }
}

// DEPOIS (deixe limpo):
export class UpdateSupplierService {
```

```
  async execute({ id, data }: UpdateSupplierRequest) {
    return await this.repository.update(id, data);
  }
}

```

Checklist de Implementação

- [ ] Decorator `@CapturePreviousData` adicionado **APENAS** em UPDATE e DELETE
- [ ] Service de captura criado com mapeamento dos campos
- [ ] Interceptor específico implementado
- [ ] Service e interceptor registrados no módulo
- [ ] Auditoria manual removida dos services
- [ ] Testado: CREATE funciona sem decorator, UPDATE/DELETE capturam dados anteriores

Exemplo Completo

Para referência, veja a implementação no módulo `branches` que já está funcionando corretamente.

Resumo dos Comportamentos

Auditoria Automática (sem configuração)

- **POST**: ✅ Auditado automaticamente
- **PUT/PATCH/DELETE**: ✅ Auditado automaticamente (sem dados anteriores)
- **GET**: ❌ Não auditado (configuar `AUDIT_INCLUDE_READS=true` se necessário)

Com `@CapturePreviousData` (implementação adicional)

- **PUT/PATCH/DELETE**: ✅ Auditado com dados anteriores incluídos

Dúvidas?

- CREATE já funciona automaticamente, não precisa de configuração
- O decorator só é necessário para incluir dados anteriores
- O `resourceIdParam` deve corresponder ao nome do parâmetro na rota
- Mapeie apenas os campos necessários no service de captura