

Implementação do Módulo de Notificações



Resumo

O módulo de notificações foi migrado de `backend/src/notificacoes/` para `backend/src/modules/notificacoes/` seguindo o padrão estabelecido no projeto DELIVEREI. A implementação inclui CRUD completo, suporte a dados mock, paginação e todos os endpoints solicitados.



Estrutura do Módulo

```
backend/src/modules/notificacoes/  
├── dto/  
│   ├── create-notificacao.dto.ts    # DTO para criação com validações  
│   ├── update-notificacao.dto.ts    # DTO para atualização (PartialType)  
│   └── index.ts                     # Exports dos DTOs  
├── notificacoes.controller.ts       # Controller com todos os endpoints  
├── notificacoes.service.ts          # Service com lógica de negócio e mock  
└── notificacoes.module.ts           # Module configurado
```



Endpoints Implementados

1. GET /api/notificacoes

- Lista notificações do usuário autenticado
- Suporta paginação: `?page=1&limit=20`
- Filtros: `?tipo=PEDIDO&lida=false`
- Retorna: Lista paginada com total, página, etc.

2. GET /api/notificacoes/:id

- Busca notificação específica por ID
- Valida se a notificação pertence ao usuário
- Retorna 404 se não encontrada

3. POST /api/notificacoes

- Cria nova notificação
- Requer role: `ADMIN_EMPRESA` ou `SUPER_ADMIN`
- Body: `{ titulo, mensagem, tipo, usuarioId, empresaId?, pedidoId? }`

4. PATCH /api/notificacoes/:id

- Atualiza notificação existente
- Valida propriedade do usuário
- Permite atualizar qualquer campo do DTO

5. PATCH /api/notificacoes/:id/marcar-lida

- Marca notificação específica como lida
- Endpoint específico para UI de notificações

6. PATCH /api/notificacoes/marcar-todas-lidas

- Marca todas as notificações não lidas do usuário como lidas
- Retorna quantidade de notificações atualizadas

7. DELETE /api/notificacoes/:id

- Remove notificação do usuário
- Validação de propriedade antes de deletar

8. GET /api/notificacoes/nao-lidas

- Conta notificações não lidas do usuário
- Útil para badge de contador na UI
- Retorna: `{ count: number }`



DTOs Implementados

CreateNotificacaoDto

```
{
  titulo: string;           // @IsString, @IsNotEmpty
  mensagem: string;        // @IsString, @IsNotEmpty
  tipo: string;             // @IsIn(['PEDIDO', 'SISTEMA', 'PROMOCAO'])
  usuarioId: string;        // @IsString, @IsNotEmpty
  empresaId?: string;       // @IsString, @IsOptional
  pedidoId?: string;        // @IsString, @IsOptional
}
```

UpdateNotificacaoDto

```
// PartialType(CreateNotificacaoDto) + campo extra:
{
  ...CreateNotificacaoDto (todos opcionais),
  lida?: boolean;           // @IsBoolean, @IsOptional
}
```



Dados Mock

Ativação

Configure a variável de ambiente:

```
USE MOCK NOTIFICACOES=true
```

Dados de Exemplo

O service possui 5 notificações mock pré-configuradas:

- 3 não lidas (Pedido Realizado, Em Preparo, Saiu para Entrega)
- 2 lidas (Promoção, Bem-vindo)
- Tipos variados: PEDIDO, PROMOCAO, SISTEMA
- Datas realistas

Funcionalidades Mock

- Todas as operações CRUD funcionam com dados em memória
- Logs identificam operações mock: [MOCK] ...
- Paginação funcional
- Filtros por tipo e status de leitura

Segurança

Guards Aplicados

- **JwtAuthGuard**: Todos os endpoints requerem autenticação
- **RolesGuard**: POST de notificações restrito a ADMIN_EMPRESA e SUPER_ADMIN

Validação de Propriedade

Todos os endpoints que manipulam notificações específicas validam se a notificação pertence ao usuário autenticado antes de realizar a operação.



Campos do Modelo

```
interface Notificacao {
  id: string;
  titulo: string;
  mensagem: string;
  tipo: 'PEDIDO' | 'SISTEMA' | 'PROMOCAO';
  lida: boolean;
  usuarioId: string;
  empresaId?: string;
  pedidoId?: string;
  createdAt: Date;
  updatedAt: Date;
}
```



Métodos Auxiliares

O service inclui métodos helper para criar notificações automáticas:

notificarNovoPedido()

```
notificarNovoPedido(
  pedidoId: string,
  usuarioId: string,
  empresaId: string,
  numeroPedido: string
)
```

notificarMudancaStatus()

```
notificarMudancaStatus(
  pedidoId: string,
  usuarioId: string,
  empresaId: string,
  numeroPedido: string,
  novoStatus: string
)
```

Suporta os status:

- CONFIRMADO
- EM_PREPARO
- SAIU_ENTREGA
- ENTREGUE
- CANCELADO



Integração

Import no app.module.ts

```
import { NotificacoesModule } from './modules/notificacoes/notificacoes.module';

@Module({
  imports: [
    // ... outros módulos
    NotificacoesModule,
  ],
})
```

Exclusão do TenantMiddleware

O módulo está configurado para ser excluído do middleware de tenant:

```
consumer
  .apply(TenantMiddleware)
  .exclude('api/notificacoes/(.*)')
  .forRoutes('*');
```



Como Testar

1. Com Dados Mock

```
# .env
USE MOCK NOTIFICACOES=true
```

2. Requisições de Exemplo

Listar notificações:

```
curl -H "Authorization: Bearer <token>" \
  http://localhost:3000/api/notificacoes?page=1&limit=10
```

Criar notificação:

```
curl -X POST \
-H "Authorization: Bearer <token>" \
-H "Content-Type: application/json" \
-d '{
  "titulo": "Nova Promoção",
  "mensagem": "50% de desconto em pizzas!",
  "tipo": "PROMOCAO",
  "usuarioId": "user-123",
  "empresaId": "empresa-456"
}' \
http://localhost:3000/api/notificacoes
```

Marcar como lida:

```
curl -X PATCH \
-H "Authorization: Bearer <token>" \
http://localhost:3000/api/notificacoes/{id}/marcar-lida
```

Contar não lidas:

```
curl -H "Authorization: Bearer <token>" \
http://localhost:3000/api/notificacoes/nao-lidas
```

✓ Checklist de Implementação

- [x] Estrutura de pastas seguindo padrão `backend/src/modules/`
- [x] DTOs com validações (create, update) e `index.ts`
- [x] Controller com todos os 8 endpoints solicitados
- [x] Service com lógica de negócio completa
- [x] Module configurado com `PrismaModule`
- [x] Dados mock para testes (`USE MOCK NOTIFICACOES=true`)
- [x] Paginação usando `calculatePagination` e `paginatedResponse`
- [x] Logger para debug
- [x] Guards JWT e Roles aplicados
- [x] Validação de propriedade do usuário
- [x] Métodos auxiliares para notificações automáticas
- [x] Módulo importado no `app.module.ts`
- [x] Exclusão do `TenantMiddleware` configurada
- [x] Módulo antigo removido de `backend/src/notificacoes`

🔄 Commit Realizado

`feat(notificacoes): Migrar módulo para modules/ e implementar CRUD completo`

Commit hash: `c4f231a`

Branch: `feature/produtos-module`

Observações

Compatibilidade com Prisma

O código está preparado para funcionar com o schema Prisma existente. Se o modelo `Notificacao` no Prisma não incluir o campo `empresaId`, ele será adicionado opcionalmente.

Próximos Passos Sugeridos

1. Testar endpoints com Postman/Insomnia
2. Verificar integração com o Prisma schema
3. Implementar notificações em tempo real com WebSocket (opcional)
4. Adicionar filtros avançados (data range, prioridade)
5. Implementar soft delete se necessário

Performance

- Paginação implementada para evitar sobrecarga
- Índices no banco de dados recomendados:
- `usuarioId` (usado em WHERE frequentemente)
- `lida` (para filtros)
- `createdAt` (para ordenação)

Troubleshooting


Erro 404 persiste

1. Verificar se o servidor foi reiniciado após as alterações
2. Confirmar que `USE MOCK NOTIFICACOES=true` está no `.env`
3. Verificar logs do servidor para erros de importação

Prisma errors

1. Rodar `npx prisma generate` após alterações no schema
2. Verificar se o modelo `Notificacao` existe no `schema.prisma`
3. Confirmar que `PrismaModule` está funcionando

Mock não está funcionando

1. Verificar variável de ambiente: `console.log(process.env.USE MOCK NOTIFICACOES)`
2. Observar logs do servidor: deve aparecer “ Usando dados MOCK...”
3. Reiniciar servidor após alterar `.env`

Implementação concluída com sucesso! ✨

O módulo de notificações está completamente funcional e seguindo todos os padrões do projeto DELIVEREI.