Documentação Técnica – Backend Node.js: Projeto Nobile

# 1. Introdução

Este documento apresenta a estrutura técnica e os componentes do backend do projeto Nobile, um marketplace de relógios de luxo. A API será construída em Node.js utilizando o framework Express, com banco de dados PostgreSQL e autenticação baseada em JWT. O backend será responsável por gerenciar usuários, produtos (relógios), compras, coleções, mensagens e funcionalidades administrativas.

# 2. Estrutura Geral da API

O backend será dividido nos seguintes módulos principais:

* - Autenticação e Autorização (JWT + roles)
* - Gerenciamento de Usuários (compradores, vendedores, admin)
* - Catálogo de Produtos (CRUD de relógios)
* - Checkout e Pagamentos (integração com Stripe)
* - Mensagens e Chat interno
* - Coleções pessoais e gestão de relógios
* - Histórico de preços (com integração analítica)
* - Painel administrativo (gestão de usuários, anúncios, relatórios)
* - Integração com serviços de frete e envio (Correios ou MelhorEnvio)

# 3. Endpoints Principais

/auth/login – Autenticação com email/senha e retorno de token JWT

/auth/register – Registro de novo usuário (comprador ou vendedor)

/users/me – Dados do usuário logado

/watches – Listagem e cadastro de relógios (GET, POST, PUT, DELETE)

/orders – Gestão de pedidos e status de compra

/collections – Relógios pessoais do usuário

/messages – Sistema de mensagens entre usuários

/admin/\* – Endpoints restritos para administração (moderação, analytics, usuários)

# 4. Modelagem de Dados (Entidades principais)

* - User: id, name, email, passwordHash, role (buyer, seller, admin)
* - Watch: id, brand, model, year, condition, price, sellerId, images[]
* - Order: id, buyerId, watchId, status, paymentInfo
* - Message: id, fromUserId, toUserId, watchId, content, timestamp
* - Collection: id, userId, watchId, estimatedValue
* - PriceHistory: id, watchModelId, date, averagePrice
* - AdminLog: id, action, userId, timestamp

# 5. Tecnologias Utilizadas

- Node.js com Express  
- PostgreSQL com Prisma ORM  
- Autenticação JWT com roles  
- Stripe API para pagamentos  
- Multer para upload de imagens  
- Cloudinary ou S3 para armazenamento  
- Swagger para documentação da API  
- Jest ou Vitest para testes unitários

# 6. Segurança

Todas as rotas protegidas exigirão autenticação JWT. Endpoints sensíveis (admin, transações, dados pessoais) terão verificação adicional de role. O backend terá proteção contra ataques de injeção, CSRF e XSS com validação de entrada e uso de middleware padrão do Express.