

---

**Exemplo Plano de Testes**  
**Inatel**  
**Engenharia de Software**

---

# **Plano de Testes**

API de Catálogo Musical (S206)

**Professor:**  
Christopher Lima

**Equipe:**  
Grupo S206

## Histórico de Revisões

Data	Versão	Descrição	Autor
24/nov/25	1.0	Release Inicial do Plano para Projeto Jest	Grupo S206

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
1.1	Objetivos . . . . .	3
1.2	A API musical . . . . .	3
1.3	Escopo . . . . .	3
<b>2</b>	<b>Requisitos a Testar</b>	<b>3</b>
2.1	Teste do Banco de Dados e Integridade . . . . .	4
2.2	Teste Funcional (Regras de Negócio) . . . . .	4
2.3	Teste do circlos de negócios (Fluxos) . . . . .	4
2.4	Perfil da Performance . . . . .	4
2.5	Teste de Carga . . . . .	4
2.6	Teste de Segurança e Controle de Acesso . . . . .	5
<b>3</b>	<b>Estratégia de Teste</b>	<b>5</b>
3.1	Tipos de Teste . . . . .	5
3.1.1	Teste de Integridade de Dados e do Banco de Dados . . . . .	5
3.1.2	Teste de Função (Integração) . . . . .	5
3.1.3	Teste de Performance e Carga . . . . .	6
3.1.4	Teste de Segurança . . . . .	6
3.2	Ferramentas . . . . .	6
<b>4</b>	<b>Recursos</b>	<b>6</b>
4.1	Sistema . . . . .	6

# 1 Introdução

## 1.1 Objetivos

Esse documento do Plano de Testes da API de Catálogo Musical compõe-se dos seguintes objetivos:

- Identificar informações de projeto existentes e os componentes de software (API, Banco de Dados) que devem ser testados.
- Listar os Requisitos a Testar recomendados (alto nível), focando na lógica de negócios e integridade de dados.
- Recomendar e descrever as estratégias de teste a serem empregadas (Testes de Integração com Jest e Supertest).
- Identificar os recursos necessários e prover uma estimativa dos esforços de teste.

## 1.2 A API musical

O sistema alvo deste plano é uma API RESTful desenvolvida em Node.js/TypeScript para gestão de um catálogo musical. O sistema permite o cadastro de Usuários, Artistas, Álbuns e Faixas musicais. O acesso às operações de escrita (POST, DELETE) é protegido via autenticação JWT (JSON Web Token), enquanto a consulta de Artistas é pública. Além dos módulos CRUD (Create, Read, Update, Delete), a aplicação integra-se com um serviço externo (Lyrics Service) para enriquecer os dados das faixas com letras de música automaticamente. O sistema utiliza Prisma ORM e banco de dados PostgreSQL.

## 1.3 Escopo

A API passará pelos testes de integração, performance básica e segurança. Os testes de integração vão lidar com a qualidade funcional dos endpoints, validação das regras de negócio, integridade referencial do banco de dados e controle de acesso. Os testes de interface gráfica (Frontend) não serão realizados, uma vez que o projeto consiste exclusivamente no Backend (API). Os testes de carga serão realizados em nível básico (até 100 usuários simultâneos) para garantir tempos de resposta aceitáveis, utilizando scripts automatizados via Jest.

Os testes mais críticos serão os testes de API (Integração), que compõem a maior parte do sistema e cobrem:

1. O fluxo de autenticação e autorização via Token.
2. A integridade das relações Artista → Álbum → Faixa.
3. A resiliência do sistema em caso de falha do serviço externo de letras.

# 2 Requisitos a Testar

A lista abaixo identifica os itens – endpoints, regras de negócio e requisitos não funcionais – identificados como alvos de teste.

## **2.1 Teste do Banco de Dados e Integridade**

- Verifique que artistas não podem ser cadastrados com nomes duplicados (Unique Constraint).
- Verifique que usuários não podem ser cadastrados com e-mails duplicados.
- Verifique que não é possível excluir um artista que possua álbuns vinculados (Restrição de Integridade).
- Verifique que ao excluir um álbum, as faixas associadas são excluídas em cascata (se aplicável) ou bloqueadas.
- Verifique a persistência correta dos dados de Artistas, Álbuns e Faixas.

## **2.2 Teste Funcional (Regras de Negócio)**

- Verifique que a duração de uma faixa deve ser um número positivo (maior que zero).
- Verifique que o sistema retorna erro 400 ao tentar cadastrar dados incompletos.
- Verifique que o sistema consome corretamente a API externa de letras (Lyrics Service).
- Verifique que o sistema trata adequadamente a indisponibilidade da API externa (Fallback/Timeout).

## **2.3 Teste do circlos de negócios (Fluxos)**

- Fluxo Completo: Criar Artista → Criar Álbum para este Artista → Criar Faixa para este Álbum → Consultar Faixa.

## **2.4 Perfil da Performance**

- Verifique se o tempo de resposta para listagem de artistas é inferior a 500ms em ambiente local.
- Verifique o comportamento do sistema sob carga leve e moderada.

## **2.5 Teste de Carga**

- Verificar a resposta do sistema com 10 usuários simultâneos.
- Verificar a resposta do sistema com 50 usuários simultâneos.
- Verificar a resposta do sistema com 100 usuários simultâneos.

## 2.6 Teste de Segurança e Controle de Acesso

- Verificar que usuários não autenticados não podem criar Artistas, Álbuns ou Faixas (Erro 401).
- Verificar que usuários não autenticados conseguem listar Artistas (Acesso Público).
- Verificar que o login com credenciais inválidas é rejeitado.
- Verificar que o token JWT gerado é válido e aceito nas rotas protegidas.

## 3 Estratégia de Teste

### 3.1 Tipos de Teste

#### 3.1.1 Teste de Integridade de Dados e do Banco de Dados

<b>Objetivo do Teste:</b>	Garantir que os métodos de acesso via Prisma ORM funcionam e respeitam as constraints do PostgreSQL.
<b>Técnica:</b>	Invocar os endpoints da API (via Supertest) enviando dados que violem restrições únicas (ex: mesmo e-mail duas vezes) e restrições de chave estrangeira (ex: álbum para artista inexistente).
<b>Critério de Finalização:</b>	O sistema deve retornar os códigos de erro HTTP apropriados (409 Conflict, 404 Not Found) e não corromper o banco.

#### 3.1.2 Teste de Função (Integração)

<b>Objetivo do Teste:</b>	Garantir a funcionalidade apropriada dos endpoints, incluindo validação de entrada e integração com serviços externos.
<b>Técnica:</b>	Executar casos de teste automatizados (Jest) para cada rota (GET, POST, DELETE): <ul style="list-style-type: none"><li>• Payload Válido: Espera-se 200/201.</li><li>• Payload Inválido (Duração negativa): Espera-se 400.</li><li>• Mocking da API de Lyrics para simular sucesso e falha.</li></ul>
<b>Critério de Finalização:</b>	Todos os testes implementados em <code>api.test.ts</code> passam com sucesso (Green).

### 3.1.3 Teste de Performance e Carga

<b>Objetivo do Teste:</b>	Verificar tempos de resposta e estabilidade sob requisições concorrentes.
<b>Técnica:</b>	Utilizar loops de promessas (Promise.all) no Jest para disparar requisições simultâneas ao endpoint <code>/artists</code> . <ul style="list-style-type: none"><li>• Cenário 1: 10 requisições simultâneas.</li><li>• Cenário 2: 100 requisições simultâneas.</li></ul>
<b>Critério de Finalização:</b>	Tempo de resposta médio abaixo de 500ms e zero falhas de conexão.

### 3.1.4 Teste de Segurança

<b>Objetivo do Teste:</b>	Verificar autenticação e proteção de rotas.
<b>Técnica:</b>	<ul style="list-style-type: none"><li>• Tentar acessar POST <code>/artists</code> sem Header Authorization.</li><li>• Tentar acessar com Token inválido.</li><li>• Registrar usuário e logar para obter Token válido e repetir acesso.</li></ul>
<b>Critério de Finalização:</b>	Rotas protegidas retornam 401/403 para acessos não autorizados e 201/200 para autorizados.

## 3.2 Ferramentas

As seguintes ferramentas serão empregadas para esse projeto:

Ferramenta	Função
Jest	Runner de Testes e Asserções
Supertest	Cliente HTTP para testes de integração
Prisma Client	Acesso ao Banco de Dados e Setup/Teardown
Docker	Containerização do Banco de Dados (Postgres)
Node.js / TypeScript	Ambiente de Execução

## 4 Recursos

### 4.1 Sistema

A tabela seguinte expõe os recursos do sistema para o projeto de teste.

<b>Recursos do Sistema</b>
<b>Servidor de Banco de Dados</b>
– PostgreSQL 13 (via Docker Container)
<b>Ambiente de Execução</b>
– Node.js v18 ou superior
– NPM para gerenciamento de dependências
<b>Repositório de Código</b>
– Git / GitHub (Projeto S206)