

Plano de Implementação: Plataforma SaaS de Escola de Música

Fase 1: Planejamento Detalhado e Refinamento de Requisitos

Nesta fase inicial, o foco é solidificar a visão do produto e detalhar todos os requisitos necessários para o desenvolvimento. A clareza nesta etapa é crucial para evitar retrabalho e garantir que o produto final atenda às expectativas.

1.1. Detalhamento dos Requisitos Funcionais

Com base nas suas ideias, vamos refinar e detalhar as funcionalidades:

- **Plataforma SaaS (Software as a Service):** O sistema será acessível por meio de um navegador web, com os usuários (alunos e professores) pagando por assinaturas ou comissões, sem a necessidade de instalar software localmente.
- **Modelo Marketplace (iFood-like):** A plataforma conectará diretamente alunos a professores, oferecendo um ambiente para busca, agendamento, pagamento e comunicação.
- **Dashboards Personalizados:**
 - **Dashboard do Aluno:** Apresentará informações como próximas aulas, professores favoritos, histórico de aulas, progresso em determinados instrumentos, e um feed de novidades (novos professores, materiais, etc.).
 - **Dashboard do Professor:** Exibirá o calendário de aulas, ganhos (mensais, semanais), estatísticas de visualização do perfil, novas solicitações de aula, e ferramentas para gerenciar seu perfil e materiais.
- **Busca Avançada de Professores:** Os alunos poderão pesquisar professores utilizando filtros como:
 - Instrumento (violão, piano, canto, etc.)
 - Estilo Musical (rock, clássico, jazz, etc.)

- Nível (iniciante, intermediário, avançado)
- Preço (faixa de valor por hora/aula)
- Localização (para aulas presenciais, com mapa interativo)
- Disponibilidade de horário
- Avaliação (média de notas dadas por outros alunos)
- **Perfil do Professor (Página Pessoal):** Cada professor terá uma página personalizável que funcionará como seu portfólio digital, contendo:
 - **Informações Básicas:** Nome, foto, biografia.
 - **Mídia:** Links para vídeos (YouTube, Vimeo), áudios (SoundCloud), e redes sociais (Instagram, etc.).
 - **Materiais:** Seção para upload de materiais em PDF, que podem ser acessíveis para todos os visitantes ou apenas para alunos matriculados.
 - **Calendário de Disponibilidade:** Visualização clara dos horários livres para agendamento.
 - **Avaliações e Comentários:** Feedback de alunos anteriores.
- **Chat Integrado:** Um sistema de mensagens em tempo real para comunicação entre alunos e professores, permitindo tirar dúvidas, combinar detalhes das aulas e enviar arquivos.
- **Sistema de Pagamento:**
 - **Para Alunos:** Pagamento por aula avulsa ou pacotes de aulas, utilizando cartão de crédito, PIX, ou boleto.
 - **Para Professores:** Recebimento dos valores das aulas (com a comissão da plataforma descontada) e pagamento da assinatura do plano premium.
- **Área Administrativa:** Um painel de controle para os administradores da plataforma gerenciarem usuários, pagamentos, conteúdo e configurações gerais.

1.2. Pesquisa e Refinamento das Tecnologias

As tecnologias que você mencionou são excelentes para construir uma aplicação web moderna e escalável. Vamos detalhar cada uma:

- **Frontend (Interface do Usuário):**

- **React:** Uma biblioteca JavaScript robusta e popular para a construção de interfaces de usuário interativas. O uso de componentes reutilizáveis acelera o desenvolvimento.
- **TypeScript:** Adiciona tipagem estática ao JavaScript, o que ajuda a prevenir bugs e melhora a manutenibilidade do código, especialmente em projetos grandes.
- **Frameworks (Next.js):** O uso de Next.js (um framework React) é altamente recomendado, pois oferece renderização no lado do servidor (SSR) e geração de sites estáticos (SSG), o que melhora significativamente o SEO (otimização para mecanismos de busca) e a performance inicial da aplicação.
- **Backend (Lógica do Servidor):**
 - **Node.js:** Um ambiente de execução JavaScript que permite utilizar a mesma linguagem tanto no frontend quanto no backend. É conhecido por sua alta performance em operações de entrada e saída (I/O), ideal para aplicações em tempo real como o chat.
 - **TypeScript:** Assim como no frontend, o TypeScript no backend traz mais segurança e organização ao código.
 - **Frameworks (Express.js ou NestJS):** Express.js é um framework minimalista e flexível para Node.js, enquanto NestJS é um framework mais opinativo e estruturado, que utiliza TypeScript por padrão e organiza o código de forma modular, similar ao Angular. A escolha entre eles dependerá do nível de estrutura desejado para o projeto.
- **Banco de Dados:**
 - **Neon Database:** Uma excelente escolha. Neon é uma plataforma de banco de dados PostgreSQL serverless, o que significa que ele escala automaticamente a capacidade de computação e armazenamento conforme a demanda, e você paga apenas pelo que usa. Isso é ideal para um SaaS que pode ter picos de uso e crescimento rápido. A compatibilidade com PostgreSQL garante um banco de dados relacional poderoso e confiável.

1.3. Definição do Escopo do MVP (Produto Mínimo Viável)

Para lançar o produto mais rapidamente e começar a coletar feedback real dos usuários, é fundamental definir um MVP. O MVP conterá as funcionalidades essenciais para resolver o problema principal do seu público-alvo.

Funcionalidades do MVP:

- **Cadastro e Login:** Para alunos e professores.
- **Perfil do Professor:** Com informações básicas, preço e calendário de disponibilidade.
- **Busca Simples de Professores:** Com filtros por instrumento e localização.
- **Agendamento de Aulas:** Funcionalidade básica de agendamento.
- **Pagamento por Aula:** Integração com um gateway de pagamento para transações de aulas.
- **Dashboard do Aluno:** Com visualização das próximas aulas.
- **Dashboard do Professor:** Com visualização das próximas aulas e gerenciamento de agendamentos.

Funcionalidades a serem deixadas para fases futuras:

- Chat integrado
- Planos premium para professores
- Upload de materiais em PDF
- Busca avançada com mais filtros
- Dashboards com estatísticas detalhadas
- Avaliações e comentários

Esta abordagem permite que você valide a ideia central do negócio com um investimento inicial menor, e então, com base no feedback, priorize o desenvolvimento das demais funcionalidades.

Fase 2: Design Técnico e Arquitetura

Nesta fase, o foco é traduzir os requisitos funcionais e não funcionais em um design técnico robusto e escalável. Serão definidos os componentes do sistema, a arquitetura geral, os fluxos de interação do usuário e o modelo de dados.

2.1. Arquitetura do Sistema

Considerando a necessidade de escalabilidade, flexibilidade e a natureza de um SaaS, uma arquitetura de microsserviços é a mais indicada a longo prazo. No entanto, para um MVP, uma arquitetura monolítica bem estruturada pode ser mais rápida de implementar e evoluir para microsserviços posteriormente, se necessário. Para este plano, vamos considerar uma abordagem que permita essa transição.

Abordagem Híbrida (Monolítico Modular com Visão de Microsserviços):

Inicialmente, o backend pode ser construído como um serviço monolítico, mas com módulos bem definidos e independentes (domínios de negócio), como `AuthService`, `UserService`, `SchedulingService`, `PaymentService`, `MediaService`, `ChatService`. Isso facilita a separação futura em microsserviços dedicados.

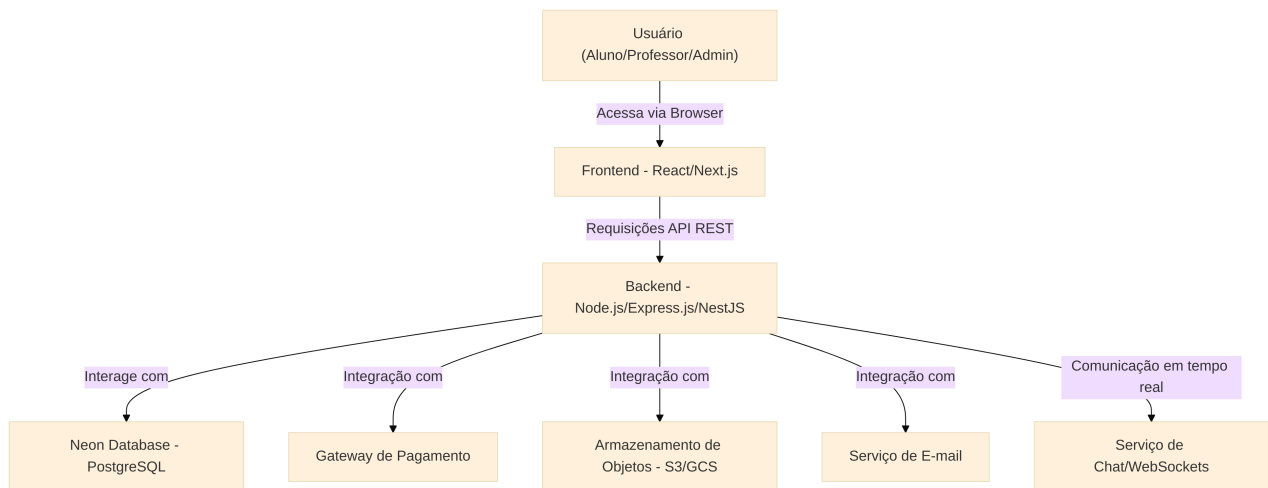
Componentes Principais:

- **Frontend (Web Application):**
 - Desenvolvido com React/Next.js e TypeScript.
 - Responsável pela interface do usuário para Alunos, Professores e Administradores.
 - Consome dados e funcionalidades via API RESTful do Backend.
 - Gerenciamento de estado global (ex: Redux, Zustand, React Context).
 - Componentes reutilizáveis e design system.
- **Backend (API Gateway / Services):**
 - Desenvolvido com Node.js/Express.js (ou NestJS) e TypeScript.
 - Atua como um API Gateway para o frontend, roteando requisições para os serviços internos.
 - Contém a lógica de negócio principal.

- Interage com o banco de dados (Neon Database) e serviços de terceiros (pagamento, armazenamento de mídia).
- **Módulos/Serviços Lógicos (para futura separação em microserviços):**
 - **Serviço de Autenticação e Usuários (AuthService , UserService):** Gerencia cadastro, login, perfis de usuário (aluno, professor, admin), permissões e roles.
 - **Serviço de Professores e Busca (ProfessorService , SearchService):** Gerencia perfis de professores, instrumentos, disponibilidade, e a lógica de busca e filtragem.
 - **Serviço de Agendamento (SchedulingService):** Gerencia a criação, confirmação, cancelamento e histórico de aulas.
 - **Serviço de Pagamentos (PaymentService):** Integração com gateways de pagamento, processamento de transações, gerenciamento de comissões e assinaturas premium.
 - **Serviço de Mídia (MediaService):** Gerencia o upload e armazenamento de vídeos e áudios dos professores.
 - **Serviço de Chat (chatService):** Gerencia a comunicação em tempo real entre alunos e professores.
 - **Serviço Administrativo (AdminService):** Funções de gerenciamento de usuários, conteúdo, relatórios e monitoramento para administradores.
- **Banco de Dados (Neon Database - PostgreSQL Serverless):**
 - Armazenamento primário de todos os dados da aplicação (usuários, perfis, aulas, pagamentos, etc.).
 - Benefícios: Escalabilidade automática, alta disponibilidade, separação de computação e armazenamento, pagamento por uso.
- **Serviços de Terceiros:**
 - **Gateway de Pagamento:** Stripe, PagSeguro, Mercado Pago (para processamento de transações financeiras).
 - **Armazenamento de Objetos:** AWS S3, Google Cloud Storage (para armazenar vídeos, áudios e PDFs).

- **Serviço de E-mail:** SendGrid, Mailgun (para notificações, confirmações de cadastro/aula).
- **Serviço de Chat (Opcional para MVP):** WebSockets implementados no backend ou integração com serviços como Pusher, Firebase Realtime Database para chat em tempo real.

Diagrama de Arquitetura (Conceitual):



2.2. Fluxos de Usuário Detalhados

Vamos detalhar os fluxos de usuário para cada perfil, com foco nas interações e transições entre as telas/funcionalidades.

2.2.1. Fluxo do Aluno

1. Acesso à Plataforma:

- Aluno acessa a URL da plataforma.
- Opção de Login ou Cadastro.

2. Cadastro de Aluno:

- Seleciona

o tipo de perfil 'Aluno'. - Preenche formulário (nome, email, senha). - Confirma email (se necessário). - Redirecionado para o Dashboard do Aluno. 3. **Login do Aluno:** - Insere email e senha. - Redirecionado para o Dashboard do Aluno. 4. **Dashboard do Aluno:** - Visualiza próximas aulas agendadas. - Acesso rápido à busca de professores. - Histórico de aulas. - Seção de

mensagens. 5. **Busca de Professores:** - Aluno acessa a página de busca. - Utiliza filtros (instrumento, estilo, preço, online/presencial, etc.). - Visualiza resultados em lista ou mapa. - Clica em um professor para ver o perfil detalhado. 6. **Perfil do Professor (Visualização Aluno):** - Visualiza biografia, experiência, instrumentos ensinados. - Assiste vídeos e ouve áudios de demonstração. - Verifica disponibilidade no calendário. - Vê avaliações e comentários de outros alunos. - Opção de agendar aula ou enviar mensagem. 7. **Agendamento de Aula:** - Aluno seleciona data e hora no calendário do professor. - Confirma detalhes da aula (duração, preço). - É redirecionado para a página de pagamento. 8. **Pagamento da Aula:** - Aluno escolhe método de pagamento (cartão, PIX). - Insere dados de pagamento. - Confirma transação. - Recebe confirmação de agendamento e pagamento. 9. **Comunicação (Chat):** - Aluno inicia chat com professor (após agendamento ou via perfil). - Troca mensagens em tempo real. - Recebe notificações de novas mensagens.

2.2.2. Fluxo do Professor

1. Acesso à Plataforma:

- Professor acessa a URL da plataforma.
- Opção de Login ou Cadastro.

2. Cadastro de Professor:

- Seleciona o tipo de perfil 'Professor'.
- Preenche formulário inicial (nome, email, senha).
- Preenche informações detalhadas do perfil (instrumentos, experiência, biografia, preços, disponibilidade, localização).
- Faz upload de vídeos/áudios de demonstração.
- Envia perfil para aprovação (se houver moderação inicial).
- Redirecionado para o Dashboard do Professor.

3. Login do Professor:

- Insere email e senha.
- Redirecionado para o Dashboard do Professor.

4. Dashboard do Professor:

- Visualiza próximas aulas agendadas.
- Ganhos recentes e estatísticas.

- Novas solicitações de agendamento.
- Acesso rápido ao gerenciamento de perfil e calendário.
- Seção de mensagens.

5. Gerenciamento de Perfil:

- Edita informações pessoais e profissionais.
- Atualiza instrumentos, estilos, preços.
- Gerencia disponibilidade no calendário.
- Adiciona/remove vídeos, áudios e links de redes sociais.
- Gerencia materiais em PDF (para alunos e não-alunos).

6. Gerenciamento de Aulas:

- Visualiza lista de aulas (pendentes, confirmadas, concluídas, canceladas).
- Aceita ou recusa solicitações de agendamento.
- Marca aulas como concluídas após a realização.
- Visualiza detalhes do aluno para cada aula.

7. Comunicação (Chat):

- Professor inicia chat com aluno (após agendamento ou via dashboard).
- Troca mensagens em tempo real.
- Recebe notificações de novas mensagens.

2.2.3. Fluxo do Administrador

1. Acesso à Plataforma:

- Administrador acessa a URL da área administrativa.
- Login com credenciais de administrador.

2. Dashboard do Administrador:

- Visão geral de métricas (usuários ativos, aulas agendadas, receita).
- Acesso rápido a módulos de gerenciamento.

3. Gerenciamento de Usuários:

- Busca por alunos ou professores.
- Visualiza, edita ou exclui perfis.

- Aprova/rejeita perfis de professores (se houver moderação).
- Reseta senhas, bloqueia usuários.

4. Gerenciamento de Pagamentos:

- Visualiza todas as transações (aulas, planos premium).
- Filtra por data, usuário, status.
- Gera relatórios financeiros.
- Gerencia repasses para professores.

5. Gerenciamento de Conteúdo:

- Modera vídeos, áudios e materiais de professores.
- Gerencia categorias de instrumentos, estilos musicais.
- Gerencia conteúdo da landing page.

6. Configurações do Sistema:

- Define comissões da plataforma.
- Gerencia planos premium.
- Configura integrações com gateways de pagamento.
- Gerencia notificações e emails automáticos.

7. Monitoramento e Logs:

- Acesso a logs de sistema e erros.
- Monitoramento de performance e uso de recursos.

2.3. Telas (Wireframes Conceituais)

Para cada fluxo de usuário, vamos descrever as telas principais e seus elementos, dando uma visão mais concreta da interface. Isso serve como um wireframe conceitual, que será refinado em uma fase de design de UI/UX.

2.3.1. Telas Comuns

- **Landing Page:**

- **Objetivo:** Atrair novos usuários e direcioná-los para cadastro/login.
- **Elementos:**
 - Cabeçalho: Logo, botões

de

Login/Cadastro, Navegação (Como Funciona, Professores, Contato). - Seção Hero: Título chamativo, subtítulo, CTA (Call to Action) para

cadastro. - Seções de Destaque: Benefícios para alunos e professores, depoimentos, professores em destaque. - Rodapé: Links úteis, redes sociais.

- **Página de Cadastro/Login:**

- **Objetivo:** Permitir que novos usuários se registrem e usuários existentes façam login.
- **Elementos:**
 - Opção para escolher entre

Cadastro de Aluno ou Cadastro de Professor. - Formulário de Cadastro: Campos para nome, email, senha, confirmação de senha. - Formulário de Login: Campos para email, senha. - Links:

Esqueceu a senha?, Criar conta. - Opções de Login Social (Google, Facebook - opcional).

2.3.2. Telas do Aluno

- **Dashboard do Aluno:**

- **Objetivo:** Visão geral e acesso rápido às funcionalidades principais para o aluno.
- **Elementos:**
 - Cabeçalho: Foto de perfil, nome do aluno, links para perfil, mensagens, configurações.
 - Seção

Próximas Aulas: Lista de aulas agendadas com data, hora, professor e link para a sala de aula virtual (se online). - Seção

Professores Favoritos: Lista de professores marcados como favoritos. - CTA para

Buscar Professor. - Histórico de Aulas: Link para a página de histórico.

- **Página de Busca de Professores:**

- **Objetivo:** Permitir que o aluno encontre professores.

- **Elementos:**

- Barra de Pesquisa: Campo de texto para nome do professor, instrumento.
- Filtros: Dropdowns/checkboxes para instrumento, estilo musical, nível, preço, online/presencial, localização.
- Resultados da Busca: Lista de cards de professores, cada um com foto, nome, instrumento, avaliação, preço por hora e um botão

Ver Perfil.

- **Página de Perfil do Professor (Visão do Aluno):**

- **Objetivo:** Apresentar informações detalhadas sobre o professor para o aluno.

- **Elementos:**

- Foto de Perfil Grande, Nome do Professor, Instrumentos Ensinados, Avaliação Média.
- Seções: Biografia, Experiência, Metodologia.
- Galeria de Mídia: Vídeos (embedados do YouTube/Vimeo), Áudios (embedados do SoundCloud).
- Disponibilidade: Calendário interativo com horários disponíveis para agendamento.
- Preços: Tabela de preços por hora/pacote.
- Avaliações: Lista de comentários e notas de outros alunos.
- Botões: Agendar Aula, Enviar Mensagem.

- **Página de Agendamento de Aula:**

- **Objetivo:** Permitir que o aluno selecione e confirme os detalhes da aula.

- **Elementos:**

- Resumo do Professor: Nome, foto, instrumento.
- Seleção de Data e Hora: Calendário e slots de horário disponíveis.
- Duração da Aula: Dropdown para selecionar duração (ex: 30min, 60min, 90min).
- Tipo de Aula: Opção Online/Presencial.

- Resumo do Pedido: Preço total, data, hora, professor.
- Botão: Ir para Pagamento.
- **Página de Pagamento:**
 - **Objetivo:** Coletar informações de pagamento e processar a transação.
 - **Elementos:**
 - Resumo do Pedido: Detalhes da aula e valor total.
 - Opções de Pagamento: Cartão de Crédito, PIX, Boleto (dependendo da integração do gateway).
 - Formulário de Cartão de Crédito: Número do cartão, nome no cartão, validade, CVV.
 - Instruções para PIX/Boleto.
 - Botão: Confirmar Pagamento.
 - Mensagem de Sucesso/Erro.
- **Página de Mensagens (Chat):**
 - **Objetivo:** Facilitar a comunicação em tempo real entre aluno e professor.
 - **Elementos:**
 - Lista de Conversas: Exibe as conversas ativas com professores.
 - Área de Chat: Exibe histórico de mensagens.
 - Campo de Entrada de Texto: Para digitar novas mensagens.
 - Botão: Enviar.
 - Indicador de Mensagem Lida/Não Lida.

2.3.3. Telas do Professor

- **Dashboard do Professor:**
 - **Objetivo:** Visão geral e acesso rápido às funcionalidades principais para o professor.
 - **Elementos:**
 - Cabeçalho: Foto de perfil, nome do professor, links para perfil, mensagens, configurações.

- Seção Próximas Aulas: Lista de aulas agendadas com data, hora, aluno.
- Seção Ganhos: Resumo dos ganhos recentes (ex:

últimos 30 dias). - Seção Solicitações de Agendamento: Lista de novas solicitações de aula para aprovação/rejeição. - CTA para Gerenciar Perfil, Gerenciar Disponibilidade. - Seção de Mensagens.

- **Página de Gerenciamento de Perfil do Professor:**

- **Objetivo:** Permitir que o professor edite suas informações e personalize seu perfil público.
- **Elementos:**
 - Formulários para editar: Informações pessoais, biografia, experiência, metodologia.
 - Seção de Instrumentos: Adicionar/remover instrumentos, definir nível de proficiência.
 - Seção de Preços: Definir preço por hora, pacotes de aulas.
 - Seção de Localização: Endereço para aulas presenciais, opção para aulas online.
 - Upload de Mídia: Botões para upload de vídeos (links externos) e áudios (links externos).
 - Links para Redes Sociais.
 - Botão: Salvar Alterações.

- **Página de Gerenciamento de Disponibilidade (Calendário do Professor):**

- **Objetivo:** Permitir que o professor defina seus horários disponíveis para aulas.
- **Elementos:**
 - Calendário Mensal/Semanal: Exibe slots de tempo.
 - Opções para Bloquear/Desbloquear Horários: Por dia, por hora, por período.
 - Definição de Recorrência: Para horários fixos semanais.
 - Visualização de Aulas Agendadas: Slots ocupados por aulas confirmadas.

- Botão: Salvar Disponibilidade.
- **Página de Gerenciamento de Aulas (Professor):**
 - **Objetivo:** Visualizar e gerenciar todas as aulas.
 - **Elementos:**
 - Abas/Filtros: Próximas, Pendentes, Concluídas, Canceladas.
 - Lista de Aulas: Cada item com data, hora, nome do aluno, status.
 - Detalhes da Aula: Ao clicar, exibe informações completas do aluno, link para chat, opção de aceitar/recusar (para pendentes), marcar como concluída.
- **Página de Mensagens (Chat):**
 - **Objetivo:** Facilitar a comunicação em tempo real entre professor e aluno.
 - **Elementos:** (Similar à tela de chat do aluno)
 - Lista de Conversas: Exibe as conversas ativas com alunos.
 - Área de Chat: Exibe histórico de mensagens.
 - Campo de Entrada de Texto: Para digitar novas mensagens.
 - Botão: Enviar.
 - Indicador de Mensagem Lida/Não Lida.

2.3.4. Telas do Administrador

- **Dashboard do Administrador:**
 - **Objetivo:** Visão geral das métricas do sistema e acesso rápido às ferramentas de gerenciamento.
 - **Elementos:**
 - Cards de Métricas: Total de usuários (alunos/professores), aulas agendadas (mês/total), receita gerada (mês/total), professores premium.
 - Gráficos: Tendências de cadastro, aulas, receita.
 - Links Rápidos: Gerenciar Usuários, Gerenciar Pagamentos, Gerenciar Conteúdo, Configurações.
- **Página de Gerenciamento de Usuários:**

- **Objetivo:** Visualizar, editar e gerenciar perfis de alunos e professores.
- **Elementos:**
 - Tabela de Usuários: Lista de todos os usuários com filtros (aluno/professor, ativo/inativo, premium/não premium).
 - Campos de Busca: Por nome, email.
 - Ações por Usuário: Visualizar detalhes, editar, bloquear/desbloquear, excluir.
 - Detalhes do Usuário: Exibe todas as informações do perfil, histórico de aulas/pagamentos.
- **Página de Gerenciamento de Pagamentos:**
 - **Objetivo:** Monitorar e gerenciar todas as transações financeiras da plataforma.
 - **Elementos:**
 - Tabela de Transações: Lista de pagamentos de aulas e assinaturas premium.
 - Filtros: Por data, tipo de transação, status, aluno/professor.
 - Resumo Financeiro: Total de receita, comissões, repasses.
 - Geração de Relatórios: Opção para exportar dados (CSV, PDF).
- **Página de Gerenciamento de Conteúdo:**
 - **Objetivo:** Moderar e organizar o conteúdo da plataforma.
 - **Elementos:**
 - Moderação de Perfis de Professor: Lista de perfis pendentes de aprovação, com links para visualização e botões de aprovar/rejeitar.
 - Gerenciamento de Categorias: Adicionar/editar/remover instrumentos, estilos musicais.
 - Gerenciamento de Materiais: Visualizar e remover materiais em PDF ou links de mídia.
- **Página de Configurações do Sistema:**
 - **Objetivo:** Ajustar parâmetros globais da plataforma.
 - **Elementos:**

- Configuração de Comissão: Campo para definir a porcentagem da plataforma.
- Gerenciamento de Planos Premium: Adicionar/editar/remover planos, preços, benefícios.
- Configurações de Integração: Chaves de API para gateways de pagamento, serviços de email.
- Configurações de Notificações: Modelos de email, tipos de notificação.

2.4. Modelo de Dados Detalhado

O modelo de dados é a espinha dorsal do sistema, definindo como as informações serão armazenadas e relacionadas no banco de dados (Neon Database - PostgreSQL). Vamos expandir o esquema simplificado anterior com mais detalhes e considerações para o desenvolvimento.

2.4.1. Entidades e Atributos

- **Usuarios (Tabela Central de Autenticação e Perfil Básico)**
 - `id` (UUID/PK): Identificador único do usuário.
 - `email` (VARCHAR(255), UNIQUE, NOT NULL): Email do usuário, usado para login.
 - `senha_hash` (VARCHAR(255), NOT NULL): Hash da senha do usuário (nunca armazenar senhas em texto puro).
 - `tipo_perfil` (ENUM("aluno", "professor", "administrador"), NOT NULL): Define o tipo de usuário.
 - `nome_completo` (VARCHAR(255), NOT NULL): Nome completo do usuário.
 - `foto_perfil_url` (TEXT): URL para a foto de perfil do usuário.
 - `data_cadastro` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data e hora do cadastro.
 - `ultimo_login` (TIMESTAMP WITH TIME ZONE): Último acesso do usuário.
 - `ativo` (BOOLEAN, NOT NULL, DEFAULT TRUE): Indica se o usuário está ativo ou bloqueado.
- **Alunos (Informações Específicas de Alunos)**

- `id` (UUID/PK): Identificador único do aluno.
 - `usuario_id` (UUID/FK, UNIQUE, NOT NULL): Chave estrangeira para `Usuarios.id`.
 - `data_nascimento` (DATE): Data de nascimento do aluno.
 - `telefone` (VARCHAR(20)): Telefone de contato.
 - `endereco` (TEXT): Endereço completo (opcional, para aulas presenciais).
 - `preferencias_musicais` (JSONB): Preferências de estilo musical, instrumentos de interesse.
- **Professores (Informações Específicas de Professores)**
 - `id` (UUID/PK): Identificador único do professor.
 - `usuario_id` (UUID/FK, UNIQUE, NOT NULL): Chave estrangeira para `Usuarios.id`.
 - `biografia` (TEXT): Descrição detalhada do professor.
 - `experiencia` (TEXT): Experiência profissional e acadêmica.
 - `metodologia` (TEXT): Abordagem de ensino.
 - `localizacao_presencial` (TEXT): Endereço para aulas presenciais.
 - `online_disponivel` (BOOLEAN, NOT NULL, DEFAULT FALSE): Indica se oferece aulas online.
 - `preco_hora_base` (NUMERIC(10, 2)): Preço base por hora de aula.
 - `status_aprovacao` (ENUM("pendente", "aprovado", "rejeitado"), NOT NULL, DEFAULT "pendente"): Status de moderação do perfil.
 - `media_avaliacao` (NUMERIC(2, 1)): Média das avaliações recebidas (calculado).
 - `total_avaliacoes` (INTEGER): Número total de avaliações (calculado).
 - `plano_premium_id` (UUID/FK): Chave estrangeira para `PlanosPremium.id` (se for premium).
 - `link_youtube` (TEXT): Link para canal do YouTube.
 - `link_instagram` (TEXT): Link para perfil do Instagram.
 - `link_soundcloud` (TEXT): Link para perfil do SoundCloud.

- **Instrumentos (Lista de Instrumentos Musicais)**
 - `id` (UUID/PK): Identificador único do instrumento.
 - `nome` (VARCHAR(100), UNIQUE, NOT NULL): Nome do instrumento (ex: "Violão", "Piano").
- **ProfessorInstrumentos (Associação Professor-Instrumento)**
 - `professor_id` (UUID/FK, PK, NOT NULL): Chave estrangeira para `Professores.id`.
 - `instrumento_id` (UUID/FK, PK, NOT NULL): Chave estrangeira para `Instrumentos.id`.
 - `nivel_proficiencia` (ENUM("iniciante", "intermediario", "avancado"), NOT NULL): Nível que o professor ensina para este instrumento.
- **DisponibilidadeProfessor (Slots de Horário Disponíveis)**
 - `id` (UUID/PK): Identificador único do slot de disponibilidade.
 - `professor_id` (UUID/FK, NOT NULL): Chave estrangeira para `Professores.id`.
 - `dia_semana` (INTEGER, NOT NULL): Dia da semana (0=Domingo, 6=Sábado).
 - `hora_inicio` (TIME, NOT NULL): Hora de início do slot.
 - `hora_fim` (TIME, NOT NULL): Hora de fim do slot.
 - `recorrente` (BOOLEAN, NOT NULL, DEFAULT TRUE): Indica se o slot se repete semanalmente.
 - `data_excecao` (DATE): Para slots não recorrentes ou exceções (ex: feriados).
- **Aulas (Informações sobre Aulas Agendadas)**
 - `id` (UUID/PK): Identificador único da aula.
 - `aluno_id` (UUID/FK, NOT NULL): Chave estrangeira para `Alunos.id`.
 - `professor_id` (UUID/FK, NOT NULL): Chave estrangeira para `Professores.id`.
 - `data_hora_inicio` (TIMESTAMP WITH TIME ZONE, NOT NULL): Data e hora de início da aula.

- `data_hora_fim` (TIMESTAMP WITH TIME ZONE, NOT NULL): Data e hora de fim da aula.
 - `duracao_minutos` (INTEGER, NOT NULL): Duração da aula em minutos.
 - `preco_total` (NUMERIC(10, 2), NOT NULL): Preço total da aula.
 - `status_aula` (ENUM("agendada", "confirmada", "cancelada", "concluida"), NOT NULL, DEFAULT "agendada"): Status atual da aula.
 - `tipo_aula` (ENUM("online", "presencial"), NOT NULL): Tipo de aula.
 - `link_aula_online` (TEXT): URL para a sala de aula virtual (para aulas online).
 - `data_agendamento` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data e hora do agendamento.
- **Pagamentos (Registro de Todas as Transações Financeiras)**
 - `id` (UUID/PK): Identificador único do pagamento.
 - `aula_id` (UUID/FK): Chave estrangeira para `Aulas.id` (se for pagamento de aula).
 - `assinatura_id` (UUID/FK): Chave estrangeira para `AssinaturasPremium.id` (se for pagamento de assinatura).
 - `usuario_pagador_id` (UUID/FK, NOT NULL): Chave estrangeira para `Usuarios.id` (quem pagou).
 - `usuario_recebedor_id` (UUID/FK, NOT NULL): Chave estrangeira para `Usuarios.id` (quem recebeu - professor ou plataforma).
 - `valor` (NUMERIC(10, 2), NOT NULL): Valor da transação.
 - `comissao_plataforma` (NUMERIC(10, 2)): Valor da comissão retida pela plataforma.
 - `data_pagamento` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data e hora do pagamento.
 - `status_pagamento` (ENUM("pendente", "aprovado", "rejeitado", "estornado"), NOT NULL): Status do pagamento.
 - `metodo_pagamento` (VARCHAR(50)): Método utilizado (ex: "cartao_credito", "pix", "boleto").

- `transacao_gateway_id` (VARCHAR(255)): ID da transação no gateway de pagamento.
- **Avaliacoes (Feedback de Alunos para Professores)**
 - `id` (UUID/PK): Identificador único da avaliação.
 - `aula_id` (UUID/FK, UNIQUE, NOT NULL): Chave estrangeira para `Aulas.id` (uma avaliação por aula).
 - `aluno_id` (UUID/FK, NOT NULL): Chave estrangeira para `Alunos.id`.
 - `professor_id` (UUID/FK, NOT NULL): Chave estrangeira para `Professores.id`.
 - `nota` (INTEGER, NOT NULL): Nota de 1 a 5.
 - `comentario` (TEXT): Comentário do aluno.
 - `data_avaliacao` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data e hora da avaliação.
- **PlanosPremium (Configuração dos Planos de Assinatura)**
 - `id` (UUID/PK): Identificador único do plano.
 - `nome_plano` (VARCHAR(100), UNIQUE, NOT NULL): Nome do plano (ex: "Básico", "Pro", "Premium").
 - `descricao` (TEXT): Descrição dos benefícios do plano.
 - `preco_mensal` (NUMERIC(10, 2), NOT NULL): Preço mensal do plano.
 - `recursos_adicionais` (JSONB): Detalhes dos recursos (ex: {"visibilidade": "alta", "suporte": "prioritario"}).
 - `ativo` (BOOLEAN, NOT NULL, DEFAULT TRUE): Indica se o plano está ativo para contratação.
- **AssinaturasPremium (Assinaturas Ativas de Professores)**
 - `id` (UUID/PK): Identificador único da assinatura.
 - `professor_id` (UUID/FK, NOT NULL): Chave estrangeira para `Professores.id`.

- `plano_id` (UUID/FK, NOT NULL): Chave estrangeira para `PlanosPremium.id`.
 - `data_inicio` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data de início da assinatura.
 - `data_fim` (TIMESTAMP WITH TIME ZONE): Data de término da assinatura (para planos com duração definida).
 - `status_assinatura` (ENUM("ativa", "cancelada", "expirada", "pendente_pagamento"), NOT NULL): Status da assinatura.
 - `proximo_ciclo_pagamento` (DATE): Data do próximo pagamento recorrente.
- **MateriaisProfessor (Materiais em PDF/Links Compartilhados)**
 - `id` (UUID/PK): Identificador único do material.
 - `professor_id` (UUID/FK, NOT NULL): Chave estrangeira para `Professores.id`.
 - `titulo` (VARCHAR(255), NOT NULL): Título do material.
 - `descricao` (TEXT): Descrição do material.
 - `url_arquivo` (TEXT): URL para o arquivo PDF ou outro recurso.
 - `tipo_acesso` (ENUM("publico", "apenas_alunos"), NOT NULL): Define quem pode acessar o material.
 - `data_upload` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data de upload.
 - **Mensagens (Chat Integrado)**
 - `id` (UUID/PK): Identificador único da mensagem.
 - `remetente_id` (UUID/FK, NOT NULL): Chave estrangeira para `Usuarios.id` (quem enviou).
 - `destinatario_id` (UUID/FK, NOT NULL): Chave estrangeira para `Usuarios.id` (quem recebeu).
 - `aula_id` (UUID/FK): Opcional, para associar a mensagem a uma aula específica.

- `conteudo` (TEXT, NOT NULL): Conteúdo da mensagem.
- `data_envio` (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT NOW()): Data e hora do envio.
- `lida` (BOOLEAN, NOT NULL, DEFAULT FALSE): Indica se a mensagem foi lida pelo destinatário.

2.4.2. Relacionamentos

- `Usuarios` 1:1 `Alunos`
- `Usuarios` 1:1 `Professores`
- `Professores` M:M `Instrumentos` (via `ProfessorInstrumentos`)
- `Alunos` M:1 `Aulas`
- `Professores` M:1 `Aulas`
- `Aulas` 1:1 `Pagamentos` (para pagamento da aula)
- `Aulas` 1:1 `Avaliacoes`
- `Professores` 1:M `DisponibilidadeProfessor`
- `Professores` 1:M `MateriaisProfessor`
- `Professores` 1:M `AssinaturasPremium`
- `PlanosPremium` 1:M `AssinaturasPremium`
- `Usuarios` M:M `Mensagens` (cada mensagem tem um remetente e um destinatário)

2.5. Especificação das APIs

As APIs serão o ponto de comunicação entre o frontend e o backend, seguindo o padrão RESTful. Serão definidas as rotas, métodos HTTP, payloads de requisição e respostas esperadas.

2.5.1. API de Autenticação (`/api/auth`)

- **POST** `/api/auth/register`
 - **Descrição:** Registra um novo usuário (aluno ou professor).
 - **Request Body:** `email`, `senha`, `tipo_perfil`, `nome_completo`.

- **Response:** token , usuario (dados básicos).
- **POST /api/auth/login**
 - **Descrição:** Autentica um usuário existente.
 - **Request Body:** email , senha .
 - **Response:** token , usuario (dados básicos).
- **POST /api/auth/refresh-token**
 - **Descrição:** Renova o token de acesso.
 - **Request Body:** refresh_token .
 - **Response:** token .

2.5.2. API de Usuários (/api/users)

- **GET /api/users/me**
 - **Descrição:** Retorna os dados do usuário autenticado.
 - **Response:** usuario (dados completos do aluno/professor/admin).
- **PUT /api/users/me**
 - **Descrição:** Atualiza os dados do usuário autenticado.
 - **Request Body:** Campos a serem atualizados (ex: nome_completo , foto_perfil_url).
 - **Response:** usuario (dados atualizados).

2.5.3. API de Professores (/api/professors)

- **GET /api/professors**
 - **Descrição:** Lista professores com filtros e paginação.
 - **Query Params:** instrumento , estilo , nivel , preco_min , preco_max , online , presencial , localizacao , page , limit , sort_by .
 - **Response:** lista_professores , total_paginas , pagina_atual .
- **GET /api/professors/:id**
 - **Descrição:** Retorna detalhes de um professor específico.
 - **Response:** professor (dados completos, incluindo mídias, avaliações).

- **PUT /api/professors/me**
 - **Descrição:** Atualiza o perfil do professor autenticado.
 - **Request Body:** biografia, experiencia, metodologia, localizacao_presencial, online_disponivel, preco_hora_base, link_youtube, link_instagram, link_soundcloud, instrumentos (array de objetos com instrumento_id, nivel_proficiencia).
 - **Response:** professor (dados atualizados).
- **POST /api/professors/me/materials**
 - **Descrição:** Adiciona um novo material ao perfil do professor.
 - **Request Body:** titulo, descricao, url_arquivo, tipo_acesso.
 - **Response:** material.
- **DELETE /api/professors/me/materials/:id**
 - **Descrição:** Remove um material do perfil do professor.
 - **Response:** Mensagem de sucesso.

2.5.4. API de Agendamento (/api/schedule)

- **GET /api/schedule/professors/:professorId/availability**
 - **Descrição:** Retorna a disponibilidade de um professor para agendamento.
 - **Query Params:** data_inicio, data_fim.
 - **Response:** slots_disponiveis.
- **POST /api/schedule/book**
 - **Descrição:** Cria um novo agendamento de aula.
 - **Request Body:** professor_id, data_hora_inicio, duracao_minutos, tipo_aula.
 - **Response:** aula (dados da aula agendada).
- **GET /api/schedule/my-lessons**
 - **Descrição:** Retorna as aulas do usuário autenticado (aluno ou professor).
 - **Query Params:** status, tipo_perfil.
 - **Response:** lista_aulas.

- **PUT /api/schedule/lessons/:id/status**
 - **Descrição:** Atualiza o status de uma aula (confirmar, cancelar, concluir).
 - **Request Body:** `status`.
 - **Response:** `aula` (dados atualizados).

2.5.5. API de Pagamentos (/api/payments)

- **POST /api/payments/process-lesson-payment**
 - **Descrição:** Processa o pagamento de uma aula.
 - **Request Body:** `aula_id`, `metodo_pagamento`, `token_cartao` (se cartão), `pix_data` (se PIX).
 - **Response:** `pagamento` (status da transação).
- **POST /api/payments/process-premium-subscription**
 - **Descrição:** Processa o pagamento de uma assinatura premium.
 - **Request Body:** `plano_id`, `metodo_pagamento`, `token_cartao`.
 - **Response:** `assinatura` (status da assinatura).
- **GET /api/payments/my-transactions**
 - **Descrição:** Retorna o histórico de transações do usuário autenticado.
 - **Response:** `lista_pagamentos`.

2.5.6. API de Avaliações (/api/reviews)

- **POST /api/reviews**
 - **Descrição:** Cria uma nova avaliação para uma aula concluída.
 - **Request Body:** `aula_id`, `nota`, `comentario`.
 - **Response:** `avaliacao`.
- **GET /api/reviews/professors/:professorId**
 - **Descrição:** Retorna as avaliações de um professor específico.
 - **Response:** `lista_avaliacoes`.

2.5.7. API de Planos Premium (/api/premium)

- **GET /api/premium/plans**

- **Descrição:** Lista todos os planos premium disponíveis.
- **Response:** `lista_planos`.
- **GET /api/premium/my-subscription**
 - **Descrição:** Retorna o status da assinatura premium do professor autenticado.
 - **Response:** `assinatura`.

2.5.8. API de Chat (/api/chat)

- **GET /api/chat/conversations**
 - **Descrição:** Lista as conversas do usuário autenticado.
 - **Response:** `lista_conversas` (com últimos mensagens).
- **GET /api/chat/conversations/:conversationId/messages**
 - **Descrição:** Retorna as mensagens de uma conversa específica.
 - **Response:** `lista_mensagens`.
- **POST /api/chat/messages**
 - **Descrição:** Envia uma nova mensagem.
 - **Request Body:** `destinatario_id`, `conteudo`, `aula_id` (opcional).
 - **Response:** `mensagem`.
- **WebSockets:** Para comunicação em tempo real, será implementado um endpoint WebSocket (ex: `ws://api.seusistema.com/chat`) para envio e recebimento instantâneo de mensagens.

2.5.9. API Administrativa (/api/admin)

- **GET /api/admin/users**
 - **Descrição:** Lista todos os usuários (alunos e professores) para gerenciamento.
 - **Query Params:** `tipo_perfil`, `status`, `search`.
 - **Response:** `lista_usuarios`.
- **PUT /api/admin/users/:id/status**

- **Descrição:** Atualiza o status de um usuário (ativo/inativo, aprovação de professor).
- **Request Body:** `status`, `motivo`.
- **Response:** `usuario` (atualizado).
- **GET /api/admin/payments**
 - **Descrição:** Lista todas as transações financeiras.
 - **Query Params:** `data_inicio`, `data_fim`, `tipo_transacao`, `status`.
 - **Response:** `lista_pagamentos`, `resumo_financeiro`.
- **PUT /api/admin/settings/commission**
 - **Descrição:** Define a comissão da plataforma.
 - **Request Body:** `porcentagem`.
 - **Response:** `configuracao`.

Fase 3: Desenvolvimento (Frontend e Backend)

Nesta fase, o foco é a implementação prática do sistema, traduzindo o design técnico em código funcional. O desenvolvimento será dividido em configuração do ambiente, desenvolvimento do backend e desenvolvimento do frontend, com integração contínua entre os dois.

3.1. Configuração do Ambiente de Desenvolvimento

Antes de iniciar o desenvolvimento, é essencial configurar um ambiente consistente e produtivo para toda a equipe.

3.1.1. Ferramentas e Dependências

Ferramentas de Desenvolvimento: - **Node.js:** Versão LTS mais recente (recomendado 18.x ou superior). - **npm ou yarn:** Gerenciador de pacotes para JavaScript. - **Git:** Controle de versão. - **Visual Studio Code:** IDE recomendada com extensões para TypeScript, React, Node.js. - **Docker:** Para containerização e padronização do ambiente (opcional, mas recomendado). - **Postman ou Insomnia:** Para testes de API.

Configuração do Banco de Dados: - **Neon Database:** Criar uma conta no Neon e configurar um banco PostgreSQL. - **Prisma ou TypeORM:** ORM (Object-Relational Mapping) para facilitar as operações com o banco de dados. - **Migrations:** Scripts para criação e atualização da estrutura do banco.

3.1.2. Estrutura de Pastas do Projeto

Uma estrutura bem organizada facilita a manutenção e colaboração:

```
music-school-platform/
├── backend/
│   ├── src/
│   │   ├── controllers/
│   │   ├── services/
│   │   ├── models/
│   │   ├── routes/
│   │   ├── middleware/
│   │   ├── utils/
│   │   └── config/
│   ├── prisma/ (ou migrations/)
│   ├── tests/
│   ├── package.json
│   └── tsconfig.json
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── hooks/
│   │   ├── services/
│   │   ├── utils/
│   │   ├── types/
│   │   └── styles/
│   ├── public/
│   ├── package.json
│   └── tsconfig.json
├── shared/
│   └── types/ (tipos TypeScript compartilhados)
├── docs/
├── docker-compose.yml (opcional)
└── README.md
```

3.1.3. Configuração de Variáveis de Ambiente

Criar arquivos `.env` para configurações sensíveis:

Backend (.env):

```
DATABASE_URL=postgresql://username:password@host:port/database
JWT_SECRET=your-jwt-secret-key
STRIPE_SECRET_KEY=your-stripe-secret-key
AWS_ACCESS_KEY_ID=your-aws-access-key
AWS_SECRET_ACCESS_KEY=your-aws-secret-key
EMAIL_SERVICE_API_KEY=your-email-service-key
```

Frontend (.env):

```
REACT_APP_API_BASE_URL=http://localhost:3001/api
REACT_APP_STRIPE_PUBLISHABLE_KEY=your-stripe-publishable-key
```

3.2. Desenvolvimento do Backend

O backend será desenvolvido com Node.js, Express.js (ou NestJS) e TypeScript, seguindo princípios de arquitetura limpa e padrões de desenvolvimento.

3.2.1. Configuração Inicial do Backend

Inicialização do Projeto:

```
mkdir backend && cd backend
npm init -y
npm install express cors helmet morgan dotenv bcryptjs jsonwebtoken
npm install -D @types/node @types/express @types/cors @types/bcryptjs
@types/jsonwebtoken typescript ts-node nodemon
```

Configuração do TypeScript (tsconfig.json):

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "lib": ["ES2020"],
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true,
    "declaration": true,
    "declarationMap": true,
    "sourceMap": true
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules", "dist"]
}
```

Configuração do Prisma (ORM):

```
npm install prisma @prisma/client  
npx prisma init
```

3.2.2. Estrutura de Código do Backend

Arquivo Principal (src/app.ts):

```

import express from 'express';
import cors from 'cors';
import helmet from 'helmet';
import morgan from 'morgan';
import dotenv from 'dotenv';

import authRoutes from './routes/auth';
import userRoutes from './routes/users';
import professorRoutes from './routes/professors';
import scheduleRoutes from './routes/schedule';
import paymentRoutes from './routes/payments';
import reviewRoutes from './routes/reviews';
import chatRoutes from './routes/chat';
import adminRoutes from './routes/admin';

import { errorHandler } from './middleware/errorHandler';
import { authenticateToken } from './middleware/auth';

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3001;

// Middlewares
app.use(helmet());
app.use(cors());
app.use(morgan('combined'));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/users', authenticateToken, userRoutes);
app.use('/api/professors', professorRoutes);
app.use('/api/schedule', authenticateToken, scheduleRoutes);
app.use('/api/payments', authenticateToken, paymentRoutes);
app.use('/api/reviews', authenticateToken, reviewRoutes);
app.use('/api/chat', authenticateToken, chatRoutes);
app.use('/api/admin', authenticateToken, adminRoutes);

// Error handling middleware
app.use(errorHandler);

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

export default app;

```

Middleware de Autenticação (src/middleware/auth.ts):


```

import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

interface AuthenticatedRequest extends Request {
  user?: {
    id: string;
    email: string;
    tipo_perfil: string;
  };
}

export const authenticateToken = async (
  req: AuthenticatedRequest,
  res: Response,
  next: NextFunction
) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Token de acesso requerido' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET!) as any;
    const user = await prisma.usuarios.findUnique({
      where: { id: decoded.userId }
    });

    if (!user || !user.ativo) {
      return res.status(403).json({ error: 'Usuário inválido ou inativo' });
    }

    req.user = {
      id: user.id,
      email: user.email,
      tipo_perfil: user.tipo_perfil
    };

    next();
  } catch (error) {
    return res.status(403).json({ error: 'Token inválido' });
  }
};

```

Controller de Autenticação (src/controllers/authController.ts):

```

import { Request, Response } from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { PrismaClient } from '@prisma/client';
import { z } from 'zod';

const prisma = new PrismaClient();

const registerSchema = z.object({
  email: z.string().email(),
  senha: z.string().min(6),
  tipo_perfil: z.enum(['aluno', 'professor']),
  nome_completo: z.string().min(2)
});

const loginSchema = z.object({
  email: z.string().email(),
  senha: z.string()
});

export const register = async (req: Request, res: Response) => {
  try {
    const { email, senha, tipo_perfil, nome_completo } =
      registerSchema.parse(req.body);

    // Verificar se o usuário já existe
    const existingUser = await prisma.usuarios.findUnique({
      where: { email }
    });

    if (existingUser) {
      return res.status(400).json({ error: 'Email já cadastrado' });
    }

    // Hash da senha
    const senhaHash = await bcrypt.hash(senha, 12);

    // Criar usuário
    const usuario = await prisma.usuarios.create({
      data: {
        email,
        senha_hash: senhaHash,
        tipo_perfil,
        nome_completo
      }
    });

    // Criar perfil específico (aluno ou professor)
    if (tipo_perfil === 'aluno') {
      await prisma.alunos.create({
        data: {
          usuario_id: usuario.id
        }
      });
    } else if (tipo_perfil === 'professor') {
      await prisma.professores.create({
        data: {
          usuario_id: usuario.id
        }
      });
    }
  }
}

```

```

// Gerar token JWT
const token = jwt.sign(
  { userId: usuario.id, email: usuario.email },
  process.env.JWT_SECRET!,
  { expiresIn: '7d' }
);

res.status(201).json({
  token,
  usuario: {
    id: usuario.id,
    email: usuario.email,
    nome_completo: usuario.nome_completo,
    tipo_perfil: usuario.tipo_perfil
  }
});
} catch (error) {
  if (error instanceof z.ZodError) {
    return res.status(400).json({ error: 'Dados inválidos', details:
error.errors });
  }
  console.error('Erro no registro:', error);
  res.status(500).json({ error: 'Erro interno do servidor' });
}
};

export const login = async (req: Request, res: Response) => {
  try {
    const { email, senha } = loginSchema.parse(req.body);

    // Buscar usuário
    const usuario = await prisma.usuarios.findUnique({
      where: { email }
    });

    if (!usuario || !usuario.ativo) {
      return res.status(401).json({ error: 'Credenciais inválidas' });
    }

    // Verificar senha
    const senhaValida = await bcrypt.compare(senha, usuario.senha_hash);
    if (!senhaValida) {
      return res.status(401).json({ error: 'Credenciais inválidas' });
    }

    // Atualizar último login
    await prisma.usuarios.update({
      where: { id: usuario.id },
      data: { ultimo_login: new Date() }
    });

    // Gerar token JWT
    const token = jwt.sign(
      { userId: usuario.id, email: usuario.email },
      process.env.JWT_SECRET!,
      { expiresIn: '7d' }
    );

    res.json({
      token,
      usuario: {

```

```
        id: usuario.id,
        email: usuario.email,
        nome_completo: usuario.nome_completo,
        tipo_perfil: usuario.tipo_perfil
    }
    });
} catch (error) {
    if (error instanceof z.ZodError) {
        return res.status(400).json({ error: 'Dados inválidos', details:
error.errors });
    }
    console.error('Erro no login:', error);
    res.status(500).json({ error: 'Erro interno do servidor' });
}
};
```

Service de Professores (src/services/professorService.ts):

```

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

interface ProfessorFilters {
  instrumento?: string;
  estilo?: string;
  nivel?: string;
  preco_min?: number;
  preco_max?: number;
  online?: boolean;
  presencial?: boolean;
  localizacao?: string;
  page?: number;
  limit?: number;
  sort_by?: 'preco' | 'avaliacao' | 'relevancia';
}

export class ProfessorService {
  static async searchProfessors(filters: ProfessorFilters) {
    const {
      instrumento,
      estilo,
      nivel,
      preco_min,
      preco_max,
      online,
      presencial,
      localizacao,
      page = 1,
      limit = 20,
      sort_by = 'relevancia'
    } = filters;

    const skip = (page - 1) * limit;

    // Construir filtros dinâmicos
    const whereClause: any = {
      status_aprovacao: 'aprovado',
      usuario: {
        ativo: true
      }
    };

    if (preco_min || preco_max) {
      whereClause.preco_hora_base = {};
      if (preco_min) whereClause.preco_hora_base.gte = preco_min;
      if (preco_max) whereClause.preco_hora_base.lte = preco_max;
    }

    if (online !== undefined) {
      whereClause.online_disponivel = online;
    }

    if (localizacao && presencial) {
      whereClause.localizacao_presencial = {
        contains: localizacao,
        mode: 'insensitive'
      };
    }
  }
}

```

```

if (instrumento) {
  whereClause.professor_instrumentos = {
    some: {
      instrumento: {
        nome: {
          contains: instrumento,
          mode: 'insensitive'
        }
      }
    }
  };
}

if (nivel) {
  whereClause.professor_instrumentos = {
    some: {
      nivel_proficiencia: nivel
    }
  };
}

// Definir ordenação
let orderBy: any = {};
switch (sort_by) {
  case 'preco':
    orderBy = { preco_hora_base: 'asc' };
    break;
  case 'avaliacao':
    orderBy = { media_avaliacao: 'desc' };
    break;
  default:
    orderBy = { total_avaliacoes: 'desc' };
}

const [professores, total] = await Promise.all([
  prisma.professores.findMany({
    where: whereClause,
    include: {
      usuario: {
        select: {
          id: true,
          nome_completo: true,
          foto_perfil_url: true
        }
      },
      professor_instrumentos: {
        include: {
          instrumento: true
        }
      },
      plano_premium: true
    },
    orderBy,
    skip,
    take: limit
  }),
  prisma.professores.count({ where: whereClause })
]);

return {
  professores: professores.map(professor => ({
    id: professor.id,

```

```

    nome: professor.usuario.nome_completo,
    foto_perfil: professor.usuario.foto_perfil_url,
    instrumentos: professor.professor_instrumentos.map(pi => ({
      nome: pi.instrumento.nome,
      nivel: pi.nivel_proficiencia
    })),
    preco_hora: professor.preco_hora_base,
    media_avaliacao: professor.media_avaliacao,
    total_avaliacoes: professor.total_avaliacoes,
    online_disponivel: professor.online_disponivel,
    premium: !!professor.plano_premium_id
  })),
  total_paginas: Math.ceil(total / limit),
  pagina_atual: page,
  total_professores: total
};
}

```

```

static async getProfessorById(id: string) {
  const professor = await prisma.professores.findUnique({
    where: { id },
    include: {
      usuario: {
        select: {
          id: true,
          nome_completo: true,
          foto_perfil_url: true
        }
      },
      professor_instrumentos: {
        include: {
          instrumento: true
        }
      },
      materiais_professor: {
        where: {
          tipo_acesso: 'publico'
        }
      },
      avaliacoes: {
        include: {
          aluno: {
            include: {
              usuario: {
                select: {
                  nome_completo: true,
                  foto_perfil_url: true
                }
              }
            }
          }
        }
      },
      orderBy: {
        data_avaliacao: 'desc'
      },
      take: 10
    }
  });

  if (!professor || professor.status_aprovacao !== 'aprovado') {
    return null;
  }
}

```

```

    }

    return {
      id: professor.id,
      nome: professor.usuario.nome_completo,
      foto_perfil: professor.usuario.foto_perfil_url,
      biografia: professor.biografia,
      experiencia: professor.experiencia,
      metodologia: professor.metodologia,
      instrumentos: professor.professor_instrumentos.map(pi => ({
        nome: pi.instrumento.nome,
        nivel: pi.nivel_proficiencia
      })),
      preco_hora: professor.preco_hora_base,
      media_avaliacao: professor.media_avaliacao,
      total_avaliacoes: professor.total_avaliacoes,
      online_disponivel: professor.online_disponivel,
      localizacao_presencial: professor.localizacao_presencial,
      links: {
        youtube: professor.link_youtube,
        instagram: professor.link_instagram,
        soundcloud: professor.link_soundcloud
      },
      materiais: professor.materiais_professor.map(material => ({
        id: material.id,
        titulo: material.titulo,
        descricao: material.descricao,
        url: material.url_arquivo
      })),
      avaliacoes: professor.avaliacoes.map(avaliacao => ({
        id: avaliacao.id,
        nota: avaliacao.nota,
        comentario: avaliacao.comentario,
        data: avaliacao.data_avaliacao,
        aluno_nome: avaliacao.aluno.usuario.nome_completo,
        aluno_foto: avaliacao.aluno.usuario.foto_perfil_url
      })))
    };
  }
}

```

3.2.3. Integração com Serviços de Terceiros

Gateway de Pagamento (Stripe):


```

import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2023-10-16'
});

export class PaymentService {
  static async processLessonPayment(aulaId: string, amount: number,
    paymentMethodId: string) {
    try {
      const paymentIntent = await stripe.paymentIntents.create({
        amount: Math.round(amount * 100), // Stripe usa centavos
        currency: 'brl',
        payment_method: paymentMethodId,
        confirm: true,
        return_url: `${process.env.FRONTEND_URL}/payment-success`
      });

      return {
        success: paymentIntent.status === 'succeeded',
        payment_intent_id: paymentIntent.id,
        status: paymentIntent.status
      };
    } catch (error) {
      console.error('Erro no pagamento:', error);
      return {
        success: false,
        error: error.message
      };
    }
  }

  static async createSubscription(customerId: string, priceId: string) {
    try {
      const subscription = await stripe.subscriptions.create({
        customer: customerId,
        items: [{ price: priceId }],
        payment_behavior: 'default_incomplete',
        payment_settings: { save_default_payment_method: 'on_subscription' },
        expand: ['latest_invoice.payment_intent']
      });

      return {
        subscription_id: subscription.id,
        client_secret:
subscription.latest_invoice?.payment_intent?.client_secret
      };
    } catch (error) {
      console.error('Erro na criação da assinatura:', error);
      throw error;
    }
  }
}

```

3.3. Desenvolvimento do Frontend

O frontend será desenvolvido com React, Next.js e TypeScript, focando em uma experiência de usuário intuitiva e responsiva.

3.3.1. Configuração Inicial do Frontend

Criação do Projeto React com Next.js:

```
npx create-next-app@latest frontend --typescript --tailwind --eslint --app
cd frontend
npm install @tanstack/react-query axios react-hook-form @hookform/resolvers zod
lucide-react
npm install -D @types/node
```

Configuração do Axios (src/lib/api.ts):

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.NEXT_PUBLIC_API_BASE_URL || 'http://localhost:3001/api'
});

// Interceptor para adicionar token de autenticação
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('auth_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Interceptor para tratar erros de autenticação
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('auth_token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

3.3.2. Estrutura de Componentes

Hook de Autenticação (src/hooks/useAuth.ts):

```

import { useState, useEffect, createContext, useContext } from 'react';
import api from '@lib/api';

interface User {
  id: string;
  email: string;
  nome_completo: string;
  tipo_perfil: 'aluno' | 'professor' | 'administrador';
}

interface AuthContextType {
  user: User | null;
  login: (email: string, senha: string) => Promise<void>;
  register: (data: RegisterData) => Promise<void>;
  logout: () => void;
  loading: boolean;
}

interface RegisterData {
  email: string;
  senha: string;
  tipo_perfil: 'aluno' | 'professor';
  nome_completo: string;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({
  children }) => {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const token = localStorage.getItem('auth_token');
    if (token) {
      fetchUser();
    } else {
      setLoading(false);
    }
  }, []);

  const fetchUser = async () => {
    try {
      const response = await api.get('/users/me');
      setUser(response.data.usuario);
    } catch (error) {
      localStorage.removeItem('auth_token');
    } finally {
      setLoading(false);
    }
  };

  const login = async (email: string, senha: string) => {
    const response = await api.post('/auth/login', { email, senha });
    const { token, usuario } = response.data;

    localStorage.setItem('auth_token', token);
    setUser(usuario);
  };

  const register = async (data: RegisterData) => {

```

```

    const response = await api.post('/auth/register', data);
    const { token, usuario } = response.data;

    localStorage.setItem('auth_token', token);
    setUser(usuario);
  };

  const logout = () => {
    localStorage.removeItem('auth_token');
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, register, logout, loading }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};

```

Componente de Dashboard do Aluno (src/components/StudentDashboard.tsx):

```

'use client';

import { useEffect, useState } from 'react';
import { useAuth } from '@/hooks/useAuth';
import api from '@/lib/api';
import { Calendar, Clock, User, Search } from 'lucide-react';

interface Aula {
  id: string;
  data_hora_inicio: string;
  data_hora_fim: string;
  professor: {
    nome: string;
    foto_perfil: string;
  };
  status_aula: string;
  tipo_aula: 'online' | 'presencial';
}

export default function StudentDashboard() {
  const { user } = useAuth();
  const [proximasAulas, setProximasAulas] = useState<Aula[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchProximasAulas();
  }, []);

  const fetchProximasAulas = async () => {
    try {
      const response = await api.get('/schedule/my-lessons?status=confirmada');
      setProximasAulas(response.data.lista_aulas.slice(0, 3));
    } catch (error) {
      console.error('Erro ao buscar aulas:', error);
    } finally {
      setLoading(false);
    }
  };

  const formatarDataHora = (dataHora: string) => {
    const data = new Date(dataHora);
    return {
      data: data.toLocaleDateString('pt-BR'),
      hora: data.toLocaleTimeString('pt-BR', { hour: '2-digit', minute: '2-digit' })
    };
  };

  if (loading) {
    return <div className="flex justify-center items-center h-64">Carregando...</div>;
  }

  return (
    <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
      <div className="mb-8">
        <h1 className="text-3xl font-bold text-gray-900">
          Olá, {user?.nome_completo}!
        </h1>
        <p className="text-gray-600 mt-2">
          Bem-vindo ao seu painel de estudos
        </p>
      </div>
    </div>
  );
}

```

```

    </p>
  </div>

  <div className="grid grid-cols-1 lg:grid-cols-3 gap-8">
    { /* Próximas Aulas */ }
    <div className="lg:col-span-2">
      <div className="bg-white rounded-lg shadow-md p-6">
        <div className="flex items-center justify-between mb-6">
          <h2 className="text-xl font-semibold text-gray-900 flex items-
center">
            <Calendar className="mr-2 h-5 w-5" />
            Próximas Aulas
          </h2>
          <button className="text-blue-600 hover:text-blue-800 text-sm
font-medium">
            Ver todas
          </button>
        </div>

        {proximasAulas.length === 0 ? (
          <div className="text-center py-8">
            <Calendar className="mx-auto h-12 w-12 text-gray-400" />
            <h3 className="mt-2 text-sm font-medium text-gray-900">
              Nenhuma aula agendada
            </h3>
            <p className="mt-1 text-sm text-gray-500">
              Que tal procurar um professor?
            </p>
            <div className="mt-6">
              <button className="inline-flex items-center px-4 py-2 border
border-transparent shadow-sm text-sm font-medium rounded-md text-white bg-blue-
600 hover:bg-blue-700">
                <Search className="mr-2 h-4 w-4" />
                Buscar Professores
              </button>
            </div>
          </div>
        ) : (
          <div className="space-y-4">
            {proximasAulas.map((aula) => {
              const { data, hora } =
formatarDataHora(aula.data_hora_inicio);
              return (
                <div
                  key={aula.id}
                  className="flex items-center p-4 border border-gray-200
rounded-lg hover:bg-gray-50"
                >
                  <img
                    src={aula.professor.foto_perfil || '/default-
avatar.png'}
                    alt={aula.professor.nome}
                    className="h-12 w-12 rounded-full object-cover"
                  />
                  <div className="ml-4 flex-1">
                    <h3 className="text-sm font-medium text-gray-900">
                      {aula.professor.nome}
                    </h3>
                    <div className="flex items-center text-sm text-gray-500
mt-1">
                      <Clock className="mr-1 h-4 w-4" />
                      {data} às {hora}

```

```

        </div>
      </div>
      <div className="flex items-center">
        <span className={`inline-flex items-center px-2.5 py-
0.5 rounded-full text-xs font-medium ${
          aula.tipo_aula === 'online'
            ? 'bg-green-100 text-green-800'
            : 'bg-blue-100 text-blue-800'
        }`}>
          {aula.tipo_aula === 'online' ? 'Online' :
'Presencial'}
        </span>
        <button className="ml-3 text-blue-600 hover:text-blue-
800 text-sm font-medium">
          Detalhes
        </button>
      </div>
    </div>
  );
  }}}
</div>
)}
</div>
</div>

{/* Sidebar com ações rápidas */}
<div className="space-y-6">
  <div className="bg-white rounded-lg shadow-md p-6">
    <h3 className="text-lg font-medium text-gray-900 mb-4">
      Ações Rápidas
    </h3>
    <div className="space-y-3">
      <button className="w-full flex items-center justify-center px-4
py-2 border border-transparent text-sm font-medium rounded-md text-white bg-
blue-600 hover:bg-blue-700">
        <Search className="mr-2 h-4 w-4" />
        Buscar Professor
      </button>
      <button className="w-full flex items-center justify-center px-4
py-2 border border-gray-300 text-sm font-medium rounded-md text-gray-700 bg-
white hover:bg-gray-50">
        <User className="mr-2 h-4 w-4" />
        Meu Perfil
      </button>
    </div>
  </div>
</div>

<div className="bg-white rounded-lg shadow-md p-6">
  <h3 className="text-lg font-medium text-gray-900 mb-4">
    Estatísticas
  </h3>
  <div className="space-y-4">
    <div className="flex justify-between">
      <span className="text-sm text-gray-500">Aulas realizadas</span>
      <span className="text-sm font-medium text-gray-900">12</span>
    </div>
    <div className="flex justify-between">
      <span className="text-sm text-gray-500">Professores</span>
      <span className="text-sm font-medium text-gray-900">3</span>
    </div>
    <div className="flex justify-between">
      <span className="text-sm text-gray-500">Horas de estudo</span>

```

```
        <span className="text-sm font-medium text-gray-900">18h</span>
      </div>
    </div>
  </div>
</div>
</div>
</div>
);
}
```

Componente de Busca de Professores (src/components/ProfessorSearch.tsx):


```

'use client';

import { useState, useEffect } from 'react';
import { useSearchParams } from 'next/navigation';
import api from '@lib/api';
import { Search, Filter, MapPin, Star, Clock } from 'lucide-react';

interface Professor {
  id: string;
  nome: string;
  foto_perfil: string;
  instrumentos: Array<{ nome: string; nivel: string }>;
  preco_hora: number;
  media_avaliacao: number;
  total_avaliacoes: number;
  online_disponivel: boolean;
  premium: boolean;
}

interface SearchFilters {
  instrumento: string;
  preco_min: string;
  preco_max: string;
  online: boolean;
  presencial: boolean;
  localizacao: string;
}

export default function ProfessorSearch() {
  const searchParams = useSearchParams();
  const [professores, setProfessores] = useState<Professor[]>([]);
  const [loading, setLoading] = useState(false);
  const [filters, setFilters] = useState<SearchFilters>({
    instrumento: searchParams.get('instrumento') || '',
    preco_min: '',
    preco_max: '',
    online: true,
    presencial: true,
    localizacao: ''
  });

  const [showFilters, setShowFilters] = useState(false);

  useEffect(() => {
    searchProfessors();
  }, []);

  const searchProfessors = async () => {
    setLoading(true);
    try {
      const params = new URLSearchParams();

      Object.entries(filters).forEach(([key, value]) => {
        if (value !== '' && value !== false) {
          params.append(key, value.toString());
        }
      });

      const response = await api.get(`/professors?${params.toString()}`);
      setProfessores(response.data.professores);
    } catch (error) {
      console.error('Erro na busca:', error);
    }
  };
}

```



```

        </button>
      </div>
    </div>

    /* Filtros expandidos */
    {showFilters && (
      <div className="bg-white rounded-lg shadow-md p-6 mb-6">
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-
4">
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-2">
              Preço mínimo
            </label>
            <input
              type="number"
              placeholder="R$ 0"
              value={filters.preco_min}
              onChange={(e) => handleFilterChange('preco_min',
e.target.value)}
              className="w-full px-3 py-2 border border-gray-300 rounded-md
focus:ring-2 focus:ring-blue-500 focus:border-transparent"
            />
          </div>

          <div>
            <label className="block text-sm font-medium text-gray-700 mb-2">
              Preço máximo
            </label>
            <input
              type="number"
              placeholder="R$ 200"
              value={filters.preco_max}
              onChange={(e) => handleFilterChange('preco_max',
e.target.value)}
              className="w-full px-3 py-2 border border-gray-300 rounded-md
focus:ring-2 focus:ring-blue-500 focus:border-transparent"
            />
          </div>

          <div>
            <label className="block text-sm font-medium text-gray-700 mb-2">
              Localização
            </label>
            <input
              type="text"
              placeholder="Cidade, estado..."
              value={filters.localizacao}
              onChange={(e) => handleFilterChange('localizacao',
e.target.value)}
              className="w-full px-3 py-2 border border-gray-300 rounded-md
focus:ring-2 focus:ring-blue-500 focus:border-transparent"
            />
          </div>

          <div>
            <label className="block text-sm font-medium text-gray-700 mb-2">
              Tipo de aula
            </label>
            <div className="space-y-2">
              <label className="flex items-center">
                <input
                  type="checkbox"

```

```

checked={filters.online}
onChange={(e) => handleFilterChange('online',
e.target.checked)}
className="rounded border-gray-300 text-blue-600
focus:ring-blue-500"
/>
<span className="ml-2 text-sm text-gray-700">Online</span>
</label>
<label className="flex items-center">
<input
type="checkbox"
checked={filters.presencial}
onChange={(e) => handleFilterChange('presencial',
e.target.checked)}
className="rounded border-gray-300 text-blue-600
focus:ring-blue-500"
/>
<span className="ml-2 text-sm text-gray-
700">Presencial</span>
</label>
</div>
</div>
</div>
</div>
)}

{/* Resultados */}
{loading ? (
<div className="flex justify-center items-center h-64">
<div className="animate-spin rounded-full h-12 w-12 border-b-2
border-blue-600"></div>
</div>
) : (
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
{professores.map((professor) => (
<div
key={professor.id}
className="bg-white rounded-lg shadow-md overflow-hidden
hover:shadow-lg transition-shadow"
>
{professor.premium && (
<div className="bg-gradient-to-r from-yellow-400 to-yellow-600
text-white text-xs font-bold px-3 py-1">
PROFESSOR PREMIUM
</div>
)}
</div>
<div className="p-6">
<div className="flex items-center mb-4">
<img
src={professor.foto_perfil || '/default-avatar.png'}
alt={professor.nome}
className="h-16 w-16 rounded-full object-cover"
/>
<div className="ml-4">
<h3 className="text-lg font-semibold text-gray-900">
{professor.nome}
</h3>
<div className="flex items-center mt-1">
{renderStars(professor.media_avaliacao)}
<span className="ml-2 text-sm text-gray-600">
({professor.total_avaliacoes})

```

```

        </span>
      </div>
    </div>
  </div>

  <div className="mb-4">
    <div className="flex flex-wrap gap-2">
      {professor.instrumentos.slice(0, 3).map((instrumento,
index) => (
        <span
          key={index}
          className="inline-flex items-center px-2.5 py-0.5
rounded-full text-xs font-medium bg-blue-100 text-blue-800"
        >
          {instrumento.nome}
        </span>
      ))}
      {professor.instrumentos.length > 3 && (
        <span className="text-xs text-gray-500">
          +{professor.instrumentos.length - 3} mais
        </span>
      )}
    </div>
  </div>

  <div className="flex items-center justify-between mb-4">
    <div className="flex items-center text-sm text-gray-600">
      <Clock className="mr-1 h-4 w-4" />
      R$ {professor.preco_hora}/hora
    </div>
    <div className="flex items-center space-x-2">
      {professor.online_disponivel && (
        <span className="inline-flex items-center px-2 py-1
rounded-full text-xs font-medium bg-green-100 text-green-800">
          Online
        </span>
      )}
    </div>
  </div>

  <div className="flex space-x-2">
    <button className="flex-1 bg-blue-600 text-white px-4 py-2
rounded-md hover:bg-blue-700 font-medium text-sm">
      Ver Perfil
    </button>
    <button className="px-4 py-2 border border-gray-300 rounded-
md hover:bg-gray-50 font-medium text-sm">
      Mensagem
    </button>
  </div>
</div>
</div>
  )}
</div>
)}

{!loading && professores.length === 0 && (
  <div className="text-center py-12">
    <Search className="mx-auto h-12 w-12 text-gray-400" />
    <h3 className="mt-2 text-sm font-medium text-gray-900">
      Nenhum professor encontrado
    </h3>
  </div>
)}

```

```
        <p className="mt-1 text-sm text-gray-500">
            Tente ajustar os filtros de busca
        </p>
    </div>
    })
</div>
);
}
```

Fase 4: Testes e Garantia de Qualidade

A fase de testes é crucial para garantir que o sistema funcione corretamente, seja seguro e ofereça uma boa experiência ao usuário. Esta fase inclui diferentes tipos de testes, desde unitários até testes de aceitação do usuário.

4.1. Estratégia de Testes

A estratégia de testes seguirá a pirâmide de testes, priorizando testes unitários na base, seguidos por testes de integração e, no topo, testes end-to-end (E2E).

4.1.1. Tipos de Testes

Testes Unitários: - Testam funções e métodos individuais. - Executados rapidamente e com frequência. - Cobertura mínima de 80% do código.

Testes de Integração: - Testam a interação entre diferentes módulos. - Incluem testes de API e integração com banco de dados. - Verificam fluxos completos de funcionalidades.

Testes End-to-End (E2E): - Simulam o comportamento real do usuário. - Testam fluxos críticos como cadastro, login, agendamento e pagamento. - Executados em ambiente similar à produção.

Testes de Performance: - Verificam tempos de resposta das APIs. - Testam capacidade de carga do sistema. - Monitoram uso de recursos (CPU, memória, banco de dados).

Testes de Segurança: - Verificam autenticação e autorização. - Testam proteção contra ataques comuns (SQL injection, XSS). - Validam criptografia de dados sensíveis.

4.2. Implementação de Testes no Backend

4.2.1. Configuração do Ambiente de Testes

Instalação de Dependências:

```
npm install -D jest @types/jest supertest @types/supertest
npm install -D ts-jest jest-environment-node
```

Configuração do Jest (jest.config.js):

```
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  roots: ['<rootDir>/src', '<rootDir>/tests'],
  testMatch: ['**/__tests__/**/*.ts', '**/?(*.)+(spec|test).ts'],
  collectCoverageFrom: [
    'src/**/*.ts',
    '!src/**/*.d.ts',
    '!src/index.ts'
  ],
  coverageDirectory: 'coverage',
  coverageReporters: ['text', 'lcov', 'html'],
  setupFilesAfterEnv: ['<rootDir>/tests/setup.ts']
};
```

4.2.2. Testes Unitários

Teste de Service (tests/services/professorService.test.ts):

```

import { ProfessorService } from '../../../src/services/professorService';
import { PrismaClient } from '@prisma/client';

// Mock do Prisma
jest.mock('@prisma/client');
const mockPrisma = new PrismaClient() as jest.Mocked<PrismaClient>;

describe('ProfessorService', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  describe('searchProfessors', () => {
    it('deve retornar lista de professores com filtros aplicados', async () => {
      // Arrange
      const mockProfessores = [
        {
          id: '1',
          usuario: {
            id: '1',
            nome_completo: 'João Silva',
            foto_perfil_url: 'photo.jpg'
          },
          professor_instrumentos: [
            {
              instrumento: { nome: 'Violão' },
              nivel_proficiencia: 'avancado'
            }
          ],
          preco_hora_base: 50,
          media_avaliacao: 4.5,
          total_avaliacoes: 10,
          online_disponivel: true,
          plano_premium_id: null
        }
      ];

      mockPrisma.professores.findMany.mockResolvedValue(mockProfessores as any);
      mockPrisma.professores.count.mockResolvedValue(1);

      // Act
      const result = await ProfessorService.searchProfessors({
        instrumento: 'violão',
        page: 1,
        limit: 20
      });

      // Assert
      expect(result.professores).toHaveLength(1);
      expect(result.professores[0].nome).toBe('João Silva');
      expect(result.total_professores).toBe(1);
      expect(mockPrisma.professores.findMany).toHaveBeenCalledWith(
        expect.objectContaining({
          where: expect.objectContaining({
            status_aprovacao: 'aprovado',
            professor_instrumentos: {
              some: {
                instrumento: {
                  nome: {

```



```

        contains: 'violão',
        mode: 'insensitive'
      }
    }
  }
}
}))
);
});

it('deve aplicar filtros de preço corretamente', async () => {
  // Arrange
  mockPrisma.professores.findMany.mockResolvedValue([]);
  mockPrisma.professores.count.mockResolvedValue(0);

  // Act
  await ProfessorService.searchProfessors({
    preco_min: 30,
    preco_max: 80
  });

  // Assert
  expect(mockPrisma.professores.findMany).toHaveBeenCalled();
  expect(expect.objectContaining({
    where: expect.objectContaining({
      preco_hora_base: {
        gte: 30,
        lte: 80
      }
    })
  })).toHaveBeenCalled();
});

describe('getProfessorById', () => {
  it('deve retornar dados completos do professor', async () => {
    // Arrange
    const mockProfessor = {
      id: '1',
      status_aprovacao: 'aprovado',
      usuario: {
        id: '1',
        nome_completo: 'João Silva',
        foto_perfil_url: 'photo.jpg'
      },
      biografia: 'Professor experiente',
      experiencia: '10 anos',
      metodologia: 'Método tradicional',
      professor_instrumentos: [
        {
          instrumento: { nome: 'Violão' },
          nivel_proficiencia: 'avancado'
        }
      ],
      preco_hora_base: 50,
      media_avaliacao: 4.5,
      total_avaliacoes: 10,
      online_disponivel: true,
      localizacao_presencial: 'São Paulo, SP',
      link_youtube: 'youtube.com/joao',
    };
  });
});

```

```

        link_instagram: 'instagram.com/joao',
        link_soundcloud: 'soundcloud.com/joao',
        materiais_professor: [],
        avaliacoes: []
    };

    mockPrisma.professores.findUnique.mockResolvedValue(mockProfessor as
any);

    // Act
    const result = await ProfessorService.getProfessorById('1');

    // Assert
    expect(result).toBeDefined();
    expect(result?.nome).toBe('João Silva');
    expect(result?.biografia).toBe('Professor experiente');
    expect(result?.instrumentos).toHaveLength(1);
    expect(result?.links.youtube).toBe('youtube.com/joao');
});

it('deve retornar null para professor não aprovado', async () => {
    // Arrange
    const mockProfessor = {
        id: '1',
        status_aprovacao: 'pendente',
        usuario: { nome_completo: 'João Silva' }
    };

    mockPrisma.professores.findUnique.mockResolvedValue(mockProfessor as
any);

    // Act
    const result = await ProfessorService.getProfessorById('1');

    // Assert
    expect(result).toBeNull();
});
});
});
});

```

4.2.3. Testes de Integração

Teste de API (tests/integration/auth.test.ts):

```

import request from 'supertest';
import app from '../../src/app';
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

describe('Auth API', () => {
  beforeAll(async () => {
    // Limpar banco de dados de teste
    await prisma.usuarios.deleteMany();
  });

  afterAll(async () => {
    await prisma.$disconnect();
  });

  describe('POST /api/auth/register', () => {
    it('deve registrar um novo aluno com sucesso', async () => {
      const userData = {
        email: 'aluno@teste.com',
        senha: '123456',
        tipo_perfil: 'aluno',
        nome_completo: 'Aluno Teste'
      };

      const response = await request(app)
        .post('/api/auth/register')
        .send(userData)
        .expect(201);

      expect(response.body).toHaveProperty('token');
      expect(response.body.usuario.email).toBe(userData.email);
      expect(response.body.usuario.tipo_perfil).toBe('aluno');

      // Verificar se o usuário foi criado no banco
      const usuario = await prisma.usuarios.findUnique({
        where: { email: userData.email }
      });
      expect(usuario).toBeDefined();
      expect(usuario?.nome_completo).toBe(userData.nome_completo);
    });

    it('deve registrar um novo professor com sucesso', async () => {
      const userData = {
        email: 'professor@teste.com',
        senha: '123456',
        tipo_perfil: 'professor',
        nome_completo: 'Professor Teste'
      };

      const response = await request(app)
        .post('/api/auth/register')
        .send(userData)
        .expect(201);

      expect(response.body).toHaveProperty('token');
      expect(response.body.usuario.tipo_perfil).toBe('professor');

      // Verificar se o perfil de professor foi criado
      const professor = await prisma.professores.findFirst({
        where: {

```

```

        usuario: {
          email: userData.email
        }
      }
    });
    expect(professor).toBeDefined();
  });

  it('deve retornar erro para email já cadastrado', async () => {
    const userData = {
      email: 'aluno@teste.com', // Email já usado no teste anterior
      senha: '123456',
      tipo_perfil: 'aluno',
      nome_completo: 'Outro Aluno'
    };

    const response = await request(app)
      .post('/api/auth/register')
      .send(userData)
      .expect(400);

    expect(response.body.error).toBe('Email já cadastrado');
  });

  it('deve retornar erro para dados inválidos', async () => {
    const userData = {
      email: 'email-inválido',
      senha: '123', // Senha muito curta
      tipo_perfil: 'aluno',
      nome_completo: 'A' // Nome muito curto
    };

    const response = await request(app)
      .post('/api/auth/register')
      .send(userData)
      .expect(400);

    expect(response.body.error).toBe('Dados inválidos');
    expect(response.body.details).toBeDefined();
  });
});

describe('POST /api/auth/login', () => {
  it('deve fazer login com credenciais válidas', async () => {
    const loginData = {
      email: 'aluno@teste.com',
      senha: '123456'
    };

    const response = await request(app)
      .post('/api/auth/login')
      .send(loginData)
      .expect(200);

    expect(response.body).toHaveProperty('token');
    expect(response.body.usuario.email).toBe(loginData.email);
  });

  it('deve retornar erro para credenciais inválidas', async () => {
    const loginData = {
      email: 'aluno@teste.com',
      senha: 'senha-errada'
    };
  });
});

```

```

    });

    const response = await request(app)
      .post('/api/auth/login')
      .send(loginData)
      .expect(401);

    expect(response.body.error).toBe('Credenciais inválidas');
  });

  it('deve retornar erro para usuário inexistente', async () => {
    const loginData = {
      email: 'inexistente@teste.com',
      senha: '123456'
    };

    const response = await request(app)
      .post('/api/auth/login')
      .send(loginData)
      .expect(401);

    expect(response.body.error).toBe('Credenciais inválidas');
  });
});
});
});

```

4.3. Implementação de Testes no Frontend

4.3.1. Configuração do Ambiente de Testes

Instalação de Dependências:

```

npm install -D @testing-library/react @testing-library/jest-dom @testing-library/user-event
npm install -D jest jest-environment-jsdom

```

Configuração do Jest (jest.config.js):

```
const nextJest = require('next/jest');

const createJestConfig = nextJest({
  dir: './',
});

const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  moduleNameMapping: {
    '^@/components/(.*)$': '<rootDir>/components/`$1',
    '^@/pages/(.*)$': '<rootDir>/pages/`$1',
  },
  testEnvironment: 'jest-environment-jsdom',
};

module.exports = createJestConfig(customJestConfig);
```

4.3.2. Testes de Componentes

Teste de Componente (tests/components/StudentDashboard.test.tsx):

```

import { render, screen, waitFor } from '@testing-library/react';
import { AuthProvider } from '@hooks/useAuth';
import StudentDashboard from '@components/StudentDashboard';
import api from '@lib/api';

// Mock da API
jest.mock('@lib/api');
const mockApi = api as jest.Mocked<typeof api>;

// Mock do hook de autenticação
const mockUser = {
  id: '1',
  email: 'aluno@teste.com',
  nome_completo: 'Aluno Teste',
  tipo_perfil: 'aluno' as const
};

const MockAuthProvider = ({ children }: { children: React.ReactNode }) => (
  <AuthProvider value={{
    user: mockUser,
    login: jest.fn(),
    register: jest.fn(),
    logout: jest.fn(),
    loading: false
  }}>
    {children}
  </AuthProvider>
);

describe('StudentDashboard', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  it('deve renderizar o nome do aluno', async () => {
    mockApi.get.mockResolvedValue({
      data: { lista_aulas: [] }
    });

    render(
      <MockAuthProvider>
        <StudentDashboard />
      </MockAuthProvider>
    );

    expect(screen.getByText('Olá, Aluno Teste!')).toBeInTheDocument();
    expect(screen.getByText('Bem-vindo ao seu painel de estudos')).toBeInTheDocument();
  });

  it('deve exibir próximas aulas quando existirem', async () => {
    const mockAulas = [
      {
        id: '1',
        data_hora_inicio: '2024-03-15T10:00:00Z',
        data_hora_fim: '2024-03-15T11:00:00Z',
        professor: {
          nome: 'Professor João',
          foto_perfil: 'photo.jpg'
        },
        status_aula: 'confirmada',
      }
    ];
  });

```

```

        tipo_aula: 'online'
      }
    ];

    mockApi.get.mockResolvedValue({
      data: { lista_aulas: mockAulas }
    });

    render(
      <MockAuthProvider>
        <StudentDashboard />
      </MockAuthProvider>
    );

    await waitFor(() => {
      expect(screen.getByText('Professor João')).toBeInTheDocument();
      expect(screen.getByText('Online')).toBeInTheDocument();
    });
  });

  it('deve exibir mensagem quando não há aulas agendadas', async () => {
    mockApi.get.mockResolvedValue({
      data: { lista_aulas: [] }
    });

    render(
      <MockAuthProvider>
        <StudentDashboard />
      </MockAuthProvider>
    );

    await waitFor(() => {
      expect(screen.getByText('Nenhuma aula agendada')).toBeInTheDocument();
      expect(screen.getByText('Que tal procurar um professor?')).toBeInTheDocument();
    });
  });

  it('deve exibir loading enquanto carrega as aulas', () => {
    mockApi.get.mockImplementation(() => new Promise(() => {})); // Promise que nunca resolve

    render(
      <MockAuthProvider>
        <StudentDashboard />
      </MockAuthProvider>
    );

    expect(screen.getByText('Carregando...')).toBeInTheDocument();
  });
});

```

4.4. Testes End-to-End (E2E)

4.4.1. Configuração do Cypress

Instalação:


```
npm install -D cypress
```

Configuração (cypress.config.js):

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    supportFile: 'cypress/support/e2e.js',
    specPattern: 'cypress/e2e/**/*.cy.{js,jsx,ts,tsx}',
    video: true,
    screenshot: true
  }
});
```

4.4.2. Testes E2E Críticos

Teste de Fluxo de Cadastro e Login (cypress/e2e/auth.cy.ts):

```

describe('Autenticação', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('deve permitir cadastro de aluno', () => {
    cy.get('[data-cy=cadastro-button]').click();
    cy.get('[data-cy=tipo-aluno]').click();

    cy.get('[data-cy=nome-input]').type('Aluno E2E Teste');
    cy.get('[data-cy=email-input]').type('aluno-e2e@teste.com');
    cy.get('[data-cy=senha-input]').type('123456');
    cy.get('[data-cy=confirmar-senha-input]').type('123456');

    cy.get('[data-cy=cadastrar-button]').click();

    cy.url().should('include', '/dashboard');
    cy.get('[data-cy=user-name]').should('contain', 'Aluno E2E Teste');
  });

  it('deve permitir login de usuário existente', () => {
    // Primeiro, criar um usuário via API
    cy.request('POST', '/api/auth/register', {
      email: 'login-test@teste.com',
      senha: '123456',
      tipo_perfil: 'aluno',
      nome_completo: 'Usuário Login'
    });

    cy.get('[data-cy=login-button]').click();
    cy.get('[data-cy=email-input]').type('login-test@teste.com');
    cy.get('[data-cy=senha-input]').type('123456');
    cy.get('[data-cy=entrar-button]').click();

    cy.url().should('include', '/dashboard');
    cy.get('[data-cy=user-name]').should('contain', 'Usuário Login');
  });
});

```

Teste de Fluxo de Agendamento (cypress/e2e/booking.cy.ts):

```

describe('Agendamento de Aulas', () => {
  beforeEach(() => {
    // Login como aluno
    cy.request('POST', '/api/auth/login', {
      email: 'aluno@teste.com',
      senha: '123456'
    }).then((response) => {
      window.localStorage.setItem('auth_token', response.body.token);
    });

    cy.visit('/dashboard');
  });

  it('deve permitir buscar e agendar aula com professor', () => {
    cy.get('[data-cy=buscar-professor-button]').click();

    // Buscar por violão
    cy.get('[data-cy=instrumento-input]').type('violão');
    cy.get('[data-cy=buscar-button]').click();

    // Selecionar primeiro professor
    cy.get('[data-cy=professor-card]').first().within(() => {
      cy.get('[data-cy=ver-perfil-button]').click();
    });

    // Agendar aula
    cy.get('[data-cy=agendar-aula-button]').click();

    // Selecionar data e hora
    cy.get('[data-cy=calendar]').within(() => {
      cy.get('[data-available=true]').first().click();
    });

    cy.get('[data-cy=duracao-select]').select('60');
    cy.get('[data-cy=tipo-online]').click();

    cy.get('[data-cy=ir-pagamento-button]').click();

    // Simular pagamento (em ambiente de teste)
    cy.get('[data-cy=cartao-numero]').type('4242424242424242');
    cy.get('[data-cy=cartao-nome]').type('Teste Usuário');
    cy.get('[data-cy=cartao-validade]').type('12/25');
    cy.get('[data-cy=cartao-cvv]').type('123');

    cy.get('[data-cy=confirmar-pagamento-button]').click();

    // Verificar sucesso
    cy.get('[data-cy=success-message]').should('be.visible');
    cy.url().should('include', '/dashboard');

    // Verificar se a aula aparece no dashboard
    cy.get('[data-cy=proximas-aulas]').should('contain', 'Professor');
  });
});

```

4.5. Testes de Performance

4.5.1. Testes de Carga com Artillery

Instalação:

```
npm install -D artillery
```

Configuração de Teste (tests/performance/load-test.yml):

```
config:
  target: 'http://localhost:3001'
  phases:
    - duration: 60
      arrivalRate: 10
      name: "Warm up"
    - duration: 120
      arrivalRate: 50
      name: "Load test"
    - duration: 60
      arrivalRate: 100
      name: "Stress test"
  defaults:
    headers:
      Content-Type: 'application/json'

scenarios:
- name: "API Health Check"
  weight: 20
  flow:
    - get:
        url: "/api/health"

- name: "User Registration and Login"
  weight: 30
  flow:
    - post:
        url: "/api/auth/register"
        json:
          email: "user-{{ $randomString() }}@test.com"
          senha: "123456"
          tipo_perfil: "aluno"
          nome_completo: "Test User {{ $randomString() }}"
    - post:
        url: "/api/auth/login"
        json:
          email: "{{ email }}"
          senha: "123456"
        capture:
          - json: "$.token"
            as: "authToken"

- name: "Search Professors"
  weight: 40
  flow:
    - get:
        url: "/api/professors"
        qs:
          instrumento: "violão"
          page: 1
          limit: 20

- name: "Get Professor Details"
  weight: 10
  flow:
    - get:
        url: "/api/professors/{{ $randomUUID() }}"
```

4.5.2. Monitoramento de Performance

Teste de Performance de API (tests/performance/api-performance.test.ts):

```
import request from 'supertest';
import app from '../../src/app';

describe('API Performance Tests', () => {
  it('deve responder busca de professores em menos de 500ms', async () => {
    const startTime = Date.now();

    const response = await request(app)
      .get('/api/professores')
      .query({
        instrumento: 'violão',
        page: 1,
        limit: 20
      })
      .expect(200);

    const endTime = Date.now();
    const responseTime = endTime - startTime;

    expect(responseTime).toBeLessThan(500);
    expect(response.body.professores).toBeDefined();
  });

  it('deve responder login em menos de 200ms', async () => {
    // Primeiro, criar um usuário
    await request(app)
      .post('/api/auth/register')
      .send({
        email: 'performance@test.com',
        senha: '123456',
        tipo_perfil: 'aluno',
        nome_completo: 'Performance Test'
      });

    const startTime = Date.now();

    const response = await request(app)
      .post('/api/auth/login')
      .send({
        email: 'performance@test.com',
        senha: '123456'
      })
      .expect(200);

    const endTime = Date.now();
    const responseTime = endTime - startTime;

    expect(responseTime).toBeLessThan(200);
    expect(response.body.token).toBeDefined();
  });
});
```

4.6. Testes de Segurança

4.6.1. Testes de Autenticação e Autorização

Teste de Segurança (tests/security/auth-security.test.ts):

```

import request from 'supertest';
import app from '../src/app';
import jwt from 'jsonwebtoken';

describe('Security Tests', () => {
  describe('Authentication', () => {
    it('deve rejeitar acesso sem token', async () => {
      await request(app)
        .get('/api/users/me')
        .expect(401);
    });

    it('deve rejeitar token inválido', async () => {
      await request(app)
        .get('/api/users/me')
        .set('Authorization', 'Bearer token-invalido')
        .expect(403);
    });

    it('deve rejeitar token expirado', async () => {
      const expiredToken = jwt.sign(
        { userId: '1', email: 'test@test.com' },
        process.env.JWT_SECRET!,
        { expiresIn: '-1h' }
      );

      await request(app)
        .get('/api/users/me')
        .set('Authorization', `Bearer ${expiredToken}`)
        .expect(403);
    });
  });

  describe('Input Validation', () => {
    it('deve sanitizar entrada para prevenir SQL injection', async () => {
      const maliciousInput = "'; DROP TABLE usuarios; --";

      const response = await request(app)
        .post('/api/auth/login')
        .send({
          email: maliciousInput,
          senha: '123456'
        })
        .expect(400);

      expect(response.body.error).toBe('Dados inválidos');
    });

    it('deve validar formato de email', async () => {
      await request(app)
        .post('/api/auth/register')
        .send({
          email: 'email-invalido',
          senha: '123456',
          tipo_perfil: 'aluno',
          nome_completo: 'Test User'
        })
        .expect(400);
    });

    it('deve exigir senha com tamanho mínimo', async () => {

```



```

    await request(app)
      .post('/api/auth/register')
      .send({
        email: 'test@test.com',
        senha: '123', // Muito curta
        tipo_perfil: 'aluno',
        nome_completo: 'Test User'
      })
      .expect(400);
  });
});

describe('Authorization', () => {
  let alunoToken: string;
  let professorToken: string;

  beforeAll(async () => {
    // Criar aluno
    const alunoResponse = await request(app)
      .post('/api/auth/register')
      .send({
        email: 'aluno-auth@test.com',
        senha: '123456',
        tipo_perfil: 'aluno',
        nome_completo: 'Aluno Auth'
      });
    alunoToken = alunoResponse.body.token;

    // Criar professor
    const professorResponse = await request(app)
      .post('/api/auth/register')
      .send({
        email: 'professor-auth@test.com',
        senha: '123456',
        tipo_perfil: 'professor',
        nome_completo: 'Professor Auth'
      });
    professorToken = professorResponse.body.token;
  });

  it('aluno não deve acessar rotas de professor', async () => {
    await request(app)
      .put('/api/professors/me')
      .set('Authorization', `Bearer ${alunoToken}`)
      .send({ biografia: 'Nova biografia' })
      .expect(403);
  });

  it('professor não deve acessar dados de outros professores', async () => {
    await request(app)
      .put('/api/professors/outro-id')
      .set('Authorization', `Bearer ${professorToken}`)
      .send({ biografia: 'Tentativa de hack' })
      .expect(403);
  });
});
});
});

```

4.7. Automação e Integração Contínua

4.7.1. Configuração do GitHub Actions

Workflow de CI/CD (.github/workflows/ci.yml):

```

name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test-backend:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: music_school_test
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 5432:5432

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'
          cache-dependency-path: backend/package-lock.json

      - name: Install backend dependencies
        working-directory: ./backend
        run: npm ci

      - name: Run database migrations
        working-directory: ./backend
        run: npx prisma migrate deploy
        env:
          DATABASE_URL:
            postgresql://postgres:postgres@localhost:5432/music_school_test

      - name: Run backend tests
        working-directory: ./backend
        run: npm run test:coverage
        env:
          DATABASE_URL:
            postgresql://postgres:postgres@localhost:5432/music_school_test
          JWT_SECRET: test-secret-key

      - name: Upload backend coverage
        uses: codecov/codecov-action@v3
        with:
          file: ./backend/coverage/lcov.info
          flags: backend

```

```

test-frontend:
  runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18'
        cache: 'npm'
        cache-dependency-path: frontend/package-lock.json

    - name: Install frontend dependencies
      working-directory: ./frontend
      run: npm ci

    - name: Run frontend tests
      working-directory: ./frontend
      run: npm run test:coverage

    - name: Upload frontend coverage
      uses: codecov/codecov-action@v3
      with:
        file: ./frontend/coverage/lcov.info
        flags: frontend

e2e-tests:
  runs-on: ubuntu-latest
  needs: [test-backend, test-frontend]

  steps:
    - uses: actions/checkout@v3

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18'

    - name: Install dependencies
      run: |
        cd backend && npm ci
        cd ../frontend && npm ci

    - name: Start backend
      working-directory: ./backend
      run: npm start &
      env:
        NODE_ENV: test
        DATABASE_URL: ${ secrets.TEST_DATABASE_URL }

    - name: Start frontend
      working-directory: ./frontend
      run: npm run build && npm start &
      env:
        NEXT_PUBLIC_API_BASE_URL: http://localhost:3001/api

    - name: Wait for services
      run: |
        npx wait-on http://localhost:3001/api/health
        npx wait-on http://localhost:3000

```

```

- name: Run E2E tests
uses: cypress-io/github-action@v5
with:
  working-directory: frontend
  browser: chrome
  record: true
env:
  CYPRESS_RECORD_KEY: ${ secrets.CYPRESS_RECORD_KEY }}

security-scan:
  runs-on: ubuntu-latest

steps:
- uses: actions/checkout@v3

- name: Run security audit
  run: |
    cd backend && npm audit --audit-level high
    cd ../frontend && npm audit --audit-level high

- name: Run SAST scan
uses: github/super-linter@v4
env:
  DEFAULT_BRANCH: main
  GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }}
  VALIDATE_TYPESCRIPT_ES: true
  VALIDATE_JAVASCRIPT_ES: true

```

4.8. Métricas de Qualidade

4.8.1. Cobertura de Testes

Configuração de Cobertura (package.json):

```

{
  "scripts": {
    "test": "jest",
    "test:watch": "jest --watch",
    "test:coverage": "jest --coverage",
    "test:ci": "jest --ci --coverage --watchAll=false"
  },
  "jest": {
    "collectCoverageFrom": [
      "src/**/*.{ts,tsx}",
      "!src/**/*.d.ts",
      "!src/index.ts"
    ],
    "coverageThreshold": {
      "global": {
        "branches": 80,
        "functions": 80,
        "lines": 80,
        "statements": 80
      }
    }
  }
}

```

4.8.2. Relatórios de Qualidade

Configuração do SonarQube (sonar-project.properties):

```

sonar.projectKey=music-school-platform
sonar.projectName=Music School Platform
sonar.projectVersion=1.0

sonar.sources=src
sonar.tests=tests
sonar.test.inclusions=**/*.test.ts,**/*.test.tsx
sonar.typescript.lcov.reportPaths=coverage/lcov.info

sonar.coverage.exclusions=**/*.test.ts,**/*.test.tsx,**/node_modules/**

```

Esta fase de testes garante que o sistema seja robusto, seguro e ofereça uma experiência confiável aos usuários. A implementação de testes automatizados e integração contínua permite detectar problemas rapidamente e manter a qualidade do código ao longo do desenvolvimento.

Fase 5: Implantação e Lançamento

Esta fase envolve a preparação e execução do lançamento do sistema em produção, incluindo configuração de infraestrutura, deploy das aplicações, configuração de monitoramento e estratégias de lançamento.

5.1. Estratégia de Implantação

5.1.1. Ambientes

Desenvolvimento (Development): - Ambiente local dos desenvolvedores - Banco de dados local ou de desenvolvimento - Configurações de debug habilitadas

Homologação/Staging: - Réplica do ambiente de produção - Dados de teste similares aos reais - Testes finais antes do deploy

Produção (Production): - Ambiente final acessível aos usuários - Configurações otimizadas para performance - Monitoramento e logs completos

5.1.2. Estratégia de Deploy

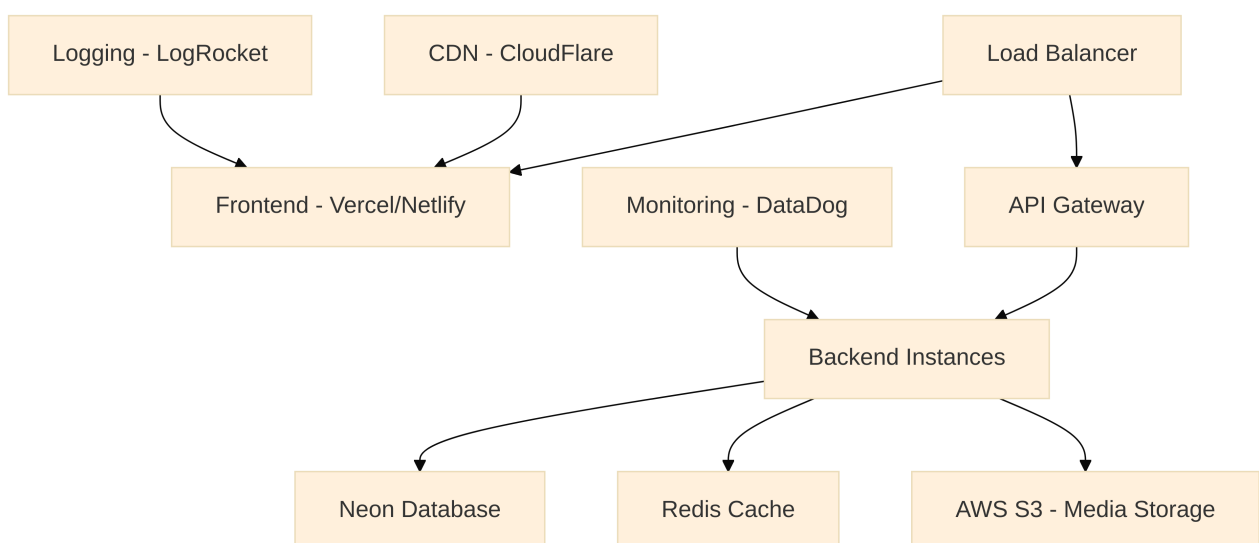
Blue-Green Deployment: - Manter dois ambientes idênticos (Blue e Green) - Deploy no ambiente inativo - Switch instantâneo após validação - Rollback rápido em caso de problemas

Rolling Updates: - Atualização gradual dos serviços - Manutenção da disponibilidade durante o deploy - Ideal para atualizações menores

5.2. Configuração da Infraestrutura

5.2.1. Arquitetura de Produção

Componentes da Infraestrutura:



5.2.2. Configuração do Backend (Node.js/Express)

Dockerfile para Backend:

```
# Dockerfile
FROM node:18-alpine AS builder

WORKDIR /app

# Copiar package.json e package-lock.json
COPY package*.json ./
RUN npm ci --only=production

# Copiar código fonte
COPY . .

# Build da aplicação
RUN npm run build

# Estágio de produção
FROM node:18-alpine AS production

WORKDIR /app

# Instalar apenas dependências de produção
COPY package*.json ./
RUN npm ci --only=production && npm cache clean --force

# Copiar build da aplicação
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/prisma ./prisma

# Criar usuário não-root
RUN addgroup -g 1001 -S nodejs
RUN adduser -S backend -u 1001

# Mudar ownership dos arquivos
RUN chown -R backend:nodejs /app
USER backend

# Expor porta
EXPOSE 3001

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD node dist/healthcheck.js

# Comando de inicialização
CMD ["node", "dist/index.js"]
```

docker-compose.yml para Desenvolvimento:


```
version: '3.8'

services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "3001:3001"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=${DATABASE_URL}
      - JWT_SECRET=${JWT_SECRET}
      - STRIPE_SECRET_KEY=${STRIPE_SECRET_KEY}
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - REDIS_URL=${REDIS_URL}
    depends_on:
      - redis
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - backend
    restart: unless-stopped
```

5.2.3. Configuração do Frontend (Next.js)

next.config.js para Produção:

```

/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  swcMinify: true,

  // Otimizações de produção
  compiler: {
    removeConsole: process.env.NODE_ENV === 'production',
  },

  // Configurações de imagem
  images: {
    domains: ['your-domain.com', 's3.amazonaws.com'],
    formats: ['image/webp', 'image/avif'],
  },

  // Headers de segurança
  async headers() {
    return [
      {
        source: '/(.*)',
        headers: [
          {
            key: 'X-Frame-Options',
            value: 'DENY',
          },
          {
            key: 'X-Content-Type-Options',
            value: 'nosniff',
          },
          {
            key: 'Referrer-Policy',
            value: 'origin-when-cross-origin',
          },
          {
            key: 'Content-Security-Policy',
            value: "default-src 'self'; script-src 'self' 'unsafe-eval' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; img-src 'self' data: https:; font-src 'self' data:;",
          },
        ],
      },
    ];
  },

  // Redirects
  async redirects() {
    return [
      {
        source: '/home',
        destination: '/',
        permanent: true,
      },
    ];
  },
};

module.exports = nextConfig;

```

Dockerfile para Frontend:

```
# Dockerfile
FROM node:18-alpine AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

COPY package.json package-lock.json ./
RUN npm ci --frozen-lockfile

FROM node:18-alpine AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

ENV NEXT_TELEMETRY_DISABLED 1
RUN npm run build

FROM node:18-alpine AS runner
WORKDIR /app

ENV NODE_ENV production
ENV NEXT_TELEMETRY_DISABLED 1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./next/static

USER nextjs

EXPOSE 3000

ENV PORT 3000

CMD ["node", "server.js"]
```

5.3. Deploy em Plataformas Cloud

5.3.1. Deploy do Backend (Railway/Render/DigitalOcean)

Configuração para Railway:

```
// railway.json
{
  "build": {
    "builder": "NIXPACKS"
  },
  "deploy": {
    "startCommand": "npm run start:prod",
    "healthcheckPath": "/api/health",
    "healthcheckTimeout": 100,
    "restartPolicyType": "ON_FAILURE",
    "restartPolicyMaxRetries": 10
  }
}
```

Script de Deploy (deploy-backend.sh):

```
#!/bin/bash

echo "🚀 Iniciando deploy do backend..."

# Verificar se todas as variáveis de ambiente estão definidas
if [ -z "$DATABASE_URL" ] || [ -z "$JWT_SECRET" ]; then
  echo "❌ Variáveis de ambiente obrigatórias não definidas"
  exit 1
fi

# Executar migrations
echo "📊 Executando migrations do banco de dados..."
npx prisma migrate deploy

if [ $? -ne 0 ]; then
  echo "❌ Erro ao executar migrations"
  exit 1
fi

# Gerar cliente Prisma
echo "🔧 Gerando cliente Prisma..."
npx prisma generate

# Build da aplicação
echo "🏗️ Fazendo build da aplicação..."
npm run build

if [ $? -ne 0 ]; then
  echo "❌ Erro no build da aplicação"
  exit 1
fi

# Executar testes de smoke
echo "🔥 Executando testes de smoke..."
npm run test:smoke

if [ $? -ne 0 ]; then
  echo "⚠️ Testes de smoke falharam, mas continuando deploy..."
fi

echo "✅ Deploy do backend concluído com sucesso!"
```

5.3.2. Deploy do Frontend (Vercel/Netlify)

Configuração para Vercel (vercel.json):

```
{
  "version": 2,
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/next"
    }
  ],
  "routes": [
    {
      "src": "/api/(.*)",
      "dest": "https://your-backend-url.railway.app/api/$1"
    }
  ],
  "env": {
    "NEXT_PUBLIC_API_BASE_URL": "https://your-backend-url.railway.app/api"
  },
  "functions": {
    "app/api/**/*.js": {
      "maxDuration": 30
    }
  }
}
```

Script de Deploy Frontend (deploy-frontend.sh):

```
#!/bin/bash

echo "🚀 Iniciando deploy do frontend..."

# Verificar variáveis de ambiente
if [ -z "$NEXT_PUBLIC_API_BASE_URL" ]; then
    echo "❌ NEXT_PUBLIC_API_BASE_URL não definida"
    exit 1
fi

# Instalar dependências
echo "📦 Instalando dependências..."
npm ci

# Executar testes
echo "🔧 Executando testes..."
npm run test:ci

if [ $? -ne 0 ]; then
    echo "❌ Testes falharam"
    exit 1
fi

# Build da aplicação
echo "🔨 Fazendo build da aplicação..."
npm run build

if [ $? -ne 0 ]; then
    echo "❌ Erro no build da aplicação"
    exit 1
fi

# Deploy para Vercel
echo "🌐 Fazendo deploy para Vercel..."
vercel --prod --confirm

if [ $? -ne 0 ]; then
    echo "❌ Erro no deploy para Vercel"
    exit 1
fi

echo "✅ Deploy do frontend concluído com sucesso!"
```

5.4. Configuração de Domínio e SSL

5.4.1. Configuração de DNS

Registros DNS necessários:

```
# Domínio principal
A      @          192.168.1.100  (IP do servidor)
CNAME  www        yourdomain.com
CNAME  api        your-backend-url.railway.app

# Subdomínios
CNAME  app        your-frontend-url.vercel.app
CNAME  admin      your-admin-panel-url.vercel.app

# Email
MX      @          10 mail.yourdomain.com
TXT     @          "v=spf1 include:_spf.google.com ~all"
TXT     _dmarc     "v=DMARC1; p=none; rua=mailto:dmarc@yourdomain.com"
```

5.4.2. Configuração SSL/TLS

Nginx Configuration (nginx.conf):

```

server {
    listen 80;
    server_name yourdomain.com www.yourdomain.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name yourdomain.com www.yourdomain.com;

    ssl_certificate /etc/nginx/ssl/yourdomain.com.crt;
    ssl_certificate_key /etc/nginx/ssl/yourdomain.com.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-
RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    # Security headers
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains"
always;
    add_header X-Frame-Options DENY always;
    add_header X-Content-Type-Options nosniff always;
    add_header X-XSS-Protection "1; mode=block" always;

    # API proxy
    location /api/ {
        proxy_pass http://backend:3001/api/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }

    # Frontend
    location / {
        proxy_pass http://frontend:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}

```

5.5. Configuração de Monitoramento e Logs

5.5.1. Monitoramento de Aplicação

Configuração do DataDog (backend):


```

// src/monitoring/datadog.ts
import { StatsD } from 'node-statsd';
import tracer from 'dd-trace';

// Inicializar tracer
tracer.init({
  service: 'music-school-backend',
  env: process.env.NODE_ENV,
  version: process.env.APP_VERSION
});

// Configurar StatsD
const statsd = new StatsD({
  host: process.env.DD_AGENT_HOST || 'localhost',
  port: 8125,
  prefix: 'music_school.'
});

export class MonitoringService {
  static recordMetric(name: string, value: number, tags?: string[]) {
    statsd.gauge(name, value, tags);
  }

  static incrementCounter(name: string, tags?: string[]) {
    statsd.increment(name, 1, tags);
  }

  static recordTiming(name: string, duration: number, tags?: string[]) {
    statsd.timing(name, duration, tags);
  }

  static recordApiCall(method: string, endpoint: string, statusCode: number,
    duration: number) {
    const tags = [
      `method:${method}`,
      `endpoint:${endpoint}`,
      `status_code:${statusCode}`
    ];

    this.incrementCounter('api.requests', tags);
    this.recordTiming('api.response_time', duration, tags);
  }
}

// Middleware para monitoramento de APIs
export const monitoringMiddleware = (req: any, res: any, next: any) => {
  const startTime = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - startTime;
    MonitoringService.recordApiCall(
      req.method,
      req.route?.path || req.path,
      res.statusCode,
      duration
    );
  });

  next();
};

```

5.5.2. Logging Estruturado

Configuração do Winston (backend):

```
// src/utils/logger.ts
import winston from 'winston';

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  defaultMeta: {
    service: 'music-school-backend',
    version: process.env.APP_VERSION
  },
  transports: [
    new winston.transports.File({
      filename: 'logs/error.log',
      level: 'error'
    }),
    new winston.transports.File({
      filename: 'logs/combined.log'
    })
  ]
});

// Em desenvolvimento, também log no console
if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.combine(
      winston.format.colorize(),
      winston.format.simple()
    )
  }));
}

// Em produção, enviar logs para serviço externo
if (process.env.NODE_ENV === 'production') {
  logger.add(new winston.transports.Http({
    host: process.env.LOG_SERVICE_HOST,
    port: process.env.LOG_SERVICE_PORT,
    path: '/logs'
  }));
}

export default logger;
```

5.6. Configuração de Backup e Recuperação

5.6.1. Backup do Banco de Dados

Script de Backup (scripts/backup-database.sh):

```
#!/bin/bash

# Configurações
BACKUP_DIR="/backups/database"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="music_school_backup_${DATE}.sql"
RETENTION_DAYS=30

# Criar diretório se não existir
mkdir -p $BACKUP_DIR

# Fazer backup
echo "📁 Iniciando backup do banco de dados..."
pg_dump $DATABASE_URL > "${BACKUP_DIR}/${BACKUP_FILE}"

if [ $? -eq 0 ]; then
    echo "✅ Backup criado com sucesso: ${BACKUP_FILE}"

    # Comprimir backup
    gzip "${BACKUP_DIR}/${BACKUP_FILE}"
    echo "📦 Backup comprimido: ${BACKUP_FILE}.gz"

    # Upload para S3 (opcional)
    if [ ! -z "$AWS_S3_BACKUP_BUCKET" ]; then
        aws s3 cp "${BACKUP_DIR}/${BACKUP_FILE}.gz"
"s3://${AWS_S3_BACKUP_BUCKET}/database/"
        echo "📤 Backup enviado para S3"
    fi

    # Remover backups antigos
    find $BACKUP_DIR -name "*.gz" -mtime +$RETENTION_DAYS -delete
    echo "🧹 Backups antigos removidos"

else
    echo "❌ Erro ao criar backup"
    exit 1
fi
```

5.6.2. Plano de Recuperação de Desastres

Procedimento de Recuperação:

```
#!/bin/bash
# scripts/restore-database.sh

BACKUP_FILE=$1

if [ -z "$BACKUP_FILE" ]; then
    echo "❌ Uso: $0 <backup_file>"
    exit 1
fi

echo "⚠️ ATENÇÃO: Este processo irá substituir todos os dados atuais!"
echo "Arquivo de backup: $BACKUP_FILE"
read -p "Deseja continuar? (yes/no): " confirm

if [ "$confirm" != "yes" ]; then
    echo "Operação cancelada"
    exit 0
fi

# Parar aplicação
echo "🛑 Parando aplicação..."
docker-compose down

# Restaurar backup
echo "🔄 Restaurando backup..."
if [[ $BACKUP_FILE == *.gz ]]; then
    gunzip -c $BACKUP_FILE | psql `DATABASE_URL`
else
    psql $DATABASE_URL < $BACKUP_FILE
fi

if [ $? -eq 0 ]; then
    echo "✅ Backup restaurado com sucesso"

    # Reiniciar aplicação
    echo "🚀 Reiniciando aplicação..."
    docker-compose up -d

    echo "✅ Sistema restaurado e funcionando"
else
    echo "❌ Erro ao restaurar backup"
    exit 1
fi
```

5.7. Estratégia de Lançamento

5.7.1. Lançamento Gradual (Canary Deployment)

Configuração de Feature Flags:

```

// src/utils/featureFlags.ts
interface FeatureFlags {
  newDashboard: boolean;
  premiumFeatures: boolean;
  chatIntegration: boolean;
  paymentV2: boolean;
}

class FeatureFlagService {
  private flags: FeatureFlags;

  constructor() {
    this.flags = {
      newDashboard: process.env.FEATURE_NEW_DASHBOARD === 'true',
      premiumFeatures: process.env.FEATURE_PREMIUM === 'true',
      chatIntegration: process.env.FEATURE_CHAT === 'true',
      paymentV2: process.env.FEATURE_PAYMENT_V2 === 'true'
    };
  }

  isEnabled(flag: keyof FeatureFlags, userId?: string): boolean {
    // Lógica para habilitar features gradualmente
    if (userId && this.isUserInBeta(userId)) {
      return true;
    }

    // Habilitar para porcentagem de usuários
    if (userId && this.shouldEnableForUser(flag, userId)) {
      return this.flags[flag];
    }

    return this.flags[flag];
  }

  private isUserInBeta(userId: string): boolean {
    // Lista de usuários beta
    const betaUsers = process.env.BETA_USERS?.split(',') || [];
    return betaUsers.includes(userId);
  }

  private shouldEnableForUser(flag: keyof FeatureFlags, userId: string):
boolean {
    // Hash do userId para distribuição consistente
    const hash = this.hashUserId(userId);
    const percentage = this.getFeaturePercentage(flag);
    return hash % 100 < percentage;
  }

  private hashUserId(userId: string): number {
    let hash = 0;
    for (let i = 0; i < userId.length; i++) {
      const char = userId.charCodeAt(i);
      hash = ((hash << 5) - hash) + char;
      hash = hash & hash; // Convert to 32bit integer
    }
    return Math.abs(hash);
  }

  private getFeaturePercentage(flag: keyof FeatureFlags): number {
    const percentages = {
      newDashboard: 25,

```

```

    premiumFeatures: 10,
    chatIntegration: 50,
    paymentV2: 5
  };
  return percentages[flag] || 0;
}
}

export const featureFlags = new FeatureFlagService();

```

5.7.2. Checklist de Lançamento

Pre-Launch Checklist:

Checklist de Pré-Lançamento

Infraestrutura

- [] Ambientes de produção configurados
- [] DNS configurado e propagado
- [] SSL/TLS configurado e testado
- [] Load balancer configurado
- [] CDN configurado
- [] Backup automático configurado

Aplicação

- [] Todos os testes passando
- [] Performance testada e otimizada
- [] Segurança auditada
- [] Logs e monitoramento configurados
- [] Feature flags implementadas
- [] Documentação da API atualizada

Banco de Dados

- [] Migrations executadas
- [] Índices otimizados
- [] Backup testado
- [] Plano de recuperação documentado

Integrações

- [] Gateway de pagamento testado
- [] Serviços de email configurados
- [] Armazenamento de mídia testado
- [] APIs de terceiros testadas

Monitoramento

- [] Alertas configurados
- [] Dashboards criados
- [] Logs centralizados
- [] Health checks funcionando

Equipe

- [] Runbook de produção criado
- [] Equipe treinada nos procedimentos
- [] Contatos de emergência definidos
- [] Plano de comunicação preparado

5.7.3. Plano de Rollback

Procedimento de Rollback:

```
#!/bin/bash
# scripts/rollback.sh

VERSION_TO_ROLLBACK=$1

if [ -z "$VERSION_TO_ROLLBACK" ]; then
    echo "❌ Uso: $0 <version>"
    echo "Versões disponíveis:"
    git tag -l | tail -10
    exit 1
fi

echo "⚠️ ROLLBACK para versão: $VERSION_TO_ROLLBACK"
read -p "Confirma o rollback? (yes/no): " confirm

if [ "$confirm" != "yes" ]; then
    echo "Rollback cancelado"
    exit 0
fi

# 1. Rollback do frontend
echo "🔄 Fazendo rollback do frontend..."
vercel rollback $VERSION_TO_ROLLBACK --confirm

# 2. Rollback do backend
echo "🔄 Fazendo rollback do backend..."
git checkout $VERSION_TO_ROLLBACK

# 3. Rebuild e redeploy
docker-compose down
docker-compose build
docker-compose up -d

# 4. Verificar saúde do sistema
echo "🏠 Verificando saúde do sistema..."
sleep 30

curl -f http://localhost:3001/api/health
if [ $? -eq 0 ]; then
    echo "✅ Rollback concluído com sucesso"
else
    echo "❌ Erro após rollback - verificar logs"
    exit 1
fi

# 5. Notificar equipe
echo "📢 Notificando equipe sobre rollback..."
# Integração com Slack/Discord/Email
```

5.8. Pós-Lançamento

5.8.1. Monitoramento Inicial

Métricas Críticas para Monitorar: - Tempo de resposta das APIs (< 500ms) - Taxa de erro (< 1%) - Disponibilidade (> 99.9%) - Uso de recursos (CPU < 70%, Memória < 80%) - Transações de pagamento (taxa de sucesso > 95%)

5.8.2. Coleta de Feedback

Implementação de Feedback Widget:


```

// src/components/FeedbackWidget.tsx
import { useState } from 'react';
import api from '@lib/api';

export default function FeedbackWidget() {
  const [isOpen, setIsOpen] = useState(false);
  const [feedback, setFeedback] = useState('');
  const [rating, setRating] = useState(0);

  const submitFeedback = async () => {
    try {
      await api.post('/feedback', {
        rating,
        message: feedback,
        page: window.location.pathname,
        userAgent: navigator.userAgent
      });

      setIsOpen(false);
      setFeedback('');
      setRating(0);

      // Mostrar mensagem de sucesso
    } catch (error) {
      console.error('Erro ao enviar feedback:', error);
    }
  };

  return (
    <div className="fixed bottom-4 right-4 z-50">
      {!isOpen ? (
        <button
          onClick={() => setIsOpen(true)}
          className="bg-blue-600 text-white p-3 rounded-full shadow-lg
hover:bg-blue-700"
        >
          ... Feedback
        </button>
      ) : (
        <div className="bg-white p-4 rounded-lg shadow-xl w-80">
          <h3 className="font-semibold mb-3">Como foi sua experiência?</h3>

          <div className="flex mb-3">
            {[1, 2, 3, 4, 5].map((star) => (
              <button
                key={star}
                onClick={() => setRating(star)}
                className={`text-2xl ${star <= rating ? 'text-yellow-400' :
'text-gray-300'}`}
              >
                ★
              </button>
            ))}
          </div>

          <textarea
            value={feedback}
            onChange={(e) => setFeedback(e.target.value)}
            placeholder="Conte-nos sobre sua experiência..."
            className="w-full p-2 border rounded mb-3"
            rows={3}
          />
        </div>
      )}
    </div>
  );
}

```

```

/>

<div className="flex justify-end space-x-2">
  <button
    onClick={() => setIsOpen(false)}
    className="px-3 py-1 text-gray-600 hover:text-gray-800"
  >
    Cancelar
  </button>
  <button
    onClick={submitFeedback}
    className="px-4 py-2 bg-blue-600 text-white rounded hover:bg-
blue-700"
  >
    Enviar
  </button>
</div>
</div>
  )}
</div>
  );
}

```

Esta fase de implantação e lançamento garante que o sistema seja colocado em produção de forma segura, monitorada e com capacidade de resposta rápida a problemas. O foco está na estabilidade, performance e experiência do usuário desde o primeiro dia de operação.

Fase 6: Monitoramento e Manutenção

Esta fase é contínua e envolve o monitoramento constante do sistema em produção, a manutenção preventiva e corretiva, além do planejamento de melhorias e novas funcionalidades baseadas no feedback dos usuários.

6.1. Estratégia de Monitoramento

6.1.1. Monitoramento de Infraestrutura

Métricas de Sistema: - **CPU:** Utilização média, picos, tendências - **Memória:** Uso de RAM, vazamentos de memória - **Disco:** Espaço disponível, I/O, latência - **Rede:** Throughput, latência, perda de pacotes - **Banco de Dados:** Conexões ativas, queries lentas, locks

Configuração de Alertas:

```
# alerts.yml
alerts:
- name: "High CPU Usage"
  condition: "cpu_usage > 80%"
  duration: "5m"
  severity: "warning"
  channels: ["slack", "email"]

- name: "Critical CPU Usage"
  condition: "cpu_usage > 95%"
  duration: "2m"
  severity: "critical"
  channels: ["slack", "email", "sms"]

- name: "Memory Usage High"
  condition: "memory_usage > 85%"
  duration: "5m"
  severity: "warning"

- name: "Database Connection Pool Full"
  condition: "db_connections > 90%"
  duration: "1m"
  severity: "critical"

- name: "API Response Time High"
  condition: "api_response_time > 1000ms"
  duration: "3m"
  severity: "warning"

- name: "Error Rate High"
  condition: "error_rate > 5%"
  duration: "2m"
  severity: "critical"
```

6.1.2. Monitoramento de Aplicação

Métricas de Negócio:

```

// src/monitoring/businessMetrics.ts
import { MonitoringService } from '../datadog';

export class BusinessMetrics {
  static trackUserRegistration(userType: 'aluno' | 'professor') {
    MonitoringService.incrementCounter('user.registration',
    [`type:${userType}`]);
  }

  static trackLessonBooking(amount: number, paymentMethod: string) {
    MonitoringService.incrementCounter('lesson.booked');
    MonitoringService.recordMetric('lesson.revenue', amount);
    MonitoringService.incrementCounter('payment.method',
    [`method:${paymentMethod}`]);
  }

  static trackSearchQuery(instrument: string, resultsCount: number) {
    MonitoringService.incrementCounter('search.query',
    [`instrument:${instrument}`]);
    MonitoringService.recordMetric('search.results', resultsCount,
    [`instrument:${instrument}`]);
  }

  static trackPremiumSubscription(planId: string, amount: number) {
    MonitoringService.incrementCounter('premium.subscription',
    [`plan:${planId}`]);
    MonitoringService.recordMetric('premium.revenue', amount);
  }

  static trackUserEngagement(action: string, userId: string) {
    MonitoringService.incrementCounter('user.engagement',
    [`action:${action}`]);
  }
}

// Middleware para tracking automático
export const businessMetricsMiddleware = (req: any, res: any, next: any) => {
  // Track API usage
  res.on('finish', () => {
    if (req.path.includes('/api/professors') && req.method === 'GET') {
      BusinessMetrics.trackSearchQuery(
        req.query.instrumento || 'all',
        res.locals.resultsCount || 0
      );
    }

    if (req.path.includes('/api/schedule/book') && req.method === 'POST') {
      BusinessMetrics.trackLessonBooking(
        res.locals.lessonAmount || 0,
        res.locals.paymentMethod || 'unknown'
      );
    }
  });
  next();
};

```

6.1.3. Monitoramento de Frontend

Configuração do LogRocket:

```

// src/monitoring/frontend.ts
import LogRocket from 'logrocket';
import setupLogRocketReact from 'logrocket-react';

// Inicializar LogRocket apenas em produção
if (process.env.NODE_ENV === 'production') {
  LogRocket.init(process.env.NEXT_PUBLIC_LOGROCKET_APP_ID!);
  setupLogRocketReact(LogRocket);
}

export class FrontendMonitoring {
  static identify(user: { id: string; email: string; name: string }) {
    if (process.env.NODE_ENV === 'production') {
      LogRocket.identify(user.id, {
        email: user.email,
        name: user.name
      });
    }
  }

  static trackEvent(event: string, properties?: Record<string, any>) {
    if (process.env.NODE_ENV === 'production') {
      LogRocket.track(event, properties);
    }

    // Também enviar para analytics
    if (typeof window !== 'undefined' && window.gtag) {
      window.gtag('event', event, properties);
    }
  }

  static captureException(error: Error, context?: Record<string, any>) {
    if (process.env.NODE_ENV === 'production') {
      LogRocket.captureException(error);
    }

    console.error('Frontend Error:', error, context);
  }
}

// Hook para monitoramento de performance
export const usePerformanceMonitoring = () => {
  useEffect(() => {
    // Monitorar Core Web Vitals
    if (typeof window !== 'undefined' && 'performance' in window) {
      const observer = new PerformanceObserver((list) => {
        list.getEntries().forEach((entry) => {
          if (entry.entryType === 'navigation') {
            const navEntry = entry as PerformanceNavigationTiming;
            FrontendMonitoring.trackEvent('page_load_time', {
              loadTime: navEntry.loadEventEnd - navEntry.loadEventStart,
              domContentLoaded: navEntry.domContentLoadedEventEnd -
                navEntry.domContentLoadedEventStart,
              firstContentfulPaint: navEntry.loadEventEnd - navEntry.fetchStart
            });
          }
        });
      });

      observer.observe({ entryTypes: ['navigation'] });
    }
  });
}

```

```
}, []);  
};
```

6.2. Dashboards e Relatórios

6.2.1. Dashboard Operacional

Configuração do Grafana:

```
{  
  "dashboard": {  
    "title": "Music School Platform - Operations",  
    "panels": [  
      {  
        "title": "API Response Times",  
        "type": "graph",  
        "targets": [  
          {  
            "expr": "avg(api_response_time) by (endpoint)",  
            "legendFormat": "{{endpoint}}"  
          }  
        ]  
      },  
      {  
        "title": "Error Rate",  
        "type": "singlestat",  
        "targets": [  
          {  
            "expr": "rate(api_errors_total[5m]) * 100"  
          }  
        ]  
      },  
      {  
        "title": "Active Users",  
        "type": "graph",  
        "targets": [  
          {  
            "expr": "active_users_total"  
          }  
        ]  
      },  
      {  
        "title": "Database Connections",  
        "type": "graph",  
        "targets": [  
          {  
            "expr": "database_connections_active"  
          }  
        ]  
      }  
    ]  
  }  
}
```

6.2.2. Dashboard de Negócio

Métricas de Negócio:


```

// src/analytics/businessDashboard.ts
export interface BusinessMetrics {
  dailyActiveUsers: number;
  monthlyActiveUsers: number;
  newRegistrations: {
    students: number;
    teachers: number;
  };
  lessonsBooked: number;
  revenue: {
    lessons: number;
    premium: number;
    total: number;
  };
  conversionRates: {
    signupToFirstLesson: number;
    freeToPremium: number;
  };
};

export class BusinessDashboardService {
  static async getDailyMetrics(date: Date): Promise<BusinessMetrics> {
    const startOfDay = new Date(date.setHours(0, 0, 0, 0));
    const endOfDay = new Date(date.setHours(23, 59, 59, 999));

    const [
      dailyActiveUsers,
      newStudents,
      newTeachers,
      lessonsBooked,
      lessonRevenue,
      premiumRevenue
    ] = await Promise.all([
      this.getDailyActiveUsers(startOfDay, endOfDay),
      this.getNewRegistrations('aluno', startOfDay, endOfDay),
      this.getNewRegistrations('professor', startOfDay, endOfDay),
      this.getLessonsBooked(startOfDay, endOfDay),
      this.getLessonRevenue(startOfDay, endOfDay),
      this.getPremiumRevenue(startOfDay, endOfDay)
    ]);

    return {
      dailyActiveUsers,
      monthlyActiveUsers: await this.getMonthlyActiveUsers(date),
      newRegistrations: {
        students: newStudents,
        teachers: newTeachers
      },
      lessonsBooked,
      revenue: {
        lessons: lessonRevenue,
        premium: premiumRevenue,
        total: lessonRevenue + premiumRevenue
      },
      conversionRates: {
        signupToFirstLesson: await this.getSignupToFirstLessonRate(date),
        freeToPremium: await this.getFreeToPremiumRate(date)
      }
    };
  }
}

```

```

    private static async getDailyActiveUsers(start: Date, end: Date):
Promise<number> {
    return await prisma.usuarios.count({
        where: {
            ultimo_login: {
                gte: start,
                lte: end
            }
        }
    });
}

    private static async getNewRegistrations(type: string, start: Date, end:
Date): Promise<number> {
    return await prisma.usuarios.count({
        where: {
            tipo_perfil: type,
            data_cadastro: {
                gte: start,
                lte: end
            }
        }
    });
}

    // Implementar outros métodos...
}

```

6.3. Manutenção Preventiva

6.3.1. Rotinas de Manutenção

Script de Manutenção Diária:

```
#!/bin/bash
# scripts/daily-maintenance.sh

echo "🔧 Iniciando manutenção diária - $(date)"

# 1. Limpeza de logs antigos
echo "📄 Limpando logs antigos..."
find /var/log/app -name "*.log" -mtime +7 -delete
find /var/log/nginx -name "*.log" -mtime +30 -delete

# 2. Limpeza de cache
echo "🧹 Limpando cache..."
redis-cli FLUSHDB

# 3. Otimização do banco de dados
echo "🗄️ Otimizando banco de dados..."
psql $DATABASE_URL -c "VACUUM ANALYZE;"

# 4. Verificação de espaço em disco
echo "💾 Verificando espaço em disco..."
DISK_USAGE=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')
if [ $DISK_USAGE -gt 80 ]; then
    echo "⚠️ ALERTA: Uso de disco alto: ${DISK_USAGE}%"
    # Enviar notificação
    curl -X POST $SLACK_WEBHOOK_URL -d "{\"text\": \"⚠️ Uso de disco alto: ${DISK_USAGE}%\"}"
fi

# 5. Backup incremental
echo "💾 Executando backup incremental..."
./scripts/backup-database.sh

# 6. Verificação de saúde dos serviços
echo "🏥 Verificando saúde dos serviços..."
curl -f http://localhost:3001/api/health || echo "❌ Backend não está respondendo"
curl -f http://localhost:3000 || echo "❌ Frontend não está respondendo"

# 7. Atualização de estatísticas de negócio
echo "📊 Atualizando estatísticas..."
node scripts/update-business-stats.js

echo "✅ Manutenção diária concluída - $(date)"
```

6.3.2. Manutenção Semanal

Script de Manutenção Semanal:

```
#!/bin/bash
# scripts/weekly-maintenance.sh

echo "🔧 Iniciando manutenção semanal - $(date)"

# 1. Análise de performance
echo "📊 Analisando performance..."
node scripts/performance-analysis.js

# 2. Auditoria de segurança
echo "🔒 Executando auditoria de segurança..."
npm audit --audit-level high
docker scout cves

# 3. Limpeza de dados órfãos
echo "🧹 Limpando dados órfãos..."
psql $DATABASE_URL -f scripts/cleanup-orphaned-data.sql

# 4. Otimização de índices
echo "🗄️ Otimizando índices do banco..."
psql $DATABASE_URL -c "REINDEX DATABASE music_school;"

# 5. Verificação de integridade
echo "🔍 Verificando integridade dos dados..."
node scripts/data-integrity-check.js

# 6. Relatório semanal
echo "📄 Gerando relatório semanal..."
node scripts/generate-weekly-report.js

echo "✅ Manutenção semanal concluída - $(date)"
```

6.4. Gestão de Incidentes

6.4.1. Classificação de Incidentes

Níveis de Severidade: - **P1 - Crítico:** Sistema completamente indisponível - **P2 - Alto:** Funcionalidade principal afetada - **P3 - Médio:** Funcionalidade secundária afetada - **P4 - Baixo:** Problema cosmético ou de usabilidade

6.4.2. Procedimento de Resposta a Incidentes

Runbook de Incidentes:

Runbook de Resposta a Incidentes

P1 - Sistema Indisponível

Ações Imediatas (0-5 minutos)

1. Confirmar o incidente
2. Ativar página de status
3. Notificar equipe de plantão
4. Iniciar bridge de comunicação

Investigação (5-15 minutos)

1. Verificar logs de aplicação
2. Verificar métricas de infraestrutura
3. Verificar status de serviços externos
4. Identificar causa raiz

Resolução (15-60 minutos)

1. Implementar fix ou rollback
2. Verificar restauração do serviço
3. Monitorar estabilidade
4. Atualizar página de status

Pós-Incidente

1. Documentar cronologia
2. Realizar post-mortem
3. Implementar melhorias
4. Comunicar usuários

P2 - Funcionalidade Principal Afetada

Ações (0-10 minutos)

1. Confirmar impacto
2. Notificar equipe relevante
3. Iniciar investigação

Investigação (10-30 minutos)

1. Identificar escopo do problema
2. Verificar logs específicos
3. Reproduzir problema

Resolução (30-120 minutos)

1. Implementar correção
2. Testar funcionalidade
3. Monitorar métricas
4. Comunicar resolução

6.4.3. Automação de Resposta

Sistema de Auto-Healing:

```

// src/monitoring/autoHealing.ts
export class AutoHealingService {
  static async handleHighMemoryUsage() {
    console.log('🔴 Detectado uso alto de memória, iniciando limpeza...');

    // Limpar cache
    await this.clearCache();

    // Forçar garbage collection
    if (global.gc) {
      global.gc();
    }

    // Reiniciar workers se necessário
    if (await this.getMemoryUsage() > 0.9) {
      await this.restartWorkers();
    }
  }

  static async handleDatabaseConnectionIssues() {
    console.log('🔴 Problemas de conexão com banco detectados...');

    // Tentar reconectar
    await this.reconnectDatabase();

    // Se falhar, usar réplica de leitura
    if (!(await this.testDatabaseConnection())) {
      await this.switchToReadReplica();
    }
  }

  static async handleHighErrorRate() {
    console.log('🔴 Taxa de erro alta detectada...');

    // Ativar circuit breaker
    await this.activateCircuitBreaker();

    // Redirecionar tráfego
    await this.redirectTrafficToHealthyInstances();

    // Notificar equipe
    await this.notifyTeam('Alta taxa de erro detectada');
  }

  private static async clearCache() {
    // Implementar limpeza de cache
  }

  private static async getMemoryUsage(): Promise<number> {
    const usage = process.memoryUsage();
    return usage.heapUsed / usage.heapTotal;
  }

  private static async restartWorkers() {
    // Implementar reinicialização de workers
  }

  private static async reconnectDatabase() {
    // Implementar reconexão com banco
  }
}

```

```
private static async testDatabaseConnection(): Promise<boolean> {  
    // Testar conexão com banco  
    return true;  
}  
  
private static async switchToReadReplica() {  
    // Alternar para réplica de leitura  
}  
  
private static async activateCircuitBreaker() {  
    // Ativar circuit breaker  
}  
  
private static async redirectTrafficToHealthyInstances() {  
    // Redirecionar tráfego  
}  
  
private static async notifyTeam(message: string) {  
    // Notificar equipe via Slack/Discord  
}  
}
```

6.5. Planejamento de Capacidade

6.5.1. Análise de Crescimento

Script de Análise de Tendências:

```

// scripts/capacity-planning.ts
interface GrowthMetrics {
  userGrowthRate: number;
  lessonGrowthRate: number;
  dataGrowthRate: number;
  trafficGrowthRate: number;
}

class CapacityPlanningService {
  static async analyzeGrowthTrends(): Promise<GrowthMetrics> {
    const last30Days = await this.getMetricsForPeriod(30);
    const last60Days = await this.getMetricsForPeriod(60);

    return {
      userGrowthRate: this.calculateGrowthRate(
        last60Days.users,
        last30Days.users
      ),
      lessonGrowthRate: this.calculateGrowthRate(
        last60Days.lessons,
        last30Days.lessons
      ),
      dataGrowthRate: this.calculateGrowthRate(
        last60Days.dataSize,
        last30Days.dataSize
      ),
      trafficGrowthRate: this.calculateGrowthRate(
        last60Days.requests,
        last30Days.requests
      )
    };
  }

  static async projectCapacityNeeds(months: number):
  Promise<CapacityProjection> {
    const currentMetrics = await this.getCurrentMetrics();
    const growthTrends = await this.analyzeGrowthTrends();

    return {
      projectedUsers: this.projectMetric(
        currentMetrics.users,
        growthTrends.userGrowthRate,
        months
      ),
      projectedTraffic: this.projectMetric(
        currentMetrics.traffic,
        growthTrends.trafficGrowthRate,
        months
      ),
      projectedStorage: this.projectMetric(
        currentMetrics.storage,
        growthTrends.dataGrowthRate,
        months
      ),
      recommendedInfrastructure: this.calculateInfrastructureNeeds(
        // ... parâmetros de projeção
      )
    };
  }

  private static calculateGrowthRate(oldValue: number, newValue: number):

```



```

number {
    return ((newValue - oldValue) / oldValue) * 100;
}

private static projectMetric(current: number, growthRate: number, months:
number): number {
    return current * Math.pow(1 + (growthRate / 100), months);
}
}

```

6.5.2. Alertas de Capacidade

Configuração de Alertas:

```

# capacity-alerts.yml
alerts:
  - name: "Database Storage 80% Full"
    condition: "database_storage_usage > 80%"
    severity: "warning"
    action: "Consider scaling storage"

  - name: "CPU Usage Trending Up"
    condition: "avg_over_time(cpu_usage[7d]) > avg_over_time(cpu_usage[14d]) *
1.2"
    severity: "info"
    action: "Monitor for scaling needs"

  - name: "Memory Usage High Consistently"
    condition: "memory_usage > 75% for 24h"
    severity: "warning"
    action: "Scale memory or optimize application"

  - name: "Request Rate Growing Fast"
    condition: "rate(requests[1d]) > rate(requests[7d]) * 1.5"
    severity: "info"
    action: "Prepare for traffic scaling"

```

6.6. Atualizações e Melhorias

6.6.1. Processo de Atualização

Pipeline de Atualização:

```

# .github/workflows/update.yml
name: System Update

on:
  schedule:
    - cron: '0 2 * * 1' # Segunda-feira às 2h
  workflow_dispatch:

jobs:
  security-updates:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Check for security updates
        run: |
          npm audit --audit-level high
          npm audit fix

      - name: Update dependencies
        run: |
          npm update

      - name: Run tests
        run: |
          npm test

      - name: Create PR if changes
        uses: peter-evans/create-pull-request@v5
        with:
          title: 'Security and dependency updates'
          body: 'Automated security and dependency updates'
          branch: 'updates/security-deps'

  performance-optimization:
    runs-on: ubuntu-latest
    steps:
      - name: Analyze bundle size
        run: npm run analyze

      - name: Check for performance regressions
        run: npm run lighthouse-ci

      - name: Database optimization
        run: |
          npm run db:analyze
          npm run db:optimize

```

6.6.2. Roadmap de Melhorias

Planejamento Trimestral:

Roadmap Q1 2024

Performance

- ☐ Implementar cache Redis para queries frequentes
- ☐ Otimizar queries N+1 no backend
- ☐ Implementar lazy loading no frontend
- ☐ Comprimir imagens automaticamente

Funcionalidades

- ☐ Sistema de chat em tempo real
- ☐ Integração com calendário externo
- ☐ Notificações push
- ☐ App mobile (React Native)

Infraestrutura

- ☐ Migrar para Kubernetes
- ☐ Implementar CDN para assets
- ☐ Configurar multi-região
- ☐ Automatizar scaling horizontal

Segurança

- ☐ Implementar 2FA
- ☐ Auditoria de segurança externa
- ☐ Criptografia de dados sensíveis
- ☐ Rate limiting avançado

UX/UI

- ☐ Redesign do dashboard
- ☐ Modo escuro
- ☐ Acessibilidade (WCAG 2.1)
- ☐ PWA features

6.7. Feedback e Iteração

6.7.1. Coleta de Feedback

Sistema de Feedback Integrado:

```

// src/services/feedbackService.ts
export interface UserFeedback {
  id: string;
  userId: string;
  type: 'bug' | 'feature' | 'improvement' | 'complaint';
  category: string;
  title: string;
  description: string;
  priority: 'low' | 'medium' | 'high' | 'critical';
  status: 'open' | 'in_progress' | 'resolved' | 'closed';
  votes: number;
  createdAt: Date;
  updatedAt: Date;
}

export class FeedbackService {
  static async submitFeedback(feedback: Omit<UserFeedback, 'id' | 'votes' | 'createdAt' | 'updatedAt'>): Promise<UserFeedback> {
    const newFeedback = await prisma.feedback.create({
      data: {
        ...feedback,
        votes: 0,
        status: 'open'
      }
    });

    // Notificar equipe de produto
    await this.notifyProductTeam(newFeedback);

    // Categorizar automaticamente usando IA
    await this.categorizeFeedback(newFeedback.id);

    return newFeedback;
  }

  static async voteFeedback(feedbackId: string, userId: string): Promise<void> {
    await prisma.feedbackVote.upsert({
      where: {
        feedbackId_userId: {
          feedbackId,
          userId
        }
      },
      create: {
        feedbackId,
        userId
      },
      update: {}
    });

    // Atualizar contador de votos
    await prisma.feedback.update({
      where: { id: feedbackId },
      data: {
        votes: {
          increment: 1
        }
      }
    });
  }
}

```

```

static async getFeedbackTrends(): Promise<FeedbackTrends> {
  const last30Days = new Date(Date.now() - 30 * 24 * 60 * 60 * 1000);

  const [
    totalFeedback,
    bugReports,
    featureRequests,
    topCategories,
    sentimentAnalysis
  ] = await Promise.all([
    prisma.feedback.count({
      where: { createdAt: { gte: last30Days } }
    }),
    prisma.feedback.count({
      where: {
        type: 'bug',
        createdAt: { gte: last30Days }
      }
    }),
    prisma.feedback.count({
      where: {
        type: 'feature',
        createdAt: { gte: last30Days }
      }
    }),
    this.getTopCategories(last30Days),
    this.analyzeSentiment(last30Days)
  ]);

  return {
    totalFeedback,
    bugReports,
    featureRequests,
    topCategories,
    sentimentAnalysis
  };
}

private static async notifyProductTeam(feedback: UserFeedback): Promise<void>
{
  // Integração com Slack/Discord
  const message = `
  Novo feedback recebido
  **Tipo:** ${feedback.type}
  **Prioridade:** ${feedback.priority}
  **Título:** ${feedback.title}
  **Descrição:** ${feedback.description}
  `;

  await this.sendSlackMessage(process.env.PRODUCT_SLACK_CHANNEL!, message);
}

private static async categorizeFeedback(feedbackId: string): Promise<void> {
  // Usar IA para categorizar automaticamente
  // Implementação com OpenAI ou similar
}
}

```

6.7.2. Análise de Dados de Uso

Analytics de Comportamento:

```

// src/analytics/userBehavior.ts
export class UserBehaviorAnalytics {
  static async analyzeUserJourney(): Promise<UserJourneyInsights> {
    const journeyData = await prisma.$queryRaw`
      SELECT
        page_path,
        COUNT(*) as visits,
        AVG(time_on_page) as avg_time,
        AVG(bounce_rate) as bounce_rate,
        COUNT(DISTINCT user_id) as unique_users
      FROM page_views
      WHERE created_at >= NOW() - INTERVAL '30 days'
      GROUP BY page_path
      ORDER BY visits DESC
    `;

    return this.processJourneyData(journeyData);
  }

  static async identifyDropoffPoints(): Promise<DropoffAnalysis> {
    const funnelSteps = [
      'landing_page',
      'signup_page',
      'profile_completion',
      'first_search',
      'first_booking',
      'payment_completion'
    ];

    const dropoffData = await this.calculateFunnelDropoff(funnelSteps);
    return this.analyzeDropoffReasons(dropoffData);
  }

  static async analyzeFeatureUsage(): Promise<FeatureUsageReport> {
    const features = [
      'professor_search',
      'lesson_booking',
      'chat_system',
      'profile_editing',
      'payment_processing',
      'review_system'
    ];

    const usageData = await Promise.all(
      features.map(feature => this.getFeatureUsageStats(feature))
    );

    return {
      features: usageData,
      trends: await this.calculateUsageTrends(features),
      recommendations: await this.generateUsageRecommendations(usageData)
    };
  }

  private static async calculateFunnelDropoff(steps: string[]):
  Promise<FunnelData> {
    // Implementar cálculo de funil de conversão
    return {} as FunnelData;
  }

  private static async getFeatureUsageStats(feature: string):

```

```
Promise<FeatureStats> {  
    // Implementar estatísticas de uso de features  
    return {} as FeatureStats;  
}
```

Esta fase de monitoramento e manutenção é essencial para garantir que o sistema continue funcionando de forma otimizada, segura e alinhada com as necessidades dos usuários. O foco está na proatividade, na melhoria contínua e na capacidade de resposta rápida a problemas e oportunidades de crescimento.

Cronograma de Implementação

Timeline Detalhado

Fase 1: Planejamento Detalhado e Refinamento de Requisitos (2 semanas) -

Semana 1: Análise de requisitos e definição do MVP - Semana 2: Pesquisa de tecnologias e validação da arquitetura

Fase 2: Design Técnico e Arquitetura (3 semanas) -

Semana 3: Design da arquitetura do sistema - Semana 4: Modelagem do banco de dados e especificação das APIs - Semana 5: Criação de wireframes e fluxos de usuário

Fase 3: Desenvolvimento (8 semanas) -

Semanas 6-7: Configuração do ambiente e desenvolvimento do backend (autenticação, usuários) - Semanas 8-9: Desenvolvimento do backend (professores, busca, agendamento) - Semanas 10-11: Desenvolvimento do frontend (landing page, autenticação, dashboards) - Semanas 12-13: Desenvolvimento do frontend (busca, perfis, agendamento, pagamento)

Fase 4: Testes e Garantia de Qualidade (3 semanas) -

Semana 14: Testes unitários e de integração - Semana 15: Testes end-to-end e de performance - Semana 16: Testes de segurança e correção de bugs

Fase 5: Implantação e Lançamento (2 semanas) -

Semana 17: Configuração da infraestrutura e deploy em staging - Semana 18: Deploy em produção e lançamento beta

Fase 6: Monitoramento e Manutenção (Contínuo) -

A partir da semana 19: Monitoramento, coleta de feedback e melhorias iterativas

Duração Total: 18 semanas (aproximadamente 4,5 meses) + manutenção contínua

Recursos Necessários

Equipe Mínima Recomendada: - 1 Tech Lead/Arquiteto de Software - 2 Desenvolvedores Full-Stack (React/Node.js) - 1 Designer UI/UX - 1 DevOps/SRE - 1 QA Engineer - 1 Product Manager

Orçamento Estimado (Mensal): - Infraestrutura: R2.000 — R 5.000 - Serviços de Terceiros: R1.500 — R 3.000 - Ferramentas e Licenças: R1.000 — R 2.000 - **Total Operacional Mensal: R4.500 — R 10.000**

Custos de Desenvolvimento (One-time): - Equipe de desenvolvimento (4,5 meses): R 180.000 — R 300.000 - Design e UX: R20.000 — R 40.000 - Infraestrutura inicial: R 10.000 — R 20.000 - **Total de Desenvolvimento: R210.000 — R 360.000**

Riscos e Mitigações

Riscos Técnicos

Alto Risco: - **Integração com Gateway de Pagamento:** Complexidade na implementação e certificação PCI DSS - *Mitigação:* Usar soluções como Stripe que já são certificadas, implementar em fases - **Escalabilidade do Banco de Dados:** Performance pode degradar com crescimento - *Mitigação:* Usar Neon Database com auto-scaling, implementar cache Redis

Médio Risco: - **Performance do Frontend:** Aplicação pode ficar lenta com muitos dados - *Mitigação:* Implementar lazy loading, paginação, otimização de bundle - **Segurança de Dados:** Vazamento de informações pessoais e financeiras - *Mitigação:* Criptografia end-to-end, auditoria de segurança, compliance LGPD

Riscos de Negócio

Alto Risco: - **Adoção de Usuários:** Professores e alunos podem não aderir à plataforma - *Mitigação:* MVP focado, feedback constante, programa de incentivos - **Concorrência:** Entrada de players estabelecidos no mercado - *Mitigação:* Diferenciação por nicho, foco na experiência do usuário

Médio Risco: - **Regulamentação:** Mudanças nas leis de pagamento ou educação - *Mitigação:* Acompanhamento jurídico, flexibilidade na arquitetura - **Sazonalidade:**

Demanda pode variar conforme período letivo - *Mitigação:* Diversificação de serviços, modelo de assinatura

Métricas de Sucesso

Métricas de Produto (Primeiros 6 meses)

Crescimento: - 500+ professores cadastrados - 2.000+ alunos registrados - 1.000+ aulas agendadas por mês - Taxa de crescimento mensal de usuários: 20%

Engajamento: - Taxa de retenção de usuários (30 dias): 60% - Tempo médio na plataforma: 15 minutos por sessão - Taxa de conversão de cadastro para primeira aula: 25% - NPS (Net Promoter Score): 50+

Financeiro: - Receita mensal recorrente: R\$ 50.000 - Ticket médio por aula: R\$ 80 - Taxa de conversão para planos premium: 15% - Margem de contribuição: 30%

Métricas Técnicas

Performance: - Tempo de carregamento da página: < 3 segundos - Tempo de resposta da API: < 500ms (95th percentile) - Uptime: > 99.5% - Taxa de erro: < 1%

Qualidade: - Cobertura de testes: > 80% - Vulnerabilidades de segurança: 0 críticas, < 5 médias - Bugs em produção: < 10 por mês - Tempo médio de resolução de bugs: < 24 horas

Próximos Passos

Imediatos (Semanas 1-2)

1. **Validação do Plano:** Revisar este documento com stakeholders
2. **Formação da Equipe:** Recrutar ou alocar desenvolvedores
3. **Setup Inicial:** Configurar repositórios, ferramentas de projeto
4. **Pesquisa de Mercado:** Validar premissas com potenciais usuários

Curto Prazo (Mês 1)

1. **Prototipagem:** Criar protótipos navegáveis das telas principais
2. **Arquitetura Técnica:** Finalizar decisões de tecnologia e infraestrutura
3. **MVP Definition:** Definir escopo exato do produto mínimo viável
4. **Parcerias:** Estabelecer parcerias com gateways de pagamento

Médio Prazo (Meses 2-4)

1. **Desenvolvimento:** Executar as fases de desenvolvimento conforme cronograma
2. **Testes Beta:** Recrutar usuários beta para feedback antecipado
3. **Marketing:** Preparar estratégia de lançamento e aquisição de usuários
4. **Legal:** Finalizar termos de uso, política de privacidade, compliance

Longo Prazo (Meses 5-12)

1. **Lançamento:** Executar estratégia de go-to-market
2. **Iteração:** Implementar melhorias baseadas em feedback
3. **Expansão:** Adicionar novas funcionalidades e mercados
4. **Escala:** Otimizar para crescimento e eficiência operacional

Conclusão

Este plano de implementação fornece um roadmap detalhado para o desenvolvimento de uma plataforma SaaS de escola de música robusta e escalável. O projeto está estruturado em fases bem definidas, com tecnologias modernas e práticas de desenvolvimento comprovadas.

Principais Diferenciais do Plano: - **Tecnologia Moderna:** React, Node.js, TypeScript, Neon Database - **Arquitetura Escalável:** Preparada para crescimento desde o início - **Foco na Experiência:** Dashboards personalizados e interface intuitiva - **Modelo de Negócio Validado:** Marketplace com múltiplas fontes de receita - **Qualidade Garantida:** Testes abrangentes e monitoramento contínuo

Fatores Críticos de Sucesso: 1. **Execução Disciplinada:** Seguir o cronograma e manter qualidade 2. **Feedback Constante:** Envolver usuários desde o início 3. **Equipe Qualificada:** Ter desenvolvedores experientes nas tecnologias escolhidas 4. **Foco no MVP:** Lançar rapidamente para validar o mercado 5. **Monitoramento Ativo:** Acompanhar métricas e ajustar estratégia

O sucesso deste projeto dependerá da execução cuidadosa de cada fase, mantendo sempre o foco na experiência do usuário e na criação de valor real para professores e alunos de música. Com o plano detalhado apresentado, a equipe terá todas as informações necessárias para transformar esta visão em realidade.

Documento preparado por: Manus AI

Data: Janeiro 2024

Versão: 1.0

Status: Pronto para Implementação