Curso de graduação tecnológica em Analise e Desenvolvimento de Sistemas

Projeto integrador Programação Web

Lucas da Silva Damaso

Documentação Técnica de Sistema – Damaso Barber

Polo Educacional de Embu Guaçu - SP 2025

Lucas da Silva Damaso

Damaso Barber

Trabalho de Projeto Integrador apresentado como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, no Curso de Graduação Tecnológica em Análise e Desenvolvimento de Sistemas da Universidade Santo Amaro – UNISA.

Luis Fernando dos Santos Pires

Link do Repositorio: https://github.com/luucdamaso/damasoBarber

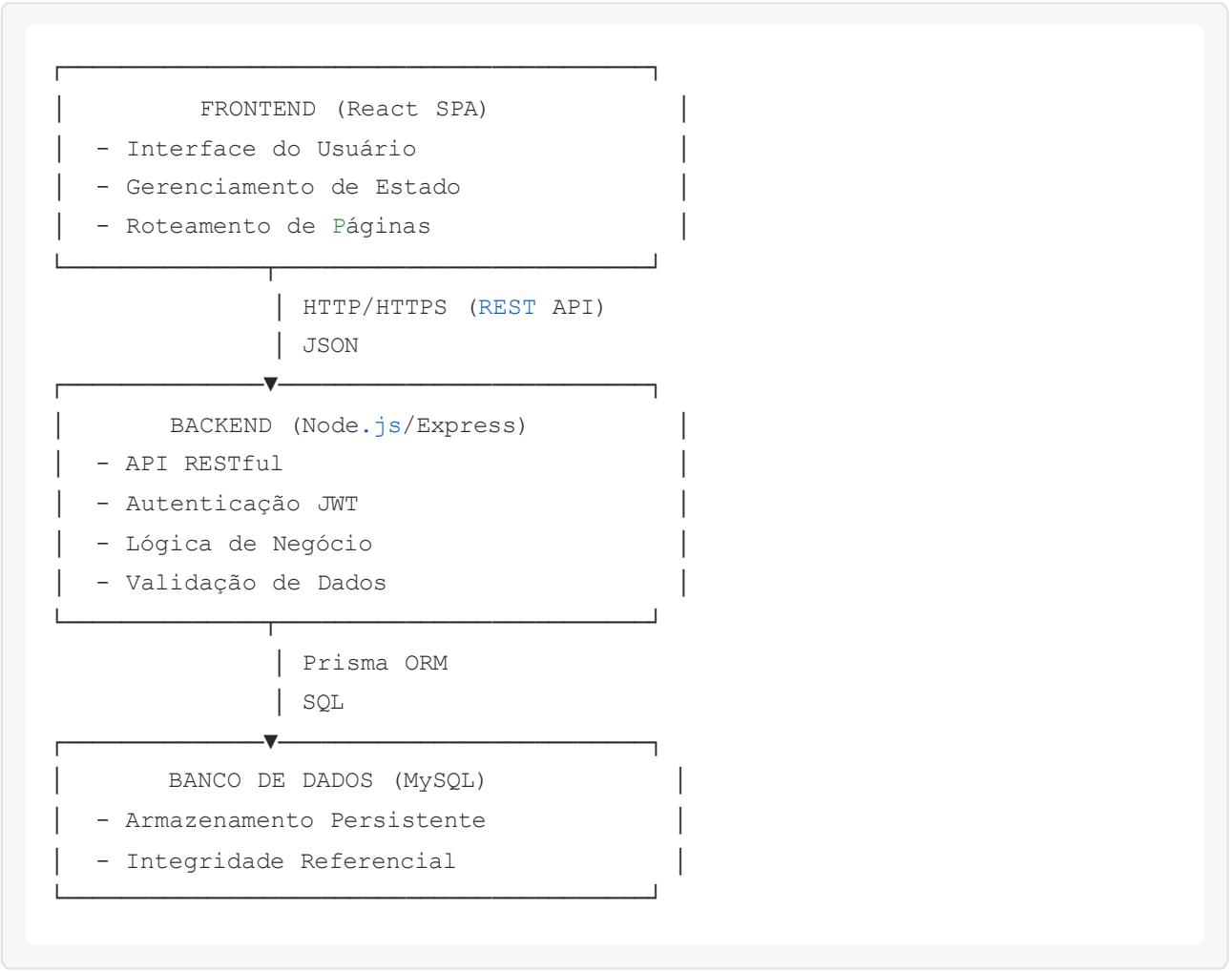
Polo Educacional de Embu Guaçu - SP 2025

Sumário

1. Visão Geral da Arquitetura

O sistema Damaso Barber foi desenvolvido seguindo uma arquitetura de três camadas (three-tier architecture), com separação clara entre apresentação, lógica de negócio e persistência de dados.

1.1 Arquitetura Geral



2. Backend - Estrutura e Componentes

2.1 Estrutura de Diretórios

```
backend/
├── prisma/
│   ├── migrations/          # Histórico de migrações do banco
│   └── schema.prisma        # Schema do banco de dados
├── src/
│   ├── controllers/         # Controladores das rotas
│   │   ├── auth.controller.js
│   │   ├── bookings.controller.js
│   │   ├── clients.controller.js
│   │   └── services.controller.js
│   ├── database/
│   │   └── prismaClient.js  # Instância do Prisma Client
│   ├── middlewares/         # Middlewares da aplicação
│   │   ├── auth.middleware.js
│   │   ├── admin.middleware.js
│   │   ├── error.middleware.js
│   │   ├── jsonError.middleware.js
│   │   └── logger.middleware.js
│   ├── routes/              # Definição de rotas
│   │   ├── index.js
│   │   ├── auth.routes.js
│   │   ├── bookings.routes.js
│   │   ├── clientes.routes.js
│   │   ├── health.routes.js
│   │   └── services.routes.js
│   ├── services/            # Lógica de negócio
│   │   └── booking.service.js
│   ├── utils/               # Utilitários
│   │   ├── jwt.js
│   │   └── validators.js
│   ├── seed.js              # Script de população do banco
│   └── server.js            # Ponto de entrada da aplicação
├── .env                     # Variáveis de ambiente
└── package.json             # Dependências e scripts
```

2.2 Fluxo de RequisiÇão

```
Cliente HTTP Request
    ↓
Express Middleware Chain
    ↓
1. CORS Middleware (cors)
    ↓
2. JSON Parser (express.json)
    ↓
3. JSON Error Handler (jsonError.middleware)
    ↓
4. Logger Middleware (logger.middleware)
    ↓
5. Router (/api)
    ↓
6. Auth Middleware (auth.middleware) [rotas protegidas]
    ↓
7. Controller (processa requisição)
    ↓
8. Service (lógica de negócio) [opcional]
    ↓
9. Prisma Client (acesso ao banco)
    ↓
10. Response (JSON)
    ↓
11. Error Handler (error.middleware) [em caso de erro]
```

2.3 Controladores (Controllers)

Os controladores são responsáveis por receber requisições HTTP, validar dados de entrada, invocar a lógica de negócio apropriada e retornar respostas HTTP.

Auth.controller.js

Responsável pela autenticação de usuários.

Principais funções:

- `register(req, res)`: Registra novo usuário
    - Valida dados de entrada (email, senha, nome)

- Verifica se email já existe
- Criptografa senha com bcrypt
- Cria usuário no banco
- Retorna token JWT

- `login(req, res)` : Autentica usuário existente
  - Valida credenciais
  - Compara senha com hash armazenado
  - Gera token JWT
  - Retorna token e dados do usuário

clients.controller.js

Gerencia operações CRUD de clientes.

Principais funções:

- `getAllClients(req, res)` : Lista todos os clientes
- `getClientById(req, res)` : Obtém detalhes de um cliente específico
- `createClient(req, res)` : Cria novo cliente
- `updateClient(req, res)` : Atualiza dados de cliente existente
- `deleteClient(req, res)` : Remove cliente do sistema

services.controller.js

Gerencia operações CRUD de serviços.

Principais funções:

- `getAllServices(req, res)` : Lista todos os serviços
- `getServiceById(req, res)` : Obtém detalhes de um serviço específico
- `createService(req, res)` : Cria novo serviço
- `` : Atualiza dados de serviço existente
- `deleteService(req, res)` : Remove serviço do sistema

bookings.controller.js

Gerencia operações CRUD de agendamentos.

Principais funções:

- `getAllBookings(req, res)` : Lista todos os agendamentos
- `getBookingById(req, res)` : Obtém detalhes de um agendamento específico
- `createBooking(req, res)` : Cria novo agendamento
- `updateBooking(req, res)` : Atualiza dados de agendamento existente
- `deleteBooking(req, res)` : Remove agendamento do sistema
- `updateBookingStatus(req, res)` : Atualiza status do agendamento

2.4 Middlewares

Auth.middleware.js

Middleware de autenticação que verifica a validade do token JWT.

Funcionamento:

1. Extrai token do header `Authorization`
2. Verifica se token existe
3. Valida token usando JWT_SECRET
4. Decodifica payload do token
5. Anexa dados do usuário ao objeto `req.user`
6. Passa para próximo middleware/controller

Uso:

```
router.get('/protected', authMiddleware, controller);
```

Admin.middleware.js

Middleware de autorização que verifica se usuário tem papel de administrador.

Funcionamento:

1. Verifica se `req.user` existe (requer auth.middleware antes)
2. Verifica se `req.user.role === 'admin'`
3. Se sim, passa para próximo middleware
4. Se não, retorna erro 403 Forbidden

error.middleware.js

Middleware de tratamento de erros global.

Funcionamento:

1. Captura erros lançados em qualquer parte da aplicação
2. Formata resposta de erro consistente
3. Log de erro para debugging
4. Retorna resposta HTTP apropriada

logger.middleware.js

Middleware de logging de requisições.

Funcionamento:

1. Registra método HTTP, URL e timestamp
2. Útil para debugging e monitoramento
3. Pode ser expandido para logging mais detalhado

2.5 Rotas (Routes)

auth.routes.js

```
POST /api/auth/register  - Registrar novo usuário
POST /api/auth/login     - Fazer login
```

clientes.routes.js

```
GET    /api/clients      - Listar clientes (protegido)
GET    /api/clients/:id - Obter cliente específico (protegido)
POST   /api/clients      - Criar cliente (protegido)
PUT    /api/clients/:id - Atualizar cliente (protegido)
DELETE /api/clients/:id - Deletar cliente (protegido)
```

services.routes.js

```
GET    /api/services      - Listar serviços (protegido)
GET    /api/services/:id - Obter serviço específico (protegido)
POST   /api/services      - Criar serviço (protegido)
PUT    /api/services/:id - Atualizar serviço (protegido)
DELETE /api/services/:id - Deletar serviço (protegido)
```

bookings.routes.js

```
GET    /api/bookings      - Listar agendamentos (protegido)
GET    /api/bookings/:id - Obter agendamento específico (protegido)
POST   /api/bookings      - Criar agendamento (protegido)
PUT    /api/bookings/:id - Atualizar agendamento (protegido)
DELETE /api/bookings/:id - Deletar agendamento (protegido)
PATCH  /api/bookings/:id/status - Atualizar status (protegido)
```

2.6 Prisma Schema

```
 // User: Usuários do sistema
model User {
  id         String   @id @default(uuid())
  email      String   @unique
  name       String?
  password  String
  role       String   @default("user")
  createdAt DateTime @default(now())
}

// Client: Clientes da barbearia
model Client {
  id         String    @id @default(uuid())
  name       String
  phone      String
  email      String?
  notes      String?
  bookings  Booking[]
  createdAt DateTime  @default(now())
}

// Service: Serviços oferecidos
model Service {
  id              String   @id @default(uuid())
  name            String
  price           Float
  durationMinutes Int?
  createdAt       DateTime @default(now())
}

// Booking: Agendamentos
model Booking {
  id         String   @id @default(uuid())
  client     Client   @relation(fields: [clientId], references: [id])
  clientId  String
  start      DateTime
  end        DateTime
  service    String
  status     String   @default("scheduled")
  createdAt DateTime @default(now())
}
```

2.7 Utilitários

jwt.js

Funções para geração e verificação de tokens JWT.

FunÇões:

- `generateToken(payload)` : Gera token JWT com payload fornecido
- `verifyToken(token)` : Verifica e decodifica token JWT

validators.js

Funções de validação de dados.

Funções:

- `validateEmail(email)` : Valida formato de email
- `validatePassword(password)` : Valida força de senha
- `validatePhone(phone)` : Valida formato de telefone
- `validateDate(date)` : Valida formato de data

3. Frontend - Estrutura e Componentes

3.1 Estrutura de Diretórios

```
frontend/client/src/
├── components/
│   ├── layout/
│   │   └── MainLayout.tsx       # Layout principal da aplicação
│   ├── ui/                      # Componentes de UI reutilizáveis
│   │   ├── button.tsx
│   │   ├── card.tsx
│   │   ├── dialog.tsx
│   │   ├── form.tsx
│   │   ├── input.tsx
│   │   ├── table.tsx
│   │   └── ... (outros componentes)
│   ├── ErrorBoundary.tsx        # Tratamento de erros
│   ├── ProtectedRoute.tsx       # Proteção de rotas
│   └── Map.tsx                  # Componente de mapa
├── contexts/
│   ├── AuthContext.tsx          # Contexto de autenticação
│   └── ThemeContext.tsx         # Contexto de tema
├── hooks/
│   ├── useAuth.ts               # Hook de autenticação
│   ├── useMobile.tsx            # Hook para detecção mobile
│   └── ... (outros hooks)
├── lib/
│   ├── api.ts                   # Configuração do Axios
│   └── utils.ts                 # Funções utilitárias
├── pages/
│   ├── Bookings/
│   │   ├── BookingList.tsx      # Lista de agendamentos
│   │   └── BookingForm.tsx      # Formulário de agendamento
│   ├── Clients/
│   │   ├── ClientList.tsx       # Lista de clientes
│   │   └── ClientForm.tsx       # Formulário de cliente
│   ├── Services/
│   │   ├── ServiceList.tsx      # Lista de serviços
│   │   └── ServiceForm.tsx      # Formulário de serviço
│   ├── Dashboard.tsx            # Dashboard principal
│   ├── Home.tsx                 # Página inicial
│   ├── Login.tsx                # Página de login
│   ├── Register.tsx             # Página de registro
│   └── NotFound.tsx             # Página 404
```

```
├── App.tsx                    # Componente raiz
└── main.tsx                   # Ponto de entrada
```

3.2 Contextos (Contexts)

AuthContext.tsx

Gerencia o estado de autenticação global da aplicação.

Estado:

- `user`: Dados do usuário logado
- `token`: Token JWT
- `isAuthenticated`: Boolean indicando se usuário está autenticado

Funções:

- `login(email, password)` : Autentica usuário
- `register(userData)` : Registra novo usuário
- `logout()`: Desautentica usuário
- `updateUser(userData)`: Atualiza dados do usuário

Persistência:

- Token armazenado em `localStorage`
- Recuperação automática ao carregar aplicação

ThemeContext.tsx

Gerencia o tema da aplicação (claro/escuro).

Estado:

- `theme`: Tema atual ("light" | "dark")

Funções:

- `setTheme(theme)` : Define tema

- `toggleTheme()`: Alterna entre temas

3.3 Hooks Customizados

useAuth.ts

Hook que fornece acesso ao contexto de autenticação.

```
const { user, token, isAuthenticated, login, logout } = useAuth();
```

3.4 Configuração de API (lib/api.ts)

Configuração centralizada do Axios para requisições HTTP.

Funcionalidades:

- Base URL configurada para o backend
- Interceptor de requisição que adiciona token JWT automaticamente
- Interceptor de resposta para tratamento de erros
- Tratamento de erro 401 (não autorizado) com logout automático

```
// Exemplo de uso
import api from '@/lib/api';

const response = await api.get('/clients');
const data = await api.post('/bookings', bookingData);
```

3.5 Páginas (Pages)

Login.tsx

Página de autenticação de usuários.

Funcionalidades:

- Formulário de login com validação

- Integração com AuthContext
- Redirecionamento após login bem-sucedido
- Tratamento de erros de autenticação

Dashboard.tsx

Página principal após login.

Funcionalidades:

- Visão geral das operações
- Estatísticas de agendamentos
- Próximos agendamentos
- Links rápidos para funcionalidades principais

ClientList.tsx

Lista de clientes cadastrados.

Funcionalidades:

- Tabela de clientes com paginação
- Busca e filtros
- Ações de editar e deletar
- Botão para adicionar novo cliente

ClientForm.tsx

Formulário de cadastro/edição de clientes.

Funcionalidades:

- Validação de campos com Zod
- Integração com React Hook Form
- Modo criação e edição
- Feedback de sucesso/erro

ServiceList.tsx e ServiceForm.tsx

Similares a ClientList e ClientForm, mas para serviços.

BookingList.tsx e BookingForm.tsx

Similares a ClientList e ClientForm, mas para agendamentos.

Funcionalidades adicionais em BookingForm:

- Seleção de cliente (dropdown)
- Seleção de serviço (dropdown)
- Seleção de data e hora
- Cálculo automático de horário de término baseado na duração do serviço

3.6 Componentes de UI

Os componentes de UI são baseados em Radix UI e estilizados com Tailwind CSS, fornecendo uma biblioteca consistente e acessível de componentes reutilizáveis.

Principais componentes:

- `Button`: Botões com variantes (primary, secondary, outline, ghost)
- `Card`: Containers para conteúdo
- `Dialog`: Modais e diálogos
- `Form`: Componentes de formulário integrados com React Hook Form
- `Input`: Campos de entrada de texto
- `Select`: Dropdowns de seleção
- `Table`: Tabelas de dados
- `Alert`: Alertas e notificações

3.7 Roteamento

O roteamento é gerenciado pela biblioteca Wouter.

Rotas definidas:

```
/ → Login (página inicial)
/login → Login
/register → Register
/dashboard → Dashboard (protegida)
/clients → ClientList (protegida)
/clients/:id → ClientForm (protegida)
/services → ServiceList (protegida)
/services/:id → ServiceForm (protegida)
/bookings → BookingList (protegida)
/bookings/:id → BookingForm (protegida)
/404 → NotFound
```

Proteção de rotas: O componente `ProtectedRoute` verifica se o usuário está autenticado antes de renderizar a rota. Se não estiver, redireciona para a página de login.

4. Fluxos de Dados

4.1 Fluxo de Autenticação

```
1. Usuário preenche formulário de login
   ↓
2. Frontend envia POST /api/auth/login
   ↓
3. Backend valida credenciais
   ↓
4. Backend gera token JWT
   ↓
5. Backend retorna token e dados do usuário
   ↓
6. Frontend armazena token em localStorage
   ↓
7. Frontend atualiza AuthContext
   ↓
8. Frontend redireciona para dashboard
   ↓
9. Requisições subsequentes incluem token no header
```

4.2 Fluxo de Criação de Agendamento

```
1. Usuário acessa página de novo agendamento
   ↓
2. Frontend carrega lista de clientes (GET /api/clients)
   ↓
3. Frontend carrega lista de serviços (GET /api/services)
   ↓
4. Usuário preenche formulário
   ↓
5. Frontend valida dados com Zod
   ↓
6. Frontend envia POST /api/bookings
   ↓
7. Backend valida dados
   ↓
8. Backend verifica disponibilidade de horário
   ↓
9. Backend cria agendamento no banco
   ↓
10. Backend retorna agendamento criado
    ↓
11. Frontend exibe mensagem de sucesso
    ↓
12. Frontend redireciona para lista de agendamentos
```

5. Segurança

5.1 Autenticação

- Senhas criptografadas com bcrypt (salt rounds: 10)
- Tokens JWT com expiração configurável
- Tokens armazenados em localStorage (frontend)
- Validação de token em todas as rotas protegidas

5.2 Autorização

- Middleware de autenticação verifica token

- Middleware de autorização verifica papel do usuário
- Rotas administrativas requerem role "admin"

5.3 Validçaão de Dados

- Validação no frontend com Zod
- Validação no backend antes de operações no banco
- Sanitização de inputs para prevenir SQL injection (Prisma)
- Validação de tipos e formatos

5.4 CORS

- Configuração de CORS para permitir requisições do frontend
- Pode ser configurado para aceitar apenas origens específicas em produção

5.5 Variáveis de Ambiente

- Credenciais sensíveis armazenadas em .env
- .env não versionado no Git
- JWT_SECRET deve ser forte e único

6. Boas Práticas Implementadas

6.1 Backend

- Separação de responsabilidades (controllers, services, routes)
- Middlewares para funcionalidades transversais
- Tratamento centralizado de erros
- Logging de requisições
- Uso de ORM para segurança e produtividade
- Validação de dados em múltiplas camadas

6.2 Frontend

- Componentização e reutilização de código
- Gerenciamento de estado com Context API
- Custom hooks para lógica reutilizável
- Validação de formulários com bibliotecas especializadas
- Feedback visual para ações do usuário
- Tratamento de erros com Error Boundaries
- Código TypeScript para type-safety

6.3 Geral

- Versionamento com Git
- Estrutura de diretórios organizada e intuitiva
- Nomenclatura consistente e descritiva
- Documentação de código e APIs
- Separação de configurações de desenvolvimento e produção

7. Melhorias Futuras

7.1 Funcionalidades

- Sistema de notificações (email/SMS)
- Integração com calendário (Google Calendar, Outlook)
- Relatórios e análises avançadas
- Sistema de avaliações e feedback
- Aplicativo móvel (React Native)
- Integração com sistemas de pagamento
- Sistema de fidelidade/pontos

7.2 Técnicas

- Testes automatizados (unitários, integração, e2e)
- CI/CD pipeline
- Containerização com Docker
- Deploy em cloud (AWS, Azure, Google Cloud)
- Monitoramento e logging avançado
- Cache de dados (Redis)
- Otimização de performance
- Internacionalização (i18n)
- Acessibilidade (WCAG compliance)

8. Troubleshooting

8.1 Problemas Comuns

Erro de conexão com banco de dados:

- Verificar se MySQL está rodando
- Verificar credenciais em .env
- Verificar se banco de dados existe
- Executar migrações: `npx prisma migrate dev`

Erro de autenticação:

- Verificar se JWT_SECRET está configurado
- Verificar se token está sendo enviado corretamente
- Verificar validade do token

Erro CORS:

- Verificar configuração de CORS no backend
- Verificar origem da requisição

Erro ao instalar dependências:

- Limpar cache: `npm cache clean --force`
- Deletar node_modules e reinstalar
- Verificar versão do Node.js

9. Referências Técnicas

- **Express.js**: https://expressjs.com/
- **React**: https://react.dev/
- **Prisma**: https://www.prisma.io/docs
- **Node.js**: https://nodejs.org/
- **JWT**: https://jwt.io/
- **Bcrypt**: https://github.com/kelektiv/node.bcrypt.js
- **Axios**: https://axios-http.com/
- **Radix UI**: https://www.radix-ui.com/
- **Tailwind CSS**: https://tailwindcss.com/
- **Vite**: https://vitejs.dev/
- **TypeScript**: https://www.typescriptlang.org/