Documentação Técnica - LiftHub Backend

Arquitetura e Padrões

Estrutura de Diretórios

```
backend/
├── controllers/         # Camada de apresentação
├── services/           # Camada de negócio
├── models/             # Camada de dados
├── routes/             # Definição de rotas
├── utils/              # Utilitários e helpers
├── config/             # Configurações
├── tests/              # Testes automatizados
└── scripts/            # Scripts auxiliares
```

Padrões Implementados

1. MVC (Model-View-Controller)

- **Models**: Definição dos esquemas MongoDB
- **Controllers**: Manipulação de requisições HTTP
- **Services**: Lógica de negócio isolada

2. Dependency Injection

- Services injetados nos controllers
- Facilita testes e manutenção

3. Error Handling

- Tratamento centralizado de erros
- Respostas padronizadas
- Logging adequado

Componentes Principais

1. Modelo do Aluno (`models/Aluno.js`)

```
const alunoSchema = new mongoose.Schema({
  cpf: {
    type: String,
    required: true,
    unique: true,
    trim: true,
    validate: {
      validator: function(v) {
        return /^\d{11}$/.test(v.replace(/\D/g, ''));
      },
      message: 'CPF deve conter 11 dígitos'
    }
  }
}, {
  timestamps: true
});
```

Características: - Validação automática de CPF - Índice único para performance - Timestamps automáticos - Trim para limpeza de dados

2. Validador de CPF (`utils/cpfValidator.js`)

```
const validarCPF = (cpf) => {
  // Implementação completa do algoritmo de validação
  // Inclui verificação de dígitos verificadores
};
```

Funcionalidades: - Validação completa de CPF brasileiro - Formatação para exibição - Limpeza de caracteres especiais - Rejeição de sequências inválidas

3. Service de Alunos (`services/alunoService.js`)

```
class AlunoService {
  async cadastrarAluno(cpf) { /* ... */ }
  async listarAlunos() { /* ... */ }
  async buscarAlunoPorCPF(cpf) { /* ... */ }
  async atualizarAluno(cpfAtual, novoCpf) { /* ... */ }
  async removerAluno(cpf) { /* ... */ }
}
```

Responsabilidades: - Lógica de negócio isolada - Validações de entrada - Interação com o banco de dados - Tratamento de erros específicos

4. Controller de Alunos (`controllers/alunoController.js`)

```
class AlunoController {
  async cadastrarAluno(req, res) {
    try {
      const { cpf } = req.body;
      const aluno = await alunoService.cadastrarAluno(cpf);
      res.status(201).json({
        success: true,
        message: 'Aluno cadastrado com sucesso',
        data: aluno
      });
    } catch (error) {
      res.status(400).json({
        success: false,
        message: error.message
      });
    }
  }
}
```

Características: - Validação de entrada - Delegação para services - Respostas padronizadas - Tratamento de erros HTTP

Testes Automatizados

Estrutura de Testes

```
tests/
├── cpfValidator.test.js     # Testes unitários do validador
├── alunoService.test.js     # Testes unitários do service
└── alunoController.test.js  # Testes de integração
```

Cobertura de Testes

1. Validação de CPF

- CPFs válidos e inválidos
- Formatação e limpeza
- Casos extremos (null, undefined)

2. Service de Alunos

- Operações CRUD completas
- Validações de negócio
- Tratamento de erros
- Mocks do banco de dados

3. Controllers

- Requisições HTTP
- Respostas padronizadas
- Códigos de status corretos
- Integração com services

Execução dos Testes

```
npm test                    # Executa todos os testes
npm test -- --coverage      # Executa com cobertura
npm test -- --watch         # Executa em modo watch
```

Configuração e Deploy

Variáveis de Ambiente

```
PORT=3001                                          # Porta do servidor
MONGODB_URI=mongodb://localhost:27017/lifthub # String de
conexão MongoDB
NODE_ENV=development                               # Ambiente de
execução
```

Scripts Disponíveis

```
{
  "start": "node server.js",        # Produção
  "dev": "nodemon server.js",       # Desenvolvimento
  "test": "jest"                    # Testes
}
```

Inicialização do Banco

```
node scripts/initDB.js
```

Funcionalidades do Script: - Criação da coleção de alunos - Configuração de índices - Verificação de conectividade - Logs informativos

Segurança e Validações

Validações Implementadas

1. **CPF**: Algoritmo completo de validação
2. **Entrada**: Sanitização de dados
3. **Unicidade**: Índices únicos no banco
4. **Tipos**: Validação de tipos de dados

Middleware de Segurança

- **CORS**: Configurado para desenvolvimento
- **JSON Parsing**: Limitação de tamanho
- **Error Handling**: Não exposição de detalhes internos

Performance e Otimizações

Banco de Dados

- Índices únicos para CPF
- Queries otimizadas
- Conexão persistente
- Pool de conexões

API

- Respostas padronizadas
- Códigos HTTP apropriados
- Paginação preparada (futuro)
- Cache preparado (futuro)

Monitoramento e Logs

Logs Implementados

- Conexão com banco de dados
- Erros de validação
- Operações CRUD
- Inicialização do servidor

Métricas (Futuro)

- Tempo de resposta
- Taxa de erro
- Uso de memória
- Conexões ativas

Extensibilidade

Preparação para Futuras Funcionalidades

1. **Autenticação**: Estrutura preparada para JWT
2. **Autorização**: Middleware de permissões
3. **Auditoria**: Logs de operações
4. **Versionamento**: API versionada
5. **Paginação**: Estrutura preparada
6. **Filtros**: Query builders preparados

Padrões para Novos Módulos

1. Criar model em `models/`
2. Implementar service em `services/`
3. Criar controller em `controllers/`
4. Definir rotas em `routes/`
5. Adicionar testes em `tests/`
6. Documentar endpoints

Esta documentação técnica serve como guia para desenvolvedores que irão trabalhar ou dar manutenção no projeto LiftHub.