Documentação Técnica – Sistema Help Tech

1. Visão Geral

O Help Tech é um sistema web desenvolvido para o gerenciamento completo de solicitações de serviços,
voltado especificamente para empresas de assistência técnica.

A solução opera em três camadas:

• Frontend (HTML, CSS, JavaScript) — executado no navegador
• Backend (Node.js + Express) — executado localmente
• Banco de Dados MySQL — acessado via Sequelize

O sistema recebe requisições do frontend, valida os dados, processa regras de negócio, interage com o MySQL e retorna respostas estruturadas em JSON.

2. Arquitetura do Sistema

A arquitetura segue um padrão em camadas:

CAMADA DE APRESENTAÇÃO (Frontend)
• Desenvolvido com HTML, CSS e JavaScript
• Executado diretamente no navegador
• Organizado em: FrontEnd/ → css/ → js/
• Responsável pela interface, formulários e chamadas à API

CAMADA DE NEGÓCIO (Backend / API)
• Desenvolvido em Node.js + Express
• Responsável por validações, regras de negócio, autenticação e rotas REST
• Executado localmente via api.js

CAMADA DE DADOS (MySQL + Sequelize)
• Banco de dados relacional
• Modelagem acessada pelo Sequelize
• Organização: backend/db/

FLUXO DE UMA REQUISIÇÃO
1. Usuário interage com frontend
2. Frontend envia requisição HTTP para API
3. API valida e acessa o banco via Sequelize
4. API retorna JSON para o frontend

3. Tecnologias Utilizadas

FRONTEND:
• HTML5
• CSS3
• JavaScript

BACKEND:
• Node.js
• Express
• Sequelize
• mysql2
• dotenv
• cors
• body-parser
• nodemailer
• morgan

BANCO DE DADOS:
• MySQL

4. Dependências Exatas Utilizadas no Backend

As dependências utilizadas no projeto, com suas versões exatas, são:

"body-parser": "^1.20.3"
"cors": "^2.8.5"
"dotenv": "^16.0.3"
"express": "^4.21.2"
"morgan": "^1.10.0"
"mysql2": "^3.15.3"
"nodemailer": "^7.0.11"
"sequelize": "^6.37.7"

5. Estrutura de Diretórios

```
/HelpTech
 ├── FrontEnd/
 │    ├── *.html
 │    ├── css/
 │    └── js/
 └── backend/
      ├── api.js
      ├── db/
```

```
│   ├── conexão e modelos Sequelize
└── .env
```

6. Instalação do Ambiente

1. Instalar Node.js e MySQL.

2. Criar o banco:
CREATE DATABASE help_tech;

3. Criar arquivo .env:
DB_HOST=localhost
DB_PORT=3306
DB_USER=usuario
DB_PASS=senha
DB_NAME=help_tech
PORT=3000

4. Instalar dependências:
cd backend
npm install

7. Execução do Sistema

1. Iniciar o MySQL.
2. Rodar o backend:
node api.js
3. Abrir o frontend pelo navegador via index.html.

8. Rotas da API (Exatamente como definidas no projeto)

BASE URL: http://localhost:3000

POST /cad_func — cadastra funcionário
GET /list_cargo — lista cargo
GET /lista solicitações — lista todos os serviços
POST /login — realiza o login
GET /Status — lista status do serviço
GET /tp_servico — lista tipos de serviço
GET /busca_cliente/:cpf — busca cliente via CPF
POST /cria_solicit — cria solicitação
GET /lista funcionários — lista funcionários
PATCH /atualiza_status_servico/:id_solicitacao
GET /clientes — lista clientes
POST /cad_cliente — cadastra cliente

DELETE /delete_solicitacao/:id_solicitacao

9. Exemplos Detalhados das Rotas (Request e Response JSON)

A seguir, exemplos completos de entrada e saída em JSON para todas as rotas da API.

1) POST /cad_func
Request:
```
{
 "nome": "João",
 "email": "joao@empresa.com",
 "cpf": "12345678900",
 "senha": "1234"
}
```

Response:
```
{
 "message": "Funcionário cadastrado com sucesso",
 "id": 1
}
```

2) GET /list_cargo
Response:
```
[
 { "id": 1, "descricao": "Técnico" },
 { "id": 2, "descricao": "Administrador" }
]
```

3) GET /lista solicitações
Response:
```
[
 {
  "id_solicitacao": 10,
  "cliente": "Maria",
  "status": "Aberto",
  "tipo_servico": "Manutenção"
 }
]
```

4) POST /login
Request:
```
{
 "usuario": "admin",
```

```
  "senha": "1234"
}
```

Response:

```
{
  "message": "Login realizado com sucesso",
  "usuario": "admin"
}
```

5) GET /Status

Response:

```
[
  { "id": 1, "descricao": "Aberto" },
  { "id": 2, "descricao": "Em andamento" }
]
```

6) GET /tp_servico

Response:

```
[
  { "id": 1, "descricao": "Manutenção" },
  { "id": 2, "descricao": "Conserto" }
]
```

7) GET /busca_cliente/:cpf

Response:

```
{
  "id": 5,
  "nome": "Carlos",
  "cpf": "12345678900"
}
```

8) POST /cria_solicit

Request:

```
{
  "cpf_cliente": "12345678900",
  "tipo_servico": 2,
  "descricao": "Equipamento não liga"
}
```

Response:

```
{
  "message": "Solicitação criada com sucesso",
  "id_solicitacao": 12
```

}

9) GET /lista funcionários
Response:
[
 {
  "id": 1,
  "nome": "João",
  "email": "joao@empresa.com"
 }
]

10) PATCH /atualiza_status_servico/:id
Request:
{
 "status": 3
}

Response:
{
 "message": "Status atualizado",
 "id_solicitacao": 12,
 "novo_status": 3
}

11) GET /clientes
Response:
[
 {
  "id": 10,
  "nome": "Ana",
  "cpf": "98765432100"
 }
]

12) POST /cad_cliente
Request:
{
 "nome": "Ana",
 "cpf": "98765432100",
 "email": "ana@cliente.com"
}

Response:
{
 "message": "Cliente cadastrado com sucesso",
 "id": 10
}

13) DELETE /delete_solicitacao/:id
Response:
{
 "message": "Solicitação removida com sucesso"
}

10. Considerações Finais

O Help Tech adota boas práticas de arquitetura modular, organização clara e escalabilidade. Está preparado para evolução futura: dashboards, autenticação JWT, deploy em nuvem e integrações externas.