Documentação do Sistema TecAssist

1. Visão Geral

TecAssist é um sistema de gerenciamento de ordens de serviço (OS) para assistência técnica. A aplicação permite cadastrar clientes, buscar equipamentos, registrar ordens de serviço, listar e filtrar OS por status e cliente, editar registros e imprimir ordens.

2. Objetivo

O objetivo do sistema é organizar o atendimento técnico dentro de uma oficina, centralizando dados de clientes, equipamentos e ordens de serviço em uma interface web simples e responsiva.

3. Tecnologias Utilizadas

- Next.js 16.2.3
- React 19.2.4
- TypeScript 5
- ESLint 9 com “eslint-config-next”
- SQLite como banco de dados local
- better-sqlite3 para acesso a banco de dados SQLite no servidor
- Node.js para execução local do projeto
- Rotas de API do Next.js (App Router)
- Componentes React com “use cliente” para interatividade e formulários

4. Estrutura do Projeto

app/

“page.tsx”: página principal e controle de visão entre criar/listar OS e clientes.

“layout.tsx”: layout global com fontes Google e CSS global.

“globals.css”: estilos globais da aplicação.

“api/”: rotas de API do servidor.

components/

“layout/Sidebar.tsx”: navegação lateral entre telas.

“layout/Header.tsx”: cabeçalho da aplicação.

“telas/CriarOS.tsx”: tela para cadastro de ordens de serviço.

“telas/ListarOS.tsx”: tela para listagem, filtro e edição de OS.

“telas/Clientes.tsx”: tela de listagem, busca e edição de clientes.

“layout/Sidebar.tsx”: navegação lateral entre telas.

“layout/Header.tsx”: cabeçalho da aplicação.

“telas/CriarOS.tsx”: tela para cadastro de ordens de serviço.

"telas/ListarOS.tsx": tela para listagem, filtro e edição de OS.

"telas/Clientes.tsx": tela de listagem, busca e edição de clientes.

lib/

"db.ts": inicializa a conexão SQLite e aplica o schema
"schema.sql": definição das tabelas do banco de dados.
"types.ts": tipos TypeScript usados no frontend.
"db.ts": inicializa a conexão SQLite e aplica o schema.
"schema.sql": definição das tabelas do banco de dados.
"types.ts": tipos TypeScript usados no frontend.

5. Arquitetura do Sistema

A arquitetura é baseada em Next.js com App Router:

- O frontend é uma SPA com navegação interna controlada por estado em "app/page.tsx".
- API Routes em "app/api/" fornecem endpoints REST para consultar e modificar dados.
- O acesso a dados é feito de forma síncrona com "better-sqlite3" em "lib/db.ts".
- A base de dados é persistida em arquivo SQLite local ("database.db") e é criada automaticamente pelo schema.
- O frontend é uma SPA com navegação interna controlada por estado em "app/page.tsx".
- API Routes em "app/api/" fornecem endpoints REST para consultar e modificar dados.
- O acesso a dados é feito de forma síncrona com "better-sqlite3" em "lib/db.ts".
- A base de dados é persistida em arquivo SQLite local ("database.db") e é criada automaticamente pelo schema.

6. Requisitos do Sistema

6.1 Requisitos Funcionais (RF)

Os requisitos funcionais descrevem o que o sistema deve fazer, ou seja, as ações e funcionalidades que ele deve oferecer aos usuários.

- **RF01 – Cadastrar Cliente:** O sistema deve permitir o cadastro de novos clientes fornecendo nome, CPF, CNPJ, telefone, e-mail e endereço.
- **RF02 – Cadastro Rápido de Cliente:** O sistema deve permitir o cadastro de um novo cliente diretamente durante o fluxo de criação de uma ordem de serviço.
- **RF03 – Buscar Clientes:** O sistema deve permitir a busca rápida de clientes cadastrados utilizando filtros por nome, CPF ou telefone.
- **RF04 – Editar Cliente:** O sistema deve permitir a alteração e atualização dos dados de um cliente existente através de uma janela modal.
- **RF05 – Visualizar Histórico do Cliente:** O sistema deve permitir que, ao selecionar um cliente, todas as suas ordens de serviço (OS) associadas sejam exibidas.
- **RF06 – Registrar Ordem de Serviço (OS):** O sistema deve permitir a abertura de uma OS vinculando um cliente, dados do equipamento, tipo de atendimento, defeito relatado, acessórios, aparência, observações, status inicial e valor total.
- **RF07 – Listar e Filtrar Ordens de Serviço:** O sistema deve disponibilizar uma tela de listagem de OS com recursos de busca por palavra-chave, além de filtros por cliente e por status.
- **RF08 – Editar Ordem de Serviço:** O sistema deve permitir a abertura de uma OS existente para edição e atualização de suas informações.

- **RF09 – Excluir Ordem de Serviço:** O sistema deve permitir a exclusão de uma ordem de serviço e, consequentemente, a remoção do equipamento associado a ela.
- **RF10 – Impressão de Ordem de Serviço:** O sistema deve permitir a visualização e impressão dos dados de uma OS em uma nova janela do navegador.
- **RF11 – Sugestão de Dados de Equipamento:** O sistema deve gerar sugestões automáticas de preenchimento para os campos "tipo", "marca" e "modelo" do equipamento com base nos dados já existentes no banco de dados.
- **RF12 – Verificar Conexão:** O sistema deve possuir um endpoint de teste (ping) para verificar o status da conexão com o banco de dados.

6.2 Requisitos Não Funcionais (RNF)

Os requisitos não funcionais determinam as restrições, características de qualidade e propriedades técnicas que o sistema deve possuir.

- **RNF01 – Tecnologia do Frontend:** O sistema deve ser desenvolvido utilizando Next.js (versão 16.2.3), React (versão 19.2.4) e TypeScript 5.
- **RNF02 – Padrão de Código:** O código-fonte deve utilizar ESLint 9 integrado com a configuração padrão eslint-config-next para garantir a padronização e qualidade do código.
- **RNF03 – Banco de Dados:** A aplicação deve utilizar o banco de dados relacional SQLite, persistido localmente através do arquivo database.db.
- **RNF04 – Desempenho e Acesso ao Banco:** O acesso ao banco de dados no servidor deve ser feito de forma síncrona utilizando a biblioteca better-sqlite3 para garantir transações simples e consultas rápidas.
- **RNF05 – Arquitetura de Navegação:** A interface do usuário deve funcionar como uma Single Page Application (SPA), controlando a navegação entre telas via estado interno no componente principal (app/page.tsx).
- **RNF06 – Interface e Design:** A interface do usuário deve ser simples e responsiva, adaptando-se a diferentes tamanhos de tela, construída de forma global com fontes do Google e estilos CSS centralizados.
- **RNF07 – Arquitetura de API:** O backend deve expor endpoints que seguem o estilo de arquitetura REST por meio das rotas de API do Next.js (App Router).
- **RNF08 – Execução em Ambiente Local:** A aplicação deve ser compatível para execução local utilizando o ambiente de execução Node.js e gerenciador de pacotes NPM.
- **RNF09 – Restrição de Segurança (Ausência de Autenticação):** Na versão atual, o sistema opera sob a premissa de que não há controle de autenticação ou níveis de permissão, permitindo acesso livre às funções por qualquer usuário local.

7. Banco de Dados

O banco de dados contém as seguintes tabelas:

“Clientes”: “id”, “nome”, “cpf”, “cnpj”, “telefone”, “email”, “endereco”, “criado_em”.

“Equipamentos”: “id”, “client_id”, “tipo”, “marca”, “modelo”, “numero_serie”, “voltagem”, “codigos_patrimonio”.

“**Tecnicos**”**:** “id”, “nome”, “email”, “senha”, “cargo”.

“Pecas”: “id”, “nome”, “sku”, “quant_estoque”, “preco_venda”.

“Ordens_Servico”: “id”, “equipamento_id”, “tecnico_id”, “stat”, “tipo_os”, “tipo_atendimento”, “defeito_relatado”, “laudo_tecnico”, “acessorios”, “aparencia”, “observacoes”, “data_entrada”, “valor_total”.

8. Funcionalidades Principais

- Busca rápida de clientes por nome, CPF ou telefone.
- Cadastro de novos clientes direto no fluxo de criação de ordem.
- Registro de ordens de serviço com dados do equipamento, tipo de atendimento, status e valor.
- Listagem das ordens de serviço com filtros por cliente, status e campo de busca.
- Edição de ordens de serviço existentes.
- Exclusão de ordens e exclusão de equipamentos associados.
- Impressão de OS em nova janela do navegador.
- Geração de sugestões para “tipo”, “marca” e “modelo” a partir de dados existentes.
- Busca rápida de clientes por nome, CPF ou telefone.
- Cadastro de novos clientes direto no fluxo de criação de ordem.
- Registro de ordens de serviço com dados do equipamento, tipo de atendimento, status e valor.
- Listagem das ordens de serviço com filtros por cliente, status e campo de busca.
- Edição de ordens de serviço existentes.
- Exclusão de ordens e exclusão de equipamentos associados.
- Impressão de OS em nova janela do navegador.
- Geração de sugestões para “tipo”, “marca” e “modelo” a partir de dados existentes.

9. Endpoints da API

- “GET /api/clientes?busca=”: retorna clientes filtrados.
- “POST /api/clientes”: cadastra um novo cliente.
- “PUT /api/clientes/[id]”: atualiza um cliente existente.
- “GET /api/ordens?clienteId=&busca=&status=”: retorna ordens de serviço.
- “POST /api/ordens”: cria uma nova ordem de serviço.
- “PUT /api/ordens/[id]”: atualiza uma ordem de serviço.
- “DELETE /api/ordens/[id]”: exclui uma ordem de serviço e seu equipamento.
- “GET /api/sugestoes?campo=&busca=”: retorna sugestões para campos de equipamento.
- “GET /api/ping”: endpoint de ping para checar conexão com o banco.
- “GET /api/clientes?busca=”: retorna clientes filtrados.
- “POST /api/clientes”: cadastra um novo cliente.
- “PUT /api/clientes/[id]”: atualiza um cliente existente.
- “GET /api/ordens?clienteId=&busca=&status=”: retorna ordens de serviço.
- “POST /api/ordens”: cria uma nova ordem de serviço.

- “PUT /api/ordens/[id]”: atualiza uma ordem de serviço.
- “DELETE /api/ordens/[id]”: exclui uma ordem de serviço e seu equipamento.
- “GET /api/sugestoes?campo=&busca=”: retorna sugestões para campos de equipamento.
- “GET /api/ping”: endpoint de ping para checar conexão com o banco.

10. Fluxo do Sistema

10.1 Cadastro de Ordem de Serviço

- Selecionar a opção “Abrir OS” no menu lateral.
- Buscar cliente existente ou cadastrar novo cliente.
- Informar dados do equipamento, defeito relatado, acessórios, aparência e observações.
- Salvar a ordem de serviço.

10.2 Listar e Filtrar OS

- Selecionar “Listar OS” no menu.
- Filtrar por palavra-chave de buscador, status ou cliente.
- Clicar em uma OS para abrir a janela de edição.
- Salvar alterações ou excluir a OS.

10.3 Gerenciamento de Clientes

- Selecionar “Clientes” no menu lateral.
- Buscar por nome, CPF ou telefone.
- Clicar em um cliente para abrir todos os seus registros de OS.
- Editar dados do cliente via modal

11. Execução do Projeto

Passos para rodar o sistema localmente:

1. **Instalar dependências:**

   “npm install”

2. **Iniciar o servidor de desenvolvimento:**

   “npm run dev”

3. **Acessar no navegador:**

   “http://localhost:3000”

12. Considerações Técnicas

- O uso de “better-sqlite3” permite consultas rápidas e transações simples sem dependência de servidor de banco de dados.

- A aplicação mistura rotas de API e renderização de frontend pelo Next.js App Router.
- Componentes interativos usam “useState” e “useEffect” para carregamento assíncrono de dados e sugestões.
- A aplicação não possui autenticação implementada.

13. Possíveis Melhorias

1. Implementar autenticação de técnicos e contas de usuário.
2. Adicionar controle de permissões e perfil técnico.
3. Criar relatórios financeiros e gráficos de OS.
4. Validar campos com regras mais robustas (CPF, e-mail, números de série).
5. Implementar um banco de dados híbrido que opere simultaneamente em nuvem e localmente.
6. Desenvolver um módulo para a gestão de peças e componentes do estoque interno.
7. Implementar paginação para listas grandes.