SISTEMA DE CONTROLE TRACTOMAQ

Relatório Técnico de Desenvolvimento
Curso: Ciência da Computação
Aluno: João Vitor de Carvalho
Matrícula: 2024008566

Índice

---

1. Introdução

O presente relatório tem como finalidade apresentar o desenvolvimento completo do sistema Tractomaq, criado como trabalho integrador das disciplinas de Programação II, Engenharia de Software e Banco de Dados.

A aplicação foi construída com Node.js, Express, Sequelize, PostgreSQL, React, Axios e Chart.js, seguindo práticas modernas de desenvolvimento web. O sistema oferece funcionalidades completas de gerenciamento, incluindo autenticação, autorização, CRUDs completos e um dashboard analítico dinâmico.

O objetivo deste documento é explicar detalhadamente a lógica empregada na implementação, a organização do código, as decisões técnicas e o processo de desenvolvimento.

---

2. Objetivos do Sistema

O sistema Tractomaq foi projetado para:

- Gerenciar produtos, agendamentos e orçamentos com operações CRUD completas.
- Fornecer um dashboard dinâmico contendo métricas e análises.
- Permitir acesso diferenciado entre Administrador e Usuário comum.
- Aplicar boas práticas de arquitetura, modularização e organização.
- Criar uma interface moderna, responsiva e intuitiva para o usuário final.

---

3. Arquitetura Geral da Aplicação

A aplicação segue uma arquitetura em cliente-servidor, com separação clara entre camadas.

3.1. Backend – Node.js + Express + Sequelize + PostgreSQL

O backend é responsável por:

- Gerenciar autenticação via JWT.

- Implementar rotas REST.
- Realizar operações no banco usando Sequelize.
- Gerar dados consolidados para o dashboard.
- Implementar regras de negócio e validações.

O banco de dados utiliza PostgreSQL, que armazena as tabelas:

- usuários
- produtos
- agendamentos
- orçamentos
- orcamento_itens

3.2. Frontend – React + Vite + Axios + Chart.js

O frontend realiza:

- Navegação entre páginas com React Router.
- Controle de sessão armazenando token no localStorage.
- Consumo das APIs do backend via Axios.
- Exibição de gráficos com Chart.js.
- Formulários CRUD completos para todas as entidades.
- Dashboard interativo.

3.3. Comunicação entre camadas

A comunicação é feita via API REST, com chamadas AJAX usando Axios:

- POST para login e criação

- GET para listagem
- PUT para atualização
- DELETE para remoção

---

4. Organização das Pastas e Arquivos

4.1. BACKEND

/config

- database.js
  Configura e exporta a conexão Sequelize com PostgreSQL.

/controllers

- auth.controller.js
  Login, registro, criação de token, validação de senha.
- produtos.controller.js
  CRUD completo de produtos.
- agendamentos.controller.js
  CRUD de agendamentos + regras de datas.
- orcamentos.controller.js
  CRUD de orçamentos e dos itens associados.
- dashboard.controller.js
  Gera todos os dados analíticos para o dashboard.

/middleware

- auth.js
  Middleware de autenticação: valida token e repassa o userId.

/models

- Usuario.js
  Tabela de usuários, campo isAdmin, associações.

- Produto.js
  Estrutura da tabela de produtos.

- Agendamento.js
  Estrutura da tabela de agendamentos.

- Orcamento.js
  Estrutura da tabela de orçamentos.

- OrcamentoItem.js
  Itens que compõem cada orçamento.

/routes

- auth.routes.js
  Rotas de login, registro e seed admin.

- produtos.routes.js
  Rotas CRUD dos produtos.

- agendamentos.routes.js
  Rotas CRUD dos agendamentos.

- orcamentos.routes.js
  Rotas CRUD dos orçamentos.

- dashboard.routes.js
  Rota para retorno do dashboard.

/utils

- seedAdmin.js
  Garante criação do usuário administrador no banco.

---

server.js

- Inicializa servidor Express
- Carrega rotas
- Sincroniza banco
- Aplica associações
- Cria Admin automático

4.2. FRONTEND

/public

- Arquivos estáticos usados pelo Vite.

/src/assets

- Arquivos CSS separados por página ou componente:
    - login.css
    - dashboard.css
    - estoque.css
    - agenda.css
    - orcamentos.css
      Garantem estilo e responsividade.

/src/componentes

- Navbar.jsx
  Barra superior da aplicação.
- Sidebar.jsx
  Menu lateral com navegação.

- PrivateRoute.jsx
  Protege rotas internas usando token.

/src/layout

- MainLayout.jsx
  Layout base das páginas internas (sidebar + navbar + conteúdo).

/src/paginas

- Login.jsx
  Tela de autenticação e envio de credenciais.
- Dashboard.jsx
  Gráficos, cards e dados analíticos.
- Estoque.jsx
  CRUD de produtos.
- Agenda.jsx
  CRUD de agendamentos.
- Orcamentos.jsx
  CRUD de orçamentos + listagem dos itens.

App.jsx

- Define rotas, rotas privadas e estrutura geral.

main.jsx

- Ponto de entrada que renderiza o React.

package.json

- Lista bibliotecas: axios, react, react-router-dom, chart.js, etc.

---

5. Modelagem e Entidades CRUD

5.1 Usuários

- Cadastro
- Login
- Token JWT
- Controle de acesso (Admin vs Comum)

5.2 Produtos

- Criar produto
- Listar produtos
- Editar produto
- Excluir produto
- Filtro por marca no dashboard

5.3 Agendamentos

- Criar agendamento
- Listar por usuário
- Atualizar status
- Excluir
- Filtragem por data (semana no dashboard)

5.4 Orçamentos

- Criar orçamento
- Listar
- Editar
- Excluir
- Itens do orçamento vinculados
- Análise de vencimento no dashboard

---

6. Autenticação, Autorização e Controle de Acesso

O sistema utiliza um fluxo de autenticação baseado em JWT (JSON Web Tokens), garantindo segurança e isolamento entre usuários.

A autenticação ocorre quando o usuário informa e-mail e senha no frontend. O backend valida a senha usando bcrypt, e em caso de sucesso gera um token contendo:

- id do usuário
- email
- isAdmin

Os tokens são enviados em cada requisição através do header Authorization. O middleware auth.js valida o token, extrai o ID e associa a requisição, permitindo que cada usuário acesse apenas os seus dados.

A autorização distingue dois perfis:

- Administrador: pode acessar funcionalidades ampliadas (como visualizar todos os dados no dashboard).
- Usuário comum: só acessa registros vinculados ao seu próprio usuário.

Essa separação garante privacidade, segurança e atendimento ao requisito de múltiplos perfis de acesso.

---

7. Dashboard Dinâmico

O dashboard reúne informações analíticas e se atualiza dinamicamente a partir de dados do banco. O backend fornece uma única rota que retorna:

- quantidade de produtos
- quantidade de agendamentos
- quantidade de orçamentos
- total faturado
- agendamentos da semana
- orçamentos que vencem no mês
- distribuição de produtos por marca
- status dos agendamentos

O frontend consome esses dados e os converte em:

- gráficos de pizza, barras e linhas
- contadores (cards)
- tabelas e listagens de itens recentes

Esse conjunto fornece uma visão geral completa do sistema, cumprindo o requisito de visualização analítica após login.

---

8. Ordenação, Filtragem e Responsividade

O sistema oferece recursos que melhoram a navegabilidade e organização das informações.

Filtragem

- Filtragem de produtos por marca.

- Filtragem de agendamentos por data (semana atual).
- Filtragem de orçamentos por validade.

Ordenação

- Produtos podem ser ordenados por nome.
- Agendamentos são ordenados por data.
- Orçamentos são ordenados por data de validade.

Responsividade

O CSS utiliza:

- flexbox
- grid
- media queries

Isso garante que todas as telas funcionem corretamente em celulares, tablets e computadores, mantendo legibilidade e usabilidade independentemente do tamanho da tela.

---

9. Tratamento de Erros e Validações

O projeto aplica validações tanto no backend quanto no frontend.

No frontend:

- Exibição de mensagens claras (ex.: “Email ou senha inválidos”).
- Alertas amigáveis utilizando SweetAlert2.
- Verificação de campos obrigatórios antes do envio.

No backend:

- Respostas padronizadas com status HTTP adequados.

- Validação de dados antes de criação/edição.
- Try/catch para evitar falhas inesperadas no servidor.
- Bloqueio de acesso quando o token é inválido ou ausente.

Essas camadas reduzem falhas e tornam a experiência do usuário mais previsível e segura.

---

10. Processo de Desenvolvimento e Uso Complementar de IA

Foi utilizado de forma externa a Inteligência Artificial como apoio ao desenvolvimento, sempre de forma complementar, sem substituir a lógica desenvolvida por mim.

Principais usos:

- Auxílio na estruturação inicial das rotas e controllers.
- Sugestões para ajustes de lógica em casos específicos.
- Geração de exemplos de validação e tratamento de erros.
- Correção de inconsistências entre frontend e backend.
- Orientação para resolver mensagens de erro do banco e do Sequelize.
- Explicações conceituais sobre tokenização, associações e fluxo de autenticação.

Responsabilidade principal do desenvolvimento

Todo o projeto desde a estruturação, organização das pastas, lógica de CRUD, modelagem e integração foi implementado manualmente, utilizado a mesma apenas como um auxiliar em determinados momentos. Servindo assim, como ferramenta de apoio para o desenvolvimento de um projeto de qualidade, com o intuito de fortalecer o entendimento do estudante para com a matéria.

---

11. Conclusão Geral

O sistema desenvolvido atende plenamente os requisitos do Trabalho Integrador, integrando conhecimentos de Programação II, Banco de Dados e Engenharia de Software.

A aplicação implementa:

- autenticação segura
- múltiplos perfis de usuário
- três entidades CRUD completas
- dashboard dinâmico com dados analíticos reais
- interface responsiva
- filtros, ordenações e validações
- integração total entre backend (Node.js/Express) e frontend (React/Vite)
- persistência em banco relacional via Sequelize

O projeto reflete práticas reais utilizadas no desenvolvimento profissional de sistemas web, demonstrando domínio das tecnologias e capacidade de integrar múltiplos componentes em uma solução completa.
