

# **CENTRO UNIVERSITÁRIO CATÓLICA DE SANTA CATARINA**

**Título do Projeto:** Sistema de Gerenciamento de Vistorias Náuticas (SGVN)

**Nome do Estudante:** Wuelliton Christian dos Santos

**Curso:** Engenharia de Software

**Data de Entrega:** 10/07/2025

**Local:**

Joinville/SC

## Resumo

Neste documento é apresentada uma proposta para o desenvolvimento de um sistema web projetado para solucionar as ineficiências no processo de vistoria de embarcações para fins de seguro. O fluxo de trabalho atual depende de ferramentas informais como o WhatsApp, resultando em coletas de dados não padronizadas por vistoriadores, muitos sem treinamento específico. As informações são posteriormente transcritas manualmente por um digitador e, por fim, validadas por um engenheiro naval para assinatura. Este processo fragmentado gera retrabalho, custos elevados e baixa confiabilidade nos laudos enviados às seguradoras. O Sistema proposto visa substituir este fluxo, oferecendo um ambiente único onde vistorias são guiadas por checklists padronizados, eliminando a necessidade do digitador e garantindo que o engenheiro naval receba um relatório íntegro e pronto para aprovação. O sistema automatizará a geração de laudos em PDF, fornecerá um dashboard analítico e aumentará a confiabilidade e a agilidade de todo o processo.

## 1. Introdução

### Contexto

O processo atual de vistoria de embarcações para seguradoras é um fluxo de trabalho manual e suscetível a erros, orquestrado por meio de aplicativos de mensagens como o WhatsApp. O processo envolve três papéis distintos com transferências manuais de informação:

1. O Vistoriador: Muitas vezes sem treinamento formal, vai a campo e captura as imagens da embarcação de forma aleatória. A falta de um guia claro resulta em fotos de baixa qualidade ou na ausência de imagens obrigatórias.
2. O Digitador: Recebe as fotos e informações via WhatsApp e realiza a transcrição manual dos dados para um documento de texto, gerando a primeira versão do laudo.
3. O Engenheiro Naval: Revisa o laudo preenchido pelo digitador e realiza a assinatura, validando o documento para a seguradora.

Essa dinâmica frequentemente exige que o vistoriador ou mesmo o engenheiro contate o proprietário da embarcação para solicitar novas fotos, causando atrasos e uma percepção de amadorismo, além de comprometer a confiabilidade do laudo final.

## Justificativa

A criação deste sistema é de alta relevância para a Engenharia de Software, pois busca resolver problema de retrabalho, custo e risco de um processo de negócio real. O projeto se justifica ao:

- **Reduzir o Risco Operacional:** Ao padronizar a coleta de dados, o sistema mitiga erros humanos e garante que todas as informações necessárias sejam capturadas corretamente na primeira visita.
- **Otimizar o Fluxo de Trabalho:** A plataforma elimina a função do digitador e centraliza as atividades, reduzindo o tempo total do ciclo de vistoria e os custos associados.
- **Aumentar a Confiabilidade e a Rastreabilidade:** O sistema cria um registro histórico centralizado e auditável, garantindo a integridade do laudo que será assinado pelo engenheiro naval e enviado à seguradora.
- **Aplicar Conceitos Modernos de TI:** Envolve o desenvolvimento de uma aplicação web robusta (full-stack), com automação de processos, foco em usabilidade (UX/UI) e implementação de práticas de segurança para proteger dados sensíveis.

## Objetivos

### Objetivo Geral

Desenvolver um sistema web que estrutura e automatiza o gerenciamento de vistorias de embarcações, garantindo a qualidade e a integridade dos dados desde a coleta em campo até a geração do laudo final.

### Objetivos Específicos:

- Desenvolver o módulo de coleta de dados em campo, com uma interface responsiva para dispositivos móveis, que permita ao vistoriador preencher checklists dinâmicos e realizar o upload de imagens de forma estruturada.
- Implementar um serviço de backend para a geração automática de laudos, que processe os dados coletados da vistoria e os compile em um documento PDF padronizado e profissional.
- Modelar e implementar o banco de dados e as APIs (Interfaces de Programação de Aplicação) necessárias para centralizar e gerenciar o ciclo de vida completo das vistorias, incluindo histórico de alterações, status e armazenamento de documentos.
- Construir um painel administrativo (dashboard) que consuma os dados das APIs e apresente indicadores e métricas de forma visual, utilizando gráficos e tabelas, para análise gerencial.

- Estruturar um sistema de controle de acesso baseado em papéis (RBAC) e um mecanismo de fluxo de trabalho (workflow) para gerenciar as etapas do processo, como "em andamento", "em revisão" e "aprovado", garantindo que os laudos passem por um ciclo de revisão antes da finalização.

## 2. Descrição do Projeto

### Tema do Projeto

Desenvolvimento de uma plataforma web que serve como ferramenta para todo o processo de vistoria náutica. O sistema guiará o Vistoriador no preenchimento de checklists e na captura de imagens obrigatórias, consolidará as informações em um laudo digital e o disponibilizará para a aprovação final do responsável técnico (Engenheiro Naval), eliminando processos paralelos e manuais.

### Problemas a Resolver

- Coleta de dados inconsistente: Fotos e informações coletadas de forma aleatória por vistoriadores sem um guia padrão.
- Fluxo de trabalho ineficiente e caro: Dependência de três papéis (Vistoriador, Digitador, Engenheiro) com transferências manuais de informação.
- Alto índice de retrabalho: Necessidade constante de contatar o proprietário da embarcação para solicitar fotos que faltaram ou que ficaram em formato inadequado.
- Baixa confiabilidade: Risco de erros de digitação e informações incorretas no laudo final enviado à seguradora, comprometendo a credibilidade do processo.
- Falta de um repositório central: Ausência de um sistema para gerenciar e consultar o histórico de vistorias de forma rápida e segura.

### Limitações

- A plataforma não incluirá, neste estágio, integração com uma base oficial de valores de embarcações (FIPE Náutica). O valor de referência será baseado nos dados do próprio sistema.
- O sistema será desenvolvido inicialmente para uso individual, sem suporte à gestão de múltiplas empresas (multi-tenancy).
- A geração de laudo será em PDF, sem a implementação de certificação digital (ex: assinatura via token).

### 3. Especificação Técnica

#### 3.1. Requisitos de Software

##### **Requisitos Funcionais (RF):**

RF01 - O sistema deve permitir cadastro e login de Administrador e Vistoriador.

RF02 - O Sistema deve criar uma vistoria, informando dados da embarcação e local (marina ou residência) logado no usuario Adminstrador.

RF03 - O sistema deve notificar o Vistoriador designado sobre a nova vistoria.

RF04 - O Sistema deve preencher o formulário da vistoria e fazer upload das fotos necessárias logado no usuario Vistoriador, podendo trocar fotos antes de concluir.

RF05 - O sistema deve impedir o envio da vistoria sem todos os campos e fotos obrigatórios preenchidos.

RF06 - O Sistema quando estiver em vistoria deve ter status visível ao Administrador (Pendente, Em andamento, Concluída ou Cancelada).

RF07 -O sistema deve gerar automaticamente um laudo em PDF e armazená-lo.

RF08 - O laudo PDF deve ser enviado por e-mail ao Administrador e ao proprietário da embarcação.

RF09 - O sistema deve manter um histórico pesquisável de vistorias por data, vistoriador e status.

RF10 - O Sistema permitirá a desativação do Vistoriador e reatribuir vistorias pendentes a outro usuário com o usuario de Administrador.

RF11 - O Sistema deve poder editar os dados básicos de uma vistoria enquanto ela estiver Pendente ou Em andamento com o usuario Adminstrador.

RF12 - O Sistema deve poder salvar um rascunho da vistoria e continuar o preenchimento posteriormente logado no usuario Vistoriador.

RF13 - O Sistema deve deixa visualizar um painel com contagem de vistorias por status no usuario Vistoriador.

RF14 - O Sistema deve poder deixa baixar qualquer laudo PDF disponível no histórico.

RF15 - O sistema deve permitir pesquisa de vistoria pelo número de casco ou nome da embarcação.

RF16 - O sistema deve registrar data e hora de criação, alteração e conclusão de cada vistoria.

### **Requisitos Não Funcionais (RNF):**

RNF01 - A aplicação deve ser responsiva e acessível por dispositivos móveis.

RNF02 - O sistema deve seguir arquitetura REST.

RNF03 - Os dados devem ser armazenados em banco de dados relacional (PostgreSQL).

RNF04 - Os documentos PDF devem ter layout padronizado.

RNF05 - O sistema deve responder em até 3 segundos para operações de criação, edição e consulta de vistorias.

RNF06 - Arquivos enviados (fotos e laudos) devem ser armazenados criptografados em repouso.

RNF07 - A aplicação deve realizar backups automáticos diários do banco de dados e dos arquivos, com retenção mínima de 30 dias. Os backups serão armazenados de forma segura em um serviço de nuvem (ex: AWS S3) e um plano de recuperação será documentado.

RNF08 - A solução deve estar em conformidade com a LGPD quanto à coleta, armazenamento e descarte de dados pessoais.

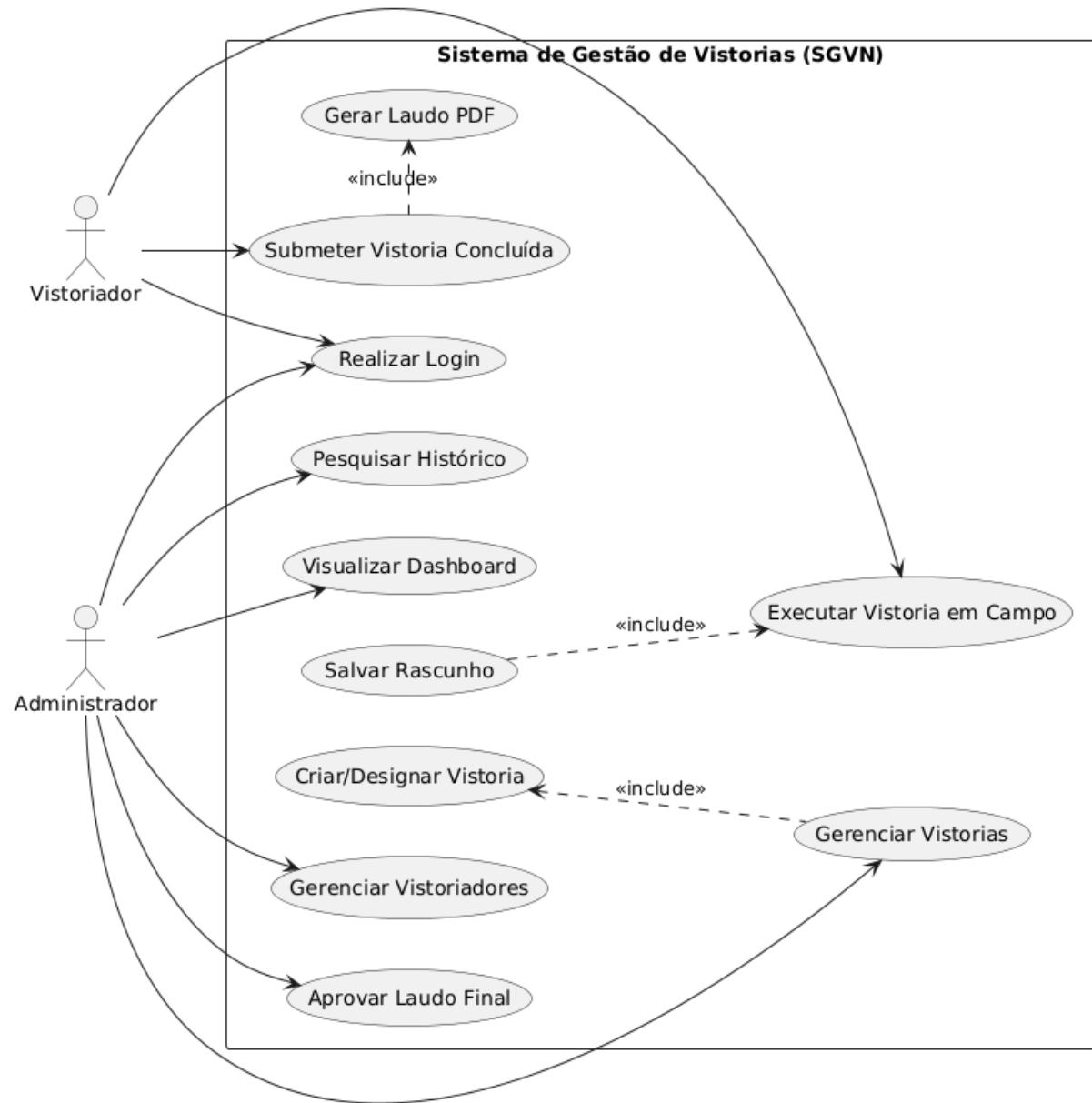
RNF09 - O sistema deve exportar a lista de vistorias para arquivo CSV.

RNF10 - A aplicação deve possuir um modo de funcionamento offline, permitindo que o Vistoriador preencha a vistoria e salve os dados localmente no dispositivo em áreas sem conectividade, para posterior sincronização com o servidor.

RNF11 - O sistema deve implementar uma estratégia de cache para otimizar o tempo de resposta de consultas frequentes, como o dashboard administrativo e históricos.

RNF12 - As imagens enviadas pelo Vistoriador devem ser comprimidas no lado do Adminstrador(frontend) para otimizar o tempo de upload e reduzir o consumo de dados e espaço de armazenamento.

### **Representação dos Requisitos:**



**Figura 1:** Diagrama de Casos de Uso UML com os principais atores e funcionalidades.

O Diagrama de Casos de Uso apresentado na **Figura 1** ilustra de forma visual a arquitetura funcional do Sistema de Gestão de Vistorias Náuticas (SGVN). Ele detalha as principais interações entre os usuários (atores) e as funcionalidades centrais do sistema (casos de uso), definindo o escopo e os limites da aplicação.

## Atores do Sistema

O sistema possui dois tipos de usuários com papéis e permissões distintas:

1. **Administrador:** É o ator com privilégios elevados, responsável pela gestão, supervisão e qualidade do processo. Suas funções incluem o gerenciamento completo das vistorias e dos vistoriadores, a análise de dados através do dashboard e, crucialmente, a aprovação final dos laudos, garantindo que o documento está em conformidade antes do envio à seguradora.
2. **Vistoriador:** É o ator de campo, focado na execução da coleta de dados. Sua principal responsabilidade é acessar as vistorias que lhe foram designadas, preencher o checklist de forma padronizada, anexar as evidências fotográficas e submeter o laudo preenchido para a aprovação do administrador.

## Principais Fluxos de Trabalho

O diagrama destaca os dois principais fluxos operacionais do sistema:

- **Fluxo de Execução da Vistoria (realizado pelo Vistoriador):** O fluxo se inicia com o Realizar Login. O Vistoriador então utiliza a função Executar Vistoria em Campo, um processo central que, conforme a notação <<include>> indica, contempla a funcionalidade de Salvar Rascunho. Ao finalizar a coleta, ele aciona a Submeter Vistoria Concluída, que por sua vez dispara a ação de Gerar Laudo PDF automaticamente, consolidando todas as informações em um documento padronizado.
- **Fluxo de Gerenciamento e Aprovação (realizado pelo Administrador):** O Administrador possui uma visão completa do sistema. Ele pode Gerenciar Vistorias (uma função que <<inclui>> a criação e designação de novas vistorias), Gerenciar Vistoriadores, Visualizar Dashboard para análises gerenciais, e Pesquisar Histórico. Sua ação mais crítica é a de Aprovar Laudo Final, que representa o último portão de qualidade do processo.

### 3.2. Considerações de Design

**Escolhas de Design:** A aplicação será desenvolvida com uma arquitetura desacoplada entre front-end e back-end. A camada de apresentação (front-end) será construída com **React**, empacotada com **Vite** para desenvolvimento e build rápidos. O back-end utilizará Node.js com o framework Express, seguindo o padrão de arquitetura MVC (Model-View-Controller).

A escolha pelo PostgreSQL como banco de dados se justifica por sua robustez, confiabilidade e suporte avançado a consultas complexas, características essenciais para a integridade dos dados das vistorias e para a futura implementação de relatórios analíticos. Pensando em escalabilidade, a arquitetura foi projetada para permitir que seus contêineres (Aplicação Web, API, Banco de Dados) possam ser escalados horizontalmente de forma independente, garantindo a performance do sistema conforme o volume de vistorias aumentar.

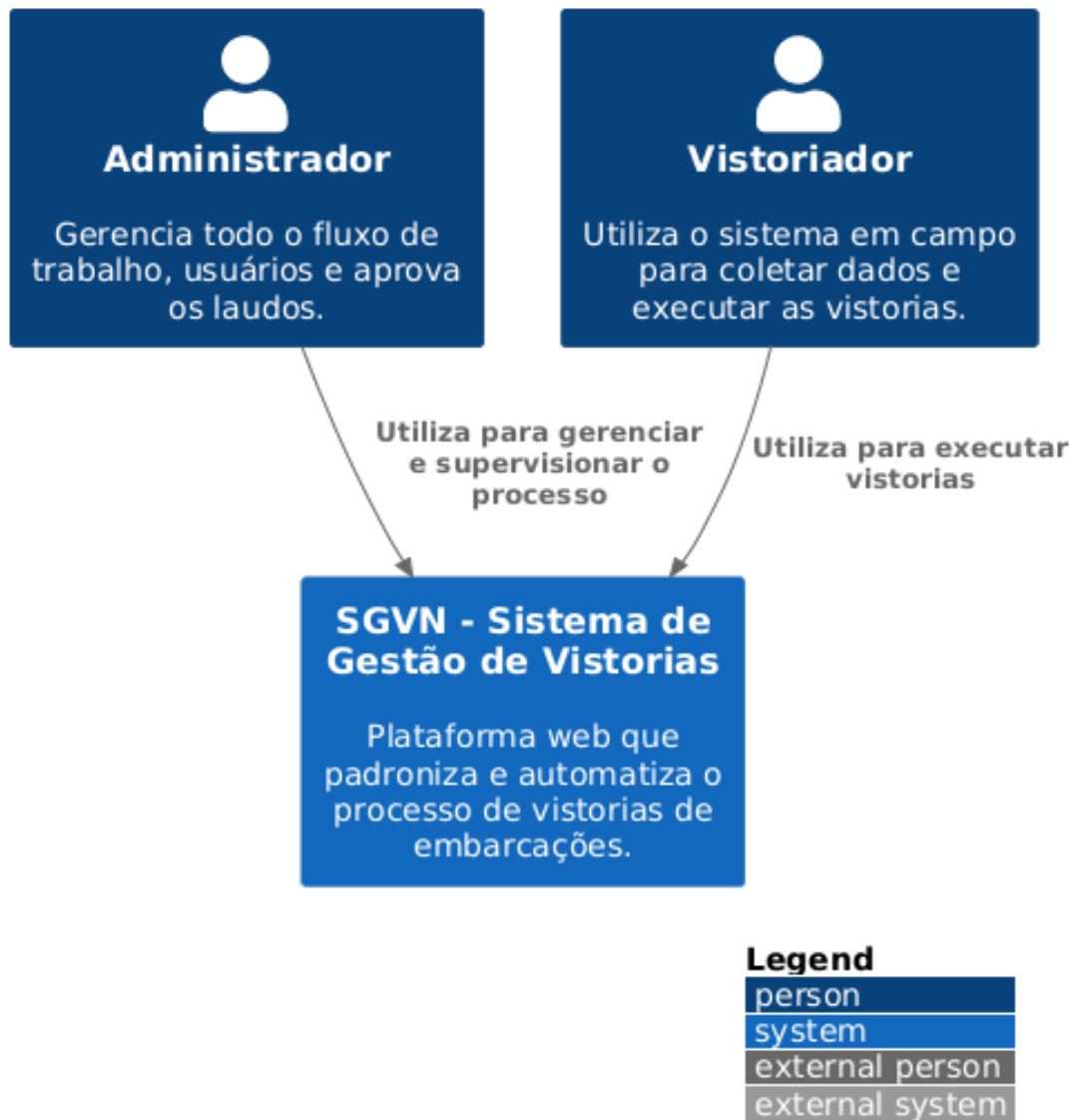
A autenticação será realizada via Clerk. O frontend (React + Vite) integra o Clerk SDK (hosted sign-in/sign-up, MFA opcional). A API (Node.js + Express) valida o JWT emitido pela Clerk via JWKS e aplica RBAC por papéis/organizações.

#### Visão Inicial da Arquitetura:

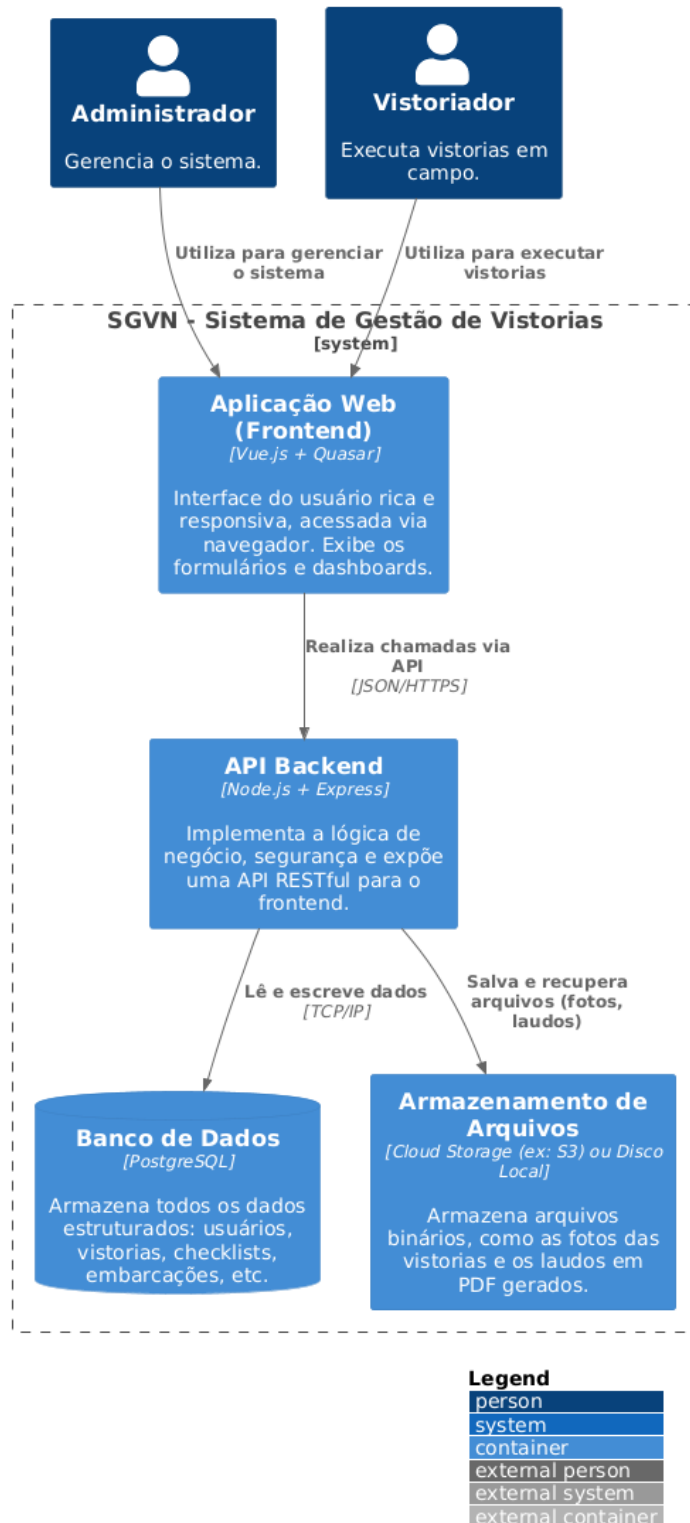
- Frontend: React, Vite, React Router, TanStack Query, React Hook Form + Zod, Axios, Clerk React SDK
- Backend: Node.js + Express.js
- Banco de Dados: PostgreSQL
- Autenticação: Clerk
- Serviço de PDF: Puppeteer ou PDF-lib
- Padrões de Arquitetura: MVC, REST
- Padrões de Arquitetura: Indicação de padrões específicos utilizados (ex.: MVC, Microserviços).

**Resumo do Modelo C4:** O Modelo C4 será utilizado para documentar a arquitetura de software em quatro níveis de detalhe:

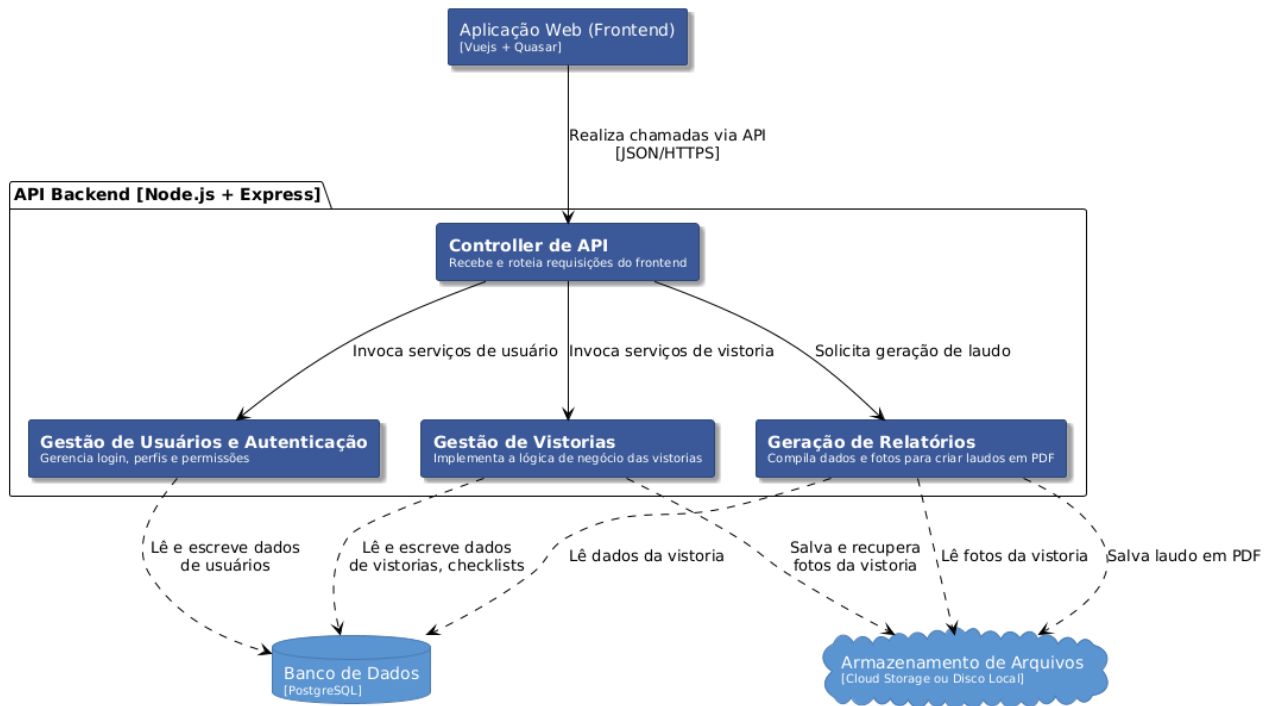
- Nível 1 – Contexto: Visão macro do sistema, mostrando suas interações com usuários (Administrador, Vistoriador) e sistemas externos (Serviço de E-mail).
- Nível 2 – Contêineres: Detalhamento do sistema em unidades executáveis, como a Aplicação Web (front-end), a API (back-end), o Banco de Dados e o Armazenamento de Arquivos.
- Nível 3 – Componentes: Divisão interna do back-end em seus principais componentes lógicos, como "Gestão de Usuários", "Gestão de Vistorias" e "Geração de Relatórios".



**Figura 2:** Nível 1: Contexto



**Figura 3: Nível 2: Contêineres**



**Figura 4:** Nível 3: Componentes

### 3.3. Stack Tecnológica

**Linguagens de Programação:** JavaScript (utilizada tanto no front-end com React quanto no back-end com Node.js).

## Frameworks e Bibliotecas

- Frontend: **React, Vite, React Router, TanStack Query, React Hook Form + Zod, Axios** (ou fetch)
- Backend: Express.js, Sequelize (ORM) , @clerk/clerk-sdk-node
- Segurança: bcryptjs (hash de senhas), JWT (autenticação) ,
- Geração de PDF: Puppeteer

## Ferramentas de Desenvolvimento e Gestão:

- Controle de Versão: GitHub
- Gerenciamento de Tarefas: Trello
- Testes de API: Postman
- Testes Automatizados: Jest (para testes unitários)
- Protótipos de Interface: Figma

### 3.4. Considerações de Segurança

A segurança é um pilar fundamental do SGVN, garantindo a proteção dos dados dos usuários e a integridade das vistorias. As seguintes medidas serão implementadas:

- **Armazenamento de senhas:** Utilização de hash com a biblioteca bcryptjs para garantir que as senhas nunca sejam armazenadas em texto plano.

- **Autenticação e Autorização:** Implementação de um fluxo de autenticação baseado em tokens JWT (JSON Web Tokens). Serão utilizados tokens de acesso de curta duração para as requisições diárias e um sistema de *refresh tokens* para renovar a sessão de forma segura, minimizando a exposição em caso de vazamento do token de acesso.
- **Proteção de Rotas:** Todas as rotas da API que acessam dados sensíveis serão protegidas, garantindo que apenas usuários autenticados e com as permissões corretas (Administrador ou Vistoriador) possam acessá-las.
- **Prevenção de Ataques:** Validação e sanitização rigorosa de todas as entradas de dados no backend para prevenir ataques de Injeção de SQL (SQL Injection) e Cross-Site Scripting (XSS).
- **Logs de Auditoria:** O sistema manterá um registro detalhado (log) de todas as operações críticas, como logins, alterações de status de vistorias e aprovações de laudos, para fins de rastreabilidade e análise de segurança.
- **Validação de Segurança:** Planejamento da execução de testes de vulnerabilidade utilizando ferramentas de análise estática de segurança (SAST) e dinâmica (DAST) para identificar e corrigir proativamente possíveis falhas de segurança.

## 4. Próximos Passos e Cronograma de Execução

Para garantir a entrega do projeto SGVN dentro do prazo estipulado, o desenvolvimento será dividido em fases, com foco em entregas contínuas. O cronograma abaixo detalha as atividades planejadas entre agosto e o início de dezembro de 2025.

### Fases do Projeto:

1. **Fundação e Estruturação (Agosto):** Foco em finalizar o design, estruturar o ambiente de desenvolvimento e criar a base de dados e as rotas essenciais da API.

2. **Desenvolvimento do Core Funcional (Setembro):** Implementação das funcionalidades centrais para os usuários, como autenticação, gerenciamento de vistorias e a lógica de negócio principal no *backend*.
3. **Funcionalidades de Campo e Geração de Laudos (Outubro):** Desenvolvimento da interface do vistoriador para coleta de dados em campo e implementação do serviço de geração automática de laudos em PDF.
4. **Finalização, Testes e Refinamento (Novembro):** Construção do *dashboard* administrativo, realização de testes integrados, correções de *bugs* e preparação da documentação final do projeto.
5. **Entrega e Apresentação (Início de Dezembro):** Consolidação do projeto, revisão final e preparação para a entrega e apresentação.

A execução seguirá o plano detalhado no cronograma abaixo, utilizando as ferramentas de gestão e desenvolvimento já definidas no projeto.

## Cronograma Detalhado (Agosto - Dezembro 2025)

### Mês: Agosto

- **Semanas 1 e 2:**
  - **Atividades:** Design e Arquitetura - Finalizar o protótipo de todas as telas no Figma.
  - **Entregáveis:** Protótipo navegável completo.
- **Semanas 3 e 4:**
  - **Atividades:** Estruturação do Backend e BD - Configurar o ambiente Node.js, estruturar o banco de dados PostgreSQL com base no DER e iniciar a API RESTful com Sequelize.
  - **Entregáveis:** Repositório no GitHub criado, schema do banco de dados implementado e rotas iniciais da API definidas.

### Mês: Setembro

- **Semanas 1 e 2:**
  - **Atividades:** Módulo de Autenticação e Usuários - Implementar as funcionalidades de login, controle de acesso (RBAC) e gerenciamento de vistoriadores (CRUD). Usar JWT e bcryptjs para segurança.
  - **Entregáveis:** Telas de login e gestão de vistoriadores funcionais. Endpoints da API protegidos.
- **Semanas 3 e 4:**
  - **Atividades:** Gerenciamento de Vistorias (Admin) - Desenvolver as funcionalidades para o Administrador criar, designar, editar e visualizar o status das vistorias.
  - **Entregáveis:** Interface Administrador para gerenciar o ciclo de vida das vistorias.

## Mês: Outubro

- **Semanas 1 e 2:**
  - **Atividades:** Execução da Vistoria (Vistoriador) - Desenvolver a interface responsiva para o Vistoriador preencher o formulário/checklist, fazer upload de fotos e salvar rascunhos.
  - **Entregáveis:** Módulo de execução de vistoria funcional em dispositivos móveis.
- **Semanas 3 e 4:**
  - **Atividades:** Geração de Laudos em PDF - Implementar o serviço no backend (usando Puppeteer) para gerar o laudo em PDF automaticamente ao concluir a vistoria e enviá-lo por e-mail.
  - **Entregáveis:** Geração e envio de laudos em PDF com dados reais da vistoria.

## Mês: Novembro

- **Semanas 1 e 2:**
  - **Atividades:** Dashboard e Pesquisa - Construir o painel administrativo com gráficos e métricas e implementar as funcionalidades de pesquisa e histórico.
  - **Entregáveis:** Dashboard funcional e sistema de busca implementado.
- **Semanas 3 e 4:**
  - **Atividades:** Testes e Refinamento - Realizar testes completos dos fluxos de trabalho (do início ao fim). Corrigir bugs, refinar a interface (UX/UI) e validar todos os requisitos funcionais e não funcionais.

- **Entregáveis:** Relatório de testes, lista de bugs corrigidos e aplicação estável.

### **Mês: Dezembro**

- **Semana 1:**
  - **Atividades:** Documentação e Preparação para Entrega - Finalizar a documentação do projeto, preparar os materiais de apresentação (slides) e realizar o deploy da aplicação em um ambiente de homologação.
  - **Entregáveis:** Documento do projeto finalizado e apresentação pronta.

## **5. Referências**

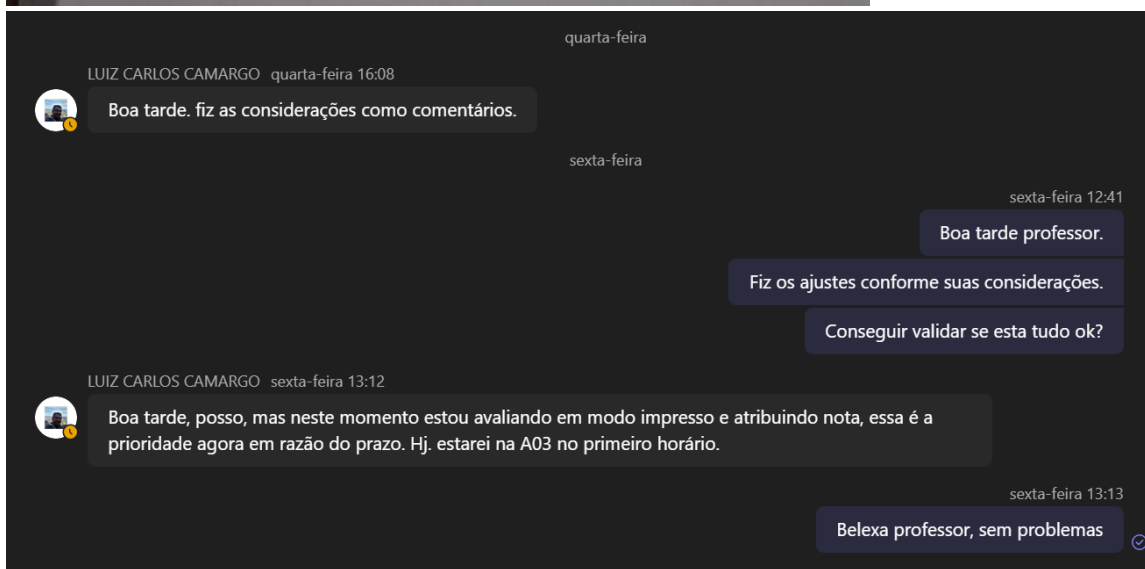
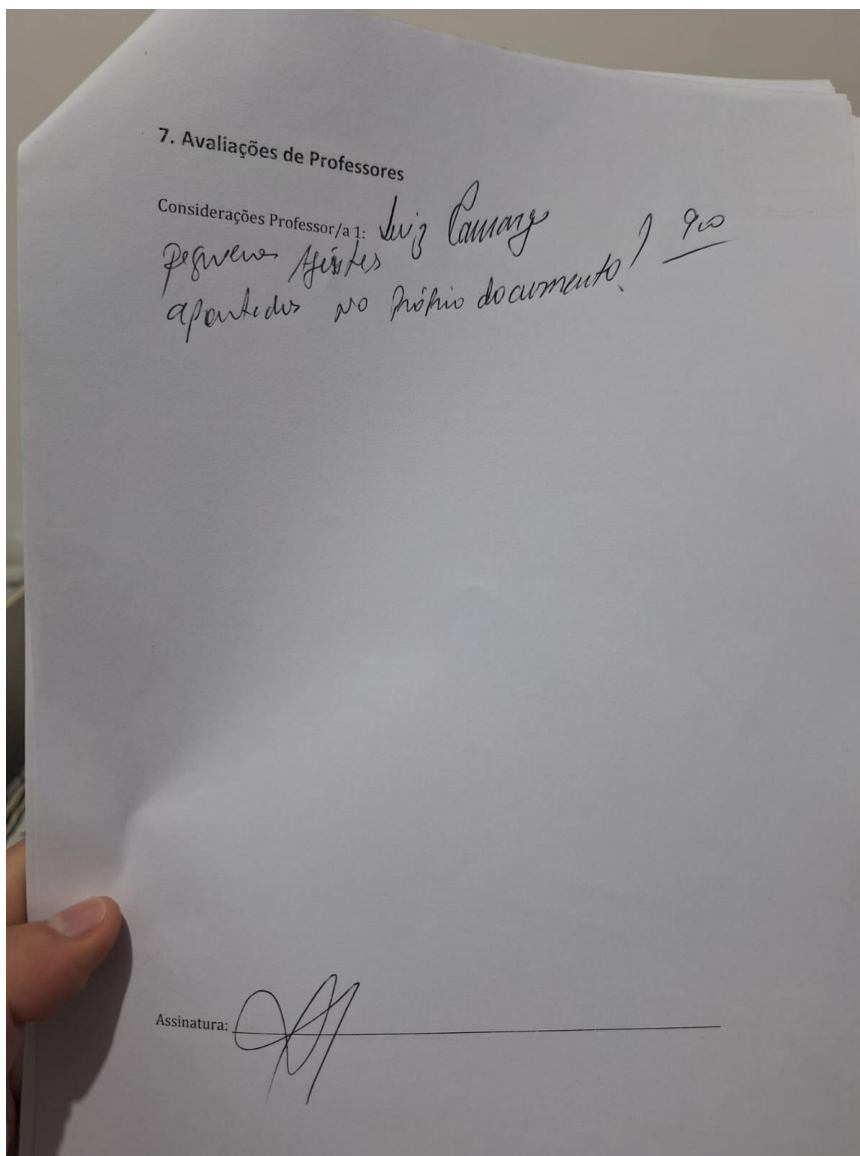
- Documentação Node.js – <https://nodejs.org/>
- React Docs
- Vite
- React Router
- TanStack Query
- React Hook Form / Zod
- OWASP Secure Coding Practices – <https://owasp.org/>
- Sequelize ORM – <https://sequelize.org/>
- Clerk Docs (React, JWT, JWKS, Webhooks).
- OIDC/JWT (RFCs), OWASP ASVS, Express best practices.

## **6. Apêndices (Opcionais)**

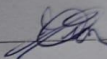
- Protótipos de Telas
- Diagrama Entidade-Relacionamento (DER) do Banco de Dados
- Exemplo de Laudo PDF Gerado
- Plano de Testes

## 7. Avaliações de Professores

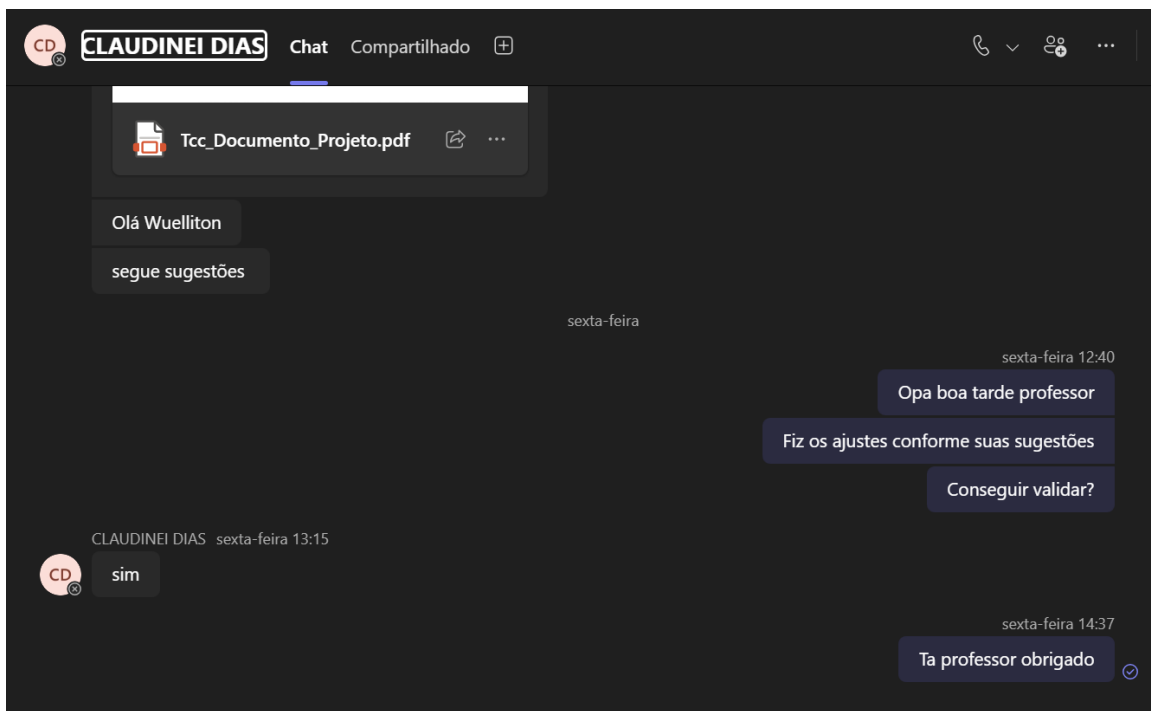
Considerações Professor/a 1: **LUIZ CARLOS CAMARGO**



Considerações Professor/a 2: **CLAUDINEI DIAS**

Assinatura: Claudinei Dias 

Considerações Professor/a 2:



Considerações

Professor/a

3:


**PAULO**

**ROGERIO**

**PIRES**

**MANSEIRA**

sugestões enviadas pelo Team

Assinatura: 



Opa, blz?

Seguem minhas considerações. Seu trabalho está bom. Eu vou focar aqui nas fraquezas que se solucionadas podem torná-lo melhor.

Sobre a especificação técnica:

- Diagrama de casos de uso mencionado, mas sem representação gráfica ou fluxos detalhados.
- Requisitos não abordam limites de armazenamento para fotos ou PDFs.
- Fluxos de erro (ex.: falha no envio de e-mail) não são especificados.

Sobre arquitetura:

- Falta de estratégias para escalabilidade ou tolerância a falhas.
- Ausência de suporte offline para vistoriadores em áreas com baixa conectividade.
- Não há menção a caching para otimizar consultas frequentes.

Sobre tecnologias:

- Ausência de ferramentas de teste (ex.: Jest para unitários, Cypress para E2E).
- Falta de justificativa para o PostgreSQL (ex.: volume de vistorias ou tipos de consultas).
- Não há menção a otimizações específicas para dispositivos móveis (ex.: compressão de imagens).
- Dependência de Puppeteer não considera alternativas leves (ex.: PDFKit).

Sobre segurança:

- Ausência de detalhes sobre armazenamento e análise de logs de auditoria.
- Falta de testes de segurança (ex.: penetração ou validação por terceiros).
- Estratégia de backups diários não especifica armazenamento ou recuperação.
- Não há menção a políticas de expiração de tokens JWT.

É isso aí. Espero que tenha ajudado. Abraço.