Documento de Arquitetura

Eagles Software

versão 1.0

Equipe:
João Victor Pereira Borges
Renan Lucas de Lima Oliveira

1 - Introdução

O Eagles Software é um sistema de cadastro, gerenciamento e divulgação de itens perdidos. Desenvolvido para o setor de achados e perdidos, com o objetivo de reduzir a quantidade de itens sob supervisão da instituição e tornar mais acessível o acervo de objetos para os frequentadores que, eventualmente, venham a perder um objeto pessoal.

2 - Visão Geral

No IFPI - Campus Picos, o setor de achados e perdidos está vinculado ao controle de disciplina. Eles recebem itens perdidos, aguardando a reclamação dos proprietários. No entanto, muitas vezes, os donos não aparecem para recuperar seus pertences, causando frustração para ambos, proprietários e equipe do controle, que busca manter a organização do ambiente. Foi focado em resolver ou pelo menos minimizar essas dores que o Eagles Software surgiu.

3 - Premissas

- Desenvolvimento para plataforma mobile e web.
- Utilização de tecnologias open source.
- Comunicação segura via HTTPS.
- Interface responsiva para dispositivos móveis.

4 - Requisitos não funcionais

Código	Requisitos	Descrição detalhada do requisito
RNF01	Escalabilidade	**Descrição:** O sistema deve ser escalável para acomodar um aumento futuro no número de usuários e objetos registrados. **Condições:** Capacidade de expansão para suportar pelo menos o dobro do número atual de usuários e registros.
RNF02	Desempenho	**Descrição:** O sistema deve ser capaz de lidar com um grande volume de registros e consultas simultâneas sem degradação significativa no desempenho. **Condições:** Tempo de resposta para operações críticas não deve exceder X segundos, mesmo em situações de carga máxima.
RNF03	Disponibilidade	**Descrição:** O sistema deve estar disponível 24 horas por dia, 7 dias por semana, com tempo de inatividade planejado mínimo para manutenção.

		Condições: Tempo de inatividade programado não deve exceder X horas por mês.
RNF04	Compatibilidade	**Descrição:** O sistema deve ser compatível com diferentes navegadores web e dispositivos para garantir uma experiência consistente para os usuários. **Condições**: Testes de compatibilidade realizados regularmente para assegurar suporte a navegadores mais utilizados e dispositivos comuns.
RNF05	Segurança	**Descrição:** Garantir a segurança dos dados armazenados no sistema, protegendo contra acessos não autorizados e ataques cibernéticos. **Condições:** Monitoramento de atividades suspeitas, conformidade com padrões de segurança.
RNF06	Usabilidade	**Descrição:** A interface do usuário deve ser intuitiva e amigável, facilitando a utilização por usuários com diferentes níveis de habilidade. **Condições:** Realização de testes de usabilidade periódicos, incorporando feedback dos usuários para melhorias

5 - Métricas

Escalabilidade:

- **Métrica:** Taxa de Crescimento de Usuários e Objetos
- **Objetivo:** Garantir que o sistema possa suportar pelo menos o dobro do número atual de usuários e registros. Monitorar a taxa de

crescimento ao longo do tempo para ajustes proativos na infraestrutura.

Desempenho:

- **Métrica:** Tempo de Resposta para Operações Críticas
- **Objetivo:** Manter o tempo de resposta para operações críticas abaixo de X segundos, mesmo em situações de carga máxima. Monitorar regularmente o desempenho em diferentes cenários para identificar e corrigir possíveis gargalos.

Disponibilidade:

- **Métrica:** Tempo de Inatividade Programado
- **Objetivo:** Garantir que o tempo de inatividade programado não exceda X horas por mês. Realizar manutenções programadas durante períodos de menor atividade para minimizar o impacto na disponibilidade.

Compatibilidade:

- **Métrica:** Índice de Compatibilidade com Navegadores e Dispositivos
- **Objetivo:** Assegurar que o sistema seja compatível com os navegadores mais utilizados e dispositivos comuns. Realizar testes de compatibilidade regularmente e manter um índice de suporte a diferentes plataformas.

Segurança:

- **Métrica:** Nível de Conformidade com Padrões de Segurança
- **Objetivo:** Garantir a conformidade com padrões de segurança estabelecidos. Monitorar atividades suspeitas, realizar auditorias de segurança e manter atualizações regulares para manter a robustez contra ameaças.

Usabilidade:

- **Métrica:** Taxa de Satisfação do Usuário
- **Objetivo:** Manter uma taxa de satisfação do usuário elevada. Realizar testes de usabilidade periódicos, coletar feedback dos usuários e implementar melhorias na interface para garantir uma experiência intuitiva e amigável.

6 - Fundamentação

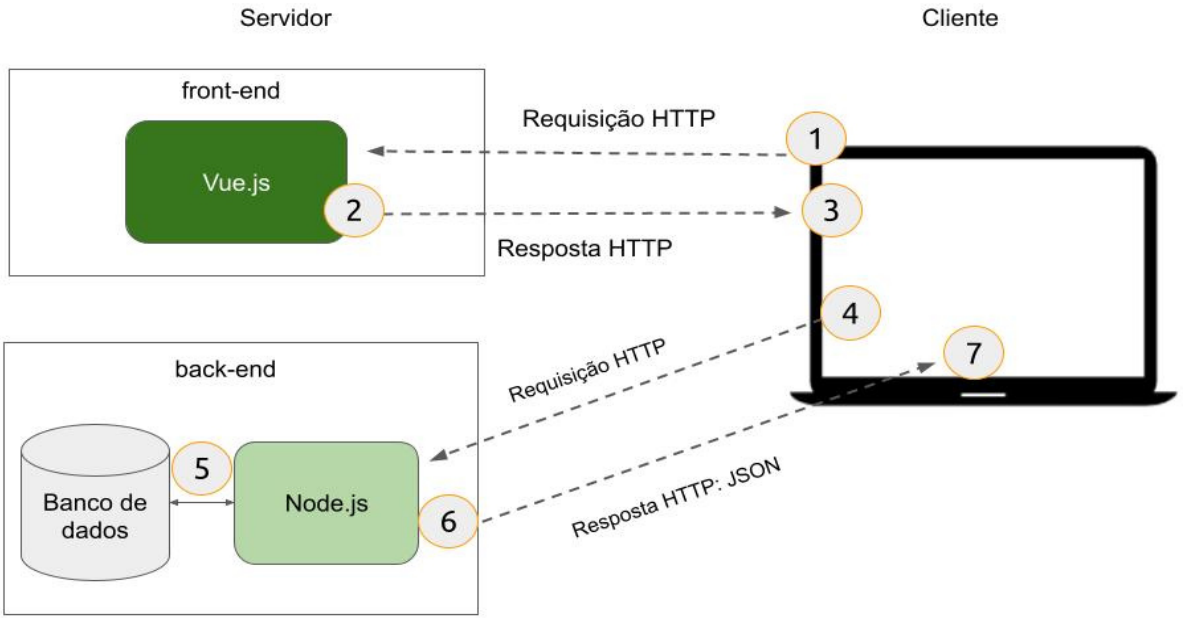
O sistema de achados e perdidos foi projetado com base em duas visões arquiteturais principais: a visão em camadas e a visão de aplicações dinâmicas.

Visão em Camadas:

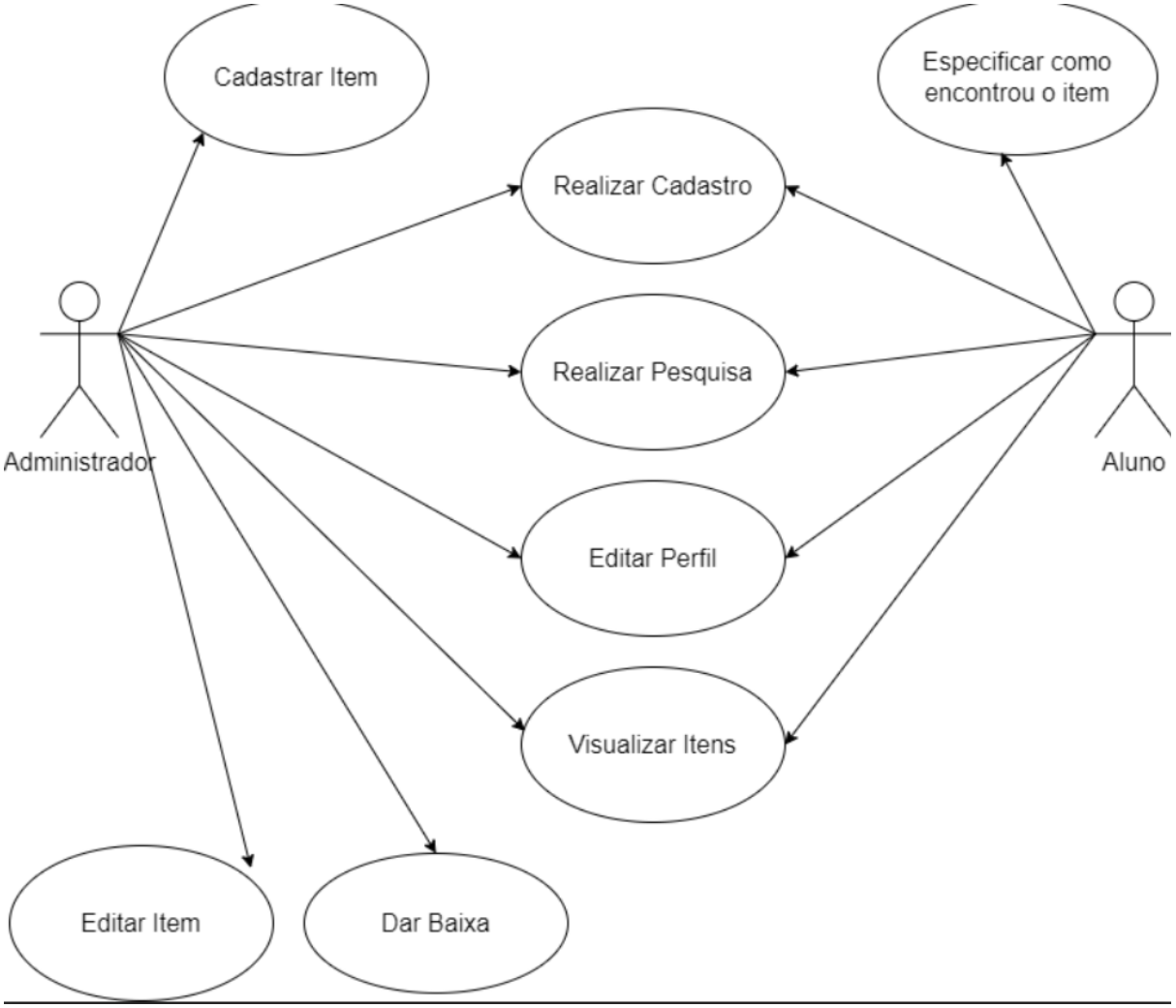
- A escolha da arquitetura em camadas é motivada pela separação clara de responsabilidades e pela organização lógica do sistema. As camadas incluem a apresentação, lógica de negócios e acesso a dados, proporcionando modularidade e facilitando a manutenção.
- A camada de apresentação, responsável pela interface do usuário, é separada da lógica de negócios, o que permite adaptações mais flexíveis no front-end sem afetar a lógica subjacente.

Visão de aplicações dinâmicas:

- Sites dinâmicos são aqueles que geram dinamicamente parte do conteúdo que é retornado para o cliente. Geralmente as páginas HTML definem o modelo com lacunas para serem preenchidas dinamicamente. O conteúdo dinâmico geralmente é armazenado em um banco de dados.
- Um site dinâmico pode retornar dados diferentes para uma mesma URL com base nas informações fornecidas pelo usuário, localização, data de acesso, etc. Exemplo: se dois usuários autenticados acessarem a página inicial do Twitter irão visualizar informações diferentes, o layout da página é o mesmo, mas o feed de tweets (conteúdo principal da página) será diferente, porque é carregado de acordo com o usuário autenticado.


Servidor
Cliente
front-end
Vue.js
Requisição HTTP
1
2
3
Resposta HTTP
4
7
back-end
Banco de dados
5
Node.js
6
Requisição HTTP
Resposta HTTP: JSON

7 - Visão de História de Usuário



8 - Visões arquiteturais

Tecnologias Front-end:

- Utilização do framework Next.js com Tailwind CSS para o desenvolvimento do front-end.

Tecnologias Back-end:

- Desenvolvimento do back-end em Node.js com Express, utilizando JavaScript como linguagem principal.

Banco de Dados Relacional:

- Adoção do Supabase, baseado em PostgreSQL, como o banco de dados principal para armazenar dados.

Armazenamento de Imagens:

- Uso do Cloudinary como banco para armazenamento de imagens, integrado ao back-end.

Plataformas de Hospedagem:

- Deploy do front-end na Vercel e do back-end no Render para garantir escalabilidade e disponibilidade.

Ambiente de Desenvolvimento:

- Utilização do Visual Studio Code (VSCode) como a principal ferramenta de desenvolvimento para codificação, depuração e gerenciamento de projetos.

Controle de Versão:

- Adoção do Git como sistema de controle de versão, com o GitHub como plataforma de hospedagem para o repositório do código-fonte. Isso permite o acompanhamento das alterações, colaboração eficiente e versionamento do código.

9 - Tratamento dos atributos de qualidade – Decisões do arquiteto

Escalabilidade:

- **Decisão do Arquiteto:** Implementação da Visão de Aplicações Dinâmicas, onde o conteúdo é gerado dinamicamente com base nas informações fornecidas pelo usuário, localização, data de acesso, etc. Utilização de técnicas de otimização de consultas no banco de dados para garantir um desempenho eficiente na geração dinâmica de conteúdo.

Desempenho:

- **Decisão do Arquiteto:** Adoção da arquitetura em camadas para separar a lógica de negócios, facilitando a otimização de consultas e

operações críticas. Utilização de técnicas de cache e indexação eficientes no banco de dados para melhorar o desempenho.

Disponibilidade:

- **Decisão do Arquiteto:** Implementação de estratégias de redundância e balanceamento de carga para garantir alta disponibilidade. Programação de manutenções durante períodos de menor atividade para minimizar o impacto no tempo de inatividade.

Compatibilidade:

- **Decisão do Arquiteto:** Adoção de práticas de desenvolvimento responsivo para garantir a compatibilidade com diferentes navegadores e dispositivos. Testes de compatibilidade regulares realizados em ambientes simulados para garantir uma experiência consistente.

Segurança:

- **Decisão do Arquiteto:** Implementação de autenticação JWT e criptografia de senhas.

Usabilidade:

- **Decisão do Arquiteto:** Realização de testes de usabilidade periódicos com usuários reais para identificar áreas de melhoria na interface. Incorporação de feedback dos usuários para ajustes contínuos, visando manter a interface intuitiva e amigável.

10 - Padrões de projeto

1. **Padrão MVC (Model-View-Controller):**
   a. Descrição: O padrão MVC será adotado para separar claramente as responsabilidades de Model (lógica de negócios), View (interface do usuário) e Controller (gestão de eventos e interação).
   b. Justificativa: Facilita a manutenção, escalabilidade e testabilidade do sistema, além de promover uma clara divisão de tarefas entre as camadas.

2. **Padrão DTO (Data Transfer Object):**
    a. Descrição: O padrão DTO será empregado para transferência eficiente de dados entre camadas, permitindo a comunicação eficaz entre o front-end e o back-end.
    b. Justificativa: Reduz a complexidade da comunicação entre diferentes partes do sistema, evitando transferências desnecessárias de dados e otimizando o desempenho.
3. **Padrão Observer:**
    a. Descrição: O padrão Observer será utilizado para implementar a comunicação entre componentes do sistema de forma assíncrona, permitindo que eventos em uma parte do sistema acionem atualizações em outras partes.
    b. Justificativa: Facilita a comunicação entre componentes independentes, garantindo que alterações em uma parte do sistema sejam refletidas em outras partes de maneira eficiente.
4. **Padrão Gateway:**
    a. Descrição: O padrão Gateway será adotado para abstrair a integração com serviços externos, como o Cloudinary para armazenamento de imagens.
    b. Justificativa: Facilita a manutenção e a substituição de serviços externos, garantindo que a lógica de integração esteja isolada e seja facilmente adaptável a mudanças.