Laboratório Júnior de Inovação e Tecnologia

Arquitetura de Software

Projeto: E-clipse

Brasília - DF

2024

Objetivo: Documentar as relações e interação entre componentes de um sistema cujo papel é enviar e armazenar de peças de Defensores, avaliar de peças para o avaliador, gerenciar das partes e avaliar de triagens para o corregedor, documentação dos processos por PDF e notificação por email.

Sumário

O sistema visa facilitar processos de avaliação e comunicação entre defensores, corregedores e avaliadores, que eram processos manuais e consumidores de tempo. O

documento a seguir mostra as relações entre as entidades e componentes, visando escalabilidade por novos membros.

2. REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

2.1 REQUISITOS FUNCIONAIS:

- Autenticar e cadastrar usuários e separá-los por níveis de acesso;
- Gerar e-mail para comunicação das partes;
- Ao defensor:
    - Controlar o histórico de suas triagens, seja negada ou aprovada.;
    - Criar recurso em relação a uma peça e defensor;
    - Enviar arquivos de peças, cadastrar relatórios, criar, enviar e refazer triagens;
    - Calcular quantidade de peças enviadas;
    - Calcular média de notas;
    - Gerar relatório mensal
- Ao avaliador:
    - Contar peças avaliadas;
    - Buscar peças pendentes;
    - Controlar histórico de avaliação de peças;
    - Avaliar peças;
- Ao corregedor:
    - Buscar triagens negadas e peças por avaliador;
    - Inativar ou reativa cadastro de corregedor, avaliador e defensor;
    - Aprovar ou negar cadastros de avaliadores e defensores;
    - Contar e buscar corregedores, avaliadores e defensores;
    - Buscar corregedores avaliadores e defensores com cadastro pendentes;
    - Trocar usuário de área;
    - Mudar senha de usuário;
    - Reunir atividades semestrais e mensais do defensor;
    - Buscar, aprovar e negar peças e triagens;
    - Buscar peças por avaliador e trocá-las de avaliador;
    - Buscar peça por defensor e ID;
    - Buscar peça por data;
    - Buscar recurso;

2.2 REQUISITOS NÃO FUNCIONAIS:

- Geração de pdf rápido e seguro;

- Envio de Emails adaptados para cada usuário;
- Busca de dados no Front End sem sobrecarregar o banco de dados;
- Segurança no salvamento de dados;
- Comunicação eficiente entre os usuários;
- Censuras em pdf;
- Controle de acesso ao software de maneira intuitiva;

3. ARQUITETURA PROPOSTA:

A divisão da lógica de front-end e back-end foi feita usando, respectivamente, React e Node JS com Express JS, além do banco MySQL modelado pelo Prisma.

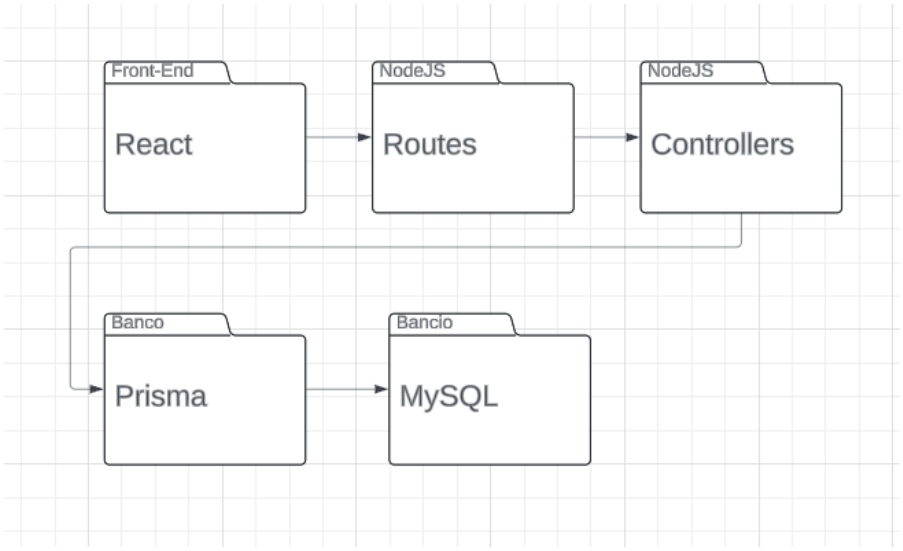
React é uma biblioteca JavaScript cuja principal proposta é abstrair sua lógica pela renderização de componentes reutilizáveis, e comunica-se por meio de envio de dados pelo cliente, tratados em formato JSON, recebidos pelo servidor, onde são performadas as devidas operações.

Node JS é um ambiente de execução baseado no Chrome V8 onde é possível desenvolver aplicações de servidor com JavaScript. É valorizado por sua versatilidade, não somente facilitando o tempo de estudo e comunicação entre as duas partes, mas também pela sua grande biblioteca de módulos. É usado para tratar dados enviados pelo front-end, e verificar o banco para performar as operações devidas.

3.1 COMPONENTES PRINCIPAIS:

Depende da lógica de negócios ou, se não, desconsiderar

3.2 DIAGRAMA DE ARQUITETURA:



4. VISÕES DA ARQUITETURA

4.1 VISÃO LÓGICA

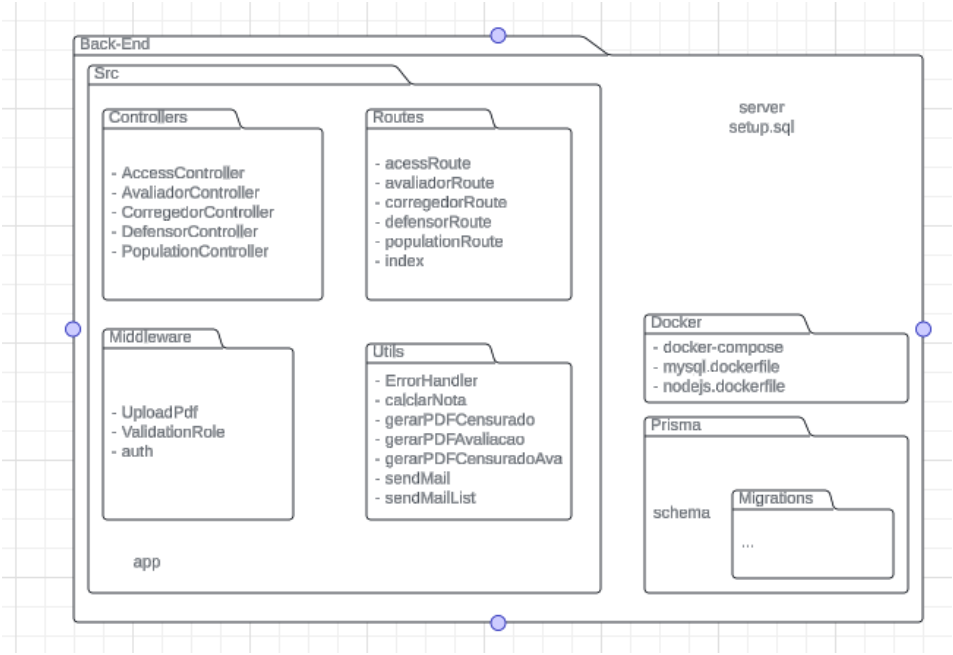
- **Diagrama de Componentes:**

4.2 VISÃO DE PROCESSOS

A FAZER

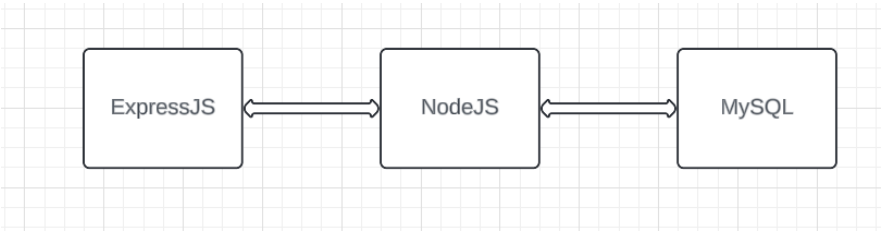
4.3 VISÃO DE IMPLEMENTAÇÃO

- **Estrutura de código:**



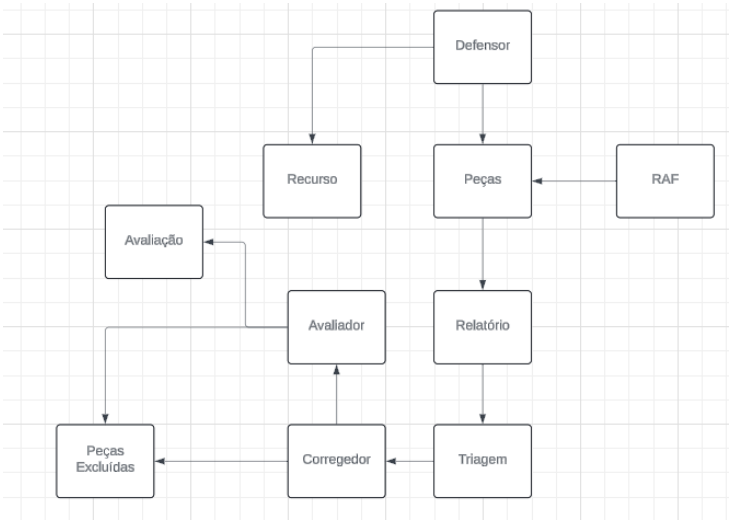
4.4 VISÃO DE IMPLANTAÇÃO

- **Diagrama de implantação:**



4.5 VISÃO DE DADOS

- **Modelo de Dados:**



5. DECISÕES ARQUITETURAIS

5.1 DECISÕES TOMADAS

- **NodeJS:** Considerado pela maior familiaridade do time em relação à ferramenta e versatilidade.
- **ExpressJS:** Framework de roteamento leve que permite flexibilidade na estrutura do projeto.
- **MySQL e Prisma:** Respectivamente banco e ORM, considerados pela familiaridade.

5.2 ALTERNATIVAS CONSIDERADAS

- **Django:** Desconsiderado pela falta de familiaridade.

5.3 TRADE-OFFS

- **Robustez, comunidade forte e funcionalidades prontas x Familiaridade**

6. TECNOLOGIAS UTILIZADAS

- **Front-end:** React, CSS;

- **Back-end:** NodeJS;
- **Banco de Dados:** MySQL;
- **Controle de Versão:** Git/GitHub/GitLab;
- **Servidor Web:** ExpressJS;
- Conteinerização**:** Docker e Docker-Compose;
- **Prototipação:** Figma e Canva;
- **Documentação:** Word;
- **Modelagem do Banco:** MySQL WorkBench
- **Desenvolvimento do Software:** Visual Studio Code.
- **Sistema Operacional:** Linux.
- **Meios de Comunicação**: Discord, Whatsapp e e-mail.

7. CONSIDERAÇÕES DE QUALIDADE

7.1 DESEMPENHO

- **Otimização de Consultas:** Uso de índices e otimização de consultas no banco de dados.
- **Cache:** Implementação de cache para reduzir a carga no servidor de banco de dados.

7.2 MANUTENIBILIDADE

- **Código modular:** Cada papel possui sua lógica bem definida em classes próprias com métodos encapsulados e claro, dentro de um arquivo para cada.
- **Documentação:** Manter documentação detalhada para facilitar estudo e escalabilidade por novos desenvolvedores.

8. CONSIDERAÇÕES DE SEGURANÇA

- São usadas bibliotecas de hash de senhas **bcrypt** e **jsonwebtoken** para tokenização de acesso.
- Separação de níveis de acesso: cada acesso é separado por caso, e cada um, quando tratada a entrada, é verificada no banco e comparada com os respectivos campos. Se nos conformes cada um receberá seu respectivo token.
- Atualização de acesso.

9. CONSIDERAÇÕES FINAIS

O sistema foi desenvolvido pensando em legibilidade, escalabilidade dada à flexibilidade das ferramentas escolhidas e abstração encapsulada, para melhor colaboração futura.

10. EQUIPE DE DESENVOLVIMENTO:

- Guilherme Araújo(Back-End)
- Pedro Lucas(Front-End)
- Guilherme Vieira(Full-Stack)
- Pedro Henrique(Back-End auxiliar)
- Gustavo Maxwell(Back-End auxiliar)