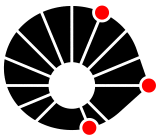

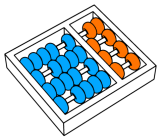

Aplicação web para automação da montagem de relatório de vistoria técnica de edifícios

Leonardo A. Lopes *Juliana F. Borin*

Relatório Técnico - IC-PFG-22-41
Projeto Final de Graduação
2022 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Aplicação web para automação da montagem de relatório de vistoria técnica de edifícios

Leonardo de Alencar Lopes* Juliana Freitag Borin*

Resumo

Empresas de engenharia civil necessitam frequentemente produzir relatórios e laudos técnicos para seus clientes. Para isso, muitas vezes é necessário que um engenheiro faça uma vistoria técnica nos edifícios, tirando fotos e descrevendo detalhes sobre patologias encontradas, melhorias recomendadas, integridade da estrutura, etc. Esses relatórios podem facilmente passar de uma centena de páginas, ocupando várias horas de um profissional muito capacitado e encarecendo a sua produção. Neste trabalho é descrita a implementação de uma aplicação web que visa automatizar a montagem desses documentos para uma empresa do ramo (Servare Engenharia). A aplicação foi desenvolvida com o framework Angular 13 no lado do cliente, NodeJS no lado do servidor e MySQL como sistema gerenciador de banco de dados. Ela foi hospedada em nuvem utilizando AWS e pode ser acessada por qualquer pessoal com usuário cadastrado. A saída da aplicação é um documento PDF, buscando ser o mais parecido possível com um laudo técnico produzido utilizando softwares convencionais, tais como editores de texto ou de slides.

1. Introdução

Os relatórios e laudos técnicos produzidos por empresas de engenharia civil são muito importantes no cotidiano de seus engenheiros. A Servare Engenharia é uma empresa do ramo de engenharia civil que precisa lidar com a produção desses relatórios. Um dos relatórios que ela precisa desenvolver é o “Relatório de não conformidades”, que busca relatar os problemas recorrentes encontrados pelo engenheiro durante uma vistoria técnica. Para cada um destes problemas é criado um quadro com fotografias tiradas no local da vistoria, além de outras informações importantes, como nome do edifício, pavimentos e locais onde o problema foi encontrado, data da vistoria, tipo de área e uma breve descrição do problema. Como um relatório pode conter centenas de problemas, a montagem e formatação destes quadros é o que demanda maior tempo e recursos para a produção do relatório.

Para a montagem desses documentos, softwares clássicos de edição de textos (*Microsoft Word*, principalmente) são muito utilizados. Esses programas convencionais são alternativas genéricas, possuindo poucas ferramentas de automação especializadas que facilitem o trabalho do engenheiro. Por exemplo, fotografias tiradas em uma vistoria técnica, ao serem incorporadas ao relatório, necessitam que sejam manualmente configuradas com tamanho adequado para ocupar o espaço correto. Cada problema encontrado pelo engenheiro pode conter de uma a seis fotografias diferentes, cada uma delas podendo estar na orientação paisagem ou retrato, e todas elas precisarão ser formatadas e revisadas pelo engenheiro que realizou a vistoria.

Visando suprir a necessidade de melhorar o tempo de produção desses documentos, este trabalho visa desenvolver um *MVP* (*Minimum Viable Product*) de uma aplicação web para

*Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP

automação da montagem desses relatórios técnicos. Em conjunto com um engenheiro especialista da Servare Engenharia, foram levantados os requisitos para a aplicação. Assim, utilizando o software implementado, o engenheiro seria responsável apenas pela coleta dos dados durante a vistoria e pela submissão dessas informações na plataforma. Essa submissão pode ser realizada tanto no momento da coleta dos dados durante a vistoria técnica (*on-the-fly*) utilizando um *smartphone*, quanto posteriormente no escritório. O aplicativo web seria responsável por salvar todos os dados em nuvem e montar o relatório no formato PDF, com as formatações corretas e buscando otimizar o tamanho da folha A4. Além disso, a aplicação deve prover um sistema de autenticação e gerenciamento de usuários (funcionários da Servare), onde cada usuário terá seus próprios projetos, podendo editá-los, excluí-los e gerar um arquivo PDF a partir dele. O usuário também deve ter a possibilidade de compartilhar seus projetos com outros usuários da sua escolha (que também poderão visualizar o arquivo PDF), funcionalidade importante para revisão de trabalhos desenvolvidos por estagiários. Ao ser compartilhado, o projeto deve aparecer na lista de projetos dos usuários escolhidos.

Por se tratar de um MVP, algumas funcionalidades não foram desenvolvidas por terem uma menor prioridade e por não proporcionar um grande desafio técnico para este trabalho. O projeto será apresentado para a Servare e, caso desejarem dar prosseguimento com o desenvolvimento, as implementações faltantes serão concluídas. Assim, foi implementado um sistema de usuários funcional, porém sem sistemas robustos de segurança, como *MFA* (*Multi Factor Authentication*), validação de dados de usuário via e-mail, entre outros. Também, o arquivo PDF gerado possui apenas os quadros de problemas encontrados na vistoria (parte mais trabalhosa para o relatório), necessitando ainda a implementação da capa, textos iniciais e finais. Ainda há outras possíveis melhorias para incrementar ao projeto, mas serão desenvolvidos caso ocorra a continuação do projeto junto à Servare.

2. Objetivo

O objetivo deste trabalho é o desenvolvimento de uma aplicação web para automação de relatórios técnicos produzidos por uma empresa de engenharia civil. O projeto desenvolvido deverá oferecer um MVP com as funcionalidades mais cruciais para a automação da produção do relatório de modo que possa ser apresentado para a Servare, exemplificando o potencial aumento de desempenho que a aplicação pode fornecer.

3. Metodologia

Nesta seção serão discutidas etapas anteriores à implementação do projeto. As decisões de implementação tomadas nesta etapa se baseiam em arquiteturas e metodologias muito consolidadas na indústria de software.

3.1 Levantamento de requisitos

Antes do desenvolvimento da aplicação foi realizada uma reunião com o engenheiro Jonathas de Alencar Lopes, funcionário da Servare Engenharia. Jonathas é responsável por realizar vistorias técnicas para encontrar patologias e problemas em edifícios e produzir o relatório de não conformidades, além de ser responsável por capacitar estagiários a realizar essa tarefa. Na reunião foram levantadas todas as dificuldades e necessidades que uma aplicação a ser desenvolvida poderia suprir. A principal questão levantada durante a discussão foi não haver uma forma prática de manipular as imagens no documento, sendo necessário um trabalho muito manual e demorado (a produção de um relatório de 100 páginas pode demorar

em torno de 30 horas para sua conclusão), o que acaba custando caro para a empresa, já que é normalmente realizado pelo engenheiro.

Assim, os principais requisitos elencados durante essa reunião foram:

- A aplicação deve automatizar o processo de dimensionamento e posicionamento das imagens do relatório;
- A aplicação deve automatizar a formatação de textos inseridos;
- A aplicação deve automatizar a criação de um quadro a partir das imagens e dados inseridos pelo engenheiro;
- A aplicação deve permitir que o engenheiro insira fotos e preencha as informações para a criação de um quadro de problemas durante uma visita técnica (*on-the-fly*). Com isso, o engenheiro pode preencher o quadro enquanto as informações encontradas na vistoria ainda estão frescas na mente, já que não seria necessário se deslocar para o escritório para preencher as informações;
- A aplicação deve permitir que engenheiros possam visualizar documentos desenvolvidos por outros funcionários;
- A aplicação não deve ser acessível para todos, apenas para pessoas autorizadas. Portanto, deve haver um sistema de *login* com usuário e senha;

3.2 Arquitetura da aplicação

Para suprir os requisitos levantados, foi desenvolvida uma arquitetura para a aplicação. A arquitetura descreve em alto nível as relações entre as principais entidades do projeto, além de dar uma visão macroscópica do que precisará ser desenvolvido. Buscando um desenvolvimento mais moderno, a arquitetura será baseada no modelo de micro serviços, causando menor acoplamento entre os componentes e maior independência entre as *stacks* de desenvolvimento.

Como a Servare pode decidir dar prosseguimento ao projeto, possivelmente contratando mais desenvolvedores, as tecnologias utilizadas na aplicação já são consolidadas na indústria. Isso facilita encontrar desenvolvedores aptos para dar continuidade ao desenvolvimento, agilizando a curva de integração de novos membros ao time e diminuindo o tempo das entregas.

Assim, a aplicação que rodará no lado do cliente foi desenvolvida utilizando o framework Angular na sua versão 13. Angular é um framework moderno mantido pela Google, permitindo o desenvolvimento de aplicações *SPA* (*Single Page Application*), além de ser bastante escalável. No momento que este trabalho começou a ser implementado, a última versão do framework lançada foi a 14, portanto a versão 13 já estava estabilizada e pronta para ser aplicada em projetos com mais robustez. A linguagem de programação que o Angular utiliza por padrão é o TypeScript, uma linguagem derivada de Javascript, porém tendo sua tipagem estática.

Para aplicar as regras de negócios desejadas, o lado do servidor rodará utilizando NodeJS, um interpretador da linguagem Javascript que permite rodá-la fora do ambiente de navegadores web. Por se tratar de uma linguagem com uma curva de aprendizado mais amigável e também já consolidada na indústria, é uma boa escolha para diminuir o tempo de implementação. Além disso, utilizando Javascript na *server side* e Typescript no *client side* tem-se uma variedade menor de tecnologias envolvidas no projeto, já que ambas são muito parecidas. Isso facilita que um desenvolvedor crie código para ambas *stacks* e possa produzir em qualquer parte do projeto.

Para persistência dos dados, foi utilizado o SGBD relacional MySQL. Ele foi escolhido por também ser uma tecnologia consolidada na indústria, possuir código aberto e ter sido utilizado durante a graduação na disciplina de banco de dados.

A partir das tecnologias selecionadas anteriormente, foi montado o diagrama de arquitetura demonstrado a seguir:

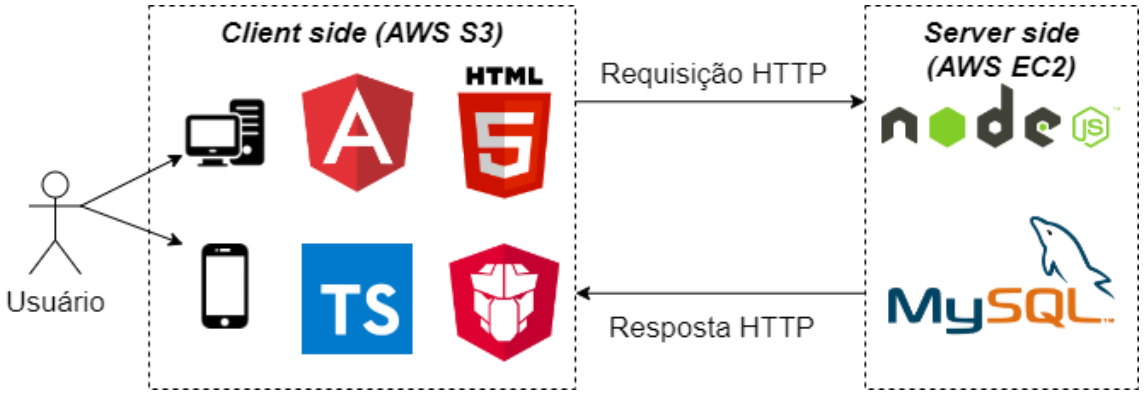

Figura 1: Diagrama da arquitetura

3.3 Git flow

Visando organizar o versionamento do código produzido durante o desenvolvimento, foi utilizado o Git Flow, com algumas alterações. A utilização deste fluxo de trabalho facilita bastante o gerenciamento de *branches*, principalmente em equipes com muitos desenvolvedores, dando maior fluidez ao processo de entregas.

Assim, foi criada a *branch release* a partir da *master*. Ela é responsável por acumular funcionalidades para serem incorporadas na *master*. A *branch master* é responsável por conter o código previamente testado que rodará para o cliente final e por definir a versão do código. Além disso, para cada *feature* a ser desenvolvida, foi criada uma *branch* a partir da *release.* Ao final do desenvolvimento da *feature*, essa branch é incorporada à *release.* Como no projeto ainda não há uma equipe de *QA* (*Quality Assurance*), não foi criada uma *branch* de homologação, que serviria para testes de novas funcionalidades. A Figura 2 ilustra o fluxo de trabalho utilizando o Git Flow.

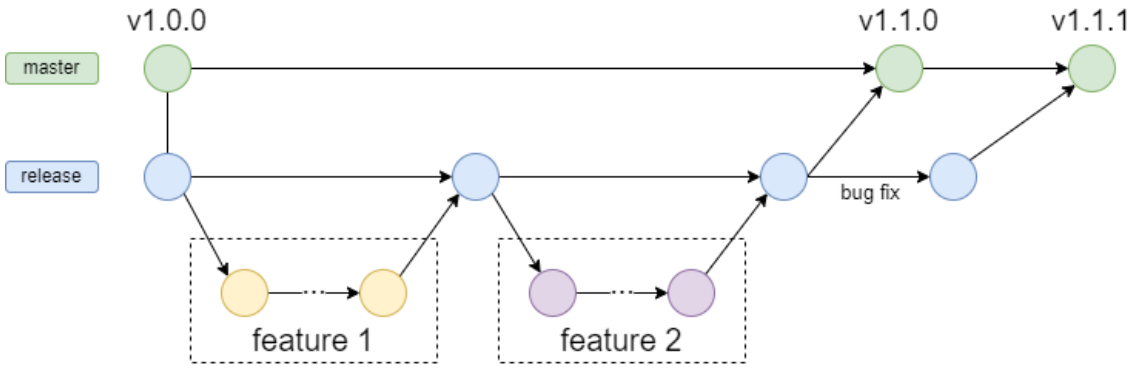

Figura 2: Git flow

3.4 Ambientes

Em concordância com as práticas *agile* de desenvolvimento de *software*, foram utilizados diferentes ambientes de utilização da aplicação para os diferentes estágios do seu ciclo de vida, cada um deles possuindo um banco de dados próprio, uma instância EC2 rodando o *backend* e um bucket S3 fornecendo o *frontend*. Com isso, um usuário criado em um ambiente não terá acesso ao outro, além de projetos criados em um ambiente não serão acessíveis em outros. Além disso, cada um dos ambientes será acessado por uma *url* diferente.

Para este projeto foram utilizados os ambientes de *staging* e *production*. A função do ambiente de *staging* é ser o mais parecido possível com o utilizado pelo usuário final, o qual engenheiros da Servare possam testar a aplicação sem prejudicar seus projetos de verdade. Esse ambiente também pode ser usado para desenvolvimento, permitindo ao desenvolvedor criar código para o *frontend* utilizando um *backend* e um banco de dados mais parecido com o real. Enquanto o ambiente de *production* é aquele que será realmente utilizado pelos engenheiros da Servare para a criação dos documentos. Por ter contato direto com o usuário final, este ambiente possui mais camadas de segurança e eficiência, como processos de *deploy* com testes automatizados, *load balancer*, entre outros. Por padrão, a *branch master* é a única que deve rodar neste ambiente.

4. Desenvolvimento

Nesta seção serão discutidos detalhes da implementação de cada parte do projeto. Também serão discutidas suas infraestruturas e a forma como foram hospedados em nuvem.

4.1 Backend

O *backend* desenvolvido é responsável por conter todas as regras de negócio da aplicação. Ele fornece uma *API* (*Application Programming Interface*) para o *frontend* consumir os dados necessários para apresentar ao usuário. Com isso, ele persiste os dados fornecidos pelo engenheiro no banco de dados e em nuvem, como informações dos usuários, projetos, quadros, imagens na AWS, etc.

A implementação do *backend* da aplicação foi feita utilizando a linguagem Javascript, por conta da sua curva de aprendizagem ser mais amigável e por ser também muito utilizada para implementação de *frontend*. Essas características da linguagem facilitam a integração de novos membros ao projeto, ajudando a Servare a dar continuidade no desenvolvimento caso queiram. Foi utilizado o interpretador NodeJS, muito usado na indústria para executar código Javascript fora do ambiente de navegadores.

Para o gerenciamento das requisições *HTTP*, foi utilizado o *framework* ExpressJS. Este *framework* é o mais utilizado na indústria para construção de servidores web, sendo rápido, confiável e robusto. Das principais características que facilitam o desenvolvimento da aplicação utilizando ExpressJS, podemos citar:

- Possui um sistema de rotas completo;
- Possibilita o tratamento de exceções dentro da aplicação;
- Permite a integração de vários sistemas de *templates* que facilitam a criação de páginas web para suas aplicações;
- Gerenciamento de diferentes requisições *HTTP* com seus mais diversos verbos;

- Feito para a criação rápida de aplicações utilizando um conjunto pequeno de arquivos e pastas;

Essas características do *framework* contribuem para que a aplicação seja desenvolvida de forma escalável e ágil.

Para persistência dos dados, no lado do *backend* foi utilizado o Sequelize. Ele é um ORM (Object/Relational Mapper) baseado em *promise*, que suporta o dialeto MySQL e recursos de transação, relacionamentos, replicação de leitura, entre outros. Também muito utilizado e consolidado na indústria, o Sequelize facilita a comunicação da aplicação com o SGBD (Sistema Gerenciador de Banco de Dados), além de prover mais segurança no processo das requisições ao banco.

Através da *API*, são fornecidas rotas que permitem a realização de todas as operações de *CRUD* (*Create, Read, Update Delete*) em todos os dados persistentes em banco. Estes dados são de usuário, projeto, quadro e imagem.

A Figura 3 ilustra os componentes do *backend* bem como sua interação com o *frontend* e o banco de dados.

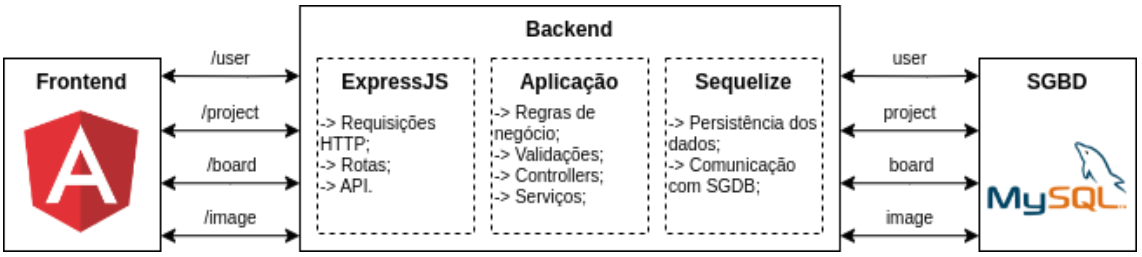

Figura 3: Diagrama do *backend*

4.2 Banco de dados

Como dito anteriormente, o SGDB utilizado é o MySQL. Ele possui código aberto e também está entre os mais utilizados na indústria. Neste tipo de aplicação, o mais comum é que o SGDB rode em uma instância *RDS* (*Relational Database Service*) na AWS, por conta de sua eficiência e segurança. Porém, para facilitar o desenvolvimento e evitar possíveis custos adicionais, ele é hospedado na mesma instância EC2 que o *backend* da aplicação.

Para a persistência dos dados, foram criadas as tabelas *Users*, *Projects*, *Boards*, *Images* e *SharedProjects.* A tabela *Users* guarda dados de usuário, como nome, email e o *hash* da senha do usuário, não sendo possível para um desenvolvedor saber uma senha apenas olhando o banco de dados. A tabela *Projects* guarda as informações dos projetos de todos os usuários, além do identificador do criador de cada projeto como chave estrangeira. A tabela *Boards* salva os dados de todos os quadros criados e também o identificador do projeto do qual ele pertence. A tabela *Images* guarda as *urls* das imagens submetidas pelos usuários, além dos identificadores para o quadro e o projeto ao qual ela pertence. Por fim, a tabela *SharedProjects* faz um relacionamento muitos-para-muitos entre as tabelas *Users* e *Projects*, sendo assim possível mapear para quais usuários cada projeto é compartilhado e quais projetos foram compartilhados com um usuário.

As tabelas e suas relações podem ser visualizadas no diagrama apresentado na Figura 4.

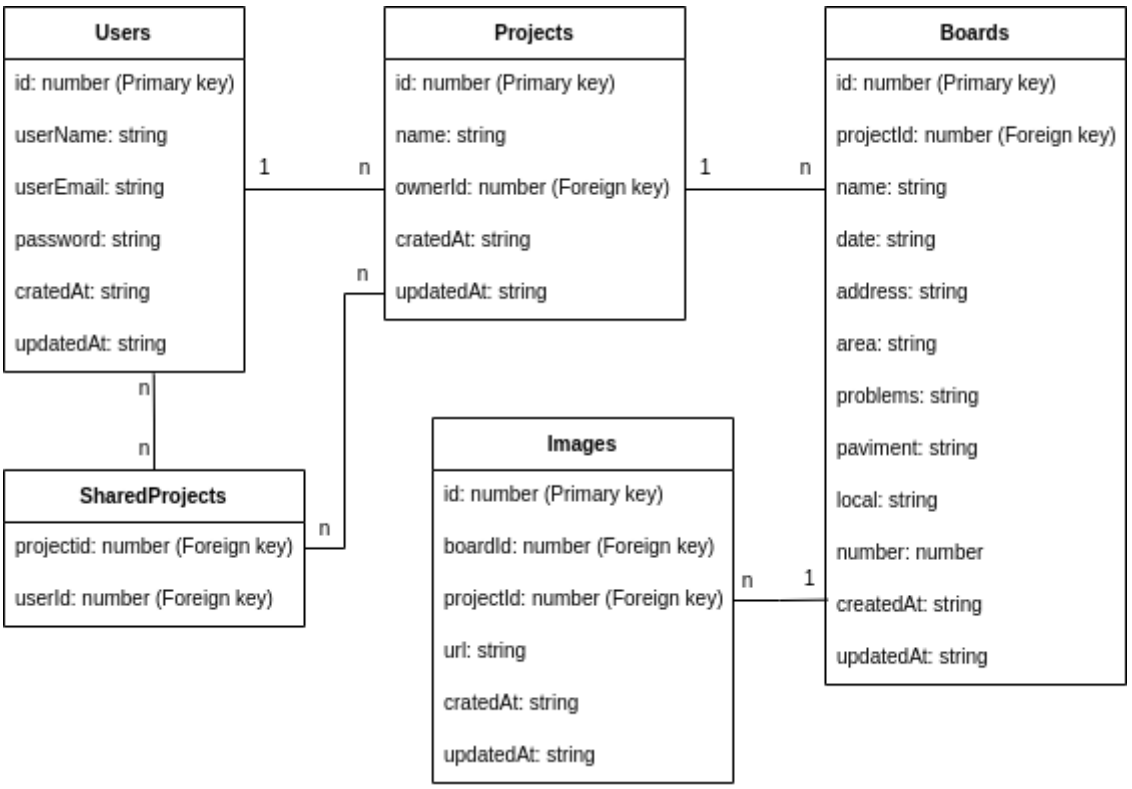

Figura 4: Diagrama do banco de dados

4.3 Frontend

O frontend da aplicação foi desenvolvido utilizando o *framework* Angular, na sua 13ª versão. Muito utilizado na indústria para desenvolvimento de *Single Page Applications*, este *framework* é mantido pela Google, recebendo atualizações constantes e uma nova versão a cada semestre.

Para uma melhor interface com o usuário, foi utilizado PrimeNG, que é uma coleção de componentes de código aberto que fornece uma vasta gama de componentes de interface nativos ao Angular e já integrados com temas escolhidos. Assim, componentes como botões, tabelas, menus, entre outros, possuem sua estilização baseada no tema Lara Light Blue. Como o projeto a princípio foi desenvolvido sem um profissional de *UI/UX*, a utilização do PrimeNG acelerou a implementação das interfaces.

Por se tratar de uma *SPA*, este framework também facilita a utilização da aplicação tanto em computadores quanto em *smartphones*, cumprindo o requisito de utilização *on-the-fly*. Apesar disso, é recomendável que seja desenvolvido um aplicativo para dispositivos móveis nativo caso a empresa deseje dar continuidade ao projeto, pois a aplicação rodaria no navegador do *smartphone*, comprometendo seu desempenho e funcionalidades.

O arquivo PDF do documento é montado pelo *frontend*, a partir dos dados adquiridos pelo *backend*. Para a montagem do documento, foram utilizadas duas bibliotecas: a **JSPdf** e **pdf-lib**. A JSPdf é utilizada para montar as páginas PDF que contém quadros montados pelo usuário. Essa montagem é realizada por código HTML e CSS, facilitando o desenvolvimento e possibilitando a visualização de uma prévia. A pdf-lib é utilizada para fazer a mesclagem de dois arquivos pdf diferentes. Isso é importante, pois nesta etapa do projeto ainda não é possível criar capa para o documento. Assim, como medida paliativa, foi construído um método do usuário poder submeter um arquivo PDF contendo a capa desejada, que será mesclada com o documento gerado pela aplicação.

Como o projeto ainda não possui um profissional em UI/UX, as telas foram desenvolvidas com um design simplificado. A Figura 6 ilustra as principais telas que serão apresentadas à Servare como *MVP*.

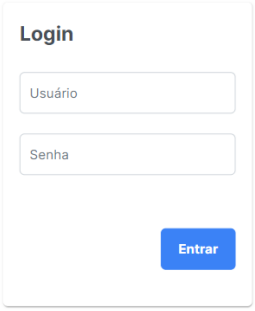

Figura 5: Tela de login

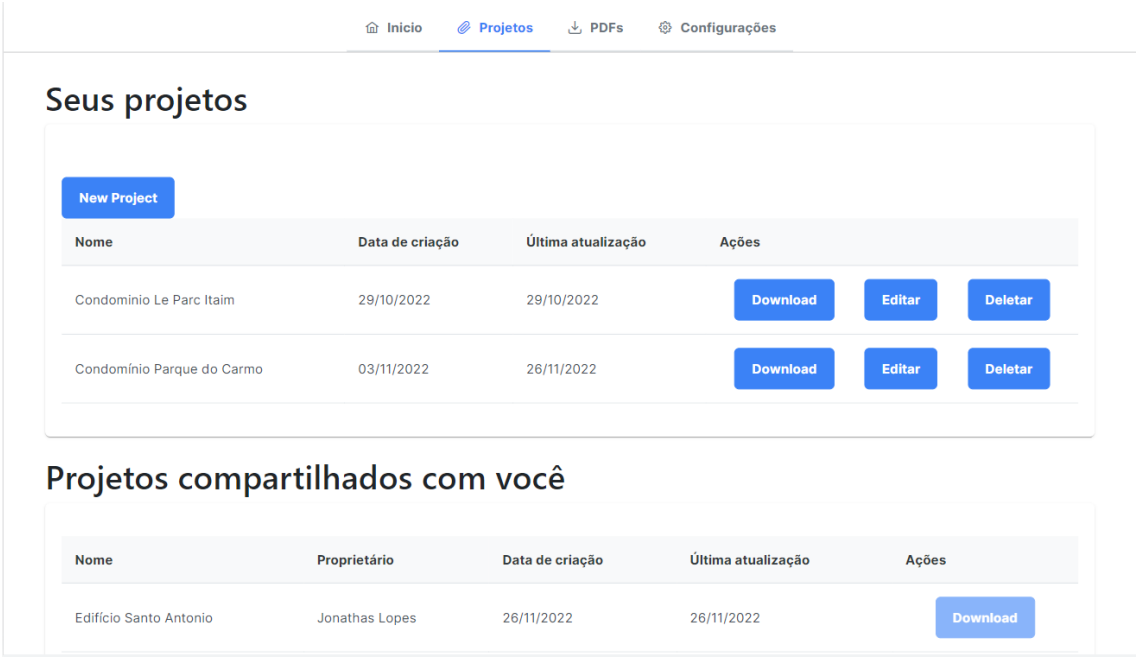

Figura 6: Tela de listagem de projetos

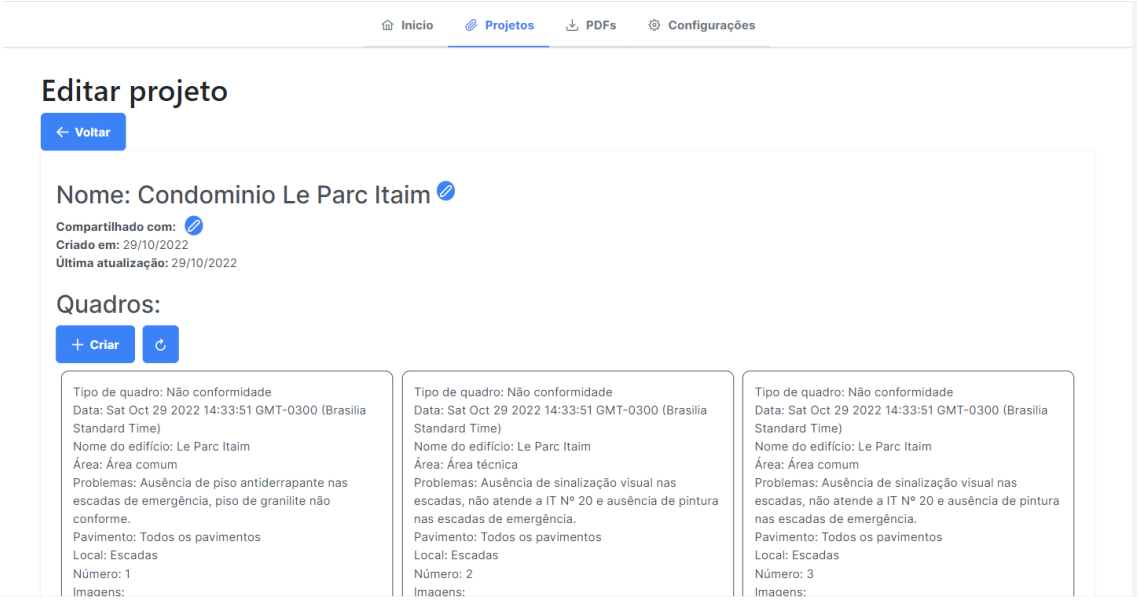

Figura 7: Tela de edição de projeto

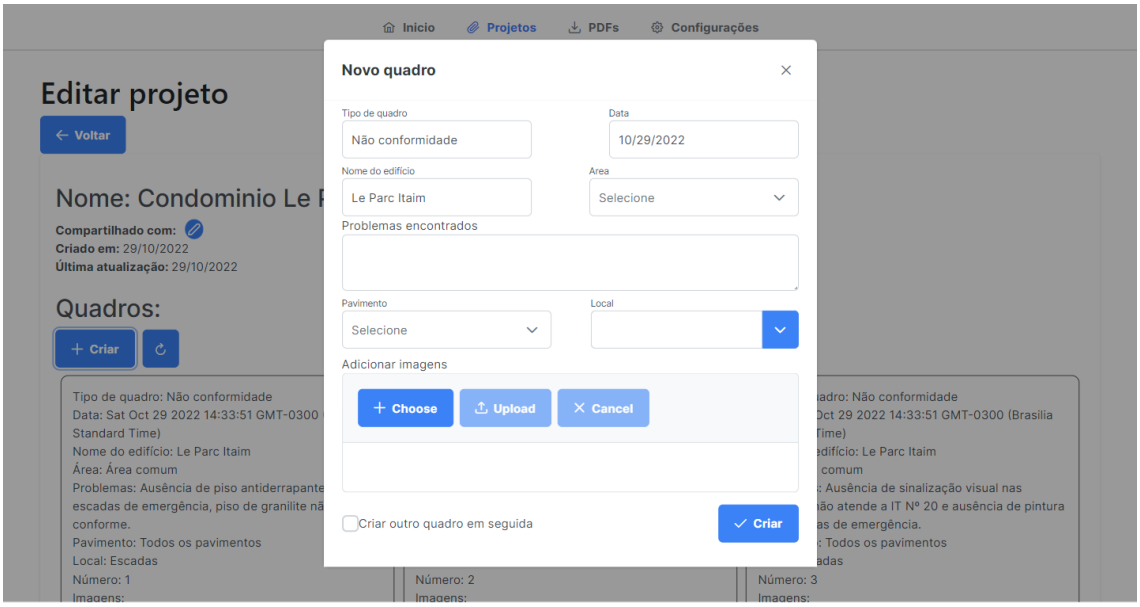

Figura 8: Tela de criação/edição de quadro

4.4 Infraestrutura

Para hospedagem da aplicação, foi utilizado a plataforma de computação em nuvem *AWS* (*Amazon Web Services*). Esta é a plataforma mais utilizada no mundo, contando com centenas de *data centers* pelo mundo inteiro. Ela foi escolhida para hospedar a aplicação por possuir diversos serviços muito robustos, seguros e confiáveis, que podem ser utilizados com a lógica de Infraestrutura como serviço (*Infrastructure as a Service*) sob demanda. Para este projeto, foi utilizado o seu nível gratuito, garantindo um ano de utilização sem custos de vários de seus serviços, com algumas restrições.

O *backend* da aplicação foi hospedado em uma instância *EC2* da *AWS*. O nível gratuito permite a utilização de 750 horas mensais da instância *t2.micro*, o suficiente para um

mês inteiro da aplicação. Esse tipo de instância possui apenas um núcleo de processamento. Além disso, possui apenas 8 GB de armazenamento, o que pode ser um problema com o passar do tempo, já que o banco de dados também está rodando na EC2. Como para este trabalho o projeto ainda é uma prova de conceito, não há uma grande demanda de poder de processamento, já que ainda há poucos usuários.

Para hospedagem do *frontend* e das imagens submetidas pelos usuários, foi utilizado o serviço *S3* da *AWS*. Assim, foram criados dois *buckets*, um para guardar o *frontend* compilado, responsável por prover para toda a internet acesso à aplicação, e outro para guardar as imagens, responsável por fornecê-las ao usuário através de suas URLs salvas no banco de dados.

5. Resultados

O resultado obtido para este trabalho é capaz de gerar um laudo técnico muito parecido com aquele utilizando *softwares* convencionais, porém, com as vantagens de poder ser feito *on-the-fly*, além de acelerar a produção do documento e aumentar a padronização dos quadros de problemas. Apesar de ainda ser necessário realizar a mesclagem de outros arquivos PDF para gerar um laudo completo, após as implementações feitas até aqui, o desenvolvimento de uma funcionalidade capaz de incorporar as páginas iniciais e finais de um relatório não representa um grande desafio técnico. Assim, o projeto está pronto para ser apresentado como um *MVP* para a *Servare*, em que deverão ser apresentadas todas as possíveis jornadas de usuário, suas vantagens e seus custos.

As Figuras 9 e 10 apresentam dois exemplos de páginas com quadros de problemas encontrados durante uma vistoria real. A imagem da Figura 9 foi gerada pela aplicação. É notável o aumento da área utilizada pelo quadro, permitindo que as imagens tenham uma resolução maior na folha A4. A imagem da Figura 10 é uma página criada por um engenheiro da Servare utilizando o software *Microsoft Word*.

SERVARE

Não conformidade	Vistoria - 29/10/2022
Le Parc Itaim	Área comum

5. Quadro:	**Pavimento:** Todos os pavimentos
Ausência de sinalização visual nas escadas, não atende a IT Nº 20 e ausência de pintura nas escadas de emergência.	**Local:** Escadas

27/11/2022 5/5

Figura 9: Página com quadro de problema gerado pela aplicação

SERVARE

Não Conformidade	Vistoria – 16/03/2022
Condomínio Le Parc Itaim	Área Comum

2. Quadro:	Pavimento:
Ausência de sinalização visual nas escadas, não atende a IT Nº 20 e ausência de pintura nas escadas de emergência.	Todos os pavimentos
	Local: Escadas de Emergência 01 e 02

Figura 10: Página com quadro de problema feito utilizando softwares convencionais

6. Conclusão

Apesar de resultados muito satisfatórios para um *MVP*, para que a aplicação possa ser utilizada de maneira robusta e segura, ainda há algumas funcionalidades que necessitam de implementação. Essas funcionalidades ficaram como pendência, para serem desenvolvidas apenas caso a Servare decida seguir com o projeto. Algumas dessas pendências são:

- Sistema de autenticação multi-fatores;
- Aprimoramento da edição de projetos/quadros/imagens;
- Tabelas com sistemas de busca e paginação;
- *User experience* e *user interface* adaptadas para aplicação móvel e pequenas telas;
- Sistema de criação de usuário disponível para usuários com tal permissão;
- Sistema de reordenamento de quadros.

Muitos conceitos vistos durante a graduação foram utilizados para o desenvolvimento deste projeto. Por exemplo, a relação muitos-para-muitos entre as tabelas de usuários e de projetos foi crucial para o sistema de compartilhamento, conceito que foi abordado na disciplina de Banco de Dados: Teoria e Prática. Além disso, as disciplinas de Redes de Computadores e Sistemas Distribuídos foram fundamentais para definir a melhor estratégia para hospedar a aplicação em nuvem na *AWS*.

Referências

[1] MARTIN, Robert Cecil. *Clean Code: A Handbook of Agile Software Craftsmanship*. Londres: Pearson Education, 2008.
[2] DRIESSEN, Vincent. *A successful Git branching model*. NVIE, 2020. Disponível em: https://nvie.com/posts/a-successful-git-branching-model/. Acesso em: 13 nov. 2022.
[3] Sequelize. Disponível em https://sequelize.org/docs/v6/getting-started/. Acesso em 07/08/2022.
[4] Codd, Edgar Frank. *A relational model of data for large shared data banks*. Communications ACM, 13(6), 377-387, 1970.
[5] ExpressJS. Disponível em https://expressjs.com/pt-br/. Acesso em 23/07/2022.
[6] Angular. Disponível em https://angular.io/start. Acesso em 21/08/2022.
[7] PrimeNG. Disponível em https://www.primefaces.org/primeng/setup. Acesso em 21/08/2022.
[8] PDF-LIB. Disponível em https://pdf-lib.js.org/. Acesso em 08/09/2022.
[9] jsPDF. Disponível em https://www.npmjs.com/package/jspdf. Acesso em 05/11/2022.