UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Victor Carrilho Marques

Desenvolvimento de um sistema web para gerenciamento de patrimônios históricos

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Victor Carrilho Marques

Desenvolvimento de um sistema web para gerenciamento de patrimônios históricos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Bruno Augusto Nassif Travençolo

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2024

Agradecimentos

Primeiramente, agradeço a Deus por me guiar e fortalecer ao longo desta jornada.

Agradeço em especial à minha esposa, Ana Jullia Carrilho, que esteve comigo desde a minha formação no ensino médio, me incentivou a escolher o curso de Sistemas de Informação para graduar, me deu todo o suporte e apoio para concluir este trabalho e com quem tenho vivido os melhores dias da minha vida. Obrigado por não deixar eu baixar a cabeça. Eu amo você, Ana Jullia.

Agradeço aos meus pais, Marivaldo Júnior e Rejane, que me deram todo carinho e amor, que sempre me apoiaram e que "... sob muito sol, fizeram-me chegar até aqui, na sombra". Amo vocês.

Agradeço ao meu orientador Professor Bruno Travençolo, pelo apoio e pela paciência durante a construção deste trabalho, e ao meu colega, Igor Abe, pela contribuição no projeto e pela parceria na empresa onde trabalhamos juntos.

Resumo

Este trabalho de conclusão de curso aborda o desenvolvimento de um sistema para o gerenciamento de patrimônios históricos, utilizando tecnologias relevantes como Java, Spring Boot, Angular, e PostgreSQL. A motivação principal do projeto foi a necessidade de facilitar o acesso e a gestão de informações sobre patrimônios históricos, visando a preservação e valorização cultural dos mesmos. O objetivo foi criar uma plataforma que permita tanto a visualização pública dos patrimônios quanto o gerenciamento restrito a usuários autorizados. O sistema desenvolvido oferece funcionalidades como cadastro, edição e busca de patrimônios, além de uma interface simples para os visitantes interagirem com as informações disponíveis. Foram realizados testes de integração para validar o funcionamento correto dos recursos disponibilizados pela API criada para atender ao sistema. Como resultado, o projeto atingiu seu objetivo, entregando um sistema eficiente para o gerenciamento de dados relacionados aos patrimônios históricos.

Palavras-chave: Patrimônio histórico, desenvolvimento de software, sistema web, teste de software.

Lista de ilustrações

Sumário

1 Introdução

O patrimônio histórico é um acervo onde se encontram diversos tipos de documentos referentes a lugares, edifícios, monumentos, vegetação, acontecimentos históricos e manifestações de um lugar, comunidade ou sociedade. Guardar esses registros e preservar esses lugares é fundamental para preservar a identidade e a memória de um povo. No entanto, a falta de incentivo, o descaso, o foco no turismo ou a perda de identidade podem levar esses bens à destruição, a demolição ou a uma transformação distorcida da realidade histórica (LOPIS, 2017).

Em relação à importância de preservação desses acervos, vemos na história que houve muitas preocupações e debates quanto ao impacto da perda desses para as gerações futuras, uma vez que essas significações transformaram a sociedade e fazem parte de quem somos hoje. Nos séculos XIX e XX, essa preocupação chegou ao Estado, que a partir dali passou a estimular a criação de leis para conservação e restauração de culturas e edificações que representaram o modo de vida de determinada época (LONDRES, 2007).

Ao olhar para a literatura, a preservação dos patrimônios históricos tem origem na França e na Europa, no século XIX, após atos de vandalismos. Foram então criadas leis, realizados inventários e criadas instituições para abrigarem esses acervos, com intuito de conservar e restaurar monumentos que remetem a acontecimentos históricos (KÜHL, 2007).

No Brasil, a política patrimonial vem sendo construída até os dias atuais, de forma lenta e gradual. Em 1937, foi criado o serviço do Patrimônio Histórico e Artístico Nacional (SPHAN), com o decreto lei federal n º 25, de 30 de novembro de 1937, que define patrimônio como "conjunto de bens móveis e imóveis existentes no País e cuja conservação seja de interesse público, quer por seu excepcional valor arqueológico ou etnográfico, bibliográfico ou artístico" (BRASIL, 1937).

Posteriormente, no ano de 1988 tem-se a promulgação da Constituição Federal, na qual consta o artigo 216, o qual aborda a definição de patrimônio cultural brasileiro, conforme citado abaixo:

> "Constituem patrimônio cultural brasileiro os bens de natureza material e imaterial, tomados individualmente ou em conjunto, portadores de referência à identidade, à ação, à memória dos diferentes grupos formadores da sociedade brasileira, nos quais se incluem: I – as formas de expressão; II – os modos de criar, fazer e viver; as criações científicas, artísticas e tecnológicas; IV – as obras, objetos, documentos, edificações e demais espaços destinados

> às manifestações artístico-culturais; V – os conjuntos urbanos e sítios de valor histórico, paisagístico, artístico, arqueológico, paleontológico, ecológico e científico" (BRASIL, 1988).

Também no artigo 216, parágrafo 1, é explicado como ocorrerá a proteção do Patrimônio Histórico Brasileiro pelo Poder Público, conforme citado abaixo:

> "Parágrafo 1º - O Poder Público, com a colaboração da comunidade, promoverá e protegerá o patrimônio cultural brasileiro, por meio de inventários, registros, vigilância, tombamento e desapropriação e de outras formas de acautelamento e preservação" (BRASIL, 1988).

Nos anos seguintes, houve conquistas, como exemplo, em 1990, foi criado o Instituto Brasileiro do Patrimônio Cultural (IBPC), depois nomeado como Instituto do Patrimônio Histórico e Artístico Nacional (IPHAN). E em 2003, o patrimônio foi definido de forma mais ampla, tanto em suas práticas quanto em seus objetos (CARVALHO, 2011).

Pensando nos dias atuais, uma das principais dificuldades encontradas é obter o apoio da sociedade para preservação desses bens, uma vez que não há educação patrimonial, o que faz com que muitos desconheçam a importância de manter a memória coletiva, rejeitando assim as medidas de preservação impostas pelo Estado. Logo, a depredação de bens patrimoniais é uma preocupação, e para que isso mude é necessário que a legislação seja efetivada na prática, com punição conforme a lei e que haja investimento em conscientização da sociedade quanto aos nossos bens culturais (MEDEIROS; SURYA, 2009).

1.1 Objetivos

Este trabalho tem como objetivo geral desenvolver um sistema web para gerenciamento de patrimônios históricos tombados da cidade de Uberlândia - Minas Gerais, com o intuito de otimizar a gestão dos ativos municipais, e a partir da base de dados, gerar uma página de visitação para o público em geral.

1.2 Justificativa

A gestão eficiente dos ativos municipais é crucial para garantir o bom funcionamento da administração pública dos municípios. Nesse sentido, pensando na cidade de Uberlândia, a implementação de um sistema de cadastramento de patrimônios surge como uma solução necessária para melhorar a administração e controle dos bens municipais.

Esse sistema proporcionará uma gestão mais eficaz dos recursos públicos, facilitando o controle, a localização e o monitoramento dos bens municipais. Além disso, promoverá a transparência na gestão pública, permitindo que os cidadãos acompanhem como os recursos estão sendo utilizados, e contribuindo para a prevenção de perdas e desvios.

Assim, a criação desse sistema representa um avanço significativo na modernização da gestão pública em Uberlândia, alinhando a cidade com as melhores práticas de governança e gestão de ativos.

Por fim, vale mencionar que este projeto foi desenvolvido em conjunto com outro trabalho de conclusão de curso e, por isso, algumas funcionalidades aqui descritas também estão presentes no outro trabalho (ABE, 2024).

1.3 Organização

Este trabalho foi desenvolvido em cinco capítulos, divididos da seguinte forma:

- Capítulo 1: traz a introdução, os objetivos e a justificativa da proposta apresentada neste trabalho;
- Capítulo 2: descreve acerca do referencial teórico, em que abordam-se as tecnologias utilizadas tanto para o desenvolvimento do *backend* quanto do *frontend*;
- Capítulo 3: aborda o desenvolvimento do sistema, em específico sua arquitetura e o banco de dados a ser usado pelo mesmo;
- Capítulo 4: aborda a interface e funcionamento do sistema, apresentando o sistema de autenticação e autorização, a página de login e as áreas internas do sistema referentes aos perfis. Por fim, também é dado destaque ao resultado dos testes automatizados implementados;
- Capítulo 5: apresenta a conclusão deste trabalho sobre o benefício do sistema de gerenciamento dos patrimônios históricos municipais, além de descrever algumas funcionalidades a serem desenvolvidas como continuação deste projeto.

2 Referencial Teórico

2.1 Trabalhos Correlatos

2.1.1 Instituto do Patrimônio Histórico e Artístico Nacional (IPHAN)

O Instituto do Patrimônio Histórico e Artístico Nacional (IPHAN) é um órgão federal criado a partir da lei nº 8.113 de 1990. Sediado em Brasília, o IPHAN está ligado ao Ministério da Cultura e atua em todo o Brasil com a missão de preservar o patrimônio cultural brasileiro de forma sustentável, promovendo a cidadania e valorizando a diversidade cultural, conforme o artigo 216 da Constituição Federal (IPHAN, 2023).

O IPHAN possui um portal próprio no qual é possível consultar diversas informações relacionadas ao patrimônio cultural brasileiro (Figura 1). O menu leva às páginas com informações sobre a estrutura organizacional do Instituto e suas superintendências, a conteúdos como fototecas e vídeos de eventos e documentários, a referências bibliográficas, além das páginas com detalhes e informações sobre bens tombados, bens inventariados e acervos. Essa última parte é semelhante ao objetivo do sistema desenvolvido a partir deste trabalho (Figura 2).

Figura 1 – Página inicial Portal IPHAN. Disponível em: http://portal.iphan.gov.br/

Vale dizer que o site está sendo migrado para dentro da plataforma gov.br, con-

Figura 2 – Página de exemplo acessível a partir do Portal IPHAN.

forme mensagem apresentada na página inicial do portal (Figura 1). Com isso, alguns links do portal levam até páginas já migradas para a plataforma do governo federal (Figura 3), assim como alguns links presentes em páginas da plataforma levam de volta ao portal.

2.1.2 Retomba

O Retomba é um aplicativo desenvolvido a partir de um projeto universitário do Núcleo de Teoria e História em Arquitetura e Urbanismo (NUTHAU) da Faculdade de Arquitetura, Urbanismo e Design (FAUeD) da UFU (Comunica UFU, 2024). O aplicativo tem como recurso principal a possibilidade de, utilizando a câmera do celular, visualizar modelos 3D de edifícios antigos sobrepostos aos edifícios atuais através de técnicas de realidade aumentada (RETOMBA, 2024). Além desse recurso como destaque, o aplicativo disponibiliza um mapa que marca a localização dos edifícios históricos, apresentando fotos e textos sobre a história dos locais, sendo possível também interagir com comentários sobre os lugares e sugerindo o registro de outros edifícios.

O pacote para baixar o aplicativo está disponível na página web do projeto Retomba. No site não é possível utilizar a funcionalidade de visualização dos modelos 3D em realidade aumentada, mas outras funções, como o mapa e a sugestão de edifícios, estão disponíveis (Figuras 4, 5 e 6).

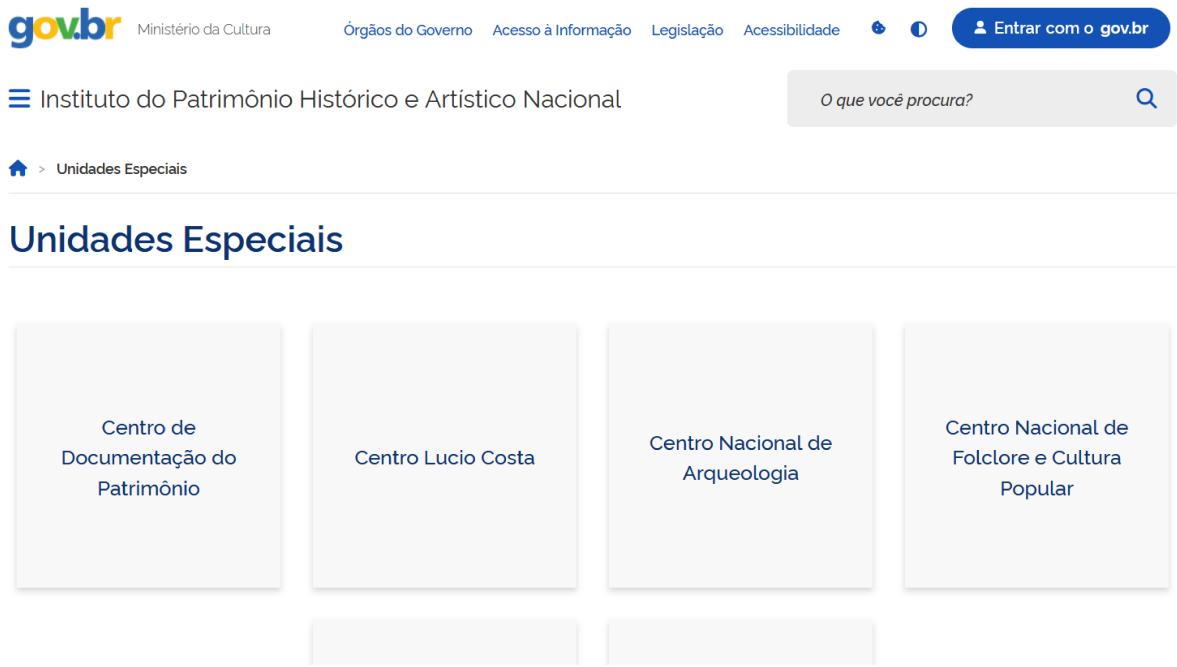

Figura 3 – Página de exemplo do IPHAN na plataforma do gov.br. Disponível em: https://www.gov.br/iphan/pt-br

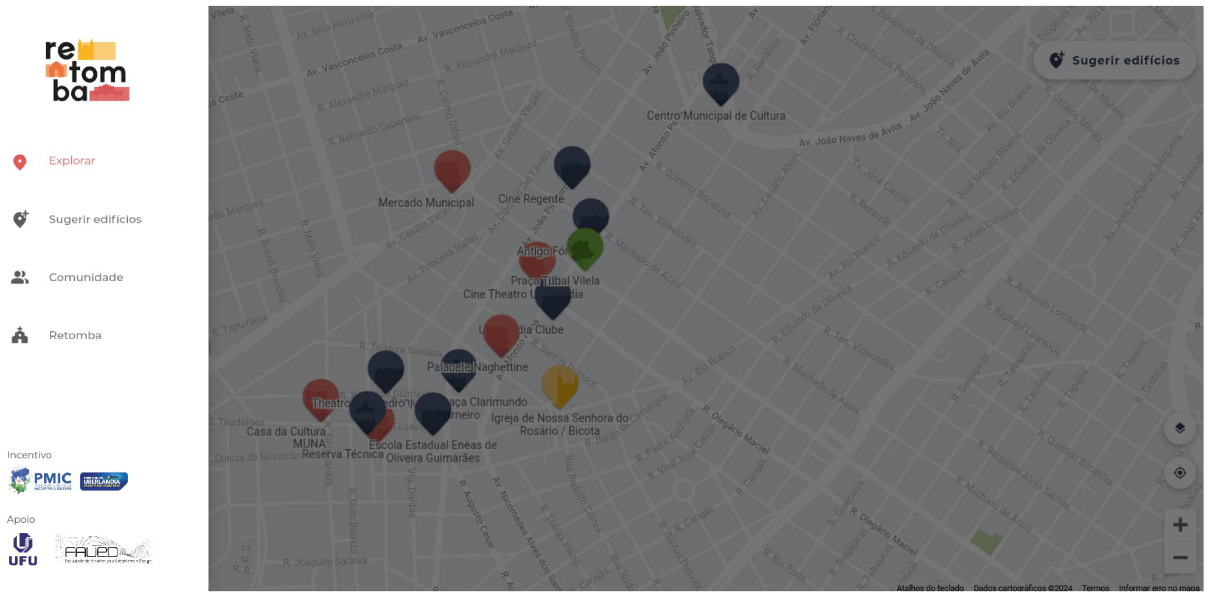

Figura 4 – Página web do Retomba. Disponível em: https://retomba.com.br/#/

2.2 Tecnologias

2.2.1 Java

Java é uma linguagem de programação orientada a objetos e plataforma de desenvolvimento de software. Tem como uma uma de suas principais características o fato de

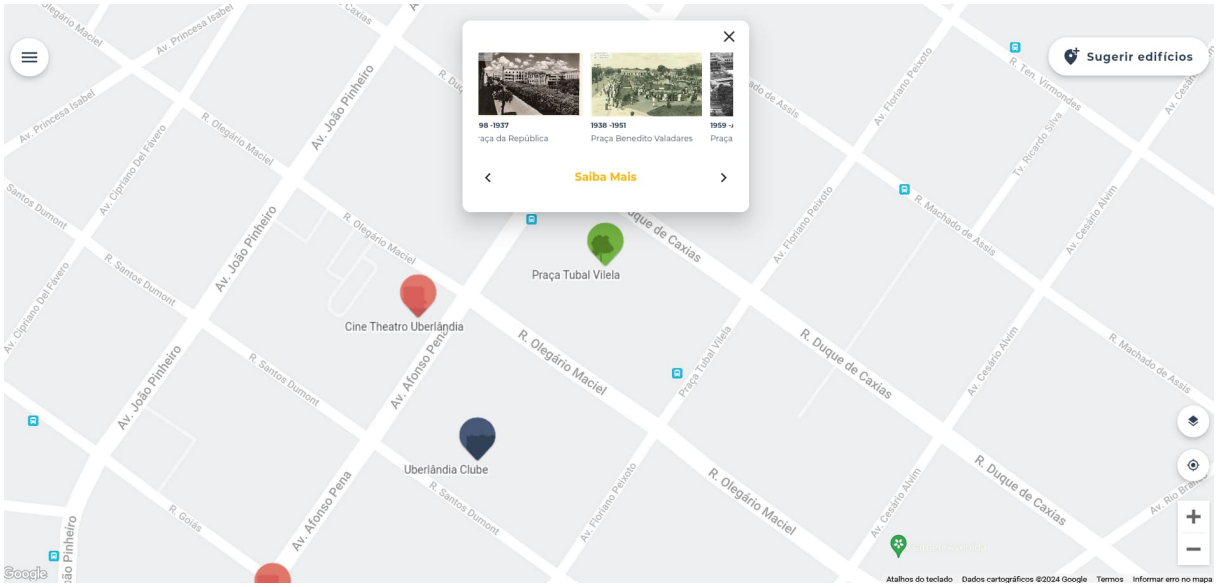

Figura 5 – Mapa na página web do Retomba.

Figura 6 – Página do Retomba. Informações sobre o lugar.

ser multiplataforma, ou seja, um mesmo código escrito em Java pode ser executado em diferentes plataformas e sistemas operacionais, bastando que a máquina possua a JVM instalada (ORACLE, 2024; IBM, 2023).

A linguagem foi lançada em 1995 pela Sun Mycrosystems, mas atualmente é desenvolvida pela Oracle após a aquisição da Sun em 2010 (O GLOBO, 2010). Mesmo com essa transferência de propriedade, a Java continou evoluindo ao longo do tempo e marca presença no mercado atual, oferecendo uma plataforma confiável e versátil para diferentes tarefas de programação (IBM, 2023).

2.2.2 Spring

O Spring é um *framework* que simplifica o desenvolvimento de software em Java, destacando-se por ser altamente configurável e por suportar variados tipos de arquiteturas, já que é dividido em módulos que podem ser usados de acordo com as necessidades da aplicação (SPRING, 2024d).

A ferramenta surgiu em 2003 em resposta à complexidade das primeiras especificações J2EE. De código aberto, hoje possui uma grande comunidade que acompanha e fornece *feedback*, contribuindo assim para a constante evolução da tecnologia. Com o passar do tempo novos projetos foram construídos sobre o Spring Framework, formando um ecossistema conhecido por muitos apenas como "Spring". Alguns desses projetos são o Spring Boot, o Spring Data e o Spring Security, que serão usados no desenvolvimento do projeto desse trabalho (SPRING, 2024d).

2.2.2.1 Spring Boot

O Spring Boot é um dos módulos construídos sobre o Spring Framework. Ao utilizá-lo, a configuração de aplicações Spring é facilitada para o desenvolvedor, pois ele realiza configurações de forma automática. Dessa forma, o desenvolvedor pode direcionar sua atenção mais para as regras de negócio e menos para configurações de projeto, ajudando assim a acelerar o processo de desenvolvimento (SPRING, 2024a).

O Spring Boot incorpora servidores web, como o Tomcat e o Jetty, dispensando a implantação de arquivo WAR para e execução da aplicação e tornando esse processo muito mais prático, como um aplicativo "autônomo" (SPRING, 2024a). Esse é apenas um exemplo de como o Spring Boot facilita o processo de desenvolvimento de aplicações web, o que justifica o uso dessa tecnologia.

2.2.2.2 Spring Data

Dentre os projetos Spring está o Spring Data, como facilitador na implementação de tarefas de acesso a dados (SPRING, 2024c). Esse módulo traz consigo uma família de subprojetos mais específicos, incluindo o Spring Data JPA, que como o nome sugere, utiliza a JPA (Java Persistence API) para a implementação de repositórios para persistência de objetos Java (SPRING, 2024b).

O Spring Data JPA será utilizado no desenvolvimento deste projeto por descomplicar a implementação das camadas de acesso a dados através de configurações simples utilizando anotações. Além disso, ele ainda traz suporte para paginação e é capaz de se integrar com código personalizado para acesso a dados, como consultas personalizadas que vão além das operações simples (persistência, atualização, seleção e remoção), sendo possível filtrar por campos específicos e utilizar ordenação, por exemplo (SPRING, 2024b).

2.2.2.3 Spring Security

O Spring Security fornece recursos para autenticação e autorização. Quando o Spring *framework* é a tecnologia escolhida para a construção de uma aplicação, esse é o projeto padrão para a implementação de um sistema seguro, pela integração com o *framework*, obviamente, e pela possibilidade de personalização ao implementar esses mecanismos de segurança (SPRING, 2024e).

2.2.3 JWT

O JSON Web Token (JWT) é um padrão aberto (RFC 7519) que define uma forma compacta e autocontida para representação de informações, através de um *token* assinado digitalmente, assim permitindo que as informações sejam transferidas com segurança entre as partes envolvidas (JSON, 2024).

É muito comum o uso de JWT na autorização de requisições feitas por um usuário em uma API, por exemplo, quando durante a autenticação do usuário o servidor gera um *token* JWT e o retorna para o cliente que está realizando a requisição. Então nas requisições seguintes o *token* JWT é passado para o servidor para que o mesmo verifique através do *token* se é permitido que o usuário tenha acesso à rota, serviço ou recurso. Dessa forma não é necessário que o usuário envie novamente suas credenciais a cada nova requisição (JSON, 2024).

2.2.4 Angular

O Angular é um *framework* de desenvolvimento de aplicativos de página única para a web que simplifica a construção de aplicações. Essa simplicidade se deve em grande parte ao Angular CLI (*Command Line Interface*), pois, através de comandos simples é possível gerar um projeto Angular base e gerar novos componentes conforme a demanda da aplicação. O Angular também fornece bibliotecas com recursos para implementação de roteamento, gerenciamento de formulários, comunicação cliente-servidor, dentre outros. Além disso, tem como característica e parte fundamental a arquitetura baseada em componentes, o que induz à construção de projetos bem estruturados, organizados, escaláveis e de fácil manutenção (GOOGLE, 2023).

2.2.5 Angular Material

O Angular Material é uma biblioteca de componentes desenvolvida pela equipe do Angular. Portanto, se integra muito bem a projetos Angular, sejam novos ou já existentes (ANGULAR MATERIAL, 2024). Essa biblioteca disponibiliza vários componentes prontos para uso, como tabelas, botões, controle de paginação, dentre outros componentes interessantes que facilitam o processo de desenvolvimento de páginas em projetos Angular.

2.2.6 TypeScript

Na abordagem sobre o Angular, foi dito que o mesmo foi construído sobre o TypeScript. Mas antes de abordar o TypeScript, será explicado de forma sucinta o que é o JavaScript, afinal, uma de suas principais fraquezas foi o motivo da criação do TypeScript (MICROSOFT, 2024c).

O JavaScript é uma linguagem de programação leve e dinâmica, muito utilizada na implementação de funcionalidades interativas de páginas web. Criada em 1995 com o propósito de que pequenos trechos de código fossem incorporados em páginas web, ao longo do tempo tornou-se tão popular que foram desenvolvidas tecnologias que permitem a execução de código JavaScript fora do navegador, sendo o Node.js um exemplo (MOZILLA, 2022).

O TypeScript é uma linguagem de programação que oferece todos os recursos do JavaScript adicionando o sistema de tipos, sendo assim uma linguagem fortemente tipada (MICROSOFT, 2024d; MICROSOFT, 2024a). O JavaScript é uma linguagem de tipagem dinâmica, o que permite falhas na escrita do código, como por exemplo, definir em algum local do código um valor de um determinado tipo que é diferente do tipo esperado, seja por um erro de digitação ou por uma suposição incorreta por parte do desenvolvedor sobre o funcionamento do *script* em tempo de execução. O TypeScript soluciona esse problema ao verificar, em tempo de compilação, que os tipos foram atribuídos corretamente (MICROSOFT, 2024c). O resultado da compilação do código TypeScript é um código JavaScript compatível com os padrões e que pode ser executado mesmo que o compilador apresente erros relacionados à atribuição de tipos. Ou seja, o TypeScript pode ser resumido como um verificador de tipos em tempo de compilação (NPM, 2024; MICROSOFT, 2024b).

2.2.7 Node.js

Para executar código JavaScript os navegadores precisam do que chamam de motor JavaScript. O Google Chrome utiliza o motor JavaScript chamado V8. Esse motor foi o escolhido para a criação do Node.js, que foi lançado em 2009 fornecendo um ambiente de execução de código JavaScript fora do navegadoe (Node.js, 2024).

Atualmente existem vários pacotes relacionados ao Node.js. E para gerenciar esses pacotes, existe o Node Package Manager (npm), que inicialmente servia apenas como forma de gerenciar dependências de pacotes Node.js, mas que passou também a ser usado em projetos *frontend* (Node.js, 2024).

No contexto do sistema que será desenvolvido ao longo desse trabalho, o Node.js será usado para gerenciar as dependências do projeto Angular através do npm.

2.2.8 PostgreSQL

O PostgreSQL será utilizado como sistema gerenciador do banco de dados da aplicação. É um sistema de banco de dados objeto-relacional de código aberto que está em desenvolvimento ativo há mais de 30 anos, sendo hoje bem reconhecido no meio tecnológico (PostgreSQL, 2024). Sua implementação foi iniciada em 1986 como um projeto acadêmico e desde então passou por várias versões, recebendo melhorias em cada uma delas conforme era testado e utilizado em aplicações de pesquisa e em ambientes produtivos. Hoje oferece muitos recursos para apoiar desenvolvedores na construção de aplicativos que demandam armazenamento e gerenciamento de dados de forma segura e íntegra (The PostgreSQL Global Development Group, 2024; PostgreSQL, 2024).

2.3 Teste de Software

O teste de software é parte crucial do processo de desenvolvimento de um sistema (GRATER, 2005) e tem como objetivo a detecção de erros ou falhas a fim de validar se a aplicação se comporta de acordo com os requisitos definidos no planejamento do sistema (JAMIL et al., 2016). Essa é a última etapa antes da entrega do produto final, sendo o último momento para a identificação de problemas que podem comprometer a qualidade do produto (NETO et al., 2006).

Os testes de software devem ser previamente planejados, para que se defina os tipos de teste que serão aplicados e assim sejam realizados testes que fazem sentido para o contexto do sistema e que sejam efetivos na identificação de problemas (SILVA, 2022). Existem alguns tipos de teste, como testes de unidade, de integração, de regressão e outros. E esses tipos se enquadram em diferentes níveis, dos testes de unidade, que são realizados no nível do código fonte da aplicação, em que são testadas as menores partes de um projeto de forma individual, até testes de mais alto nível que são realizados por usuários que validam as funcionalidades por meio da interação com a interface gráfica do sistema.

3 Desenvolvimento

3.1 Arquitetura

Para a implementação deste projeto foi construído um sistema web com base na arquitetura conhecida como cliente-servidor. De forma breve, esse modelo consiste em ter um processo cliente que realiza requisições para um processo servidor que é responsável por receber, processar e enviar uma resposta para o cliente (nessa sequência). Especificamente, foi utilizada a arquitetura cliente-servidor em 3 camadas, sendo essas camadas divididas em: camada de apresentação, camada de aplicação ou de negócios e camada de banco de dados (KUMAR, 2019).

A camada de apresentação é implementada do lado do cliente, que, em um sistema web, é um navegador, como o Google Chrome, por exemplo. O navegador roda a interface gráfica para o usuário interagir com o sistema ao mesmo tempo que pode realizar requisições ao servidor, através do protocolo HTTP. A partir dessas interações do usuário, a depender da funcionalidade utilizada, o navegador deve enviar requisições ao servidor para atender às solicitações do usuário, como realizar cadastros, consultas, edições de dados, dentre outras funcionalidades. Portanto, cada usuário terá uma cópia do projeto rodando no navegador instalado em seu dispositivo (OSÓRIO, 2023; KUMAR, 2019).

Para atender as requisições dos clientes, é necessário pelo menos um servidor web rodando a aplicação *backend*, que compõe a camada de aplicação. Essa aplicação contém as regras de negócio para o sistema funcionar conforme planejado previamente e comunica-se com o banco de dados para efetuar as operações de leitura, inserção, atualização e remoção de dados. O banco de dados, que na arquitetura cliente-servidor em 3 camadas representa a camada de banco de dados, pode estar rodando na mesma máquina do *backend* ou em um outro servidor dedicado (KUMAR, 2019).

Para este projeto, a implementação da camada de apresentação é um projeto construído com o Angular, a camada de negócios é implementada em um projeto Java utilizando o *framework* Spring, que serve como uma API Rest e comunica-se com um banco de dados PostgreSQL, sendo a camada de banco de dados. Essa arquitetura pode ser simplificada em duas partes: *frontend* e *backend*. O *frontend* é a camada de apresentação e o *backend* encapsula as camadas de negócios e de banco de dados. A Figura 7 representa esse modelo e a comunicação entre as partes que compõem o modelo.

Para detalhar melhor essa comunicação, a Figura 8 mostra o diagrama de sequência que representa o fluxo da funcionalidade de cadastro de um patrimônio histórico. Esse fluxo de comunicação serve como exemplo para praticamente todas as funcionalidades.

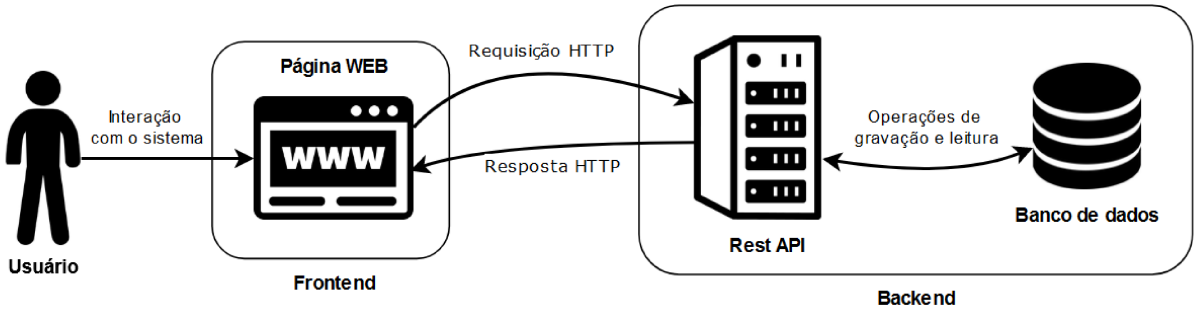

Figura 7 – Arquitetura. Comunicação entre as camadas do sistema.

Nele podemos entender também a organização dos projetos *frontend* e *backend*. O *frontend* é organizado em componentes, seguindo o padrão de projetos construídos sobre o Angular, onde cada componente possui seu arquivo HTML, um código TypeScript com a lógica de apresentação (acionada a partir de eventos e interações do usuário com o sistema) e um *service*, também escrito em TypeScript, responsável por realizar as chamadas HTTP à API. Enquanto o *backend* está organizado em classes *controller*, para processar as requisições; classes service, que contém as regras de negócio da aplicação; e classes *repository*, que se comunicam com o banco de dados e são utilizadas pelas classes *service*.

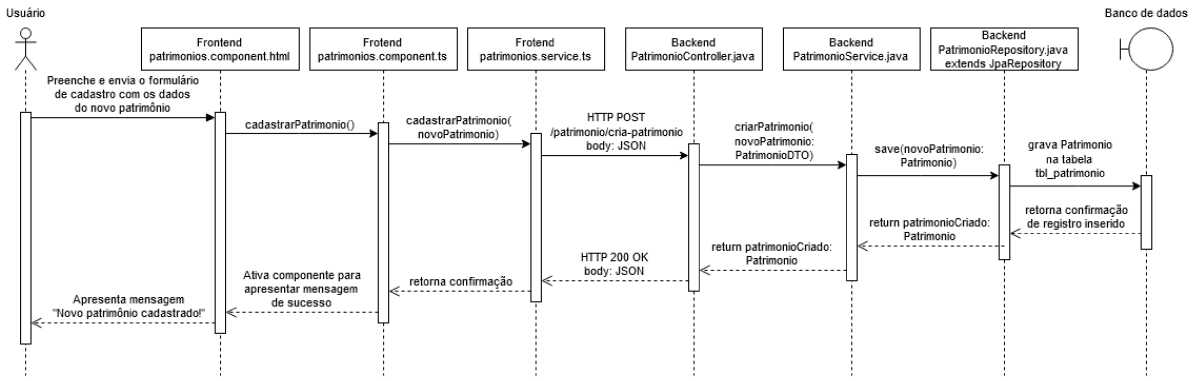

Figura 8 – Arquitetura. Diagrama de sequência do fluxo de cadastro de um patrimônio.

3.2 Banco de dados

Seguindo os princípios da engenharia de software, desde a concepção até a entrega, a construção de um sistema passa por várias etapas, com a modelagem de dados sendo uma parte fundamental. É nesse estágio que os conceitos e objetos do mundo real, relacionados ao domínio do negócio, serão estruturados e organizados de forma que posteriormente possam ser representados em um banco de dados (PRESSMAN; MAXIM, 2021).

Para o sistema desenvolvido neste trabalho, o banco de dados foi implementado com as seguintes tabelas: tbl_patrimonio, tbl_categoria, tbl_patrimonio_arquivos, tbl_tipo_arquivo, tbl_patrimonio_comentarios, tbl_usuario e tbl_tipo_usuario. Para visualizar essas entidades e suas relações, na Figura 9 é apresentado o Diagrama relacional do banco de dados do sistema.

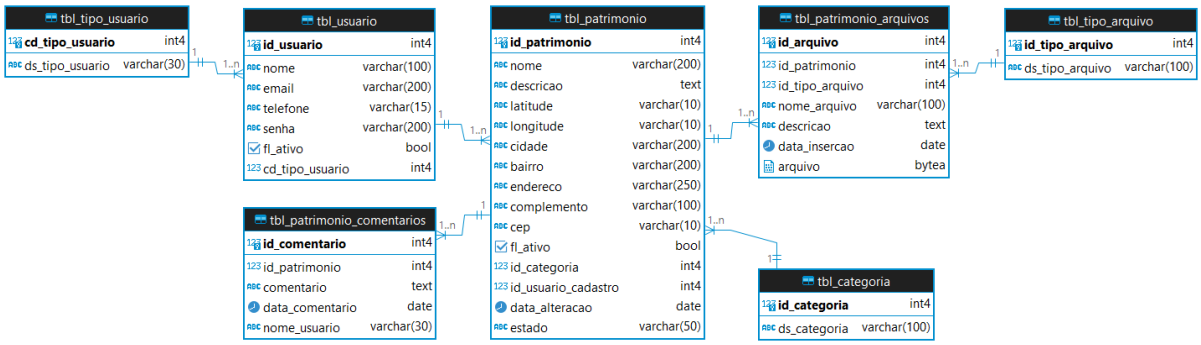

Figura 9 – Diagrama relacional do banco de dados do sistema.

Para utilizar o sistema é necessário que o usuário esteja cadastrado na `tbl_usuario`. Os campos e-mail e senha serão utilizados para realizar o login no sistema e então o usuário poderá utilizar as funcionalidades permitidas para ele, o que vai depender do tipo do usuário definido no momento do cadastro. O tipo do usuário referencia os registros armazenados na tabela `tbl_tipo_usuario`, que é uma tabela de controle para que determinadas funcionalidades fiquem disponíveis de acordo com cada perfil. Por exemplo, um usuário do tipo administrador tem permissão para acessar todas as telas e utilizar todas as funcionalidades do sistema, enquanto um usuário convencional estará limitado a algumas delas. Portanto, um usuário pode ser somente de um tipo, enquanto um determinado tipo pode estar relacionado a vários usuários cadastrados.

Uma funcionalidade permitida a todos os usuários é a de cadastrar um patrimônio. A tabela `tbl_patrimonio` irá armazenar os dados do patrimônio, como o nome, a localização, a categoria, a descrição do patrimônio - sendo este muito importante para que detalhes interessantes, como a história e o contexto da construção daquele imóvel, arte ou objeto, sejam fornecidos no momento do cadastro -, dentre outros campos para histórico e controle do sistema. Um desses campos de controle é o `id_usuario_cadastro`, que grava o id do usuário que realizou o cadastro do patrimônio, estabelecendo a relação entre as tabelas `tbl_patrimonio` e a tabela `tbl_usuario`, onde um patrimônio estará relacionado a somente um usuário, que pode ser quem realizou o cadastro ou editou por último as informações do patrimônio, mas um usuário pode estar relacionado a vários patrimônios aos quais ele realizou o cadastro ou editou as informações pela última vez.

Outra relação da tabela `tbl_patrimonio` é com a tabela `tbl_categoria`, através do campo `id_categoria`. Ao cadastrar um patrimônio deve ser escolhida uma categoria.

As categorias disponíveis para escolha ficam registradas na tabela `tbl_categoria`, permitindo que diferentes patrimônios sejam da mesma categoria, o que possibilita que quem estiver visitando o site possa fazer consultas e filtrar por patrimônios de uma categoria específica. Logo, um patrimônio deve se enquadrar em alguma categoria e uma categoria pode estar relacionada a vários patrimônios. Exemplos de categorias: bens imóveis, bens móveis, sítios naturais, dentre outras.

Além dessa relação, a tabela tbl_patrimonio se relaciona com as tabelas tbl_patrimonio_comentarios e tbl_patrimonio_arquivos. Como o nome sugere, a tabela tbl_patrimonio_comentarios guarda os comentários dos visitantes do site sobre os patrimônios, gravando o nome da pessoa, a data do comentário, o comentário e o id do patrimonio para o qual foi feito o comentário. Assim, um patrimônio pode receber inúmeros comentários, e cada um desses comentários estará vinculado a um único patrimônio.

Já a tabela `tbl_patrimonio_arquivos` grava os arquivos anexados para os patrimônios, como fotos ou algum outro tipo de documento relacionado ao patrimônio. São gravados o nome do arquivo, a data em que o documento foi anexado (já que pode ser armazenado no momento do patrimônio ou posteriormente), o arquivo codificado e o id do patrimônio, que referencia o id da tabela `tbl_patrimonio`. Sendo assim, podem ser anexados vários arquivos para um patrimônio, mas cada registro de arquivo na base é relacionado a somente um patrimônio. Além desses campos, há o campo que define o tipo do arquivo (`id_tipo_arquivo`), ou seja, se é um PDF, uma imagem, um arquivo de apresentação, dentre outros. Esse campo é uma referência à tabela `tbl_tipo_arquivo`, que já possui os tipos de arquivos esperados pelo sistema. Dessa forma, um arquivo é de um tipo específico e um tipo pode ser usado para arquivos diferentes que se enquadram no determinado tipo.

3.3 Testes

A fim de exemplificar de forma prática a implementação e execução de testes de software, foram implementados testes de integração para validar o funcionamento de alguns *endpoints* da API. Dessa forma é possível verificar o funcionamento de um fluxo completo, validando se, diante de alguns cenários, a API está respondendo com o código HTTP correto, se está retornando respostas que condizem com o que está sendo enviado no corpo da requisição, dentre outras validações.

Para implementar os testes de integração, foram utilizados alguns recursos oferecidos pelo JUnit (dependência embutida no módulo de testes do Spring) juntamente com algumas configurações do Spring que facilitam a implementação desse tipo de teste. O JUnit é um *framework* de testes de unidade para a linguagem de programação Java, sendo bastante utilizado para escrever e executar testes automatizados.

Com base no banco de dados real, foi criado um banco de dados para a execução dos testes. No momento da criação, o banco de dados real já possuía alguns dados (previamente cadastrados) necessários para o uso inicial do sistema, como tipos de usuários, tipos de arquivos, algumas categorias e um usuário administrador.

4 Resultados

4.1 Funcionalidades do Sistema

Nesta seção, será apresentado o funcionamento do sistema, utilizando capturas de tela da interface do sistema em uso para evidenciar e auxiliar na descrição de cada uma das funcionalidades disponíveis.

4.1.1 Parte Pública

A parte pública se refere às páginas disponíveis para qualquer pessoa que acessar o site, que serão referidas como "visitante". Esse tipo de usuário poderá, basicamente, visualizar os dados dos patrimônios históricos previamente catalogados, além de deixar comentários sobre o que ele observou.

Ao acessar o sistema, um visitante irá visualizar uma lista de patrimônios cadastrados (Figura 10). Para cada patrimônio, o sistema apresenta as imagens em tamanho pequeno em um carrossel, seguido da descrição ao lado, o estado e cidade onde se localiza o patrimônio e os comentários dos visitantes. Ao clicar sobre o título do patrimônio, o visitante é levado a uma página dedicada, na qual são apresentados mais dados sobre a localização daquele bem, podendo também o usuário interagir com a página enviando os próprios comentários (Figuras 11, 12 e 13) e realizando o *download* das imagens (ou documentos anexados) ao clicar sobre o nome do arquivo (Figura 14).

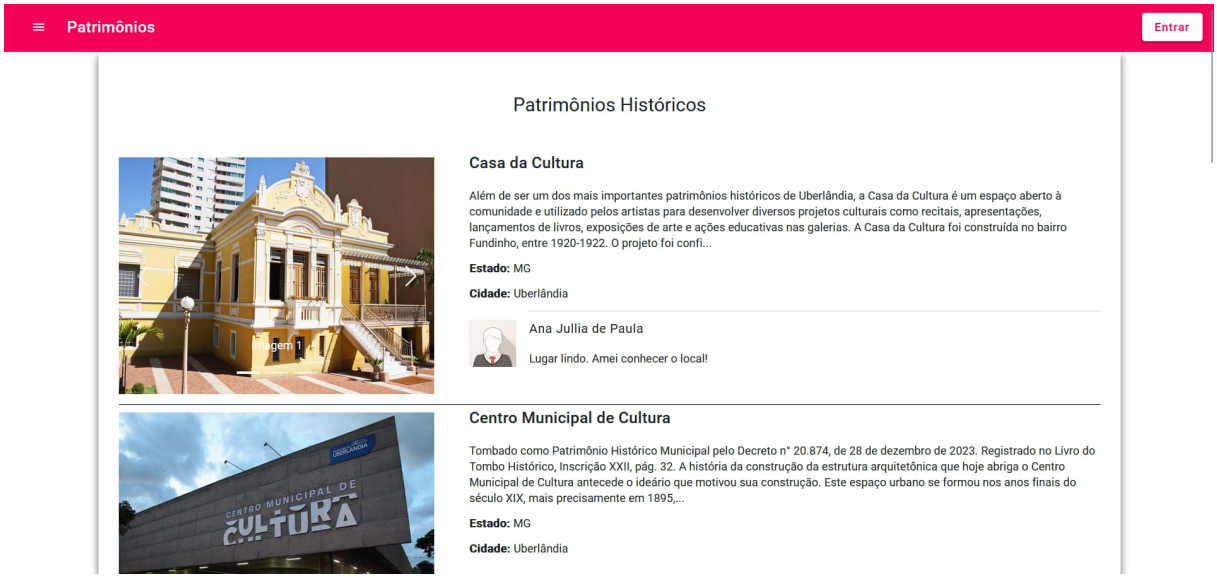

Figura 10 – Página inicial do sistema.

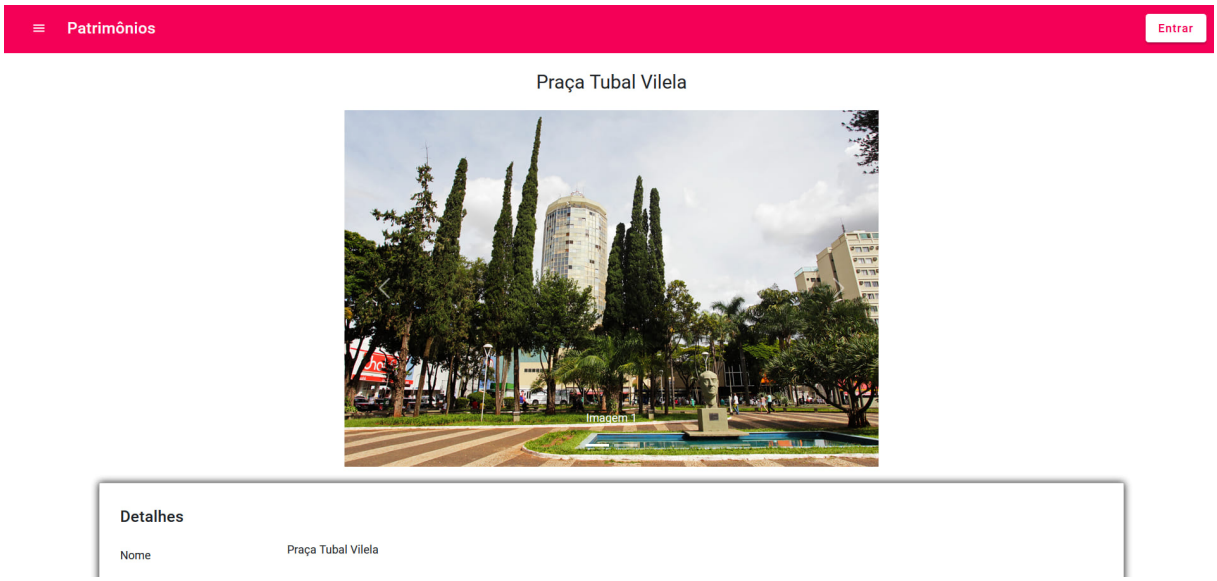

Figura 11 – Interação com o sistema. Visão de um patrimônio.

Figura 12 – Interação com o sistema. Detalhamento das informações

Figura 13 – Interação com o sistema. Comentários.

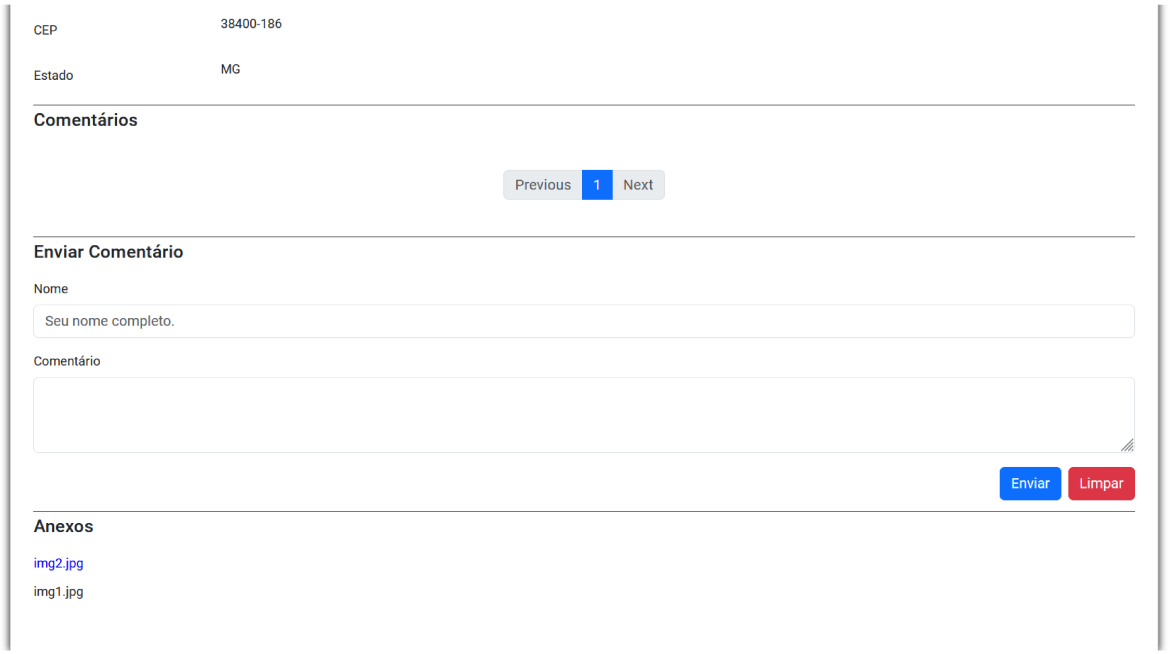

Figura 14 – Interação com o sistema. *Download* de arquivos.

Ao acessar o menu, no canto superior esquerdo da interface, clicando na opção "Categorias" o visitante pode buscar por patrimônios cadastrados em uma categoria específica (Figura 15). Ao clicar no item referente à categoria desejada, o sistema busca pelos patrimônios registrados naquela determinada categoria, apresentando uma lista simplificada, somente com o título dos patrimônios (Figura 16). Nessa tela, é possível realizar um segundo filtro, buscando por um nome ou ocorrência de letras que se enquadram no nome de algum patrimônio (Figura 17). Se clicar sobre o nome de algum dos itens listados, é carregada a mesma tela já citada anteriormente (Figura 11), que apresenta mais detalhes sobre o bem patrimonial.

Figura 15 – Busca de patrimônios por categoria.

Figura 16 – Listagem de nomes de patrimônios ao filtrar por uma categoria.

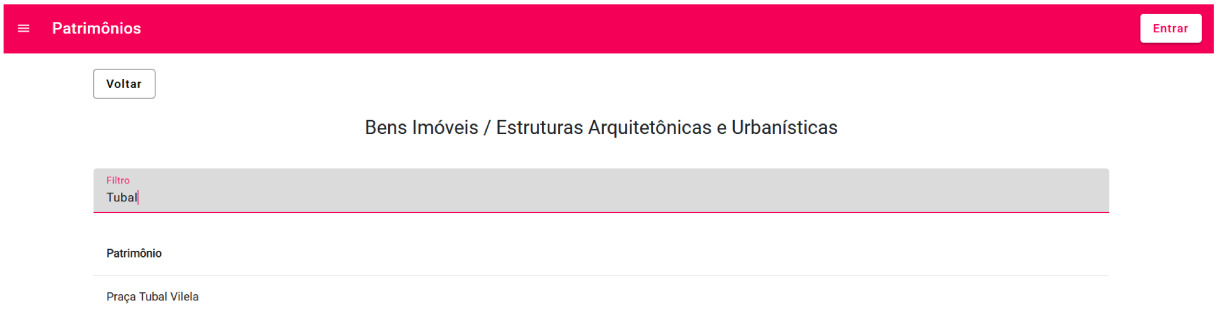

Figura 17 – Filtro de busca.

4.1.2 Gerenciamento de Usuários

Para acessar a área privada do sistema, que permite realizar o cadastro e edição dos dados dos patrimônios, é necessário estar logado como "administrador" ou como um "usuário padrão" (Figura 18). Além desses recursos principais, de gerenciamento de patrimônios, um usuário com acesso de administrador consegue visualizar a opção "Usuários" no menu e, a partir dessa opção, acessar as funcionalidades relacionadas ao gerenciamento de usuários. Vale ressaltar que tais funcionalidades só estão disponíveis para usuários do tipo "administrador" .

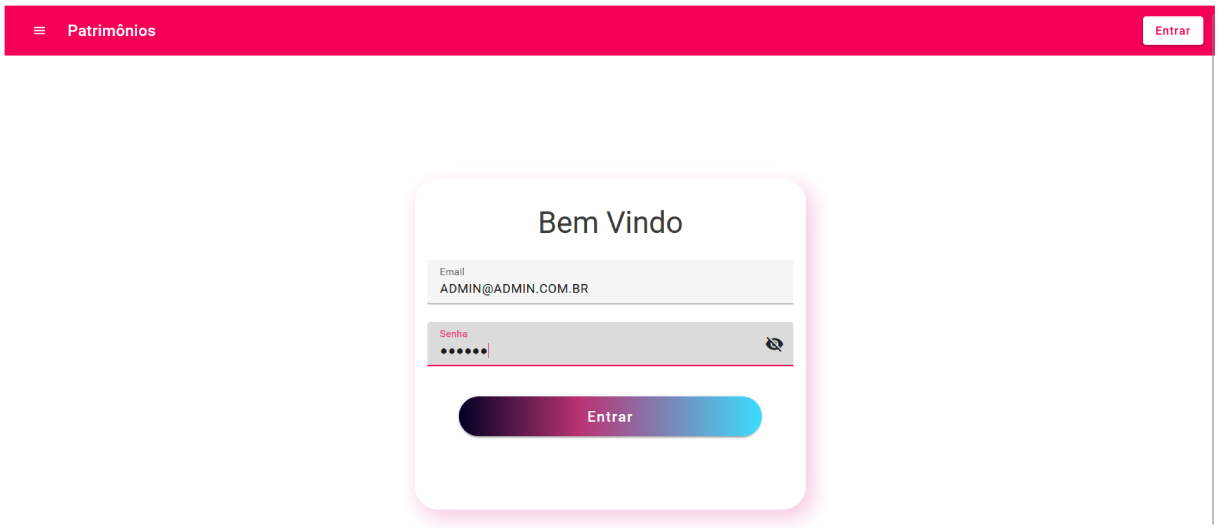

Figura 18 – Tela para login.

Um administrador pode cadastrar um novo usuário, preenchendo os campos requeridos e selecionando o tipo do usuário: "Usuário" ou "Administrador" (Figuras 19 e 20).

Na aba "Todos Usuários" o sistema carrega a lista com os dados de cada usuário em uma tabela que leva dois botões nas últimas colunas, sendo possível editar os dados do usuário ou redefinir a senha, respectivamente. Clicando no botão de edição, a aplicação abre um modal (janela suspensa), permitindo a edição do nome, telefone e o tipo do usuário (Figuras 21 e 22). O e-mail não pode ser alterado pois é a chave única que identifica cada usuário. Clicando no outro botão, o sistema gera uma nova senha e a apresenta a senha gerada (Figura 23). Também é possível ativar ou inativar um usuário por meio de um botão dedicado para essa funcionalidade (Figura 23).

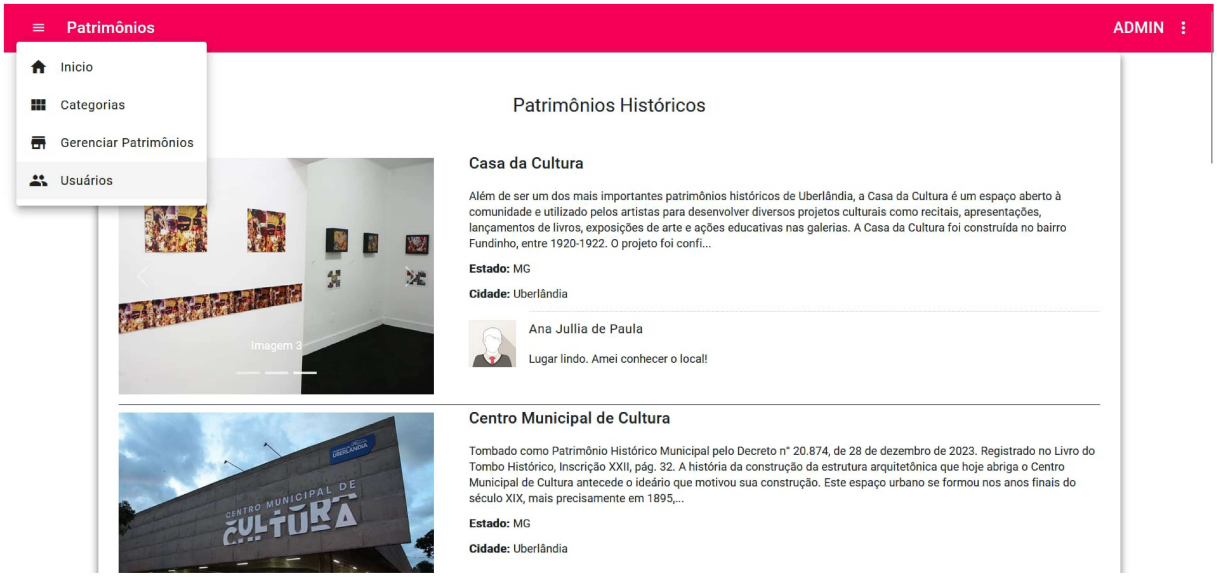

Figura 19 – Menu para novos usuários.

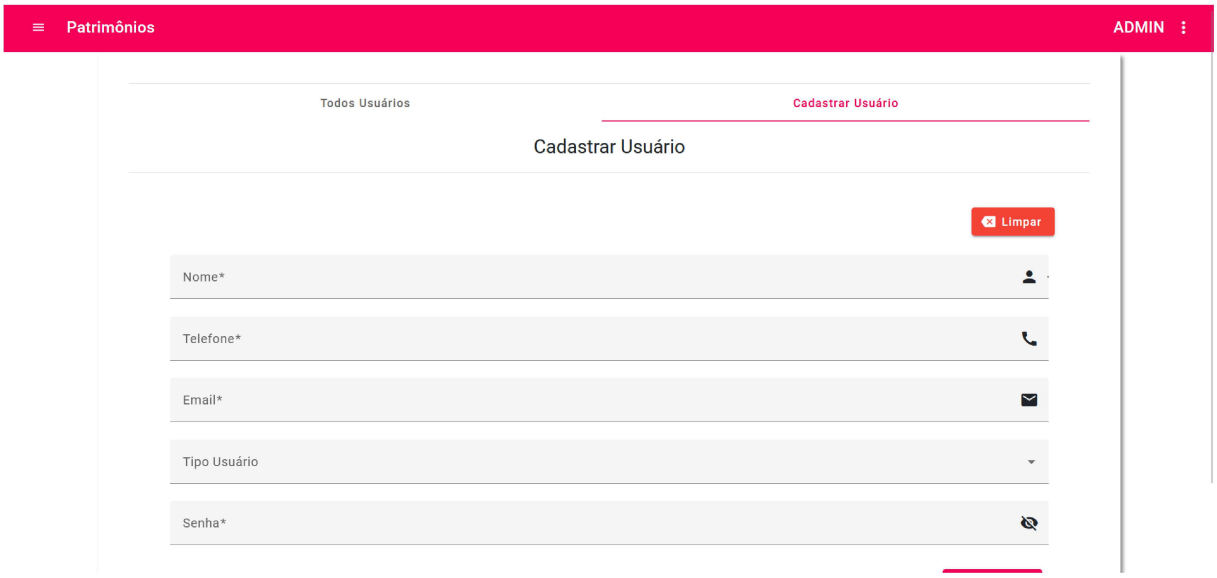

Figura 20 – Cadastro de novos usuários.

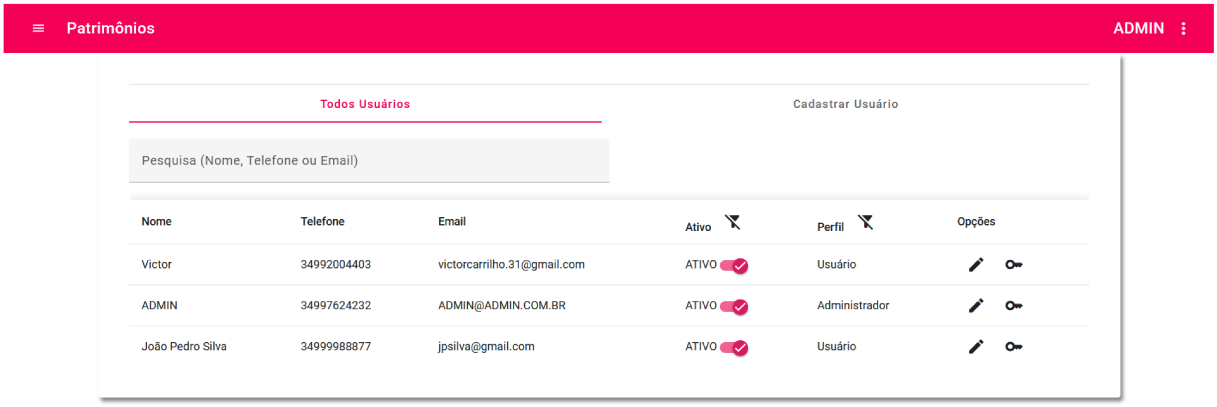

Figura 21 – Lista de usuários.

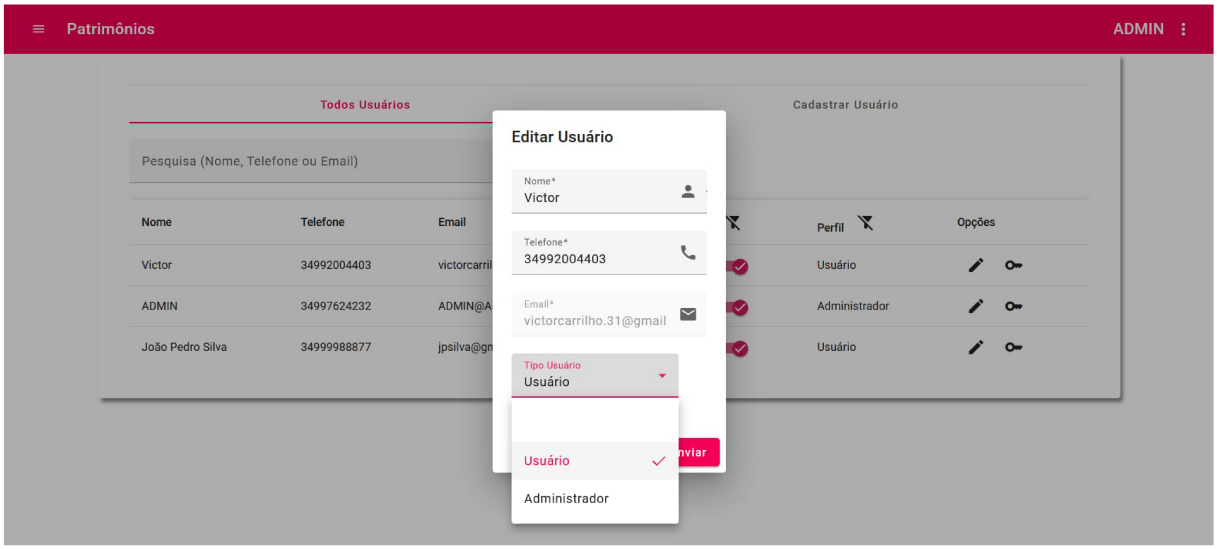

Figura 22 – Modificação dos dados do usuário.

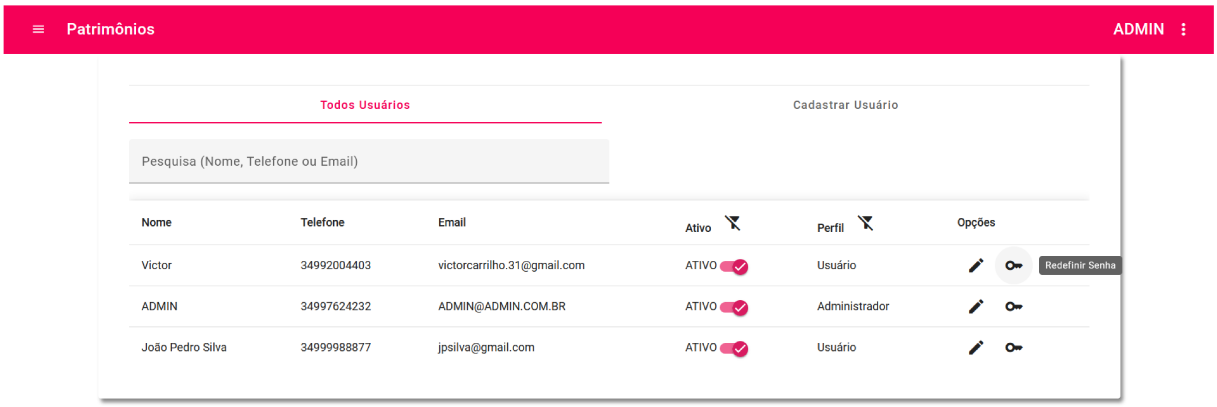

Figura 23 – Modificação de senha e alteração de status (Ativo / Inativo).

Ainda na aba “Todos Usuários”, o administrador que está verificando os usuários pode pesquisar por registros que possuam a ocorrência pesquisada nos campos nome, telefone ou e-mail, bem como utilizar os filtros específicos dos campos “Ativo” (Figura 24) e “Perfil” (Figura 26), que permitem o filtro pelos valores possíveis para esses campos.

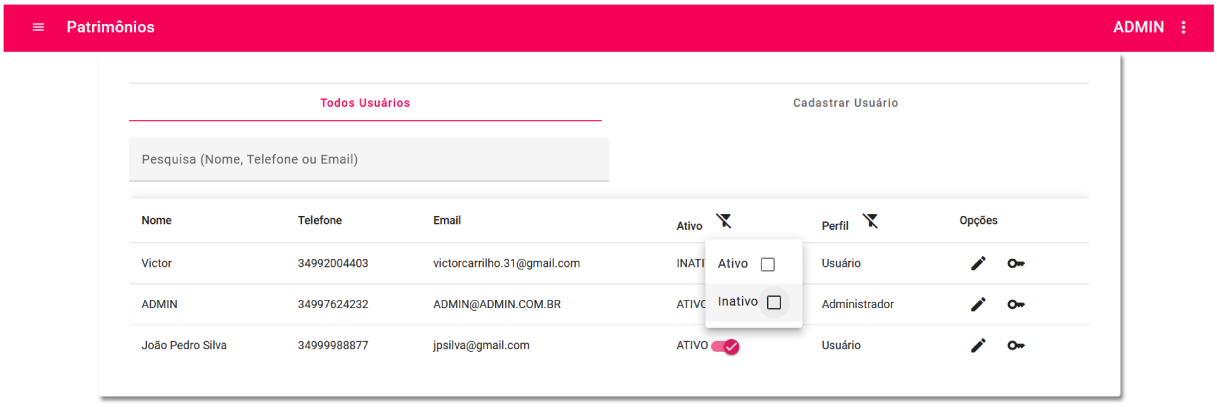

Figura 24 – Filtro para listar usuários (Ativo / Inativo).

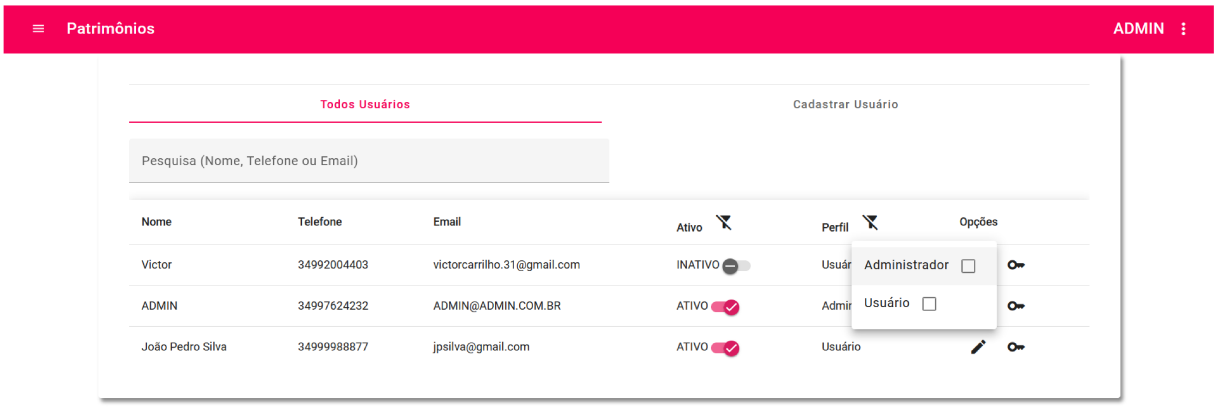

Figura 25 – Filtro para listar por perfil (Administrador / Usuário).

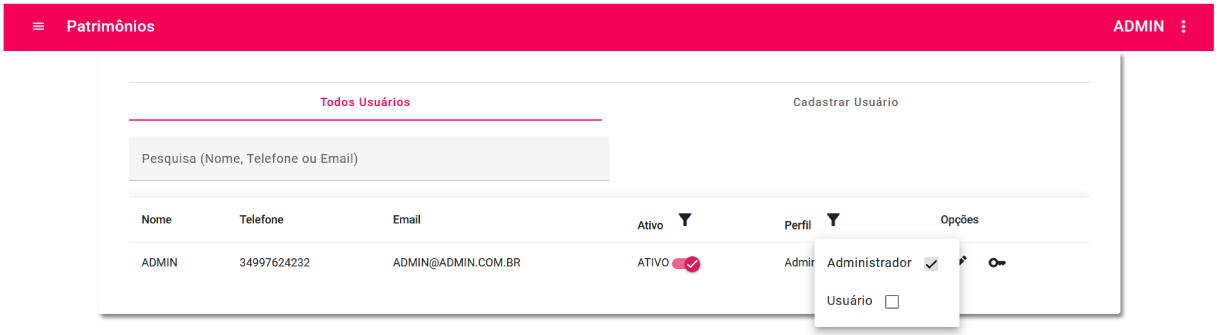

Figura 26 – Filtro para listar por perfil (Administrador / Usuário).

Após entrar no sistema, no canto superior direito, clicando no ícone de três pontos na vertical, é possível realizar o *logout*, saindo da área privada do sistema, ou acessar o dados do perfil do usuário (Figura 27). Na tela de perfil, pressionando o botão "Editar", os campos de nome e telefone são habilitados para que o usuário edite e envie as alterações (Figura 28). O usuário também tem a opção de alterar sua senha, na aba "Redefinir senha", sendo necessário digitar duas vezes a nova senha, a fim de confirmar a alteração.

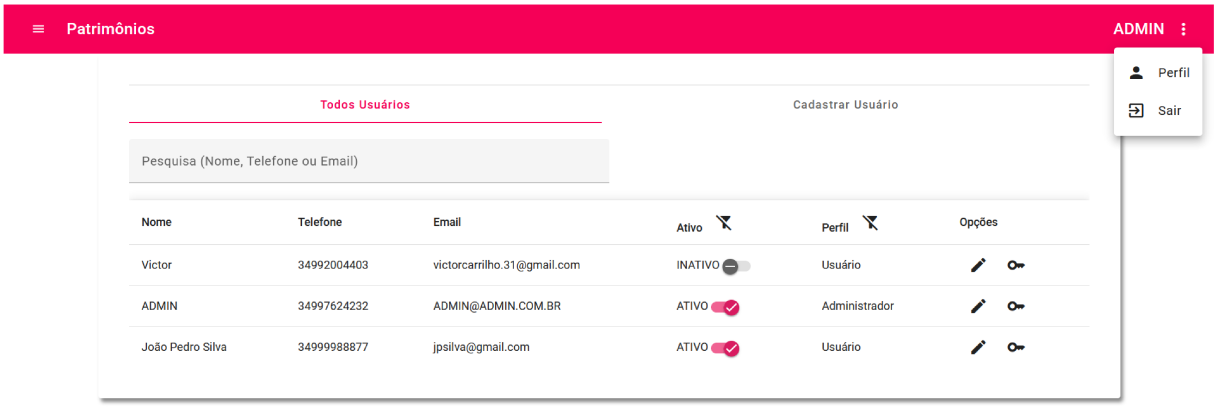

Figura 27 – Menu para acessar perfil do usuário.

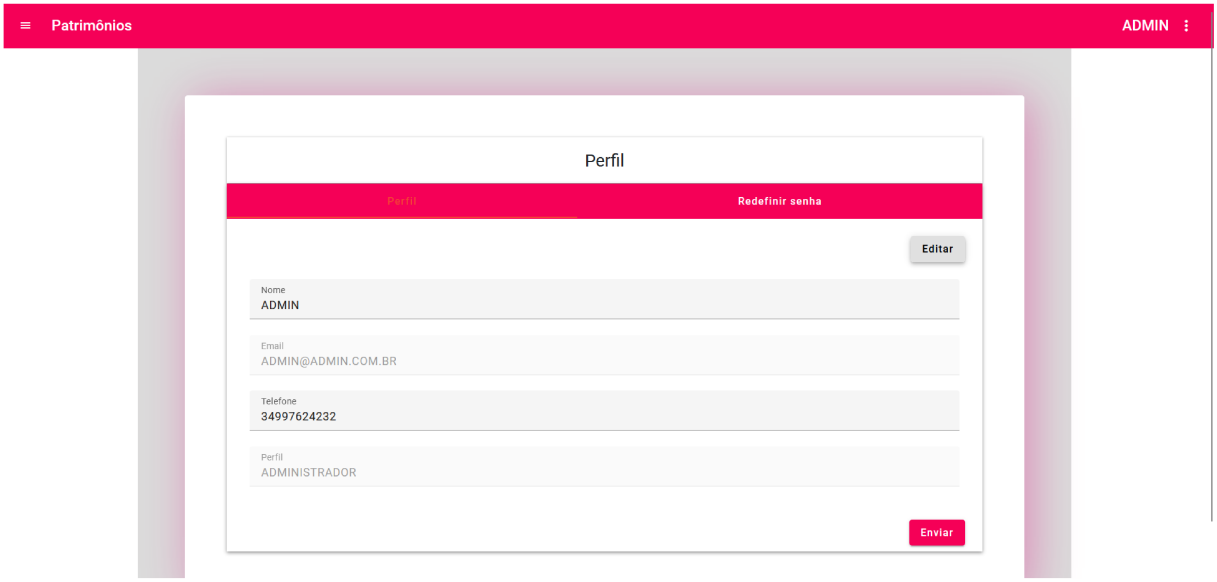

Figura 28 – Edição dos dados na tela de perfil do usuário.

4.1.3 Gerenciamento de Patrimônios

Tanto administradores quanto usuários "padrão" conseguem gerenciar os patrimônios, com os recursos acessíveis a partir da opção "Gerenciar Patrimônios" no menu (Fi-

gura 29). Automaticamente, o sistema carrega a lista completa de patrimônios carregados, já sendo possível editar os dados a partir dessa tela, clicando no ícone de edição, na coluna "Editar", e até mesmo é possível inativar um registro (Figura 30). Mais adiante, a parte de edição será abordada novamente, mas antes será detalhada a parte de cadastro de um patrimônio.

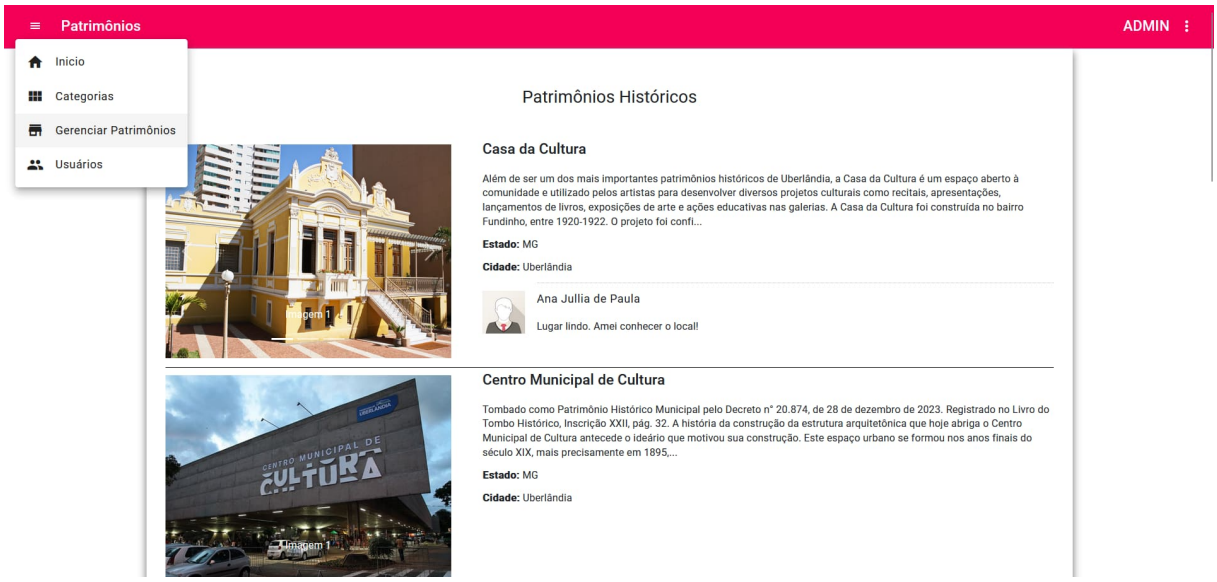

Figura 29 – Listagem de Patrimônios.

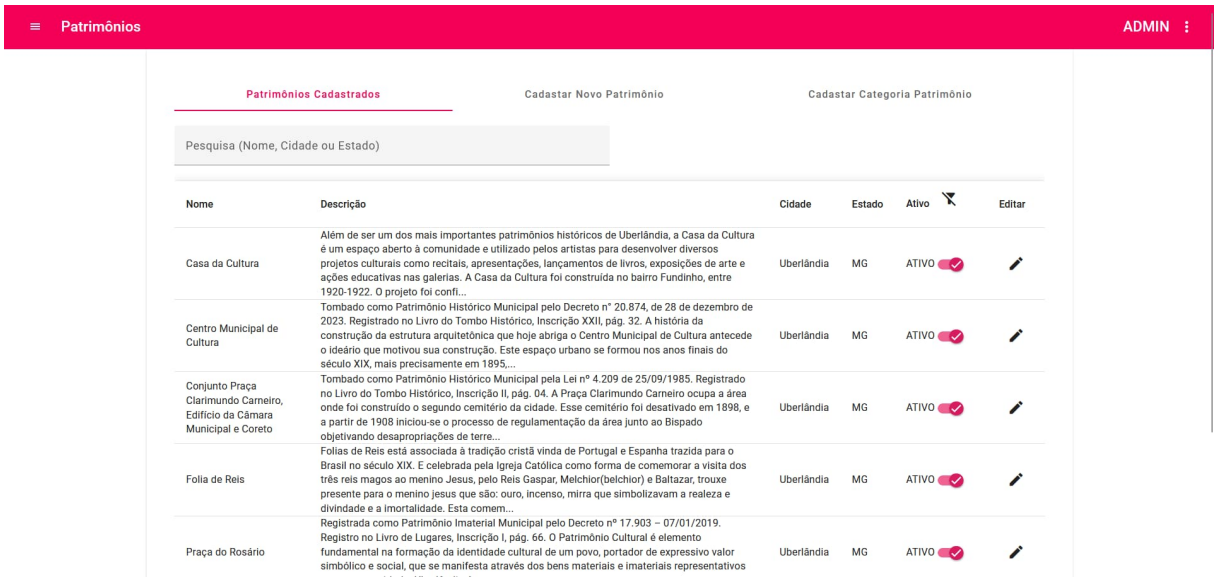

Figura 30 – Listagem de Patrimônios.

Na aba "Cadastrar Novo Patrimônio", são exibidos os campos para preenchimento, em que aqueles obrigatórios são destacados com um "*" (Figuras 31 e 32). Na parte inferior, na seção "Arquivos", ao clicar no botão "Procurar", o explorador de arquivos

do sistema operacional é aberto para que sejam selecionados os arquivos para *upload* no sistema (Figuras 33 e 34). Preenchidos os campos, basta clicar em "Cadastrar" para armazenar os dados do patrimônio juntamente com os arquivos relacionados. Ao final, a aplicação apresenta uma mensagem de sucesso ou de erro para a tentativa de cadastro. Em caso de sucesso, os campos são limpos automaticamente.

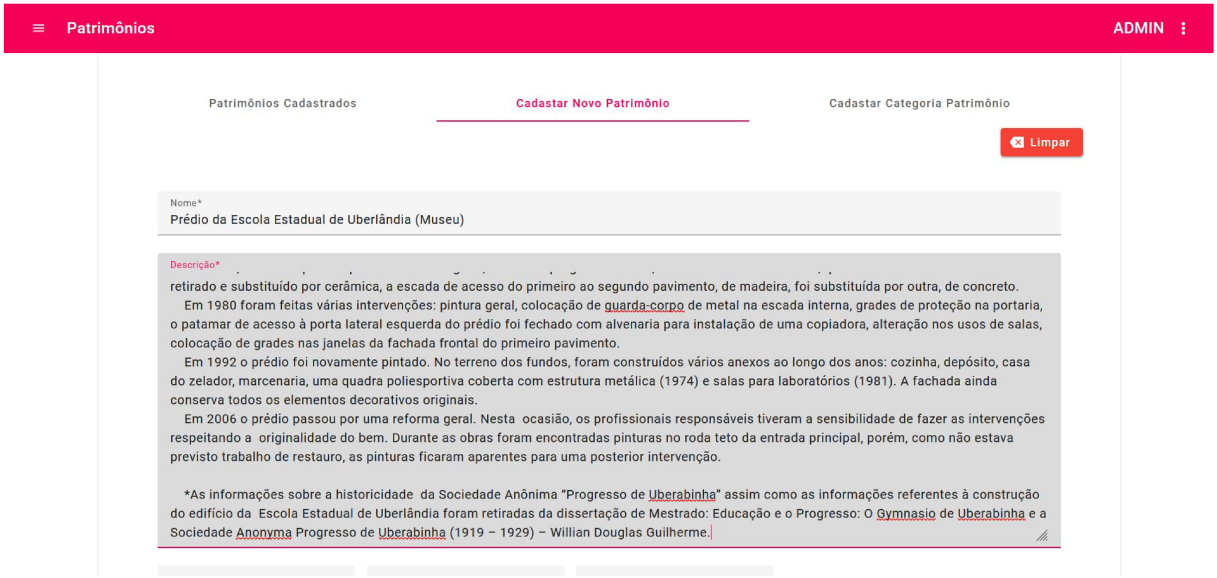

Figura 31 – Cadastro de patrimônio.

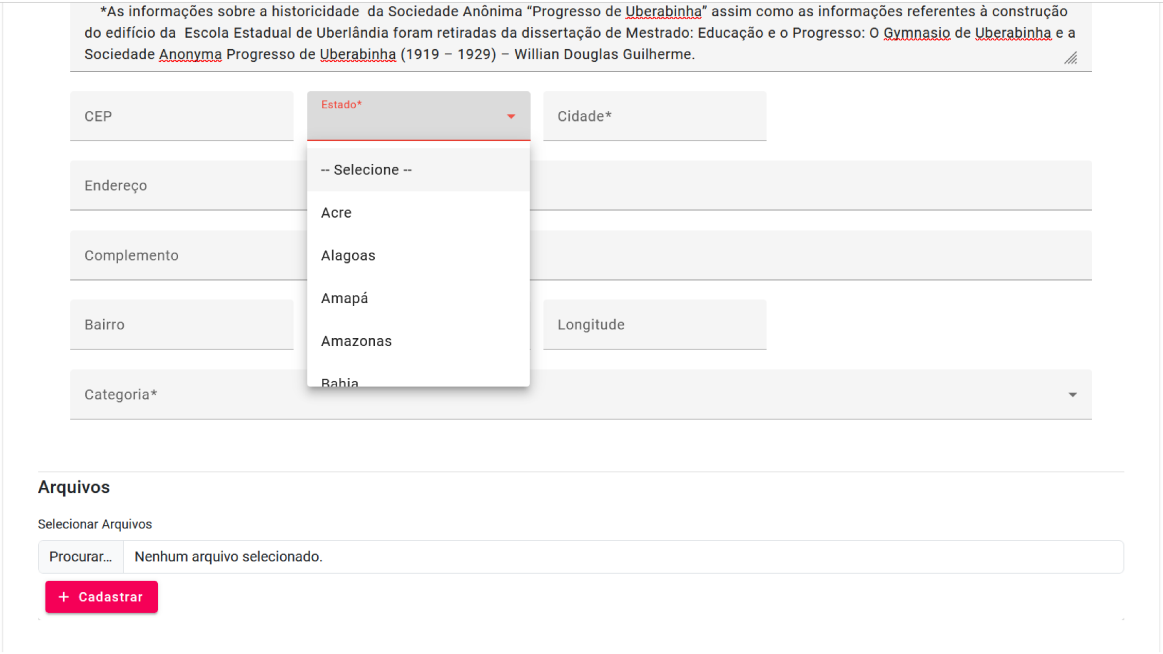

Figura 32 – Cadastro de patrimônio.

Na guia "Cadastrar Categoria Patrimônio", são exibidas as categorias existentes, que ficam disponíveis para seleção no formulário de cadastro de um novo patrimônio,

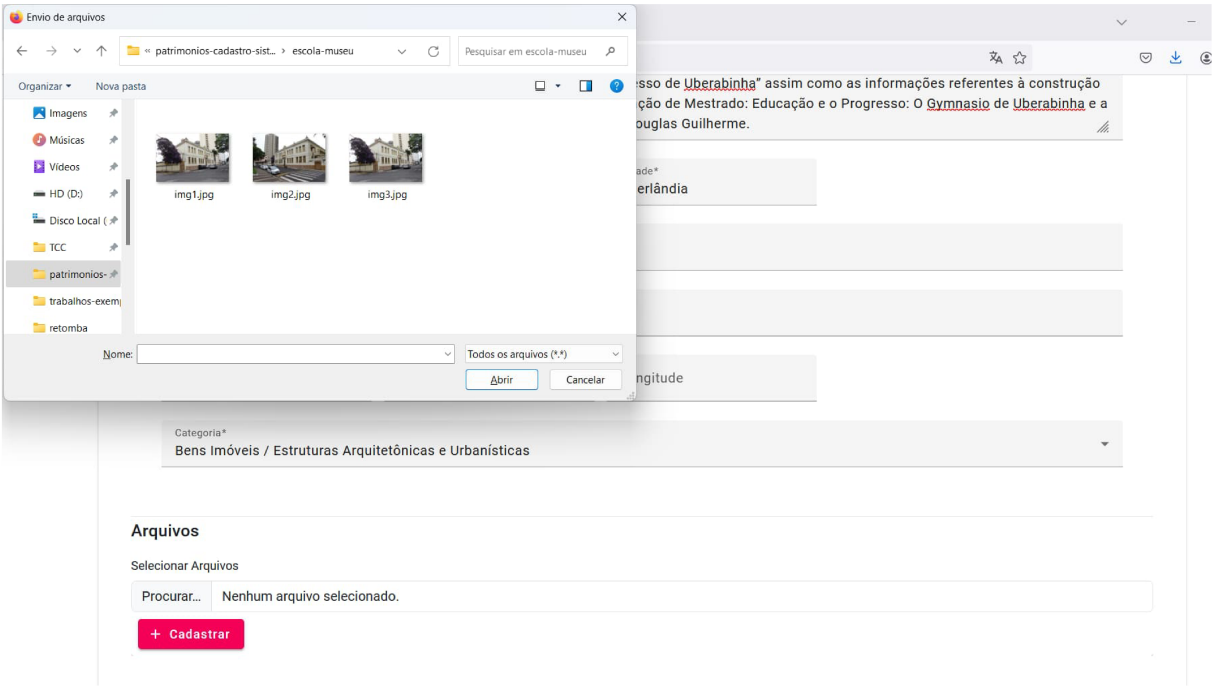

Figura 33 – *Upload* de imagens ao sistema.

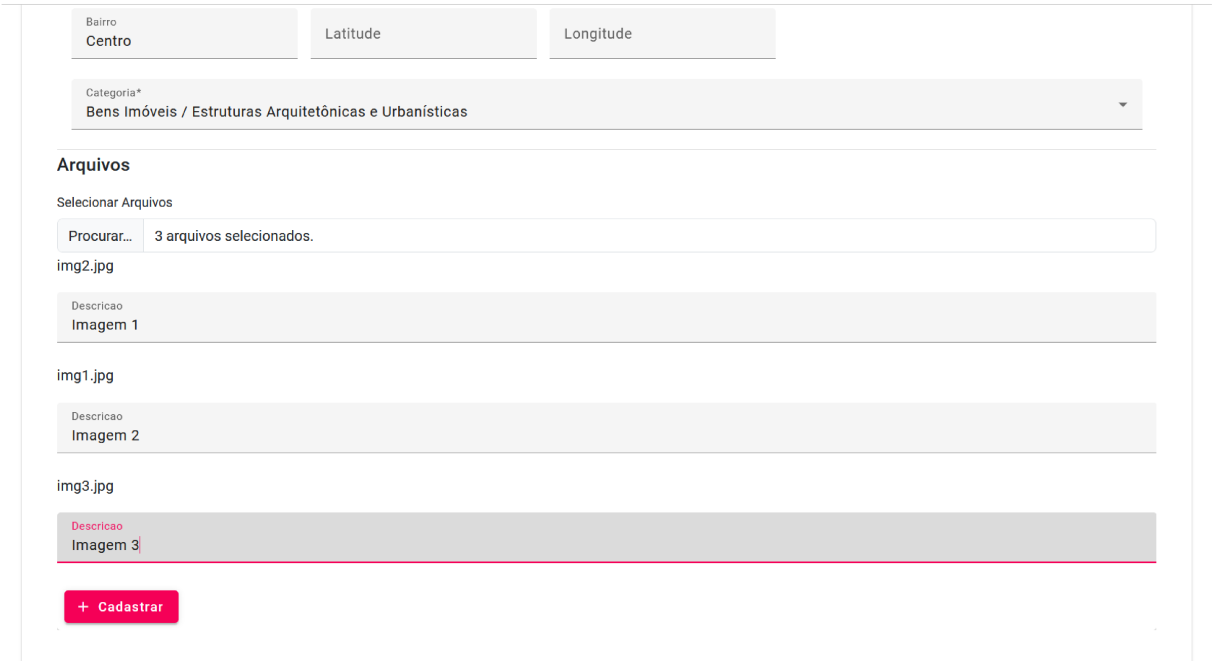

Figura 34 – *Upload* de imagens ao sistema.

sendo possível cadastrar uma nova categoria ao preencher o campo “Nova Categoria” e enviar com o botão “Cadastrar” (Figura 35). A nova categoria cadastrada passará a estar disponível para filtro na tela acessível na parte pública a partir do menu “Categorias”.

Voltando na primeira seção, na guia “Patrimônios Cadastrados”, pressionando

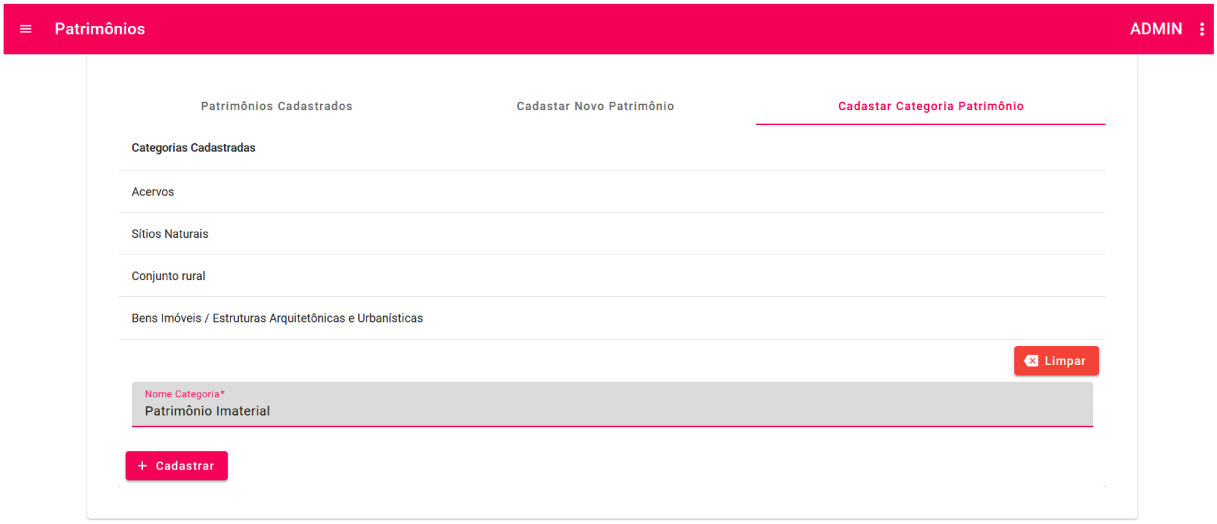

Figura 35 – Cadastro de novas categorias de patrimônios.

o botão “Editar”, é aberto um modal para edição de todos os campos disponíveis no formulário de cadastro, além de ser possível excluir os arquivos vinculados ao patrimônio, como também incluir novos arquivos (Figura 36).

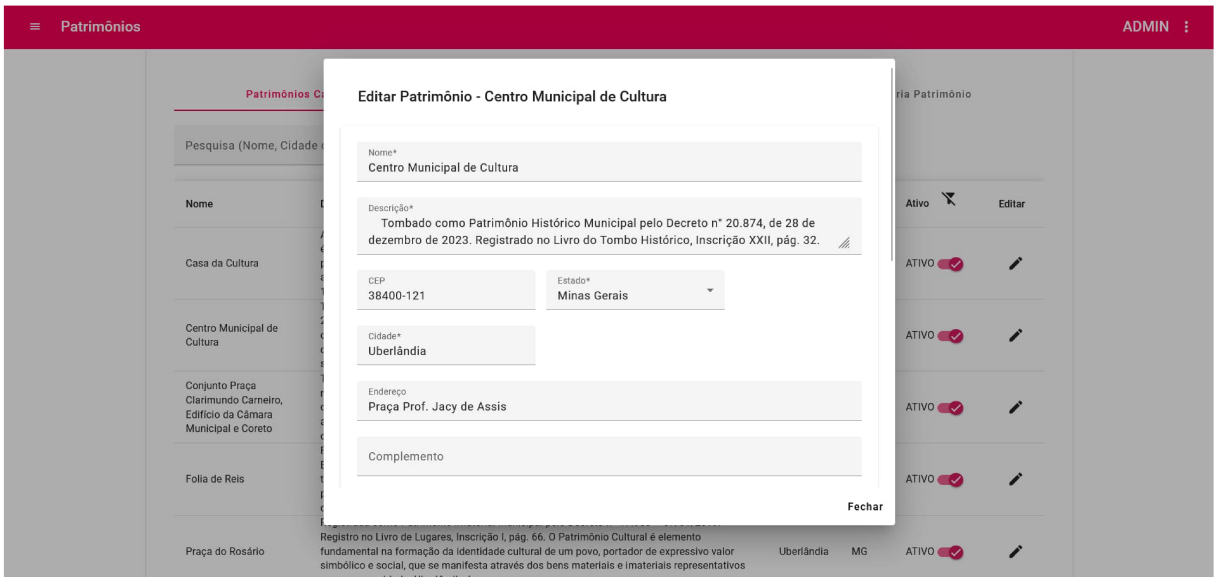

Figura 36 – Edição de patrimônio cadastrado.

4.2 Testes

A Figura 37 mostra a definição da classe de teste com sua única dependência (`TesteUtil`). A primeira anotação (`@SpringBootTest`) é uma anotação do Spring essencial para a realização dos testes de integração, pois com essa configuração é iniciado um servidor web embutido em uma porta aleatória, permitindo que sejam feitas chamadas HTTP reais à API. Com isso, é necessária a segunda anotação (`@ActiveProfiles`), que serve para indicar que o perfil de teste ("test") deve ser utilizado para a execução do código, levando em conta as configurações especificadas no arquivo `application-test.properties`, no qual se encontra a configuração para uso do banco de dados de teste. Já a terceira anotação orienta o JUnit a criar uma única instância da classe de teste e reutilizá-la para todos os métodos de teste, o que permite o compartilhamento de recursos entre os testes - essa vantagem será evidenciada mais a frente.

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class TestesIntegracao {

    @Autowired
    private TesteUtil testeUtil;

    @BeforeAll
    public void setup() throws Exception {
        testeUtil.preparaBancoDados();
    }
```

Figura 37 – Definição da classe de testes.

Já citada anteriormente como dependência da classe de teste, a classe `TesteUtil` (Figura 38) foi criada para centralizar e compartilhar recursos essenciais, como o *token* de autenticação, entre diferentes testes de integração na aplicação. Ao encapsular a lógica de obtenção e gerenciamento do *token*, bem como a configuração de *headers* e a realização de requisições HTTP, a classe facilita a reutilização desses recursos, reduz a duplicação de código nos testes, além de simplificar alguma manutenção que seja necessária no futuro após alguma alteração na aplicação (como na forma de obtenção do *token*, por exemplo).

Antes da execução dos testes, o método início (anotado com `@BeforeAll`) chama o método `preparaBancoDados` da classe `TesteUtil`. Esse método então executa outros dois, da classe `DBRepository` (Figuras 39 e 40), que irão rodar comandos SQL para limpar algumas tabelas e redefinir as sequências referentes a essas tabelas no banco de dados do

```java
@Component
public class TesteUtil {

    @Autowired
    private TestRestTemplate restTemplate;

    @Autowired
    private BDRepository repository;

    @Value("${app.tempo}")
    private Long tempoExpiraToken;

    public static final String emailAdmin = "ADMIN@ADMIN.COM.BR";
    private static final String senhaAdmin = "123";
    private static String token;
    private static LocalDateTime validadeToken;

    public void preparaBancoDados() throws Exception {
        repository.limpaBaseUsuarios();
        repository.limpaBasePatrimonios();
    }
```

Figura 38 – Classe `TesteUtil` criada para centralizar e compartilhar recursos essenciais.

ambiente de teste. Essa etapa é essencial para o sucesso dos testes, garantindo que as tabelas estejam preparadas, pois, ao término de cada execução do plano de testes, elas irão conter registros inseridos durante o processo.

Partindo para os testes, na Figura 41 é apresentado o código do teste `buscarPatrimoniosSemT` que, como indica o nome, valida se a busca por patrimônios, sem que seja passado um *token* de autenticação para a API, foi feita com sucesso, já que essa é uma chamada feita pela página pública do sistema. Para tal, é chamado um método `buscaPatrimonios` da classe `TesteUtil` (Figura 42), responsável por realizar a requisição HTTP GET no *endpoint* "/patrimonio/busca-patrimonios" e retornar a resposta para quem o chamou. Recebida a resposta, na última linha o método teste verifica se o retorno não é nulo, indicando que a consulta foi realizada com sucesso.

Abordando mais um teste, o `buscarUsuariosSemTokenDeveFalhar`, esse tem o objetivo de verificar que a busca por usuários deve falhar caso não seja enviado um *token*

```
public void limpaBaseUsuarios() throws Exception {
    String deleteQuery = "DELETE FROM tbl_usuario WHERE email <> '" + TesteUtil.emailAdmin + "';";
    executaQuery(deleteQuery);

    String alterSequenceQuery = "ALTER SEQUENCE seq_user RESTART WITH 2;";
    executaQuery(alterSequenceQuery);
}

public void limpaBasePatrimonios() throws Exception {
    String truncateQuery = "TRUNCATE TABLE tbl_patrimonio CASCADE;";
    executaQuery(truncateQuery);

    String alterSequenceQuery = "ALTER SEQUENCE seq_patrimonio RESTART WITH 1;";
    executaQuery(alterSequenceQuery);
}
```

Figura 39 – Métodos da classe `DBRepository` para preparar banco de dados para execução dos testes.

de autenticação válido (Figura 43). O método `buscaUsuarios` (Figura 44) requisita o *endpoint* que retorna a lista de usuários e foi centralizado na classe `TesteUtil`, pois será chamado em outro teste. O método de teste por sua vez valida se o código HTTP de retorno é igual a 403 e se o corpo da resposta é nulo, que são as condições necessárias para que o teste seja aprovado pelo JUnit.

Antes de seguir com a apresentação dos testes, é importante mostrar o funcionamento do método de recuperação do *token* de autenticação, que é utilizado no próximo teste que será abordado. O método `getHeadersTokenAdmin` (Figura 45) monta e retorna os *headers* com o *token* que recebeu do método `getTokenAdmin` (Figura 45). Esse por sua vez verifica se o *token* atual é válido (chamando o método *token*Valido) e, caso não for, efetua uma chamada HTTP POST no *endpoint* "/login" enviando no corpo da requisição um objeto (que será convertido em JSON) com o e-mail e senha do usuário. O *token* retornado é então atribuído ao atributo *token* e o momento de expiração do *token* é calculado e atribuído ao outro atributo de controle da classe `TesteUtil`, o `validadeToken`. Esses atributos de controle são validados no método *token*Valido (Figura 45), avaliando se já possui um *token* armazenado e se a data e hora atual é menor que a data e hora de expiração do *token*. Satisfazendo as condições, entende-se que já possui um *token* válido. Portanto, de forma resumida, se já possuir um *token* válido (que foi gerado anteriormente), o retorna, e, caso contrário, gera um novo *token*.

Voltando o foco para os testes, o teste `cadastrarUsuarioComumSucesso` (Figura 46) inicia recuperando o *token* de autenticação, que é passado para o método `buscaUsuarios` da classe `TesteUtil`, já que para recuperar a lista de usuários é necessário que o usuário esteja autenticado. A lista de usuários retornada será usada mais adiante no

```java
@Repository
public class BDRepository {

    @Autowired
    private DataSource data;

    @Value("${spring.jpa.properties.hibernate.default_schema}")
    private String schema;

    private void executaQuery(String sql) throws Exception {
        Connection conn = null;
        Statement statement = null;
        try {
            conn = data.getConnection();
            conn.setSchema(schema);
            System.out.println("Connected to database: " + conn.getMetaData().getDatabaseProductName());

            statement = conn.createStatement();
            int result = statement.executeUpdate(sql);
            System.out.println("Linhas afetadas: " + result);
        } catch (SQLException e) {
            throw new Exception("Erro ao executar query", e);
        } finally {
            try {
                if (statement != null) {
                    statement.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                throw new Exception("Erro ao fechar conexao com o banco de dados", e);
            }
        }
    }
```

Figura 40 – Métodos da classe `DBRepository` para preparar banco de dados para execução dos testes.

```java
@Test
public void buscarPatrimoniosSemTokenSucesso() {
    List<Patrimonio> patrimonios = testeUtil.buscaPatrimonios( httpHeaders: null);

    Assertions.assertNotNull(patrimonios);
}
```

Figura 41 – Código do teste *buscarPatrimoniosSemTokenSucesso()*.

```
public List<Patrimonio> buscaPatrimonios(HttpHeaders httpHeaders) {
    HttpEntity<Object> httpEntity = new HttpEntity<>( body: null, httpHeaders);

    ResponseEntity<List<Patrimonio>> response = this.restTemplate.exchange(
            url: "/patrimonio/busca-patrimonios",
            HttpMethod.GET,
            httpEntity,
            new ParameterizedTypeReference<List<Patrimonio>>() {
            }
    );

    return response.getBody();
}
```

Figura 42 – Chamada de *buscaPatrimonios()* da classe `TesteUtil`.

```
@Test
public void buscarUsuariosSemTokenDeveFalhar() {
    ResponseEntity<List<Usuario>> resposta = testeUtil.buscaUsuarios( httpHeaders: null);

    Assertions.assertEquals(HttpStatusCode.valueOf( code: 403), resposta.getStatusCode());
    Assertions.assertNull(resposta.getBody());
}
```

Figura 43 – Código do teste *buscarUsuariosSemTokenDeveFalhar()*.

código para validar o teste. Em seguida, é chamado o método `cadastraUsuarioComum` (Figura 47), em que se tem um JSON fixo que é enviado no corpo da requisição (juntamente com os *headers* que carrega o *token*) feita no *endpoint* de método POST "/usuario/cria-usuario". O método de teste guarda o corpo da resposta em um objeto Java e valida se o objeto não é nulo e se o id do objeto também não é nulo. Além disso, é feita uma nova chamada para buscar a lista de usuários, esperando que a quantidade de usuários da lista retornada nessa segunda chamada seja maior em um número que a quantidade verificada na primeira consulta.

O teste `cadastrarPatrimonioSucesso`, da Figura 48 é parecido com o teste anterior, com a diferença que neste é chamado o método `cadastrarPatrimonio` para invocar o *endpoint* POST "/patrimonio/cria-patrimonio".

A Figura 49 mostra o resultado da execução dos testes.

```java
public ResponseEntity<List<Usuario>> buscaUsuarios(HttpHeaders httpHeaders) {
    HttpEntity<Object> httpEntity = new HttpEntity<>( body: null, httpHeaders);

    ResponseEntity<List<Usuario>> response = this.restTemplate.exchange(
            url: "/usuario/busca-usuarios",
            HttpMethod.GET,
            httpEntity,
            new ParameterizedTypeReference<List<Usuario>>() {}
    );

    return response;
}
```

Figura 44 – Chamada de *buscaUsuarios()*.

```java
public String getTokenAdmin() {
    if(!tokenValido()) {
        LoginRequestDTO loginRequestDTO = new LoginRequestDTO();
        loginRequestDTO.setEmail(emailAdmin);
        loginRequestDTO.setSenha(senhaAdmin);

        LocalDateTime momentoAtual = LocalDateTime.now();
        ResponseEntity<LoginResponseDTO> response = restTemplate.postForEntity( url: "/login", loginRequestDTO, LoginResponseDTO.class);
        token = response.getBody().getToken();
        validadeToken = momentoAtual.plusMinutes(tempoExpiraToken);
    }
    return token;
}

private static boolean tokenValido() {
    return token != null && !token.isBlank()
            && LocalDateTime.now().isBefore(validadeToken);
}

public HttpHeaders getHeadersTokenAdmin() {
    String bearerToken = getTokenAdmin();

    HttpHeaders headers = new HttpHeaders();
    headers.add( headerName: "Accept", headerValue: "application/json");
    headers.setBearerAuth(bearerToken);

    return headers;
}
```

Figura 45 – Chamadas de *getHeadersTokenAdmin()*, *getTokenAdmin()* e *tokenValido()* para testes de autenticação.

```java
@Test
public void cadastrarUsuarioComumSucesso() {
    HttpHeaders httpHeaders = testeUtil.getHeadersTokenAdmin();
    List<Usuario> usuariosCadastradosPreviamente = testeUtil.buscaUsuarios(httpHeaders).getBody();

    UsuarioResponseDTO usuarioCadastrado = testeUtil.cadastraUsuarioComum();
    List<Usuario> usuariosCadastrados = testeUtil.buscaUsuarios(httpHeaders).getBody();

    Assertions.assertNotNull(usuarioCadastrado);
    Assertions.assertNotNull(usuarioCadastrado.getId());

    Assertions.assertEquals( expected: usuariosCadastradosPreviamente.size() + 1, usuariosCadastrados.size());
}
```

Figura 46 – Código do teste *cadastrarUsuarioComumSucesso()*.

```java
public UsuarioResponseDTO cadastraUsuarioComum() {
    String jsonCadastroUsuarioComum =
            """
                {
                    "nome": "USUARIO",
                    "email": "USUARIO@USUARIO.COM.BR",
                    "telefone": "34988889999",
                    "senha": "senha",
                    "tipoUsuario": 2,
                    "ativo": true
                }
            """;

    HttpHeaders httpHeaders = getHeadersTokenAdmin();
    httpHeaders.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> httpEntity = new HttpEntity<>(jsonCadastroUsuarioComum, httpHeaders);

    ResponseEntity<UsuarioResponseDTO> response = this.restTemplate.exchange(
            url: "/usuario/cria-usuario",
            HttpMethod.POST,
            httpEntity,
            UsuarioResponseDTO.class
    );

    return response.getBody();
}
```

Figura 47 – Chamada de *cadastraUsuarioComum()* para testes.

```java
@Test
public void cadastrarPatrimonioSucesso() {
    HttpHeaders headersTokenAdmin = testeUtil.getHeadersTokenAdmin();
    List<Patrimonio> patrimoniosCadastradosPreviamente = testeUtil.buscaPatrimonios(headersTokenAdmin);

    String jsonCadastroPatrimonio =
    """
        {
            "patrimonio": {
                "nome": "Nome",
                "descricao": "Descrição",
                "cep": "",
                "estado": "MG",
                "cidade": "Udia",
                "endereco": "",
                "complemento": "",
                "bairro": "",
                "latitude": "",
                "longitude": "",
                "categoria": "4"
            }
        }
    """;

    Patrimonio patrimonioCadastrado = testeUtil.cadastrarPatrimonio(jsonCadastroPatrimonio, headersTokenAdmin);
    List<Patrimonio> patrimonios = testeUtil.buscaPatrimonios(headersTokenAdmin);

    Assertions.assertNotNull(patrimonioCadastrado);
    Assertions.assertNotEquals( unexpected: 0, patrimonioCadastrado.getId());

    Assertions.assertEquals( expected: patrimoniosCadastradosPreviamente.size() + 1, patrimonios.size());
}
```

Figura 48 – Código do teste *cadastrarPatrimonioSucesso()*.

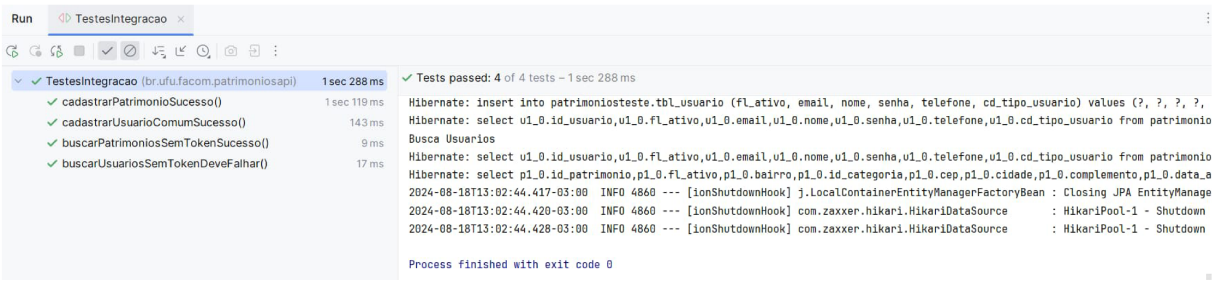

Figura 49 – Exibição dos resultados dos testes implementados.

5 Conclusão

Este trabalho apresentou o desenvolvimento de um sistema para o gerenciamento de patrimônios históricos, integrando tecnologias utilizadas no mercado, como Java, Spring Boot e Angular para oferecer uma solução eficiente. O sistema desenvolvido se propõe a facilitar a gestão de patrimônios ao permitir o cadastro, edição e exclusão de informações, além de garantir a integridade dos dados por meio de autenticação e autorização.

A implementação dos testes de integração foi importante para validar a eficácia dos *endpoints* da API, assegurando que o sistema funcione conforme os requisitos estabelecidos. Esses testes garantem que o fluxo das funcionalidades esteja corretamente implementado, validando as respostas do servidor e garantindo a confiabilidade do sistema.

Além dos aspectos técnicos, este projeto reforça a importância da preservação do patrimônio histórico, contribuindo para a valorização e a proteção de bens culturais. O sistema desenvolvido não apenas facilita a gestão administrativa desses bens, mas também promove a acessibilidade a informações relevantes sobre esse tema para o público em geral, incentivando a conscientização e a preservação cultural.

Por fim, o sistema pode ser aprimorado com novas funcionalidades, tanto na parte pública quanto na parte privada. Por exemplo, para melhorar a experiência e interação de um visitante com as páginas de visitação, poderia ser feita uma integração com uma plataforma de mapas para que o usuário situe-se melhor sobre a localização dos patrimônios. Enquanto na parte de gestão, na área privada, incluir novas funcionalidades voltadas para a gestão de reparos e inventários agregaria maior eficiência na preservação e manutenção dos patrimônios históricos. Nesse sentido, pensando na integração futura com sistemas já existentes de órgãos públicos, como os site de prefeituras, de instituições, e do governo federal, sugere-se adotar os padrões de desenvolvimento e de interface desses sistemas. Por último, além das melhorias funcionais sugeridas, incluir um link que leve à uma documentação sobre o sistema no próprio repositório Git onde se encontram os projetos, permitiria que interessados sugerissem correções, ajustes e melhorias nos mesmos.

Referências

ABE, I. M. **Sistema de gerenciamento de patrimônios tombados**. [S.l.]: Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Universidade Federal de Uberlândia, 2024. Citado na página 10.

ANGULAR MATERIAL. **Angular Material**. 2024. Disponível em: <https://material.angular.io/>. Acesso em: 19/08/2024. Citado na página 16.

BRASIL. **Decreto-Lei nº 25, de 30 de novembro de 1937. Organiza a proteção do patrimônio histórico e artístico nacional**. 1937. Diário Oficial da União, Brasília, DF, [30. nov. 1937]. Acesso em: 20/05/2024. Disponível em: <https://www.planalto.gov.br/ccivil_03/decreto-lei/del0025.htm>. Citado na página 8.

_____. **Constituição da República Federativa do Brasil**. 1988. Promulgada em 5 de outubro de 1988. Acesso em: 20/05/2024. Disponível em: <https://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm>. Citado na página 9.

CARVALHO, A. C. de. Preservação do patrimônio histórico no brasil: estratégias. **Revista Eletrônica do Programa de Pós-Graduação em Museologia e Patrimônio–PPG-PMUS Unirio**, v. 4, n. 1-2011, p. 117, 2011. Acesso em: 20/05/2024. Disponível em: <http://200.156.20.26/index.php/ppgpmus/article/view/195/158>. Citado na página 9.

Comunica UFU. **Alunos e professores do curso de Arquitetura da UFU criam aplicativo de realidade aumentada**. 2024. Disponível em: <https://comunica.ufu.br/noticias/2024/01/alunos-e-professores-do-curso-de-arquitetura-da-ufu-criam-aplicativo-de-0>. Acesso em: 02/08/2024. Citado na página 12.

GOOGLE. **What is Angular?** 2023. Disponível em: <https://angular.io/guide/what-is-angular>. Acesso em: 29/04/2024. Citado na página 16.

GRATER, M. T. Benefits of using automated software testing tools to achieve software quality assurance. 2005. Citado na página 18.

IBM. **Java Performance Monitoring**. 2023. Disponível em: <https://www.ibm.com/docs/en/aix/7.1?topic=management-java-performance-monitoring>. Acesso em: 29/04/2024. Citado na página 14.

IPHAN. **PORTARIA Nº IPHAN Nº 141, DE 12 DE DEZEMBRO DE 2023?** 2023. Disponível em: <https://www.gov.br/iphan/pt-br/centrais-de-conteudo/legislacao/atos-normativos/2023/portaria-no-iphan-no-141-de-12-de-dezembro-de-2023>. Acesso em: 02/08/2024. Citado na página 11.

JAMIL, M. A.; ARIF, M.; ABUBAKAR, N. S. A.; AHMAD, A. Software testing techniques: A literature review. In: IEEE. **2016 6th international conference on**

information and communication technology for the Muslim world (ICT4M). [S.l.], 2016. p. 177–182. Citado na página 18.

JSON. **Introduction to JSON Web Tokens**. 2024. Disponível em: <https://jwt.io/introduction>. Acesso em: 29/04/2024. Citado na página 16.

KÜHL, B. M. A restauração de monumentos históricos na frança após a revolução francesa e durante o século xix: um período crucial para o amadurecimento teórico. **Revista CPC**, n. 3, p. 110–144, 2007. Acesso em: 20/05/2024. Disponível em: <https://www.revistas.usp.br/cpc/article/view/15601/17175>. Citado na página 8.

KUMAR, S. A review on client-server based applications and research opportunity. **International Journal of Recent Scientific Research**, v. 10, n. 7, p. 33857–3386, 2019. Citado na página 19.

LONDRES, C. O patrimônio histórico na sociedade contemporânea. Rio de Janeiro: Edições Casa de Rui Barbosa, 2007, 2007. Acesso em: 20/05/2024. Disponível em: <https://rubi.casaruibarbosa.gov.br/bitstream/handle/20.500.11997/17985/FCRB_Escritos_1_7_Cecilia_Londres.pdf?sequence=1&isAllowed=y>. Citado na página 8.

LOPIS, E. A. Patrimônio histórico cultural: preservar ou transformar? uma questão conflituosa. **Mosaico**, v. 8, n. 12, p. 9–23, 2017. Acesso em: 20/05/2024. Disponível em: <https://periodicos.fgv.br/mosaico/article/view/65461/66858>. Citado na página 8.

MEDEIROS, M. C. de; SURYA, L. A importância da educação patrimonial para a preservação do patrimônio. In: **ANPUH – XXV Simpósio Nacional de História**. Fortaleza, Brasil: [s.n.], 2009. Acesso em: 20/05/2024. Disponível em: <https://www.snh2011.anpuh.org/resources/anais/anpuhnacional/S.25/ANPUH.S25.0135.pdf>. Citado na página 9.

MICROSOFT. **TypeScript for JavaScript Programmers**. 2024. Disponível em: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>. Acesso em: 20/05/2024. Citado na página 17.

_____. **TypeScript for the New Programmer**. 2024. Disponível em: <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>. Acesso em: 20/05/2024. Citado na página 17.

_____. **The TypeScript Handbook**. 2024. Disponível em: <https://www.typescriptlang.org/docs/handbook/intro.html>. Acesso em: 20/05/2024. Citado na página 17.

_____. **What is TypeScript?** 2024. Disponível em: <https://www.typescriptlang.org>. Acesso em: 29/04/2024. Citado na página 17.

MOZILLA. **JavaScript**. 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 20/05/2024. Citado na página 17.

NETO, A. C. D.; NATALI, A. C.; ROCHA, A. R.; TRAVASSOS, G. Caracterização do estado da prática das atividades de teste em um cenário de desenvolvimento de software brasileiro. In: **Anais do V Simpósio Brasileiro de Qualidade de Software**. Porto Alegre, RS, Brasil: SBC, 2006. p. 27–41. Disponível em: <https://sol.sbc.org.br/index.php/sbqs/article/view/15598>. Citado na página 18.

Node.js. **The V8 JavaScript Engine**. 2024. Disponível em: <https://nodejs.org/en/learn/getting-started/the-v8-javascript-engine>. Acesso em: 20/05/2024. Citado na página 17.

NPM. **TypeScript**. 2024. Disponível em: <https://www.npmjs.com/package/typescript>. Acesso em: 20/05/2024. Citado na página 17.

O GLOBO. **Oracle conclui aquisição da Sun Microsystems**. 2010. Disponível em: <https://oglobo.globo.com/economia/oracle-conclui-aquisicao-da-sun-microsystems-3062797>. Acesso em: 29/04/2024. Citado na página 14.

ORACLE. **O que é Java?** 2024. Disponível em: <https://www.java.com/pt-BR/download/help/whatis_java.html>. Acesso em: 29/04/2024. Citado na página 14.

OSÓRIO, V. E. P. Middlewares orientados a mensagens. 2023. Acesso em: 20/05/2024. Disponível em: <https://www.researchgate.net/profile/Victor-Osorio-10/publication/378309830_Middlewares_Orientados_a_Mensagens/links/65d392f8476dd15fb34466ed/Middlewares-Orientados-a-Mensagens.pdf>. Citado na página 19.

PostgreSQL. **The PostgreSQL Global Development Group. What is PostgreSQL?** 2024. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 20/05/2024. Citado na página 18.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software: uma abordagem profissional**. 9. ed. [S.l.]: McGraw Hill Education, 2021. Citado na página 20.

RETOMBA. **Sobre o aplicativo**. 2024. Disponível em: <https://retomba.com.br/#/>. Acesso em: 02/08/2024. Citado na página 12.

SILVA, F. R. **Testes backend automatizados no Sistema Online de Distribuição de Disciplinas**. [S.l.]: Trabalho de Conclusão de Curso (Graduação em Gestão da Informação) – Universidade Federal de Uberlândia, 2022. Citado na página 18.

SPRING. **Building an Application with Spring Boot**. 2024. Disponível em: <https://spring.io/guides/gs/spring-boot>. Acesso em: 29/04/2024. Citado na página 15.

_____. **Spring Data JPA Overview**. 2024. Disponível em: <https://spring.io/projects/spring-data-jpa#overview>. Acesso em: 29/04/2024. Citado na página 15.

_____. **Spring Data Overview**. 2024. Disponível em: <https://spring.io/projects/spring-data>. Acesso em: 29/04/2024. Citado na página 15.

_____. **Spring Framework Overview**. 2024. Disponível em: <https://docs.spring.io/spring-framework/reference/overview.html>. Acesso em: 29/04/2024. Citado na página 15.

_____. **Spring Security Overview**. 2024. Disponível em: <https://spring.io/projects/spring-security>. Acesso em: 29/04/2024. Citado na página 16.

The PostgreSQL Global Development Group. **A Brief History of PostgreSQL**. 2024. Disponível em: <https://www.postgresql.org/docs/current/history.html>. Acesso em: 20/05/2024. Citado na página 18.