UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Angélica Batassim Nunes

Avaliação experimental de técnicas de otimização de performance no website Arquigrafia

São Carlos

2022

Angélica Batassim Nunes

Avaliação experimental de técnicas de otimização de performance no website Arquigrafia

Monografia apresentada ao Curso de Engenharia Mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheira Mecatrônica.

Orientadora: Profa. Dra. Kalinka Regina Lucas Jaquie Castelo Branco

São Carlos

2022

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Nunes, Angélica Batassim
N972a Avaliação experimental de técnicas de otimização de performance no website Arquigrafia / Angélica Batassim Nunes; orientadora Kalinka Regina Lucas Jaquie Castelo Branco. São Carlos, 2022.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2022.

1. Performance web. 2. Otimização web. 3. Teste de carga. 4. Análise de performance. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: Angélica Batassim Nunes

Título: Avaliação experimental de técnicas de otimização de performance no website Arquigrafia

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Professora Dra Kalinka Regina Lucas Jaquie Castelo Branco (Orientador)

Nota atribuída: 10,0 (dez) _____ (assinatura)

Professor Dr. Rodrigo Nicoletti

Nota atribuída: 10,0 (dez) _____ (assinatura)

Professora Dra. Maíra Martins da Silva

Nota atribuída: 10,0 (dez) _____ (assinatura)

Média: 10,0 (dez)

Resultado: APROVADA

Data: 14/12/2022.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☑ NÃO ☐ Visto do orientador _____

Dedico este trabalho à todos aqueles que contribuíram com a minha educação e sobretudo a minha família, pelo apoio que sempre forneceram, independente das escolhas feitas.

AGRADECIMENTOS

Agradeço à minha família, pelo incentivo, encorajamento e suporte durante toda a trajetória da minha vida.

À minha orientadora Kalinka, por prover todo o suporte necessário para o desenvolvimento deste trabalho.

À Universidade de São Paulo pelos recursos investidos em minha permanência e educação.

À todos os professores que fizeram parte da minha vida.

"Bloom with pride"
Mana

RESUMO

NUNES, A. B. **Avaliação experimental de técnicas de otimização de performance no website Arquigrafia**. 2022. 72p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2022.

Performance web pode ser entendida como a velocidade de carregamento de um website ou serviço e a percepção que um usuário tem sobre ela. A velocidade de carregamento das páginas de um website pode definir o tempo que um usuário permanece navegando pelo conteúdo e aumenta a probabilidade de gerar conversões. Uma performance ruim, por outro lado, pode aumentar a taxa de abandono e diminuição de tráfego em um website. Este trabalho contribui com um estudo sobre técnicas de otimização de performance e uma análise experimental dessas técnicas aplicadas no website Arquigrafia, utilizando duas métricas: o tempo de resposta médio obtido por meio de testes de carga com Apache JMeter e a nota de performance do Lighthouse.

Palavras-chave: Performance web. Otimização web. Teste de carga. Análise de performance.

ABSTRACT

NUNES, A. B. **Experimental evaluation of performance optimization techniques on Arquigrafia website**. 2022. 72p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2022.

Web performance can be understood as the loading speed of a website or service and the perception that a user has about it. The loading speed of the pages of a website can define the time that a user spends browsing the content and increases the probability of generating conversions. Poor performance, on the other hand, can increase the abandonment rate and decrease traffic to a website. This work contributes with a study on performance optimization techniques and an experimental analysis of these techniques applied on the Arquigrafia website, using two metrics: the average response time obtained through load tests using Apache JMeter and Lighthouse performance score.

Keywords: Web performance. Web optimization. Load test. Performance analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MVC	*Model View Controller*
CSS	*Cascading Style Sheets*
JS	*JavaScript*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
FAQ	*Frequently Asked Questions*
SQL	*Structured Query Language*
ID	*Identity*
LAMP	*Linux, Apache, MySQL, PHP/Perl/Python*
API	*Application Programming Interface*
XML	*Extensible Markup Language*
JSON	*JavaScript Object Notation*

SUMÁRIO

1 INTRODUÇÃO

Criado para preservação e divulgação de imagens de arquitetura brasileira, o Arquigrafia <https://www.arquigrafia.org.br/> é também uma rede social (SANTOS; LIMA, 2020) gratuita e *open source*, regida pela licença *Creative Commons*. O principal objetivo do projeto Arquigrafia é ampliar a cultura visual arquitetônica, promovendo maior interação entre estudantes de arquitetura, professores, profissionais da área e, em uma instância mais ampla, entre todos os interessados no tema (SANTOS, 2012).

Atualmente, o Arquigrafia conta com um acervo de mais de 13,4 mil registros de arquitetura, entre imagens e vídeos georreferenciados e mais de 3,2 mil usuários ativos, institucionais e particulares cooperando com o crescimento contínuo *online* desde 2010. A aplicação reúne algumas características que impactam negativamente na performance web e consequentemente, na experiência do usuário:

- Excesso de requisições *Hypertext Transfer Protocol* (HTTP) que carregam *assets* como imagens, ícones, *scripts* e folhas de estilo (*Cascading Style Sheets* - CSS), e que muitas vezes são requisitados em páginas onde não são utilizados;
- Respostas HTTP muito longas e sem a possibilidade de filtrar o conteúdo desejado;
- Múltiplos processos sendo executados em um único método (*query*, registro de *logs* e renderização de página);
- Número elevado de tabelas e relações entre elas, resultando em *queries* longas no banco de dados;
- Tecnologias desatualizadas.

Esses fatores quando somados, resultam em um grande consumo de recursos do servidor (CPU, memória e armazenamento), e principalmente, aumentam o tempo de resposta das páginas e *Application Programming Interfaces* - APIs. Cada requisição feita a um servidor para retornar uma imagem, *script* ou folha de estilo tem um custo de tempo. Esse custo aumenta em dispositivos móveis, porque a latência da conexão é inerente em redes móveis 4G (GARDNER, 2011).

Melhorar a performance não apenas implica em uma experiência melhor para os usuários, mas também redução de custos e menor suscetibilidade de falhas. Atualmente, o algoritmo de indexação na busca do Google também valoriza uma boa performance, sendo uma poderosa ferramenta para a divulgação da plataforma.

1.1 Objetivos

Este trabalho visa promover um estudo de performance do website Arquigrafia. As estratégias utilizadas são *update* de versões e substituição do banco SQL (*Structured Query Language*) para NoSQL (MongoDB[1]), Redução de requisições, Cache de recursos e Compressão de imagens (WebP).

As ferramentas utilizadas para medir o impacto dessas técnicas são o JMeter[2] e Lighthouse[3].

1.2 Estrutura do trabalho

Este trabalho possui 4 capítulos. No capítulo 2 estão reunidos alguns trabalhos relacionados e um levantamento das definições, ferramentas e tecnologias utilizadas. Nesse capítulo também são detalhadas as métricas que são utilizadas no experimento de performance.

No capítulo 3, é feita uma análise do Arquigrafia legado utilizando engenharia reversa. Nessa análise, são apresentadas a estrutura do banco de dados, a arquitetura da aplicação, funcionalidades e regras de negócio. O processo de refatoração tem objetivos puramente didáticos e focados em performance, portanto funcionalidades, regras de negócio e estrutura do banco de dados não sofreram alteração, com exceção da remoção de recursos que já não são suportados pelas versões utilizadas na refatoração. São ainda apresentados nesses capítulo os processos de configuração das duas aplicações em uma máquina local, em conjunto com a arquitetura desenvolvida para a realização do experimento.

Desse modo, onze páginas e duas APIs foram selecionadas para receber os testes:

- Páginas: home, usuário, editar usuário, imagem, interpretação, contribuições, album, projeto, FAQ, cadastro e login.
- APIs: Imagens (retorna 1000 imagens) e Tags (retorna 1000 tags).

O experimento de performance é realizado aplicando as técnicas a seguir:

- Update de versões e substituição do banco SQL para NoSQL (MongoDB).
- Redução de requisições HTTP.
- Cache de recursos.
- Compressão de imagens (WebP).

[1] <https://www.mongodb.com/>
[2] <https://developer.chrome.com/docs/lighthouse/>
[3] <https://jmeter.apache.org/>

Por fim, o impacto das técnicas aplicadas é analisado por meio dos resultados obtidos pelas ferramentas JMeter e Lighthouse.

No capítulo 4 são apresentadas as conclusões, as limitações são discutidas e são apresentados os trabalhos futuros.

2 REVISÃO DA LITERATURA

Neste capítulos são apresentados os trabalhos relacionados ao desenvolvido neste trabalho de conclusão de curso, bem como as ferramentas utilizadas. Os trabalhos mencionados na seção 2.1 serviram como base para selecionar as técnicas de aumento de performance utilizadas no experimento, bem como as ferramentas destacadas na seção 2.2 e utilizadas para avaliar o impacto dessas técnicas.

2.1 Trabalhos relacionados

Algumas das técnicas de performance mencionadas por Gardner (2011) são reduzir o número de requisições HTTP que são realizadas no servidor, reduzir o tamanho dos arquivos e salvar dados em cache.

Gardner cita ainda que para reduzir as requisições que uma página faz, pode-se combinar arquivos de CSS e Javascript, incorporar dados binários para imagens em arquivos CSS por meio de URIs (*Uniform Resource Identifier*) de dados e carregar o conteúdo apenas quando o usuário deslizar a página até ele.

A fim de reduzir o tamanho das páginas também recomenda o uso de CSS ao invés de imagens para fins de estilo (como bordas arredondadas, gradientes e sombras), reduzir o tamanho das imagens ajustando compressão e qualidade, utilizar *sprite images* e compressão Gzip. Também são mencionadas ferramentas para medir a performance, como Google PageSpeed[1] e Yslow[2].

Segundo a Google Developers (2022), WebP é um método de compressão com e sem perdas que pode ser usado em uma grande variedade de imagens fotográficas, translúcidas e gráficas encontradas na web. O time conduziu uma pesquisa comparativa entre os formatos de imagens JPEG, PNG, GIF e outros, e constataram que O WebP normalmente atinge uma média de 30% mais compactação do que JPEG e JPEG 2000 (outros métodos de compressão), sem perda de qualidade de imagem.

Um estudo de Marang (2018) testou e comparou algumas técnicas de melhoria de performance como cache de recursos, diminuição de requisições HTTP, web workers e priorização de conteúdo em uma aplicação web. Analisando os dados capturados por meio de *logging* manual sobre cada função executada (registros com o nome da função, localização e tempo de execução), foi concluído que após as implementações, o tempo de carregamento diminuiu em 45%. O tempo de carregamento da página inicial diminuiu em 53% e o tempo de navegação para as demais páginas a partir da *homepage* diminuiu 61%.

[1] <https://pagespeed.web.dev/>
[2] <http://yslow.org/>

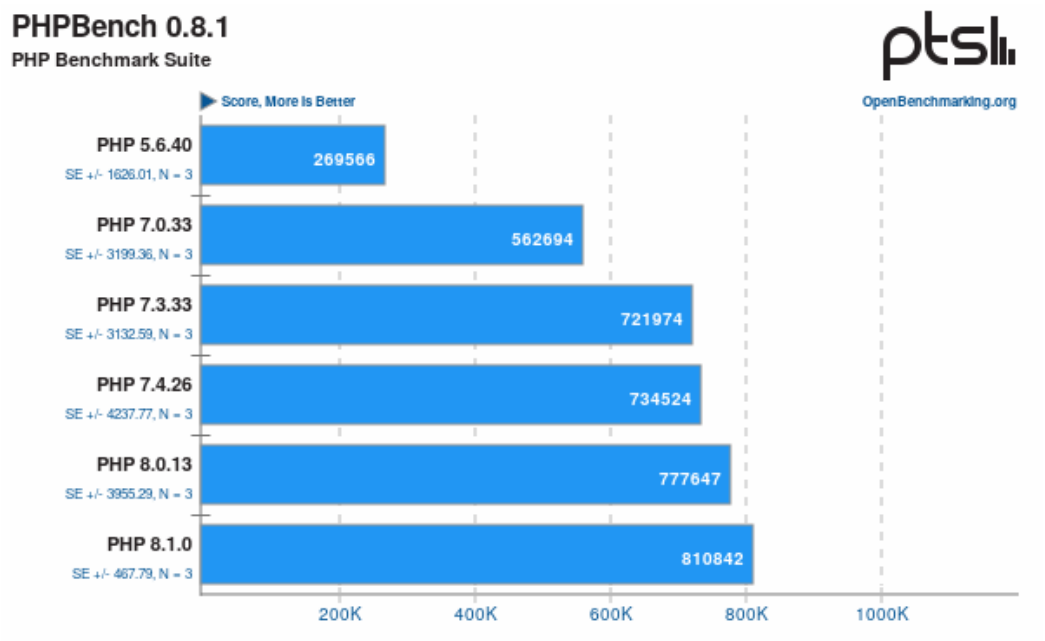
A implementação do cache de recursos resultou em uma melhora de 18% no tempo de carregamento da aplicação.

SyamimiSaid et al. (2014) levantaram uma revisão sobre performance web. Os pesquisadores focaram em formas de aumentar a performance, problemas e efeitos da má performance na experiência do usuário. Além disso, também mencionam ferramentas para analisar performance, como Yotta site optimizer[3], Google PageSpeed e Yslow.

Keshavarz (2021) atestou a hipótese de que o MongoDB performa melhor do que o MySQL por meio de experimentos que executam operações de leitura, escrita, edição e exclusão (*Create, Read, Update and Delete* - CRUD). Os experimentos foram realizados utilizando a ferramenta Jmeter para executar as *queries* nos dois bancos de dados e calcular o tempo de resposta de cada um. Os resultados mostraram que para a maioria das operações o MongoDB demonstrou performar melhor.

A Figura 1 ilustra o estudo feito por Larabel (2021), no qual foram comparadas as performances do PHP 8.1, lançado em novembro de 2021 e suas versões anteriores. Todas as versões do PHP foram recém-criadas no mesmo sistema utilizando os mesmos pacotes e as mesmas opções de compilação. O PHP 5.6 foi lançado em 2014 sendo o PHP 5.6.40 seu último lançamento, no início de 2019. O resultado dessa pontuação mostra que a versão 8.1 tem um desempenho superior a todas as versões anteriores.

Figura 1 – Pontuação de perfomance por versão do PHP



Fonte: (LARABEL, 2021).

[3] <https://www.yottaa.com/>

2.2 Ferramentas

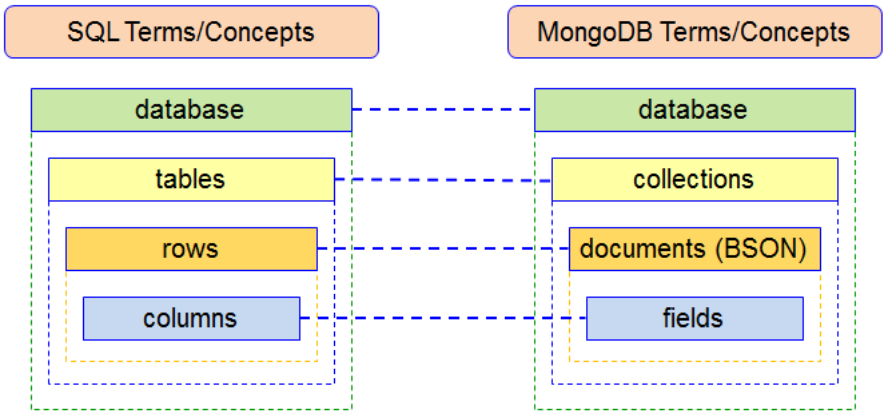
As principais ferramentas utilizadas para o desenvolvimento deste projeto são apresentadas e descritas brevemente nesta seção. Constam:

- **MySQL**: Banco de dados relacional desenvolvido pela Oracle e baseado em *Structured Query Language* (SQL). Atualmente, o Arquigrafia utiliza um banco Mysql para armazenar os dados. No MySQL os dados são gravados em linhas (registros) dentro de tabelas. Cada linha da tabela possui campos padronizados e cada uma dessas linhas é referenciada por um identificador (ID), normalmente conhecido como chave primária. Por meio dos IDs é possível montar relacionamentos entre tabelas, por exemplo, utilizando o ID (chave primária) de determinada tabela em uma outra, como chave estrangeira.

- **MongoDB**: Banco de dados não relacional que fornece suporte a armazenamento de JSON. Este tipo de banco é conhecido pela eficiência em armazenar grandes volumes de dados. No MongoDB, os dados são armazenados em coleções, que são correspondentes às tabelas dos bancos SQL. Os dados armazenados dentro das coleções são chamados de documentos, que correspondem às linhas das tabelas SQL e não precisam, necessariamente, ter o mesmo padrão.

  Na Figura 2 é ilustrada a correspondência entre os bancos de dados SQL e MongoDB.

Figura 2 – Correspondências entre os bancos de dados SQL e MongoDB



Fonte: (Studio 3T, 2022).

- **PHP**: Principal linguagem de programação usada no desenvolvimento da aplicação.

- **Laravel**[4]: Laravel é um *framework* baseado em PHP e *open source*. Por padrão, o Laravel utiliza o padrão MVC (*Model, View, Controller*), que é focado na separação de conceitos em três camadas:

  - *Model*: gerencia o comportamento dos dados por meio de regras de negócio, lógica e funções. Essa camada é responsável por interagir com os dados do banco, e no Laravel, ela se comporta como a abstração de uma tabela ou coleção.
  - *View*: onde ocorre a interação do usuário com os dados (*front-end*).
  - *Controller*: camada que faz a intermediação entre a entrada e saída dos dados (*front-end* e *back-end*). Em geral, um *controller* recebe a requisição, invoca as classes necessárias para executar uma determinada ação, por exemplo, realizar uma consulta no banco por meio da *Model*, e envia a resposta para a *view* renderizar os dados solicitados.

  Também é possível associar o Laravel à um *framework* de *front-end*, como o VueJS[5]. Uma das possibilidades é tornar a *view* responsável por invocar os componentes do VueJS, ao invés de renderizar o conteúdo diretamente.

- **Docker**[6]: O Docker é uma plataforma que possibilita criação de ambientes isolados por meio da virtualização de aplicações e ambientes dentro de *containers*. Assim, é possível utilizar versões concorrentes de um mesmo ambiente sem haver conflitos.

- **Docker Compose**[7]: Ferramenta utilizada para criar aplicações *multi-container*, ou seja, com ele é possível desenvolver uma imagem *Docker* com múltiplos *containers* de acordo com o objetivo. Neste trabalho, o Docker Compose foi utilizado para montar os ambientes de teste do Arquigrafia.

- **Composer**[8]: Gerenciador de dependências que possibilita baixar e atualizar dependências e bibliotecas em aplicações escritas em PHP. Esse gerenciador é um dos requisitos para instalação do Laravel.

- **Apache HTTP Server**[9]: O Apache é um servidor web aberto e mantido pela *Apache Software Foundation*. Sua principal função é configurar objetos que venham a ser acessados por um cliente via navegador, tanto em rede interna quando externa.

4 <https://laravel.com/>
5 <https://vuejs.org/>
6 <https://www.docker.com/>
7 <https://docs.docker.com/compose/>
8 <https://getcomposer.org/>
9 <https://httpd.apache.org/>

- **Studio3T**[10]: Software que reúne ferramentas para manipulação de coleções MongoDB por meio de uma interface gráfica. A ferramenta utilizada no trabalho foi o conversor de banco de dados SQL para MongoDB.
- **WebP Converter**[11]: Software desenvolvido pela Vertexshare. Neste trabalho foi utilizado para converter e comprimir as imagens do Arquigrafia em WebP.
- **Javascript**: Linguagem de programação com diversas aplicações, no entanto, é amplamente utilizada para desenvolvimento *front-end*.
- **CSS**: CSS é uma linguagem de estilo utilizada para descrever a forma como os elementos criados em HTML ou XML (*Extensible Markup Language*) são apresentados na tela. Arquivos do tipo CSS são geralmente chamados de folhas de estilo, por esse motivo.
- **Vue.js**[12]: É um *framework* escrito em javascript focado no desenvolvimento de interfaces gráficas.
- **Redis**[13]: Redis é um armazenamento de dados em memória orientado por dados do tipo chave-valor. É amplamente utilizado para armazenar dados em cache com uma durabilidade opcional.
- **Apache JMeter**: Software *open source* baseado em Java. O software possibilita extrair diversas métricas em tempo real e condições definidas pelo usuário, como número de requisições, número de usuários virtuais (*threads*), limite de requisições por usuário, comportamento mediante a erros e outros. Assim é possível construir *scripts* que executam testes de *stress*, carga, pico entre outros, tanto em páginas web, quanto APIs e operações em banco de dados.
- **Lighthouse 9**[14]: Ferramenta de auditoria *open source* desenvolvida pela Google que coleta dados de websites em tempo real e atribui pontuações de performance, acessibilidade, boas práticas e SEO. O Lighthouse é a versão para desenvolvedores do Google PageSpeed, utilizando simulações para avaliar as páginas, enquanto Google PageSpeed utiliza os mesmos mecanismos do Lighthouse mas os complementa com dados reais baseados nos últimos 28 dias. A ferramenta ainda contribui com relatórios e sugestões para a otimizar as métricas utilizadas no cálculo das pontuações.

Com base nas ferramentas selecionadas e também nas necessidade do projeto foram levantadas as métricas a serem averiguadas.

10 <https://studio3t.com/>
11 <https://anywebp.com/software.html>
12 <https://vuejs.org/>
13 <https://redis.io/>
14 <https://developer.chrome.com/docs/lighthouse/overview/>

2.3 Métricas

Para medir a performance foram utilizadas duas métricas: O tempo médio de resposta obtido nos testes de carga do JMeter e a nota de performance atribuída pelo Lighthouse.

O Lighthouse foi utilizado para atribuir uma nota de performance para 11 páginas do website a cada cenário proposto. Essa auditoria de performance é baseada em seis métricas:

- **First Contentful Paint (FCP)**: O FCP mede quanto tempo leva para o navegador renderizar o primeiro conteúdo da tela;
- **Speed Index (SI)**: Mede a rapidez com que o conteúdo é exibido visualmente durante o carregamento da página.
- **Largest Contentful Paint (LCP)**: Tempo que leva até o usuário visualizar o maior conteúdo da tela, que pode ser uma imagem, texto, plano de fundo ou similar.
- **Time to Interactive (TTI)**: O tempo que a página leva para se tornar interativa, por exemplo, o momento em que os botões se tornam clicáveis e redirecionam para o caminho correto.
- **Total Blocking Time (TBT)**: Mede o tempo total que uma página é impedida de responder à interação do usuário, como cliques do mouse, toques na tela ou pressionamentos do teclado. É o tempo entre o FCP e o TTI.
- **Cumulative Layout Shift (CLS)**: CLS é a medida do quanto uma página web muda inesperadamente durante sua vida. Por exemplo, se um visitante do site carregou uma página e, enquanto a estava lendo, um *banner* apareceu na tela. Esse tipo de mudança aumenta o CLS. Uma boa pontuação de CLS deve ser de 0,1 ou menos.

A nota de performance é o resultado da média ponderada ilustrada na Tabela 1.

No Lighthouse, as notas de performance seguem a classificação abaixo:

- 0 a 49: Ruim
- 50 a 89: Necessita de melhoria
- 90 a 100: Bom

Atualmente, o Lighthouse simula as condições de carregamento de página de um dispositivo intermediário (Moto G4) em uma rede móvel para celular e um *desktop*

Tabela 1 – Pesos das métricas para definição da nota de performance no Lighthouse 9

Métrica	Peso
First Contentful Paint	10%
Speed Index	10%
Largest Contentful Paint	25%
Time to Interactive	10%
Total Blocking Time	30%
Cumulative Layout Shift	15%

Fonte: (Google Developers, 2019b)

emulado com conexão cabeada para *desktop* (Google Developers, 2019a). Por isso, a nota de performance em geral é menor para a simulação mobile.

Com o objetivo avaliar a performance em condições de carga, as mesmas páginas testadas pelo Lighthouse foram analisadas por meio de requisições feitas pela ferramenta JMeter. Além das onze páginas do website, duas APIs do tipo GET também foram analisadas. O teste do JMeter garante uma grande confiabilidade nos resultados, visto que as requisições são feitas em tempo real e em grande volume, extraindo dados próximos da realidade.

2.4 Considerações Finais

Nesta seção foram apresentados trabalhos relacionados, bem como as ferramentas que foram utilizadas para o desenvolvimento deste trabalho.

No próximo capítulo é apresentado o desenvolvimento do trabalho.

3 DESENVOLVIMENTO

3.1 Desconstruindo o Arquigrafia

Existem muitos desafios no processo de refatoração de uma aplicação legado. Levando em consideração a complexidade de realizar mudanças na aplicação legado e a fim de preservar os recursos e dados foi realizado um levantamento das funcionalidades, regras de negócio e banco de dados do Arquigrafia por meio de engenharia reversa.

3.1.1 *Funcionalidades e regras de negócio*

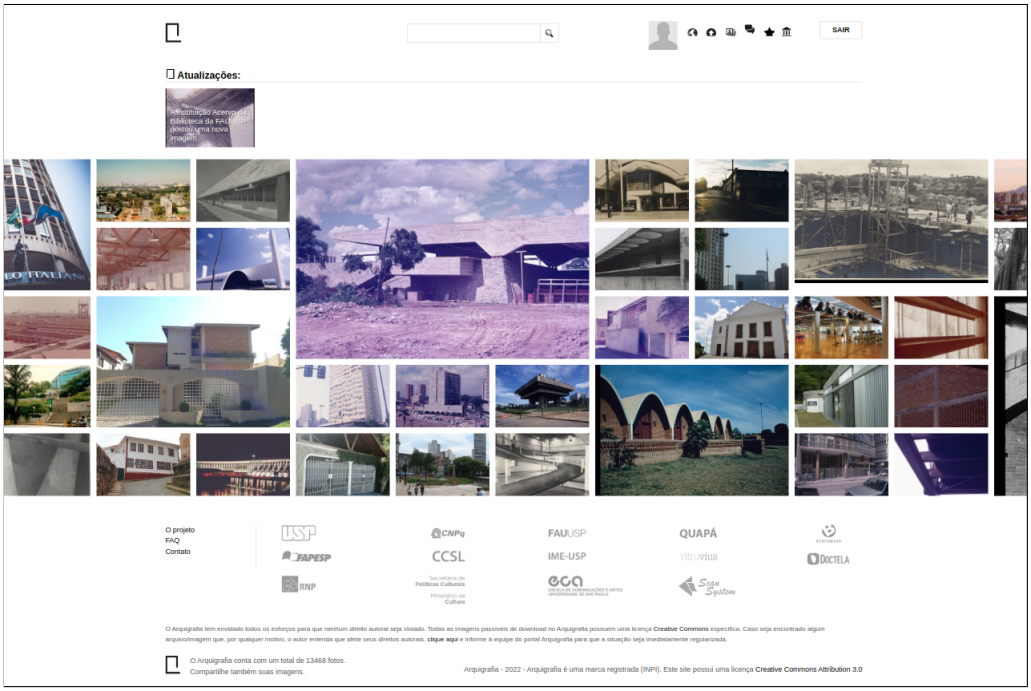
No Arquigrafia legado é possível que um usuário adicione conteúdo (imagens ou vídeos de arquitetura) e interaja com o conteúdo de outros usuários adicionando comentários, *likes* e registrando sua interpretação sobre a obra.

Existem dois recursos de *login* na aplicação, sendo um destinado aos usuários comuns e outro destinado a usuários vinculados a uma instituição participante do projeto: Equipe Arquigrafia, Acervo da Biblioteca da FAUUSP (Faculdade de Arquitetura e Urbanismo da Universidade de São Paulo), Acervo Quapá e Museu Republicano Convenção de Itu. O segundo recurso, chamado *login* institucional, permite que o usuário gerencie o conteúdo publicado pela instituição com a qual mantém o vínculo. Quando o usuário não está autenticado, a área de visitantes fornece acesso aos recursos básicos, como visualizar imagens, perfil de usuários, as páginas sobre o projeto, FAQ (*Frequently Asked Questions*) e realizar buscas.

A área logada comum fornece acesso aos seguintes recursos:

- CRUD de imagens, vídeos e álbuns (adicionar, visualizar, editar e modificar);
- Interações usuário-usuário: enviar mensagem via *chat*, seguir usuário, visualizar *leaderboards* (*ranking* de usuários que mais contribuem) e visualizar atualizações dos usuários seguidos;
- Interações usuário-imagem: adicionar comentário, dar *like*, sugerir modificações, denunciar imagem, registrar percepção sobre imagem, baixar imagem, compartilhar imagem nas redes sociais (*Facebook* e *Twitter*) e adicionar imagem (de sua autoria ou não) a um álbum;
- Museu ao ar livre: recurso focado em acessibilidade, para adicionar áudios relacionados à uma determinada obra. Atualmente o recurso está disponível no ambiente de produção, mas está incompleto.

Figura 3 – Home da área logada.



Fonte: Elaborada pela autora.

Na Figura 3 é ilustrada uma imagem do site do Arquigrafia já em área logada.

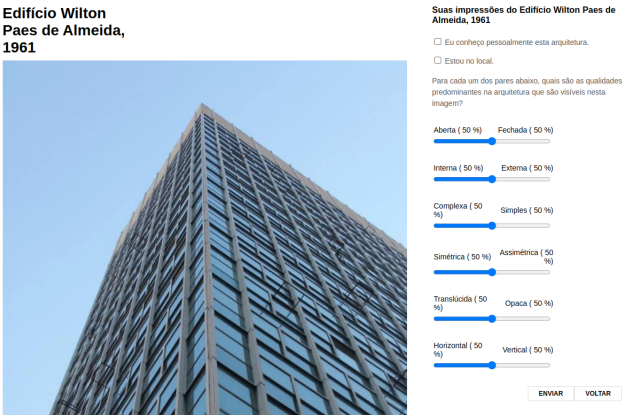
Uma sugestão pode ser do tipo revisão ou edição, sendo a primeira uma mudança em um campo existente e a segunda, o preenchimento de um campo vazio. Quando uma sugestão é feita, a imagem que recebeu a sugestão bloqueia o recebimento de outras sugestões até que o autor aceite ou rejeite a proposta. Ao enviar uma sugestão, um e-mail de aviso é enviado para o autor da imagem. Já no momento do aceite ou rejeição da sugestão, um e-mail é enviado para o autor da sugestão. Não é possível sugerir modificações em conteúdo publicado por uma instituição.

A interpretação de uma obra pode ser feita por meio de um formulário que avalia seis características da obra: aberta, interna, complexa, simétrica, translúcida e horizontal. A média das interpretações também é disponibilizada ao visualizar o respectivo conteúdo e imagens com médias similares são agrupadas ao visualizar uma interpretação.

Além dos recursos citados anteriormente, na área institucional um usuário pode visualizar rascunhos feitos por ele e/ou por outros funcionários da mesma instituição. Um rascunho pode ser criado adicionando as informações do conteúdo antes que ele seja publicado.

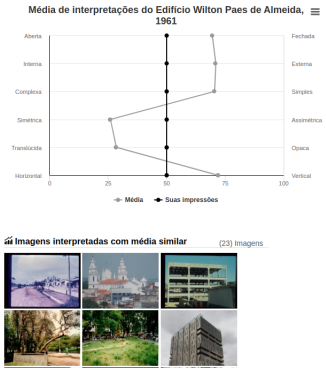
Todos esses processos são registrados diariamente em *logs* a cada acesso autenticado ou não. Esses *logs* definem a ação (como adicionar imagem e realizar busca), o usuário (quando autenticado apresenta seu identificador único) e a URL de origem.

Figura 4 – Formulário de interpretações



Fonte: Elaborada pela autora.

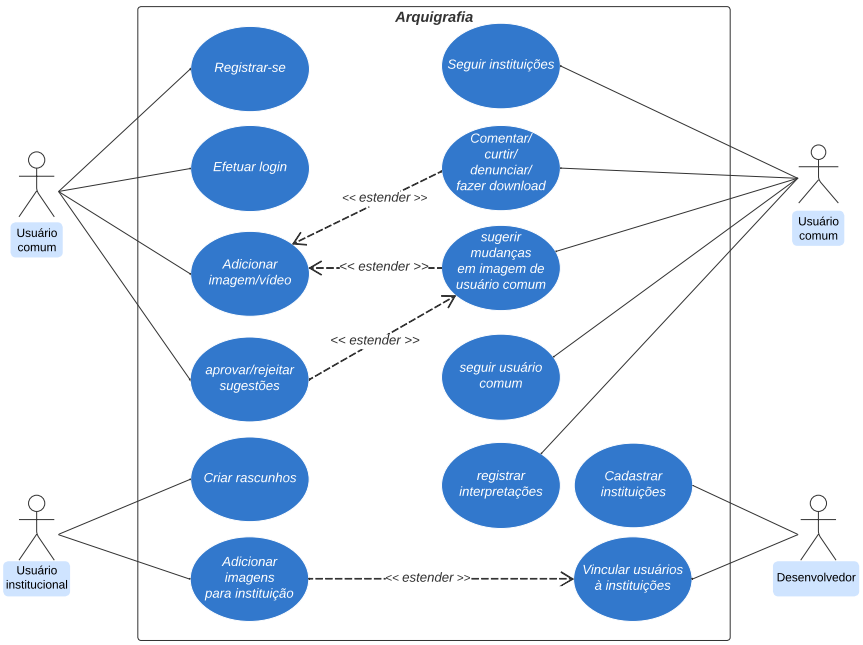
Figura 5 – Média de interpretações



Fonte: Elaborada pela autora.

O diagrama de casos de uso ilustrado na Figura 6 apresenta as funcionalidades mapeadas no estudo da aplicação legado.

Figura 6 – Diagrama de casos do Arquigrafia legado



Fonte: Elaborada pela autora.

3.1.2 *Banco de dados*

O banco de dados do Arquigrafia legado é um banco SQL com 41 tabelas que se relacionam entre si, sendo as principais *users* e *photos*.

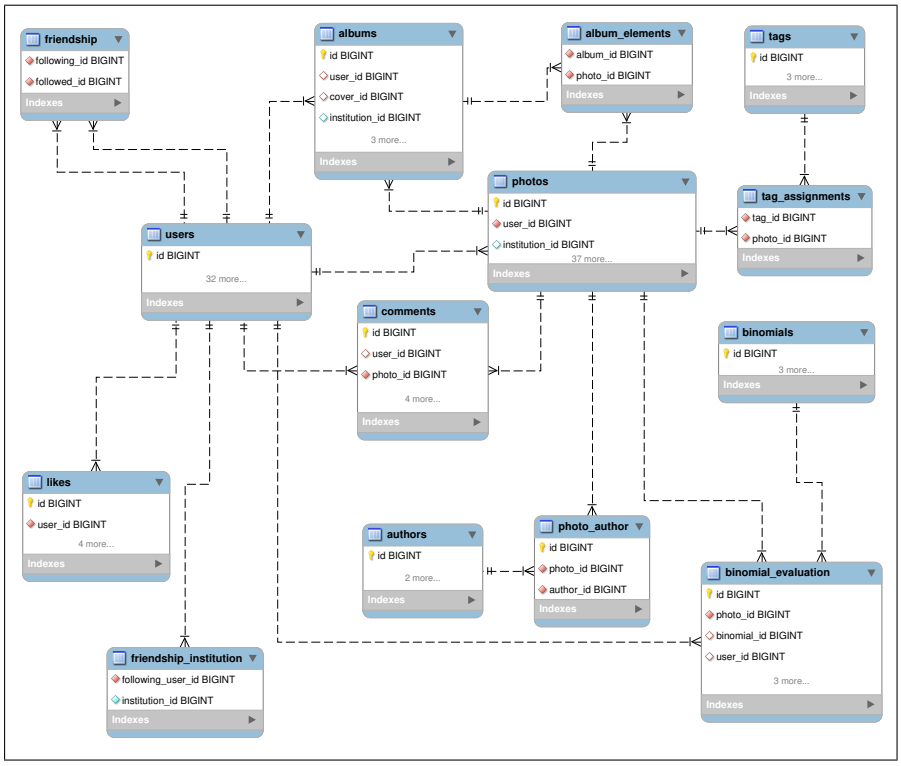
Analisando as tabelas foi possível identificar alguns recursos incompletos, como a existência de papéis nas tabelas *roles* e *users_roles*, mas que atualmente não possuem

utilidade prática na aplicação. Outro exemplo de recurso incompleto é o sistema de gamificação, que aparece nas tabelas *badges* e *user_badges*, mas que também não está implementado na aplicação.

Existem também tabelas vazias, como no caso de *external_accounts*, *blocks*, *medals* e *moderators*, além de tabelas sub-utilizadas e colunas duplicadas.

Na Figura 7 é ilustrado um resumo das principais tabelas e relacionamentos envolvidos nas *queries* que retornam o conteúdo das páginas do Arquigrafia, como visualizar imagens e perfis de usuários.

Figura 7 – Diagrama Entidade-Relacionamento na forma reduzida



Fonte: Elaborada pela autora.

3.1.3 *Arquitetura*

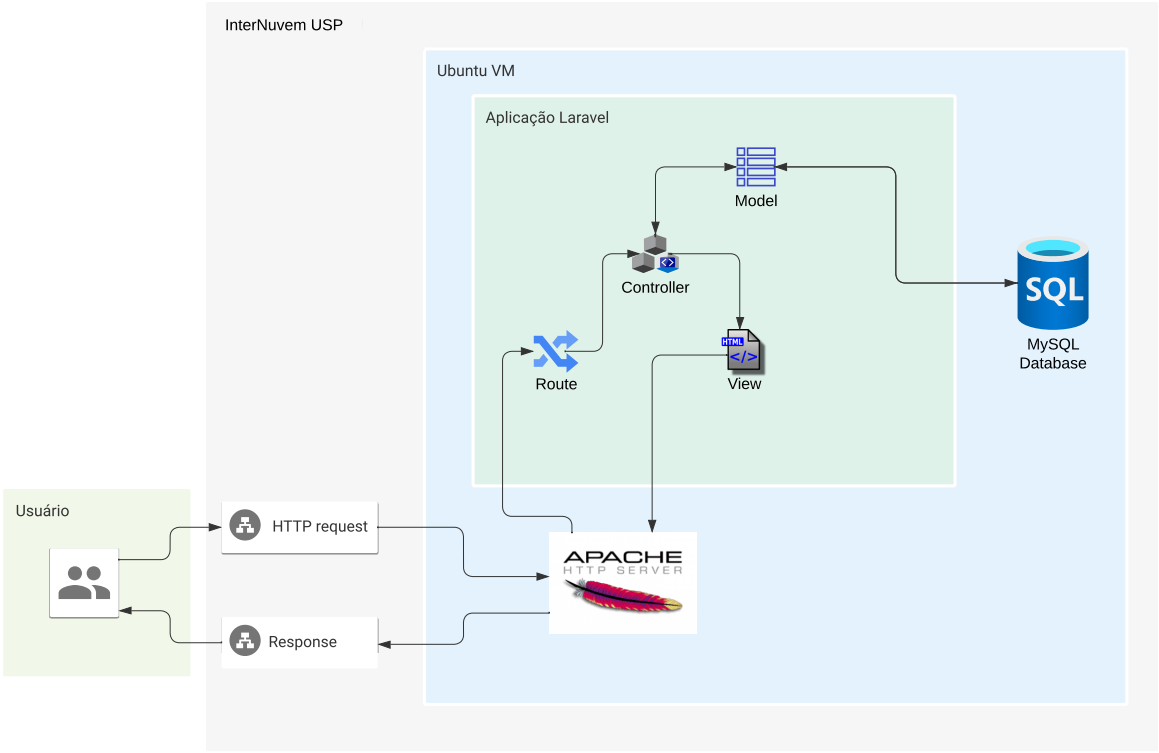
A arquitetura do Arquigrafia legado consiste da *stack LAMP* (Linux, Apache, MySQL, PHP/Perl/Python), um ambiente Linux composto pelo Apache, MySQL e PHP. O Apache2 é utilizado como servidor web que configura o caminho da aplicação Laravel e expõe as portas 80 e 443 para serem acessadas via web. O banco de dados MySQL 5.6 é instalado diretamente no servidor (InterNuvem USP).

O padrão de arquitetura do Laravel 4 é o MVC. Isso significa que os *controllers* são responsáveis por receber as requisições e acessar as *models*, que por sua vez manipulam

os registros do banco de dados. No *front-end* são utilizados os próprios recursos do *blade engine* para renderizar o conteúdo das páginas, não havendo uma divisão clara entre *front-end* e *back-end*.

Na Figura 8 é ilustrado o diagrama da arquitetura do Arquigrafia legado.

Figura 8 – Arquitetura do Arquigrafia legado



Fonte: Elaborada pela autora.

3.2 Configuração do ambiente de testes

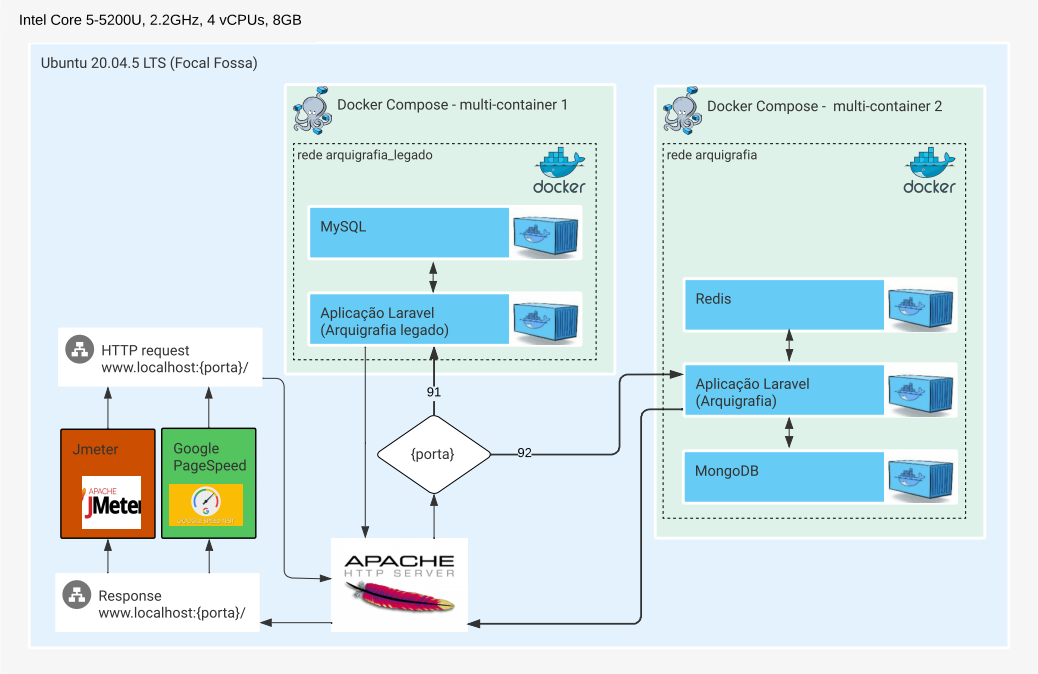
A arquitetura dos testes foi desenvolvida em uma máquina Ubuntu 20.04.5 LTS (Focal Fossa). O hardware utilizado foi um *notebook* com processador Intel Core i5-5200U, 2.2GHz, 4 vCPUs e 8GB de memória. Versão do Docker 20.10.17, Docker Compose 1.21.0, Jmeter 5.5 e Apache 2.4.41.

Para instalar a versão mais atual do Arquigrafia, foi feito um *backup* do servidor e do banco de dados. A virtualização da aplicação com Docker foi a solução escolhida para possibilitar o *update* das versões do PHP 5.6 e Laravel 4.2 para o PHP 8.1 e Laravel 8, em ambientes isolados do restante, evitando a necessidade de instalações nativas e versões concorrentes no sistema operacional. Assim, o ambiente do Arquigrafia

legado foi reproduzido na aplicação *multi-container* 1, ilustrada na Figura 9, e o ambiente Arquigrafia, onde são aplicadas as técnicas de performance, foi criado a partir do Arquigrafia legado, com suas versões atualizadas, o banco de dados NoSQL MongoDB e o Redis, que posteriormente foram utilizados no processo do experimento. Esse ambiente está representado pela aplicação *multi-container 2* ilustrado na Figura 9.

Para servir duas aplicações simultaneamente, foi utilizado o Apache, que recebe todas as requisições e redireciona para a devida aplicação, de acordo com a porta indicada. A aplicação legado foi então servida na porta 91 e a nova aplicação foi servida na porta 92.

Figura 9 – Arquitetura do experimento



Fonte: Elaborada pela autora.

O experimento do JMeter foi configurado para coletar um grande volume de dados a fim de possibilitar uma análise mais consistente. Além disso, as limitações do sistema foram levadas em consideração.

Foram utilizadas 10 *threads* (usuários virtuais) que executam 30 requisições cada durante três iterações. Assim, 900 requisições são executadas por teste.

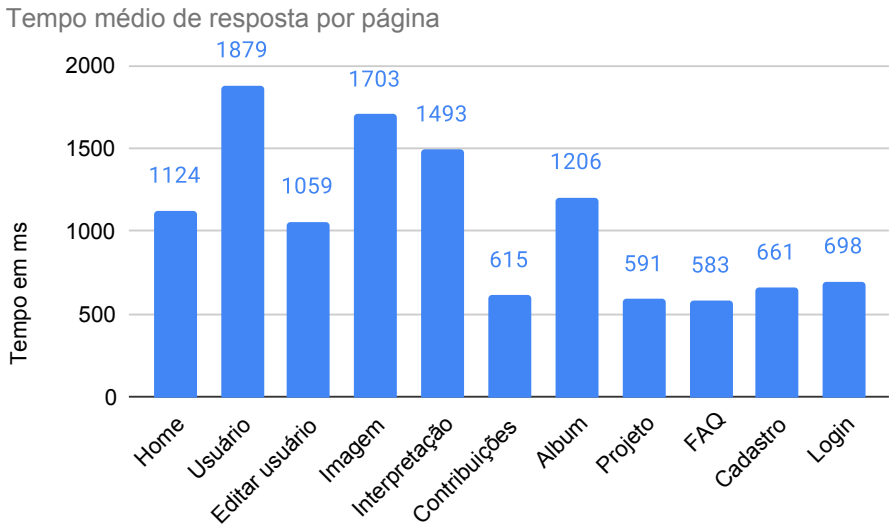
As páginas testadas foram as mesmas nas duas aplicações e o conteúdo das páginas também foi o mesmo para que não houvessem páginas mais ou menos longas sendo comparadas nas mesmas condições.

3.3 Análise de performance do Arquigrafia legado

As páginas e APIs do Arquigrafia legado foram submetidas ao teste de carga e à auditoria do Lighthouse para fins de comparação após aplicação das técnicas de otimização de performance.

É possível observar, por meio do gráfico ilustrado na Figura 10, que o maior tempo médio de resposta é da Home, com 1879ms, seguido pela página de Imagem, com 1703ms. Outras páginas que ficaram acima de 1000ms foram Interpretação, com 1493ms, Álbum, com 1206ms e Editar usuário, com 1059ms. O menor tempo de resposta médio foi da página de Projeto, com 591ms.

Figura 10 – Gráfico do tempo médio de resposta por página na aplicação legado.

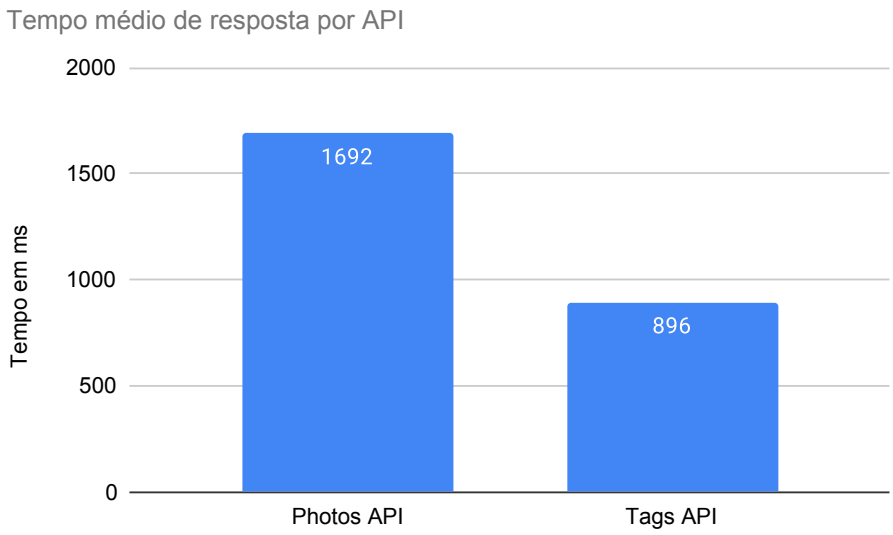


Fonte: Elaborada pela autora.

No caso das APIs, o maior tempo de resposta foi da API de Imagens, com 1692ms, conforme pode ser observado no gráfico ilustrado na Figura 11.

Como pode ser observado na Tabela 2, na classificação do Lighthouse, a página com menor performance foi a Home, com uma nota 44 na versão *desktop*. Entre as 11 páginas, 4 são classificadas como ruins na escala do Lighthouse (1-49), sendo Home, Usuário, Imagem e Interpretação. As páginas de Contribuições, Álbum e FAQ foram classificadas como intermediárias (50-89) e as páginas de Login e Projeto classificadas como boas (90-100).

Figura 11 – Gráfico do tempo médio de resposta por API na aplicação legado.



Fonte: Elaborada pela autora.

Tabela 2 – Avaliação do Lighthouse antes de aplicar as técnicas de performance

Página	Nota de performance	Classificação
Home	44	Ruim
Usuário	28	Ruim
Editar usuário	84	Intermediária
Imagem	38	Ruim
Interpretação	49	Ruim
Contribuições	63	Intermediária
Album	78	Intermediária
Projeto	91	Boa
FAQ	89	Intermediária
Cadastro	79	Intermediária
Login	90	Boa

3.4 Cenário 1: *Update* de versões e mudança do banco para MongoDB

Ao virtualizar a aplicação, foi utilizado o Laravel 8, uma versão recente e estável do *framework* utilizado no Arquigrafia legado. Os requisitos do Laravel 8 são o orquestrador de dependências Composer e o PHP 7.3 ou acima, portanto foi utilizado o PHP 8.1, baseado no score de performance dessa versão. Nesse processo foram realizadas mudanças para que a aplicação se adequasse aos novos padrões estruturais e de sintaxe. Foram desenvolvidos três *containers* para executar a aplicação:

- Arquigrafia: aplicação web e variáveis de ambiente que realizam as configurações básicas para o funcionamento do Laravel, como o banco de dados e o *driver* de *cache* que serão utilizados;
- Mongo: banco de dados MongoDB
- Redis: banco de dados de *cache*

Com o resultado do *update* de versões, algumas bibliotecas se tornaram incompatíveis, como no caso da *facebook/php-sdk-v4*, que implementa autenticação de usuários via *Facebook*. Além dessa biblioteca, foram removidos os pacotes *messenger*, *artdarek/pusherer* e *tricki/laravel-notification*, que implementam o *chat* e o envio de notificações na aplicação legado.

A fim de popular as coleções do novo banco de dados foi utilizado o software Studio 3T, executando a conversão do banco SQL para o MongoDB. O software tem uma limitação de 1000 documentos por coleção em sua versão de teste, portanto, cada tabela foi convertida em uma coleção limitada em 1000 documentos.

Do lado do *framework* Laravel, foi utilizada a biblioteca *jenssegers/mongodb*, que adiciona funcionalidades ao *Eloquent* e ao *Query Builder* para utilizar o MongoDB. Esta biblioteca estende as classes originais do Laravel, então usa exatamente os mesmos métodos nativos para executar *queries* sobre o MongoDB. Desse modo, é possível utilizar o Eloquent para definir relacionamentos entre os dados mesmo em um banco não relacional. Alternativamente, também é possível utilizar o *Eloquent* com a sintaxe própria do MongoDB para relacionar os dados por meio de referências e documentos embutidos.

Por padrão, essa biblioteca interpreta cada identificador único (*_id*) como *string*, portanto todos os identificadores e chaves estrangeiras da base legado foram convertidos em *string* para que pudessem ser encontrados nas *queries*. Essa adaptação foi realizada no processo de conversão do banco utilizando o recurso de migração SQL -> MongoDB.

No exemplo do Código 3.1, o resultado da API de buscar foto por ID deve retornar os dados da imagem juntamente com os dados do usuário, que pode ser comum ou institucional.

Código 3.1 – Método da API /photo/{id}

```
/**
 * APIPhotosController.php
 *
 * @param string $id
 * @return Response
 */
public function show($id)
{
    $photo = Photo::find($id);
```

```
    if (!$photo) {
        return \Response::json(["error" => "Image not found" ], 404);
    }

    $license  = Photo::licensePhoto($photo);

    if  (!is_null($photo->institution_id)) {
        $sender = $photo->institution;
    } else {
        $sender = $photo->user;
    }

    return \Response::json(["photo" => $photo, "license" => $license, "sender" => $sender], 200);
}
```

Fonte: Elaborado pelo autor

O método busca a imagem por meio do ID recebido na URL (*Uniform Resource Locator*). Caso a imagem tenha um *institution_id*, isso significa que ela foi publicada por um usuário institucional. Neste caso, o *Eloquent* é usado para retornar os dados da instituição pelo relacionamento 1:N entre as coleções imagem-instituição. Analogamente, se a imagem não tiver *institution_id*, significa que ela foi publicada por um usuário comum e o *Eloquent* retorna o relacionamento 1:N entre as coleções imagem-usuário. A definição desses relacionamentos é realizada na *Model Photo.php*, referenciado no Código 3.2.

Código 3.2 – Definição do relacionamento imagem-instituição por meio do Eloquent

```
/**
 * Photo.php
 *
 * @return Jenssegers\Mongodb\Relations\BelongsTo
 */
public function institution()
{
    return $this->belongsTo('App\Models\Institutions\Institution');
}

 /**
 * Photo.php
 *
 * @return Jenssegers\Mongodb\Relations\BelongsTo
 */
public function user()
{
    return $this->belongsTo('App\Models\Users\User');
}
```

Fonte: Elaborado pelo autor

A resposta da API obtida pelo *Eloquent* utilizando o MongoDB é ilustrada no Código 3.3.

Código 3.3 – Resposta da API /photo/{id} obtida pelo Eloquent utilizando MongoDB

```
{
  "photo": {
    "_id": "924",
    "aditionalImageComments": "Imagem com borda do slide; Pontos pretos;
       Azulada",
    "description": "Fachada da Residência Elza Salvatori Berquó",
    "district": "Chácara Monte Alegre",
    "imageAuthor": "FAU",
    "name": "Residência Elza Salvatori Berquó",
    "user_id": "8",
     ...
    },
     "license": [
     "by-nc-nd",
     "Não permite o uso comercial da imagem e não permite modificaões na
        imagem."
     ],
     "sender":{
     "_id": "1",
     "name": "Acervo da Biblioteca da FAUUSP",
     ...
     }
}
```

Fonte: Elaborado pelo autor

No exemplo do código 3.4, a API de comentários deve retornar os comentários de uma foto identificada pelo ID. No método referenciado, o *Eloquent* utiliza a sintaxe do MongoDB para agregar os dados dos usuários que publicaram cada comentário.

Em 3.5 há um exemplo de resposta da API de comentários com os dados dos usuários agregados.

Código 3.4 – Método da API /comments/{id}

```
    $comments = Comment::raw((function($collection) {
        return $collection->aggregate([
          [
            '$lookup' => [
              'from' => 'users',
               'localField' => 'user_id',
               'foreignField'=> '_id',
               'as' => 'user'
```

```
            ]
          ]
        ]);
    }))->where('photo_id', $id);

    return \Response::json( array_values($comments->toArray()), 200);
}
```

Fonte: Elaborado pelo autor

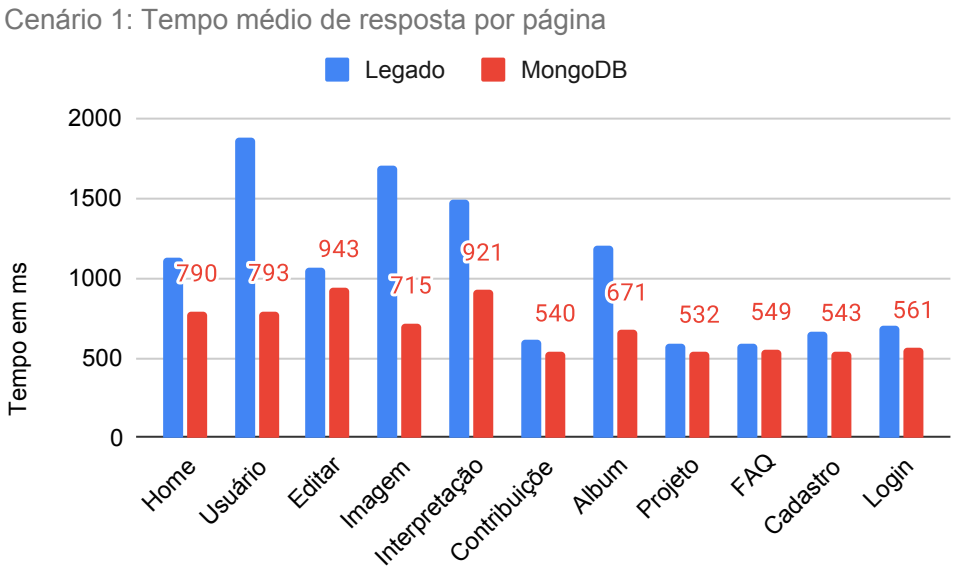
Código 3.5 – Resposta da API /comments/{id} obtida pelo Eloquent utilizando MongoDB

```
{
  "_id": "14",
  "postDate": {...}
  "text": "Acho que esta não é a fachada posterior e sim a lateral leste",
  "user_id": "74",
  "photo_id": "924",
  "user": [
    {
      "_id": "74",
      "name": "...",
      "lastName": "...",
      ...
  ]
},
{
  "_id": "6370a3f01d2fcd4d32019b72",
  "postDate": {...}
  "text": "...",
  "user_id": "6361033925868b0fc40e3252",
  "photo_id": "924",
  "user": [...]
},
...
}
```

Fonte: Elaborado pelo autor

Após todas as adaptações necessárias, os testes de carga e os testes do Lighthouse foram aplicados. A Tabela 3 e as imagens ilustradas nas Figuras 12 e 13 apresentam os resultados obtidos neste cenário.

Figura 12 – Gráfico do tempo médio de resposta por página após *update* de versões e mudança do banco para MongoDB



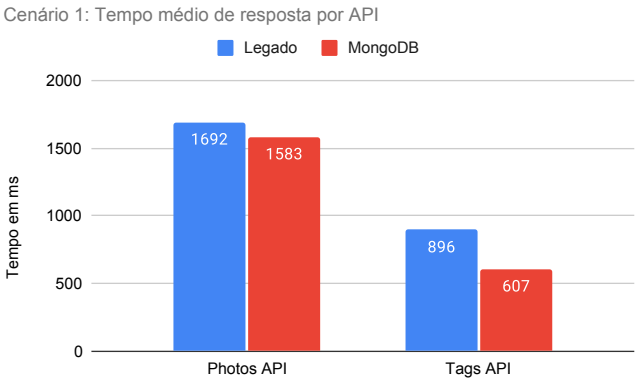
Fonte: Elaborada pela autora.

Entre as páginas testadas, o maior tempo médio de resposta observado foi na Home, com $793ms$. A página de Imagem apresentou a maior redução no tempo de resposta após as mudanças realizadas neste cenário, com uma redução de $58,02\%$ em relação à aplicação legado.

Nas páginas com poucas ou nenhuma imagem, percebe-se também um tempo de resposta mais rápido que a demais e uma redução menos significativa. O mesmo ocorre nas páginas em que não são realizadas consultas no banco de dados. Alguns exemplos de páginas com esses perfis são as páginas de Projeto (redução de $9,98\%$), FAQ (redução de $5,83\%$), Cadastro (redução de $17,85\%$) e Login (redução de $19,63\%$).

As APIs também foram testadas a fim de avaliar o tempo de resposta de consultas isoladas no banco de dados, evitando a influência na performance provocadas pelo *front-end*. Na Figura 13 é ilustrado o resultado das APIs de Imagens e *Tags*, antes e depois de aplicar as técnicas do cenário 1. As *queries* retornam 1000 documentos em cada uma das coleções.

Figura 13 – Gráfico do tempo médio de resposta por API após *update* de versões e mudança do banco para MongoDB



Fonte: Elaborada pela autora.

A API de Imagens sofreu uma redução de $6,44\%$, enquanto a API de *Tags* sofreu uma redução de $32,25\%$ em relação à aplicação legado.

Na avaliação de performance do Lighthouse as páginas com maiores mudanças na pontuação de performance foram a página de Usuário e a página de Imagem, que aumentaram em 10 pontos. Em segundo lugar estão as páginas Home e Álbum, com um aumento de 6 pontos. Três páginas tiveram variações negativas, sendo a página de Interpretação, FAQ e Login. As demais páginas variaram entre 1 e 4 pontos.

Tabela 3 – Avaliação do Lighthouse após o *update* de versões e mudança do banco para MongoDB

Página	Legado	MongoDB	Variação	Classificação
Home	44	50	6	Intermediária
Usuário	28	38	10	Ruim
Editar usuário	84	84	0	Intermediária
Imagem	38	48	10	Ruim
Interpretação	49	48	-1	Ruim
Contribuições	63	65	2	Intermediária
Album	78	84	6	Intermediária
Projeto	91	92	1	Boa
FAQ	89	87	-2	Intermediária
Cadastro	79	83	4	Intermediária
Login	90	89	-1	Intermediária

Fonte: Elaborada pela autora.

3.5 Cenário 2: Redução de requisições

A fim de reduzir as requisições no *front-end*, foi utilizado um mecanismo de minificação de *scripts* e folhas de estilo nativo do Laravel. Esse mecanismo é o Laravel Mix, que reduz para para 2 o número de arquivos de CSS e javascript que são requisitados no carregamento da página.

Também foi utilizado o VueJs como *framework* de *front-end*, que separa os elementos das páginas em componentes. Assim, ao invés de renderizar o conteúdo diretamente na *view*, a *view* apenas invoca os componentes, que por sua vez renderizam o conteúdo por meio de requisições para as APIs do *back-end*.

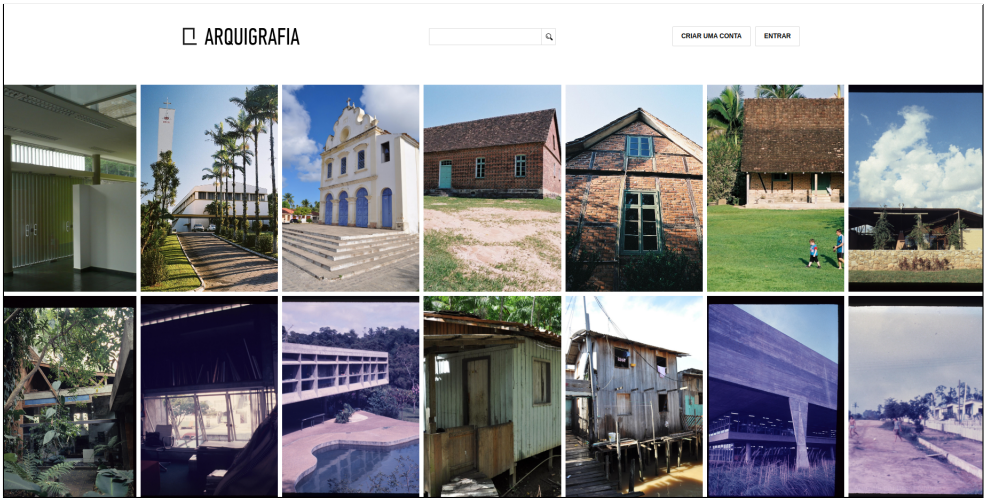
Os componentes comuns como a barra de navegação (*header*) e o rodapé (*footer*) foram reaproveitados em todas as páginas, reduzindo código repetitivo.

Em algumas páginas, como home e a página do usuário, foi implementado *lazy loading* para carregar os itens da página apenas quando requisitados.

Na home da aplicação legado, existem 300 imagens aleatórias sendo carregadas simultaneamente, onde 150 tem resolução baixa e 150 tem resolução alta. As imagens de resolução baixa são carregadas antes das de resolução alta para gerar um efeito visual de transição durante o carregamento.

Assim, a técnica de *lazy loading* foi utilizada para carregar 20 imagens e o efeito visual foi construído com CSS. Dessa forma, sempre que o usuário desliza a página, são requisitadas mais 20 imagens e assim por diante, reduzindo consideravelmente o tempo de carregamento da página. A imagem resultante da refatoração pode ser observada a partir do resultado ilustrado na Figura 14.

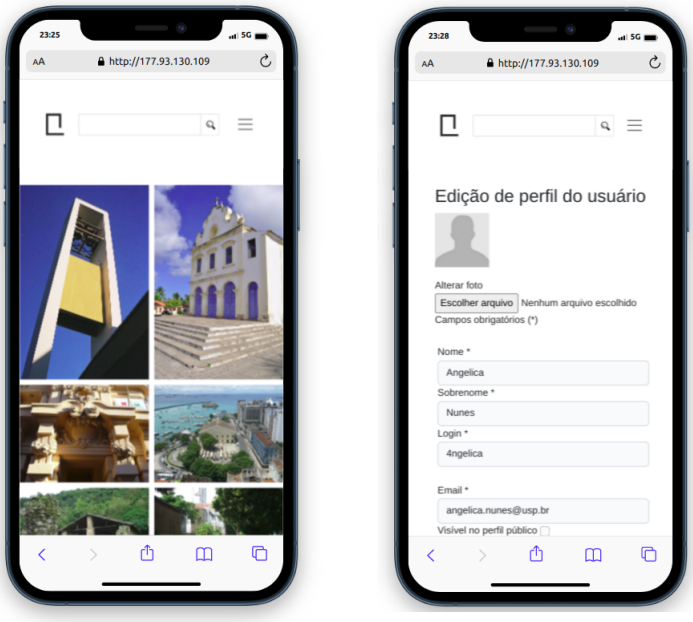
Figura 14 – Refatoração da *home* com *lazy loading*



Fonte: Elaborada pela autora.

Essa etapa da refatoração também abrangeu implementação de *design* responsivo para controlar as dimensões dos conteúdos da tela e não prejudicar a nota de LCP, e consequentemente, a nota de performance, conforme ilustrado na Figura 15.

Figura 15 – *Design* responsivo das páginas Home e Edição de usuário



Fonte: Elaborada pela autora.

Nas imagens ilustradas nas Figuras 16 e 17, é possível observar um comparativo da quantidade e parte do conteúdo das requisições na *home* antes e depois da refatoração, capturadas no *Chrome DevTools*. O total de requisições foi reduzido de 341 para 54 em um tamanho de tela que necessitou executar duas vezes a API com as 20 imagens para ser preenchida.

Nas demais páginas onde o *lazy loading* não foi implementado, também houve redução de requisições ao utilizar o mecanismo de minificação citado anteriormente entre outras adaptações e reformulações das classes pré-existentes.

Na Figura 18, são ilustrados os resultados dos testes de carga após a redução das requisições e na Tabela 4, os resultados do Lighthouse.

Como observado no cenário 1, as páginas de Projeto, FAQ, Cadastro e Login tiveram uma redução sutil no tempo médio de resposta, onde Projeto apresentou a menor redução com $0,19\%$ e Login apresentou a maior redução entre as 4, com $3,57\%$. Por outro lado, páginas como a Interpretação, Home, Usuário e Imagem apresentaram uma redução mais acentuada, porém menores do que a redução observada no cenário 1.

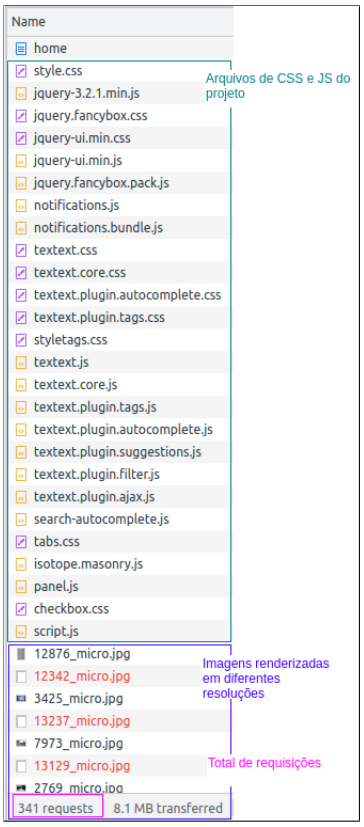


Figura 16 – Requisições na home da aplicação legado

Fonte: Elaborada pela autora.

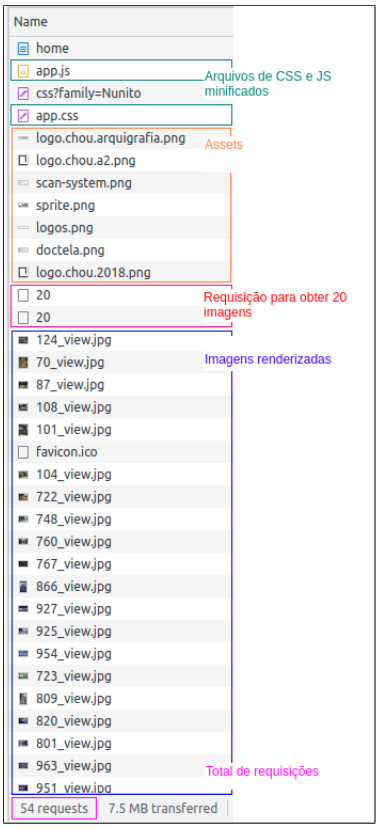
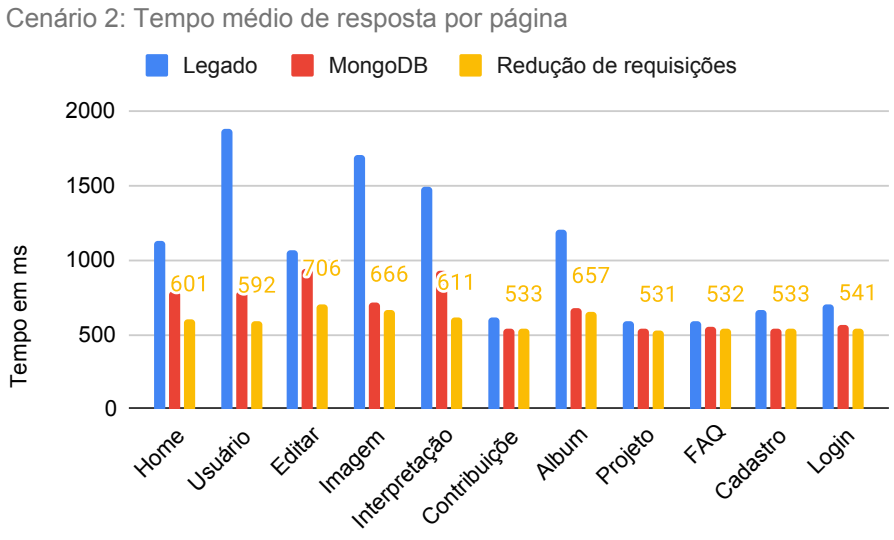


Figura 17 – Requisições na home após a refatoração

Fonte: Elaborada pela autora.

Figura 18 – Gráfico do tempo médio de resposta por página após a redução de requisições



Fonte: Elaborada pela autora.

Em uma classificação geral, a maior redução no tempo médio de resposta foi da página de Interpretação com $33,66\%$ e a menor redução foi da página de Projeto, com $0,19\%$.

Tabela 4 – Avaliação do Lighthouse após a redução de requisições

Página	MongoDB	Redução de requisições	Variação	Classificação
Home	50	77	27	Intermediária
Usuário	38	92	54	Boa
Editar usuário	84	99	15	Boa
Imagem	48	90	42	Boa
Interpretação	48	95	47	Boa
Contribuições	65	100	35	Boa
Album	84	79	-5	Intermediária
Projeto	92	97	5	Boa
FAQ	87	99	12	Boa
Cadastro	83	99	16	Boa
Login	89	100	11	Boa

Fonte: Elaborada pela autora.

A página de Interpretação mostra a interpretação de uma arquitetura registrada por um usuário e além disso, mostra a média de todas as interpretações já feitas sobre essa mesma imagem. Ainda são agregadas as imagens com médias similares, portanto, todos esses processos são resultado de *queries* grandes executadas no banco de dados e uma renderização de muitas imagens no *front-end*. Além das imagens, também ocorre a renderização do gráfico de interpretações, construído por um *script* de JS.

Os resultados do Lighthouse após a redução das requisições mostraram grandes melhorias na nota de performance, alcançando a nota máxima nas páginas de Contribuições e Login considerando *desktop*. Com exceção da página de *Album* e a *Home*, todas as as pontuações na simulação de *desktop* se mantiveram dentro da classificação boa do Lighthouse (90-100). Nenhuma nota *desktop* ficou classificada como ruim.

Como as APIs não sofrem influência do número de requisições realizadas na renderização das páginas, elas não foram submetidas ao teste de carga nesse cenário.

3.6 Cenário 3: *Cache* de dados

O *cache* dos arquivos de CSS e JS foi implementado por meio do versionamento dos arquivos que foram minificados no cenário 2, conforme ilustrado na Figura 19. Portanto, sempre que houver uma nova requisição, os arquivos são recuperados diretamente do *cache*, representados por um índice da versão (*hash*).

Figura 19 – Requisição dos arquivos de CSS e JS em memória cache

Fonte: Elaborada pela autora.

Como o conteúdo das páginas de FAQ e Projeto não são variáveis, foi viável armazenar a página por inteiro no *cache*.

No Código 3.6, os métodos *faq()* e *project()* recebem a requisição das páginas pelo usuário. No momento em que o usuário requisitar uma dessas páginas, ocorre uma verificação da existência da chave que armazena a página no Redis (*faq_page* para FAQ e *project_page* para Projeto). Caso exista, o conteúdo armazenado nessa chave é renderizado e caso contário, o conteúdo é armazenado na chave e então é renderizado.

Código 3.6 – Armazenamento das páginas FAQ e Project em cache

```
public function faq()
{
    if ( Redis::exists('faq_page') ) {
        return Redis::get('faq_page');
    } else {
        $cachedData = view('faq')->render();
        Redis::set('faq_page', $cachedData);
        return $cachedData;
    }
}

public function project()
{
    if ( Redis::exists('project_page') ) {
        return Redis::get('project_page');
    } else {
        $cachedData = view('project')->render();
        Redis::set('project_page', $cachedData);
        return $cachedData;
```

```
    }
}
```

Fonte: Elaborado pelo autor

Por outro lado, páginas como Login e Cadastro não puderam ser salvas por inteiro no cache. Como no Laravel cada sessão necessita de um token único, ao salvar essas páginas no cache, um mesmo token continua sendo utilizado, mesmo após expirar e tornando impossível realizar o login, cadastro e outras ações que necessitam de formulário.

Já para as páginas onde o conteúdo é varíavel, a solução escolhida foi salvar na memória cache a resposta das APIs utilizadas para renderizar o conteúdo exibido nas páginas.

No código 3.7 há um exemplo de cache na API de imagens. A resposta da query que busca as imagens é armazenada na instância *index_photos* durante 5 minutos, portanto durante esse intervalo de tempo o retorno da API será o conteúdo armazenado na chave, não necessitando realizar a consulta no banco novamente.

Código 3.7 – Armazenamento da API de imagens em cache

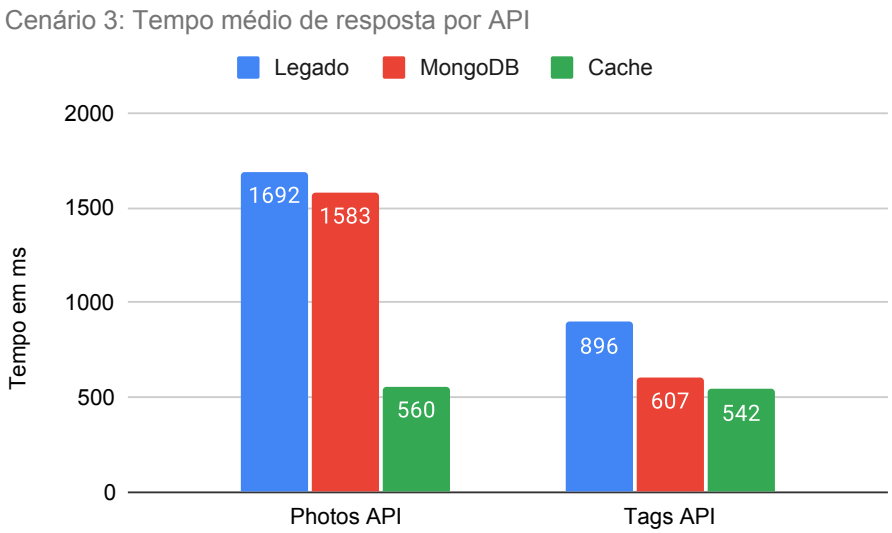
```
public function index(Request $request)
{
   $result = Cache::remember('index_photos', 60 * 5, function() {
      return Photo::where('draft', null)->get()->toArray();
   });

   return \Response::json($result);
}
```

Fonte: Elaborado pelo autor

O mesmo princípio foi aplicado nas demais APIs. Nas Figuras 20 e 21 e na Tabela 5 são ilustrados os resultados dos testes.

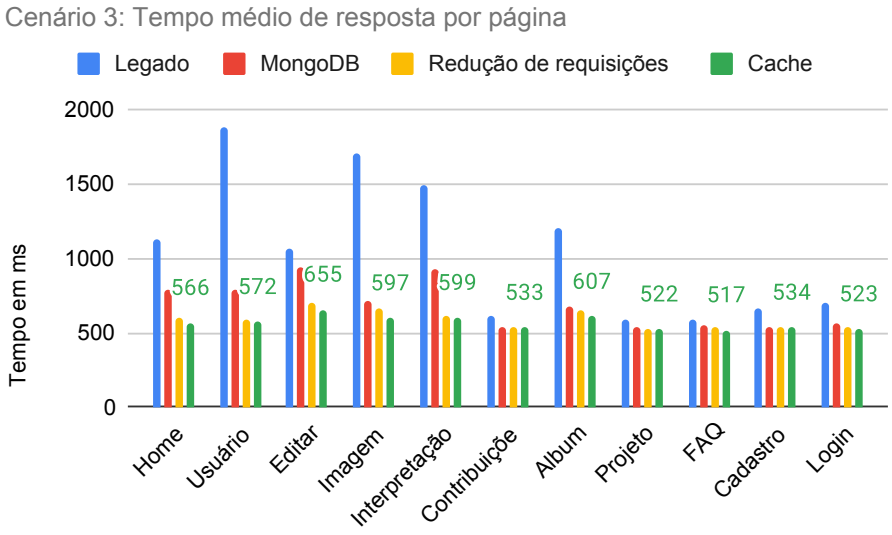
Figura 20 – Gráfico do tempo médio de resposta por API após a implementação de cache



Fonte: Elaborada pela autora.

O tempo médio de resposta da API de imagens foi reduzido significativamente em $64,6\%$, quando comparado ao cenário 2. Já a API de *Tags* sofreu uma redução de $10,7\%$, registrando um tempo de médio de resposta de $545ms$.

Figura 21 – Gráfico do tempo médio de resposta por página após a implementação de cache



Fonte: Elaborada pela autora.

As páginas do website tiveram reduções menos significativas no tempo médio de resposta em relação ao cenário 2, sendo a maior redução de $10,4\%$ na página de Imagem e $7,6\%$ na página de Álbum. Na página de Contribuições não houve variação e a página de Cadastro teve uma variação negativa de $-0,2\%$.

Tabela 5 – Avaliação do Lighthouse após implementação de cache

Página	Redução de requisições	*Cache*	Variação	Classificação
Home	77	80	3	Intermediária
Usuário	92	98	6	Boa
Editar usuário	99	97	-2	Boa
Imagem	90	94	4	Boa
Interpretação	95	98	3	Boa
Contribuições	100	100	0	Boa
Album	79	100	21	Boa
Projeto	97	99	2	Boa
FAQ	99	100	1	Boa
Cadastro	99	100	1	Boa
Login	100	100	0	Boa

Fonte: Elaborada pela autora.

Na avaliação do Lighthouse a página com o maior ganho de performance foi a página de Álbum, alcançando nota 100 com uma variação de 21 pontos desde a redução de requisições. Além dela, as páginas FAQ, Cadastro, Contribuições e Login também obtiveram nota máxima. Na página de Editar usuário houve uma variação negativa de -2 na nota de performance. Por fim, todas as notas com exceção da Home, permaneceram na classificação boa. A Home não subiu de classificação, com 80 pontos.

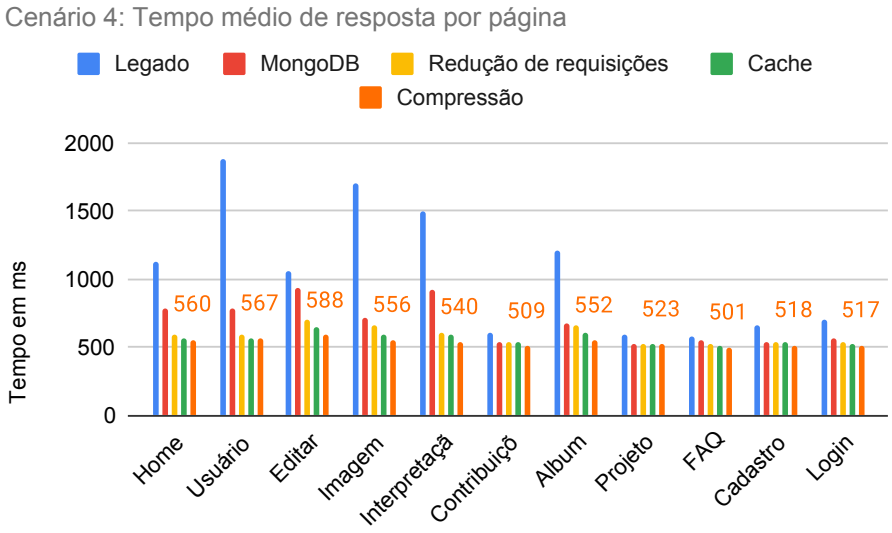
3.7 Cenário 4: Compressão de imagens (WebP)

O cenário 4 foi idealizado visando executar a compressão das imagens reduzindo o tamanho sem prejudicar a resolução e assim otimizar o carregamento. Durante o processo de desconstrução do Arquigrafia legado, foi observado que a biblioteca *intervention/image*, que implementa o armazenamento das imagens, também realiza a compressão no formato JPG.

Portanto, neste cenário as imagens foram convertidas e comprimidas em WebP utilizando o software Webp Converter, que oferece suporte para esse formato. Além da compressão das imagens herdadas da aplicação legado, a mesma biblioteca foi configurada para salvar os novos recursos no formato WebP, no processo do upload.

Os demais *assets* como ícones e logotipos também foram comprimidos em WebP. Na Figura 22 e Tabela 6 são ilustrados os resultados dos testes.

Figura 22 – Gráfico do tempo médio de resposta por página após a compressão de imagens WebP



Fonte: Elaborada pela autora.

A maior redução na velocidade média de resposta entre o cenário 3 e 4 foi de $10,2\%$ ocorreu na página de Edição do Usuário. A segunda diminuição mais significativa ocorreu na página de interpretações, com $9,8\%$.

Tabela 6 – Avaliação do Lighthouse após a compressão de imagens WebP

Página	Redução de requisições	Compressão	Variação	Classificação
Home	80	86	6	Intemediária
Usuário	98	99	1	Boa
Editar usuário	97	97	0	Boa
Imagem	94	95	1	Boa
Interpretação	98	99	1	Boa
Contribuições	100	100	0	Boa
Album	100	99	-1	Boa
Projeto	99	100	2	Boa
FAQ	100	100	1	Boa
Cadastro	100	100	0	Boa
Login	100	100	0	Boa

Fonte: Elaborada pela autora.

As páginas que tiveram as mudanças menos significativas foram Projeto, que não apresentou redução, mas um crescimento de $0,2\%$, seguida pela página de usuário, com uma diminuição de $0,9\%$. Neste cenário as APIs também não foram testadas por não envolverem elementos gráficos.

Por fim, o maior ganho em pontuação de performance do Lighthouse foi na Home, aumentando em 6 pontos sua nota de performance. A segunda página com maior ganho foi Projeto, com 2 pontos, alcançando nota máxima, conforme pode ser observado na Tabela 6. Cinco páginas alcançaram pontuação máxima, e as demais oscilaram entre -1 e 1 pontos. Todas as páginas atingiram a classificação boa, com exceção da Home, que ficou com nota 86.

3.8 Considerações Finais

Neste capítulo foi apresentado o desenvolvimento do projeto em questão. Foram apresentadas todas as modificações melhorias realizadas, bem como todas as métricas e avaliações de desempenho.

Foi comprovado que a modificação de elementos e ferramentas permitiu uma melhoria no desempenho da aplicação. Além disso pode ser observado também que as atualizações permitiram uma evolução e também melhoria no desempenho da aplicação do Arquigrafia. Sendo assim, no próximo capítulo são apresentadas as conclusões, as contribuições e os trabalhos futuros.

4 CONCLUSÃO

De modo geral, foi possível observar que o impacto de algumas técnicas são maiores em determinadas condições. Segundo o indicador de velocidade de resposta média obtido nos testes de carga, no *update* de versões e mudança do banco para MongoDB, houve um ganho de performance acentuado nas páginas de Imagem, Usuário, Interpretação e Home. As páginas de Imagem e Interpretação consomem as APIs de Imagem e de *Tags*. Já as páginas de Usuário e Home consomem, entre outras, a API de Imagens. Esses resultados estão diretamente relacionados com a velocidade de leitura dos dados no banco de dados, uma vez que as APIs apenas leem o conteúdo das coleções e devolvem a resposta. Já as páginas de Home e Usuário executam diversas *queries* no banco, a fim de renderizar na tela os conteúdos que foram requisitados pelo usuário durante a navegação.

Esses conteúdos incluem informações pessoais do usuário, seguidores, usuários seguidos, álbuns, imagens e interpretações, gerando um volume grande de dados. Enquanto isso, na Home muitas imagens são requisitadas, ou seja, a consulta não agrega um conjunto de dados, mas resulta em uma resposta muito grande.

Na Tabela 7 é ilustrado o resultado dos testes de carga por página em cada cenário.

Tabela 7 – Resultado dos testes de carga por página em cada cenário

Página	Legado	MongoDB	Redução de requisições	Cache	Compressão
Home	1124ms	790ms	601ms	566ms	560ms
Usuário	1879ms	793ms	592ms	572ms	567ms
Editar usuário	1059ms	943ms	706ms	655ms	588ms
Imagem	1703ms	715ms	666ms	597ms	556ms
Interpretação	1493ms	921ms	611ms	599ms	540ms
Contribuições	615ms	540ms	533ms	533ms	509ms
Album	1206ms	671ms	657ms	607ms	552ms
Projeto	591ms	532ms	531ms	522ms	523ms
FAQ	583ms	549ms	532ms	517ms	501ms
Cadastro	661ms	543ms	533ms	534ms	518ms
Login	698ms	561ms	541ms	523ms	517ms

Fonte: Elaborada pela autora.

Ainda no cenário 2, o Lighthouse obteve melhores resultados de performance para as páginas de Usuário, Imagem Home e Álbum. O aumento se justifica pois se enquadram no perfil de páginas que realizam consultas às APIs. No entanto, a página de Interpretação teve uma oscilação negativa de um ponto, um dos possíveis motivos é o desvio associado à execução dos testes.

Além disso, as oscilações para menos ou mais pontos não foram tão acentuadas. Isso se deve ao fato de que o tempo de resposta das páginas não depende exclusivamente de consultas no banco, mas também a renderização dos conteúdos, volume de *scripts* e *assets*. O Lighthouse também valoriza aspectos como priorização de conteúdo, o tempo que o website leva para se tornar totalmente funcional e a quantidade de *scripts* carregados, por exemplo.

Por esses motivos, no cenário de redução de requisições foi possível observar um ganho de performance muito elevado na nota do Lighthouse, visto que a refatoração compreendeu aspectos de priorização de conteúdo, redução e minificação de *scripts* e folhas de estilo.

No cenário 3, em que o *cache* foi implementado, percebe-se uma diminuição na velocidade de resposta nas APIs de Imagens e *Tags*, mais uma vez. A resposta das APIs pode ser facilmente armazenada em cache caso o conteúdo não seja tão variável, como pode ser observado na Tabela 8. Por esse motivo as páginas web não tiveram ganhos consideráveis nesse cenário, já que nem todas consomem este tipo de API.

Tabela 8 – Resultado dos testes de carga por API em cada cenário

API	Legado	MongoDB	Cache
Imagens	$1692ms$	$1583ms$	$560ms$
Tags	$896ms$	$607ms$	$542ms$

Fonte: Elaborada pela autora.

No cenário 4, em que as imagens são comprimidas em WebP, ocorre uma diminuição muito sutil no tempo de resposta, mesmo na Home, que renderiza no mínimo 20 páginas durante o arranque. Além disso, nenhuma das páginas resultou em uma velocidade média menor que $500ms$. Esse resultado pode significar que a otimização atingiu um platô dentro das condições determinadas no teste de carga e de *hardware*. Em condições de menor carga, por exemplo, com um único usuário virtual realizando as mesmas 900 requisições, a velocidade média alcançada na Home foi de $146ms$.

A oscilação dos resultados do Lighthouse justifica-se por alguns fatores, como o fato de que o mecanismo da ferramenta simula condições de conexão diferentes para *desktop* e mobile e fato do teste ter sido executado somente uma vez para cada página/cenário. A Google recomenda que o teste seja executado pelo menos 5 vezes para garantir resultados mais consistentes. Como o teste foi realizado utilizando a interface gráfica do Lighthouse e o processo é longo, seria inviável realizar uma bateria de testes para cada página a fim de calcular o erro ou desvio dessas pontuações, conforme ilustrado na Tabela 9.

Tabela 9 – Resultado da avaliação do Lighthouse por página em cada cenário

Página	Legado	MongoDB	Redução de requisições	Cache	Compressão
Home	44	50	77	80	86
Usuário	28	38	92	98	99
Editar usuário	84	84	99	97	97
Imagem	38	48	90	94	95
Interpretação	49	48	95	98	99
Contribuições	63	65	100	100	100
Album	78	84	79	100	99
Projeto	91	92	97	99	100
FAQ	89	87	99	100	100
Cadastro	79	83	99	100	100
Login	90	89	100	100	100

Fonte: Elaborada pela autora.

Em geral, o a oscilação de performance, tanto nos testes de carga quanto na avaliação do Lighthouse, pode ser justificada pelo desvio dos testes devido à condições diversas como alto consumo de memória RAM, conexão com a internet, entre outros.

É importante considerar que apenas as 11 páginas avaliadas no experimento passaram pelo processo de refatoração. Além disso, as bibliotecas obsoletas que foram removidas do projeto possivelmente impactaram nos resultados de performance, visto que a refatoração não compreendeu a substituição dos recursos implementados por elas. Um exemplo é a sessão da Home que exibe as atualizações recentes da rede de seguidores do usuário logado. Como esse recurso depende da biblioteca que implementa as notificações, ele foi removido da página, resultando menos requisições.

Uma das consequências de utilizar *cache* é não receber possíveis alterações em tempo de execução. Enquanto o *cache* persistir, a resposta de uma API, por exemplo, será a mesma, mesmo que mais recursos sejam adicionados, excluídos ou modificados.

Além disso, a maioria das páginas consideradas no experimento pertencem ao fluxo de usuário comum, portanto a área institucional não foi implementada devido à relação complexidade/tempo.

4.1 Próximos passos e trabalhos futuros

Os resultados deste estudo podem colaborar com a tomada de decisão relacionada à performance no processo de refatoração do Arquigrafia. Entre as técnicas utilizadas, a mudança do banco para o MongoDB e o armazenamento em *Cache* se destacaram para otimizar os processos do *back-end*, enquanto a redução de requisições e compressão das imagens resultou em uma otimização das interfaces gráficas entregues ao usuário.

Neste estudo, o Lighthouse foi utilizado como indicador de performance do ponto de vista do usuário, mas visto que além de classificar, a ferramenta também contribui com relatórios de otimização, seria possível implementar algumas das sugestões da ferramenta, como:

- Habilitar compressão de texto
- Eliminar bloqueios de renderização
- Servir arquivos de imagem de maneira estática via cache

Futuramente é esperado que o Arquigrafia implemente o *VRACore*, um padrão internacional de dados para a descrição de obras de cultura visual, bem como das imagens que as documentam. Essa mudança implica em grandes mudanças na forma como os dados se relacionam e também em um grande volume de dados. Uma possibilidade é implementar um banco mais rápido, como o MongoDB para atender à demanda e entregar performance para o usuário final.

Uma sugestão para recursos e funcionalidades futuras é padronizar uma arquitetura que defina os componentes gráficos separados das APIs, como foi feito no cenário 2, ou utilizando microsserviços. Isso possibilita que as APIs sejam acessadas pelo *front-end* apenas quando necessário, e também sejam passíveis de serem armazenadas em memória cache, do lado do *back-end*. Além de ganhar em performance, o projeto se torna mais escalável na medida em que as APIs podem ser utilizadas em aplicações terceiras, por exemplo, em uma aplicação mobile.

REFERÊNCIAS

GARDNER, B. S. Responsive web design: Enriching the user experience. In: **Sigma Journal: Inside the Digital Ecosystem**. [S.l.: s.n.], 2011. v. 11(1), p. 13–19.

Google Developers. **About PageSpeed Insights**: Frequently asked questions (faqs). [s.n.], 2019. Disponível em: <https://developers.google.com/speed/docs/insights/v5/about#what-device-and-network-conditions-does-lighthouse-use-to-simulate-a-page-loadhttps://web.dev/performance-scoring/>. Acesso em: 17 nov. 2022.

_____. **Lighthouse performance scoring**: How lighthouse calculates your overall performance score. [s.n.], 2019. Disponível em: <https://web.dev/performance-scoring/>. Acesso em: 17 nov. 2022.

_____. **Comparative Study**: Comparative study of webp, jpeg and jpeg 2000. [s.n.], 2022. Disponível em: <https://developers.google.com/speed/webp/docs/c_study>. Acesso em: 17 nov. 2022.

KESHAVARZ, S. Analyzing performance differences between mysql and mongodb. 02 2021.

MARANG, A. Z. Analysis of web performance optimization and its impact on user experience. In: . [S.l.: s.n.], 2018.

PHP 8.1 Benchmarks - Continuing The Nice Performance Trajectory. Phoronix, 2021. Apresenta um estudo comparativo entre a performance do PHP 8.1 e versões anteriores. Disponível em: <https://www.phoronix.com/review/php-81-benchmarks>. Acesso em: 3 out. 2022.

SANTOS, A. P. O. d. **Aplicação de práticas de usabilidade ágil em software livre**. 2012. Tese (Mestrado em Ciência da Computação) — Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012.

SANTOS, C.; LIMA, V. M. Vocabulário controlado e indexação social de imagens de arquitetura: um sistema de organização do conhecimento em ambiente colaborativo. **Revista Ibero-Americana de Ciência da Informação**, v. 13, p. 316–326, 02 2020.

Studio 3T. **Lesson 4: Comparing MongoDB vs SQL Concepts**. [s.n.], 2022. Disponível em: <https://studio3t.com/academy/topic/mongodb-vs-sql-concepts/>. Acesso em: 18 nov. 2022.

SYAMIMISAID, N. et al. Review on web performance. **Journal of Engineering and Applied Sciences**, v. 9, p. 18–23, 01 2014.

Apêndices

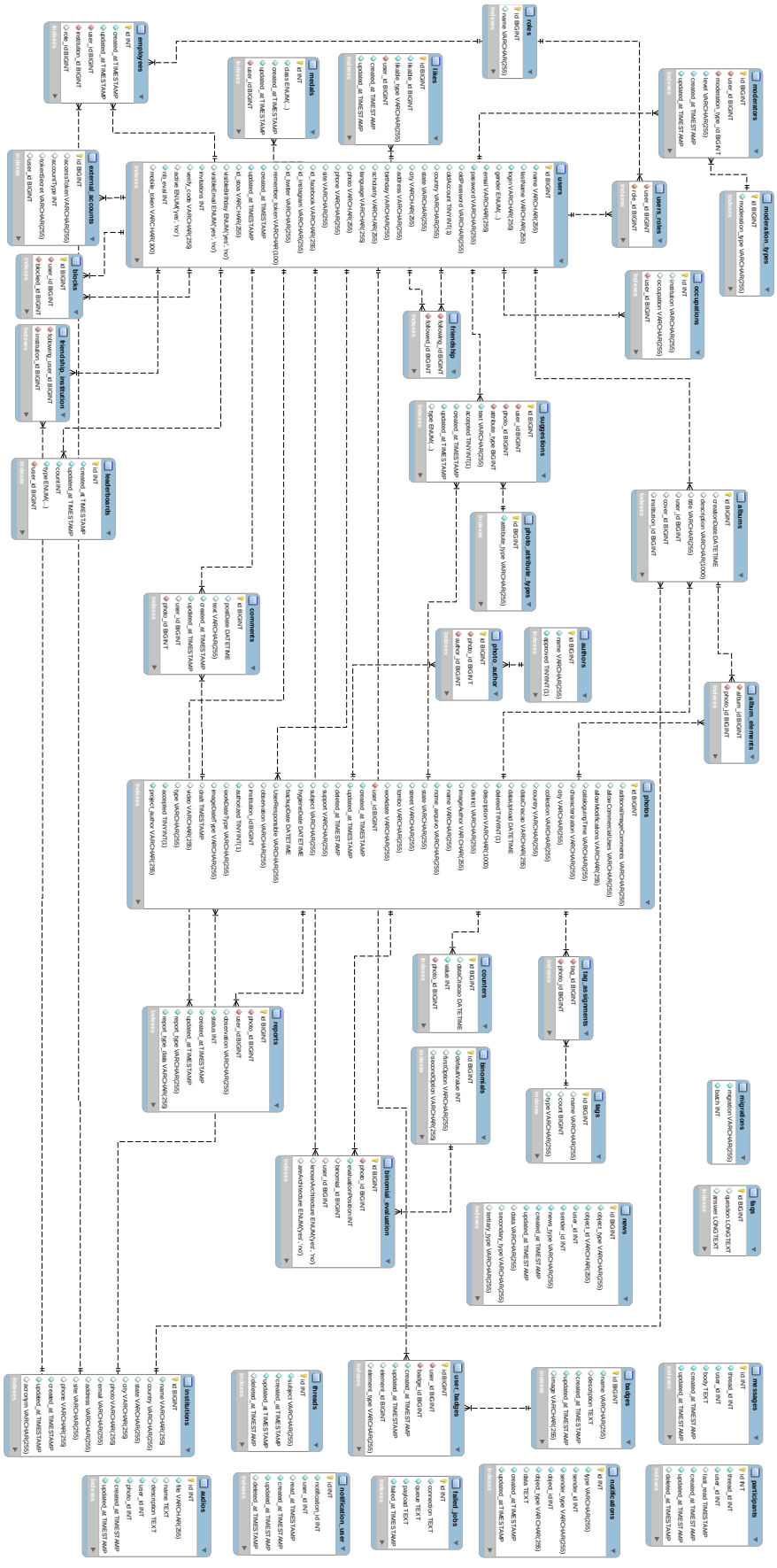
APÊNDICE A – BANCO DE DADOS

A lista abaixo descreve cada uma das tabelas do Arquigrafia Legado e seu papel dentro da estrutura do banco de dados. A Figura 23 apresenta o diagrama entidade-relacionamento do Arquigrafia Legado.

- album_elements: relação entre ID de imagens/vídeos e ID do albúm ao qual pertencem;
- albums: dados de um álbum, ID do usuário ou instituição ao qual pertence;
- audios: dados de um áudio e ID da imagem/vídeo ao qual pertence ;
- authors: Nome do autor e uma coluna de aprovado ou não;
- badges: Só tem um registro. Descrição e imagem do emblema;
- binomial_evaluation: ID da de quem interpretou, knownArchitecture e areArchitecture;
- binomials: binômios utilizados na interpretação das obras;
- blocks: tabela vazia;
- comments: relaciona Id da imagem/vídeo e id do usuário que comentou;
- counters: relaciona ID da imagem e um valor, o último registro é de fev/2015, não é mais usada;
- employees: relaciona ID de usuário, ID da instituição onde trabalha e o papel que executa (visitor, administrador...);
- external_accounts: tabela vazia;
- failed_jobs: tabela vazia;
- faqs: perguntas e respostas;
- friendship: relação entre usuário seguidor (following_id) e seguido (followed_id);
- friendship_institution: relação entre instituição e seus seguidores (following_user_id);
- institutions: instituições que fazem parte do Arquigrafia (Acervo da Biblioteca da FAUUSP, Acervo Quapá, Museu Republicano e Equipe Arquigrafia);
- leaderboards: contador de contribuições do tipo upload ou evaluation por usuário;

- likes: relaciona tipo (comentário ou imagem/vídeo) e pessoa que deu o like;
- medals: tabela vazia;
- messages: relaciona thread (conversa), usuário que enviou a mensagem e o corpo da mensagem;
- migrations: migrations;
- moderation_types: tipos de moderação (reviewer, editor, moderator, curator);
- moderators: tabela vazia;
- news: relação de mudanças que geram notificações (edição de perfil, adição de foto/vídeo), tipo do objeto que sofreu a mudança (usuário, foto) e ID do objeto. Colunas secondary_type e tertiary_type;
- notifications: relaciona tipo de notificação e os usuários envolvidos;
- notification_user: relaciona usuário que recebeu a notificação e o status da leitura ();
- occupations: relaciona usuário, instituição e a ocupação do usuário na instituição;
- participants: relação de usuários por thread (chat) e timestamp da última leitura;
- photo_attribute_types: atributos de uma imagem/video;
- photo_author: autores das imagens/vídeos;
- photos: imagens/vídeos;
- reports: relação entre imagens/vídeos reportados e o usuário que reportou;
- roles: papéis (usuário, visitante, administrador e responsible);
- suggestions: relação de sugestões, usuários envolvidos, tipo e status da sugestão;
- tag_assignments: relação entre imagens/videos e as tags relacionadas;
- tags: lista de tags adicionadas, onde os tipos são acervo ou livre;
- threads: lista de chats;
- user_badges: relaciona usuário e o emblema recebido.
- users: usuários;
- users_roles: relaciona usuário e seu papel;

Figura 23 – Diagrama Entidade-Relacionamento do Arquigrafia Legado



Fonte: Elaborada pelo autor.

APÊNDICE B – TABELAS DAS MÉTRICAS DO LIGHTHOUSE

As tabelas a seguir apresentam os resultados dos testes do Lighthouse em cada um dos cenários analisados. Aqui são detalhadas as notas obtidas para cada uma das métricas utilizadas no cálculo da nota de performance.

Tabela 10 – Avaliação desktop do Lighthouse antes de aplicar as técnicas de performance.

Página	FCP	SI	LCP	TTI	TBT	CLS	Nota de performance
Home	2,2s	2,6s	6,2s	3.0s	170ms	0,43	44
Usuário	1,9s	3,7s	13,4	5	710ms	0,001	28
Editar usuário	1,5s	1,7s	2,0s	1,5s	50ms	0	84
Imagem	2,4s	3,3s	2.9s	4,2s	650ms	0	38
Interpretação	1,8s	2,6s	2.4s	2,9s	640ms	0,001	49
Contribuições	1, 9s	2,0s	2,8s	2,2s	160ms	0,229	63
Albuns	1,6s	1,8s	2,7s	1,6s	0ms	0	78
Projeto	1,3s	1,3s	1,5s	1,4s	70ms	0	91
FAQ	1,3s	1,3s	1,8s	1,3s	50ms	0,008	89
Cadastro	1,8s	1,9s	2,3s	1,8s	60ms	0	79
Login	1,3s	1,3s	1, 7s	1,3s	40ms	0	90

Fonte: Elaborada pelo autor.

Tabela 11 – Avaliação desktop do Lighthouse no cenário 1

Página	FCP	SI	LCP	TTI	TBT	CLS	Nota de performance
Home	1,7s	2,3s	5,5s	2,4s	130ms	0,534	50
Usuário	2,0s	3,0s	5,3s	2,9s	550ms	0	38
Editar usuário	1,5s	1,8s	1,9s	1,5s	60ms	0	84
Imagem	2,0s	7,9s	2,8s	3,2s	390ms	0	48
Interpretação	1,6s	2,8s	2,3s	2,9s	760ms	0	48
Contribuições	1,6s	1,8s	2,2s	2,3s	210ms	0,299	65
Albuns	1,2s	1,5s	2,2s	1,3s	10ms	0	84
Projeto	1,2s	1,3s	1,4s	1,3s	30ms	0	92
FAQ	1,3s	1,4s	1,9s	1,8s	40ms	0	87
Cadastro	1,5s	1,8s	1,9s	2,0s	40ms	0	83
Login	1,3s	1,3s	1,8s	1,3s	30ms	0	89

Fonte: Elaborada pelo autor.

Tabela 12 – Avaliação desktop do Lighthouse no cenário 2

Página	FCP	SI	LCP	TTI	TBT	CLS	Nota de performance
Home	0,4s	1,7s	0,4s	1,2s	250ms	0,372	77
Usuário	0,4s	1,2s	1,3s	1,2s	160ms	0,021	92
Editar usuário	0,4s	1,1s	0,6s	1,1s	80ms	0	99
Imagem	0,4s	1,1s	1,4s	1,2s	130ms	0,142	90
Interpretação	0,5s	1,2s	1,4s	1,2s	80ms	0,047	95
Contribuições	0,4s	0,9s	0,6s	1,1s	70ms	0	100
Albuns	0,4s	1,2s	2,8s	1,2s	190ms	0	79
Projeto	0,4s	1,2s	1,2s	1,1s	80ms	0	97
FAQ	0,4s	1,1s	0,4s	1,0s	80ms	0	99
Cadastro	0,4s	0,9s	0,4s	1,1s	100ms	0	99
Login	0,4s	1,0s	0,6s	1,0s	60ms	0	100

Fonte: Elaborada pelo autor.

Tabela 13 – Avaliação desktop do Lighthouse no cenário 3

Página	FCP	SI	LCP	TTI	TBT	CLS	Nota de performance
Home	1,4s	3,2s	1,8s	4,6s	220ms	0,534	80
Usuário	0,6	1,1	0,6	1,2	120ms	0,007	98
Editar usuário	0,5s	0,6s	0,5s	0,8s	140ms	0	97
Imagem	0,4s	0,7s	1,2s	1,1s	140ms	0,06	94
Interpretação	0,4s	0,8s	0,9s	0,9s	90ms	0,031	98
Contribuições	0,4s	0,5s	0,5s	0,8s	70ms	0	100
Albuns	0,4s	0,9s	0,6s	0,9s	70ms	0	100
Projeto	0,4s	0,6s	0,4s	0,9s	110ms	0	99
FAQ	0,4s	0,7s	0,4s	0,8s	60ms	0	100
Cadastro	0,4s	0,5s	0,5s	0,8s	80ms	0	100
Login	0,4s	0,6s	0,6s	0,9s	80ms	0	100

Fonte: Elaborada pelo autor.

Tabela 14 – Avaliação desktop do Lighthouse no cenário 4

Página	FCP	SI	LCP	TTI	TBT	CLS	Nota de performance
Home	0,4s	0,9s	1,4s	0,7s	30ms	0,333	86
Usuário	0,4s	0,9s	0,4s	0,8s	100ms	0,019	99
Editar usuário	0,5s	0,8s	0,8s	0,8s	140ms	0	97
Imagem	0,4s	0,7s	0,9s	0,8s	120ms	0,109	95
Interpretação	0,5s	0,6s	0,8s	0,7s	80ms	0,031	99
Contribuições	0,5s	0,8s	0,5s	0,8s	50ms	0	100
Albuns	0,4s	0,8s	0,4s	0,8s	100ms	0	99
Projeto	0,4s	0,5s	0,4s	0,8s	80ms	0	100
FAQ	0,5s	0,5s	0,5s	0,7s	80ms	0	100
Cadastro	0,4s	0,6s	0,4s	0,7s	70ms	0	100
Login	0,5s	0,6s	0,5s	0,8s	50ms	0	100

Fonte: Elaborada pelo autor.