UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO DE ENGENHARIA DE COMPUTAÇÃO

NILDSON DE CASTRO PINHEIRO MELLO

PROJETO E DESENVOLVIMENTO DE SISTEMA DE COMUNICAÇÃO REMOTA ENTRE PESSOAS

SÃO LUIS
2021

NILDSON DE CASTRO PINHEIRO MELLO

PROJETO E DESENVOLVIMENTO DE SISTEMA DE COMUNICAÇÃO REMOTA ENTRE PESSOAS

Trabalho de Conclusão de Curso Apresentado ao Curso de Engenharia de Computação da Universidade Estadual do Maranhão como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Reinaldo de Jesus da Silva.

SÃO LUIS

2021

Mello, Nildson de Castro Pinheiro.

Projeto e desenvolvimento de sistema de comunicação remota entre pessoas / Nildson de Castro Pinheiro Mello. – São Luís, 2021.

64 f

TCC (Graduação) – Curso de Engenharia de Computação, Universidade Estadual do Maranhão, 2021.

Orientador: Prof. Dr. Reinaldo de Jesus da Silva.

1.Sistema web. 2.Laravel. 3.WebRTC. I.Título

CDU: 004.45

Elaborado por Giselle Frazão Tavares – CRB 13/665

NILDSON DE CASTRO PINHEIRO MELLO

PROJETO E DESENVOLVIMENTO DE SISTEMA DE COMUNICAÇÃO REMOTA ENTRE PESSOAS

Trabalho de Conclusão de Curso Apresentado ao Curso de Engenharia de Computação da Universidade Estadual do Maranhão como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Aprovado em: 19 / 04 / 2021

BANCA EXAMINADORA

Reinaldo de Jesus da Silva

Prof. Dr. Reinaldo de Jesus da Silva (Orientador)
Universidade Estadual do Maranhão

Pedro Brandão Neto

Prof. Msc. Pedro Brandão Neto (1º membro)
Universidade Estadual do Maranhão

Erick de Araújo Oliveira

Msc. Erick de Araújo Oliveira (2º membro)
Universidade Estadual do Maranhão

Este trabalho é dedicado aos meus familiares, em especial a minha mãe Nilsana, a meu pai Allan e ao meu irmão Allison pela confiança, amor e fonte de inspiração na minha vida, e ao meu primo John Ruan Oliveira de Castro (in memoriam) que me proporcionou muitas memórias boas que guardarei por toda minha vida.

À minha querida noiva, companheira e meu amor Núbia Cristina Garcia Bizerra por todo apoio, companheirismo, cumplicidade, amor e carinho na minha vida.

AGRADECIMENTOS

A Deus por me conceder saúde e forças para superar os obstáculos encontrados na jornada da minha vida.

Ao meu orientador Reinaldo de Jesus da Silva pelas oportunidades e ensinamentos que me ofereceu não somente no curso, mas na vida, desde o início do curso até o mercado de trabalho.

A toda minha família por todo suporte, apoio e colaboração durante a minha graduação. Sem vocês eu não conseguiria.

Aos meus amigos que adquiri durante o curso, por todos os momentos vividos, em especial Matheus Victor, Leonardo França, Matheus Pereira, Danilo Costa, Matheus Lacerda, Elvyson Brunno, Euclides Freire, Matheus Teixeira e “Chico” Junior.

"Por que, como imagina em sua alma, assim é"

(Provérbios 23:7)

RESUMO

Em 2020 o mundo viu-se surpreendido com o início da pandemia de covid-19 e apesar das restrições de diversas áreas, os recursos relacionados com trabalho remoto e aulas à distância foram fundamentais para a continuidade destes processos. Este trabalho de conclusão de curso tem como objetivo propôr o projeto e desenvolvimento de um sistema de comunicação remota, descrevendo as principais etapas do projeto, como a análise de requisitos, modelagem, implementação do protótipo e testes. Os termos relacionados com este trabalho e fundamentais para o projeto como conteinerização de aplicação, banco de dados relacional, falsificações de solicitação entre sites e teste de software foram conceituados e as tecnologias utilizadas como Docker, PostgreSQL, framework Laravel, PHPUnit, WebRTC e outras ferramentas utilizadas foram descritas e apresentadas. Os resultados alcançados foram satisfatórios, pois alcançaram os objetivos propostos. Por fim, propostas de melhorias foram sugeridas, como aumentar o escopo do sistema e o suporte para novas tecnologias em desenvolvimento, como por exemplo a holoportação.

Palavras-chave: Sistema Web. Laravel. WebRTC.

ABSTRACT

In 2020, the world was surprised by the beginning of the covid-19 pandemic and despite the restrictions in several areas, resources related to remote work and distance classes were essential for the continuation of these processes. This course conclusion work aims to propose the design and development of a remote communication system, describing the main stages of the project, such as requirements analysis, modeling, prototype implementation and tests. The terms related to this work and fundamental to the project such as application containerization, relational database, request forgery between sites and software testing were conceptualized and the technologies used such as Docker, PostgreSQL, Laravel framework, PHPUnit, WebRTC and other tools used were described and presented. The results achieved were satisfactory, as they achieved the proposed objectives. Finally, proposals for improvements were suggested, such as increasing the scope of the system and support for new technologies under development, such as holoporation.

Keywords: Web System. Laravel. WebRTC.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSRF	*Cross-site Request Forgery*
DCL	*Data Control Language*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
DTL	*Data Transaction Language*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
MER	Modelo Entidade Relacionamento
PC	*Personal Computer*
RTC	*Real-Time Communications*
SGBD	Sistema Gerenciador de Banco de Dados
SO	Sistema Operacional
SQL	*Structured Query Language*
TLS	*Transport Layer Security*
VM	*Virtual Machine*

.

SUMÁRIO

1 INTRODUÇÃO

A evolução tecnológica observada no último século permitiu e ainda permite mudanças significativas nos mais diversos aspectos da sociedade. Transporte, saúde, educação, entretenimento e comunicação são exemplos de áreas que progrediram consideravelmente com a evolução dos recursos tecnológicos.

Presente nas mais diversas áreas e contando com os mais variados softwares, o computador (juntamente com os dispositivos computacionais) é um dos recursos com maior empregabilidade atualmente.

Em meados da década de 1970 surgiram os primeiros computadores pessoais, de pequeno porte e destinados ao uso pessoal. Com o passar das décadas estes computadores evoluíram cada vez mais, tanto em poder computacional como em funcionalidades.

O seu barateamento, dentre outros fatores, possibilitou a globalização do uso do computador e dos demais dispositivos computacionais como *smartphones, tablets,* consoles, entre outros.

Acompanhando este progresso tem-se o desenvolvimento de novos aplicativos e sistemas, com constantes atualizações e funcionalidades novas, otimizando processos e proporcionando novas realidades tecnológicas.

Aplicativos de comunicação, plataformas de *streaming* e redes sociais são bons exemplos de tecnologias das últimas décadas que se popularizaram exponencialmente e hoje apresentam milhões de usuários.

Em meio a esse contexto de constante aperfeiçoamento percebe-se que novos métodos, processos, procedimentos, conceitos, linguagens e técnicas continuam surgindo e sendo aplicados e difundidos na sociedade, em alguns casos por necessidade e em outros pelo desejo de criar novas possibilidades para as pessoas.

Em 2020 o mundo viu-se surpreendido com a pandemia de covid-19 (vírus popularmente conhecido como coronavírus) e a população sofreu medidas restritivas de acordo com as políticas de cada país. No Brasil as medidas restritivas impediram diversas pessoas de trabalhar e/ou frequentar suas escolas e universidades. Dessa forma, conceitos como *home office* e ensino à distância (já presentes no cotidiano de diversos brasileiros) tornaram-se mais conhecidos.

Sistemas de vídeo conferência, reuniões e chamadas remotas tiveram milhares de novos usuários ativos em busca de suas funcionalidades. Em meio a este contexto de necessidades por recursos de chamadas remotas, será apresentado aqui o projeto de um sistema com interface

amigável, objetiva e de fácil entendimento, para comunicação remota por mensagens e chamadas, descrevendo todas as etapas do projeto.

1.1 Objetivos

Nesta seção estão os objetivos deste trabalho, divididos em Objetivo Geral e Objetivo Específico.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é propor o projeto e desenvolvimento de um sistema web que possibilite e dê suporte para a comunicação remota entre pessoas.

1.1.2 Objetivos Específicos

Para que o objetivo geral possa ser atingido, são propostos os seguintes objetivos específicos:

- Investigar o uso de sistemas web como ferramenta de comunicação;
- Apresentar a arquitetura do sistema proposto;
- Construir um protótipo do sistema proposto;
- Realizar testes aplicando métricas de qualidade de software.

1.2 Metodologia

Este trabalho pode ser classificado como de abordagem qualitativa, pois seu desenvolvimento e resultados são focados no caráter subjetivo do contexto do projeto. Será realizada a coleta e análise dos requisitos, e então a modelagem. Será feito o desenvolvimento do sistema com as tecnologias conceituadas no capítulo 2: framework Laravel, Docker, WebRTC, PostgreSQL, dentre outros. A etapa de testes será realizada com o PHPUnit e observada por meio de gráficos de cobertura.

1.3 Estrutura de Trabalho

O documento está estruturado da seguinte: no capítulo 2 é apresentada a fundamentação teórica dos conceitos, métodos e tecnologias utilizadas no trabalho. No capítulo 3 é apresentada

uma investigação sobre trabalhos relacionados com sistemas web para comunicação. No capítulo 4 é apresentada a arquitetura proposta. No capítulo 5 descreve-se a implementação do protótipo a cobertura dos testes realizados. O capítulo 6 apresenta as considerações finais e as propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para atender as expectativas e os objetivos do trabalho alcançando assim um trabalho satisfatório, faz-se necessário estabelecer uma base teórica já difundida e estudada, relacionada ao tema da pesquisa. Desta forma, a fundamentação teórica do trabalho foi estruturada visando esclarecer diversos conceitos relacionados e servindo de base para o desenvolvimento.

2.1 Conceitos de Software e Sistemas de Informação

Os softwares atualmente estão presentes nas mais diversas áreas, direta ou indiretamente, dando suporte à vida, presente no comércio, na cultura e agregados no cotidiano (PRESSMAN, 2011). Assim definiu PRESSMAN acerca de software:

> Software de computador é o produto que profissionais de software desenvolvem e ao qual dão suporte no longo prazo. Abrange programas executáveis em um computador de qualquer porte ou arquitetura, conteúdos (apresentados à medida que os programas são executados), informações descritivas tanto na forma impressa (hard copy) como na virtual, abrangendo praticamente qualquer mídia eletrônica. (PRESSMAN, 2011)

Já PAULA FILHO (2003) correlacionou software e sistemas de informação da seguinte forma:

> O software é a parte programável de um sistema de informática. Ele é um elemento central: realiza estruturas complexas e flexíveis que trazem funções, utilidade e valor ao sistema. Mas outros componentes são indispensáveis: as plataformas de hardware, os recursos de comunicação de informação, os documentos de diversas naturezas, as bases de dados e até os procedimentos manuais que se integram aos automatizados. (PAULA FILHO, 2003)

Assim o software é o chamado artefato (também conhecido como executável) e o sistema de informação é o conjunto destes artefatos somados e relacionados, comunicando-se para atingir o objetivo em comum (PAULA FILHO, 2003).

2.2 Engenharia e Processo de Software

Um bom desenvolvimento de software e/ou sistema requer estudo, planejamento e uma organização prévia. Assim analisou PAULA FILHO (2003) acerca da engenharia de software:

> Um dos problemas básicos da engenharia de software é o levantamento e documentação dos requisitos dos produtos de software. Quando este levantamento é bem feito, os requisitos implícitos são minimizados. Quando a documentação é bem feita, os requisitos documentados têm maiores chances de serem corretamente entendidos pelos desenvolvedores. (PAULA FILHO, 2003)

Para PRESSMAN (2011) engenharia de software engloba processos, uma gama de métodos e práticas, assim como diversas tecnologias que possibilitam o desenvolvimento de um produto de qualidade.

Basicamente, dentro das etapas de uma boa engenharia de software está uma correta coleta de requisitos, organização de prazos, arquitetar o design e modelos utilizados, o planejamento de testes e demais ciclos no projeto. Todo o fluxo e planejamento interferem na qualidade e sucesso do projeto.

PRESSMAN (2011, p 566) defende que o espectro de um gerenciamento efetivo, organizado e focado considerar os chamados 4 Ps: pessoas, produto, processo e projeto.

Estes pontos podem ser considerados da seguinte forma:

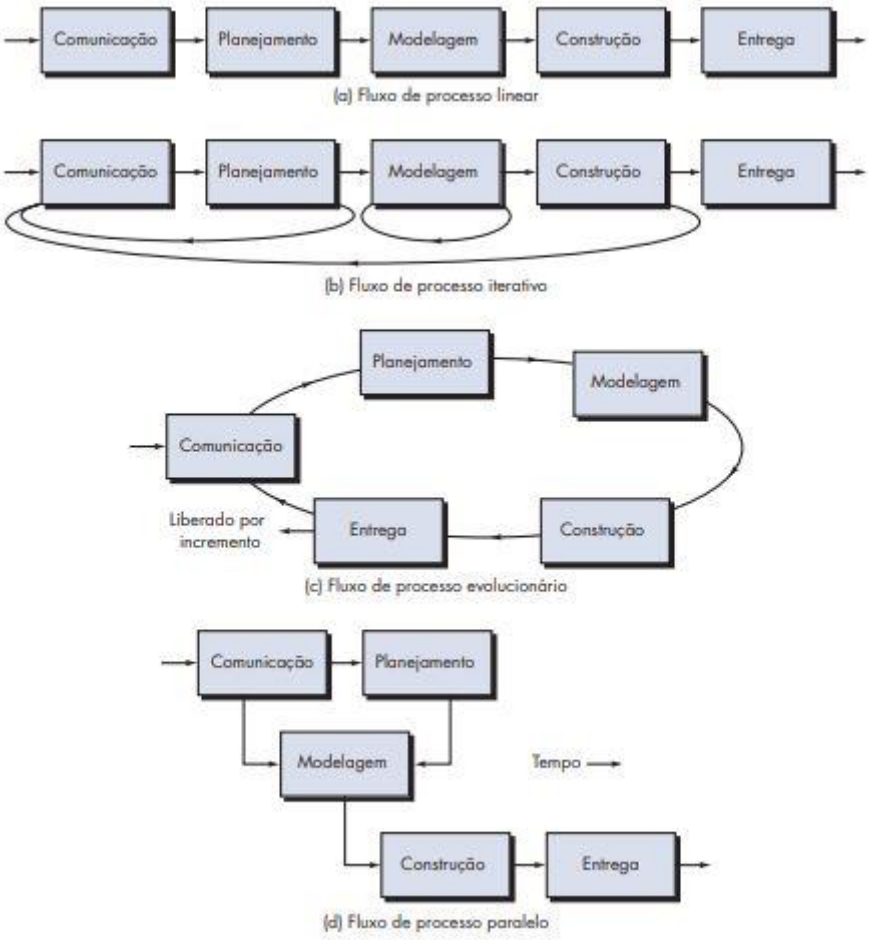
- Pessoal: consiste na gestão dos recursos humanos, indispensável para nortear e organizar os colaboradores do projeto. A correta administração do pessoal é fundamental para alcançar os objetivos;
- Produto: consiste na definição do escopo do produto, observando suas restrições técnicas e possíveis soluções e alternativas;
- Processo: consiste na metodologia e no plano do projeto, definindo pontos de controle, artefatos de software, controle de qualidade e demais métricas que ocorrem durante todo o projeto, desde a coleta e análise de requisitos até a construção do produto;
- Projeto: consiste nos detalhes intrínsecos do projeto como cronograma, custos e objetivos. A qualidade e monitoramento também estão relacionados.

Para (SOMMERVILLE, 2011) a engenharia de software não deve ser entendida como somente a criação de um programa, mas toda a configuração e análise atrelada ao para fazer esse produto, que na verdade em geral não é apenas um arquivo, mas uma série de códigos separados que interagem entre si através de arquivos de configuração e padrões estabelecidos.

Existem diversas metodologias e modelos que podem ser escolhidos para o processo de software, cada um com seu fluxo específico. O fluxo do processo pode ser entendido como a organização das etapas e tarefas em relação ao tempo (PRESSMAN 2011, p 54).

Segundo SOMMERVILLE (2011) os processos são complexos e por serem elaborados por pessoas podem apresentar um grau de subjetividade, não tendo um processo de software ideal, mas modelos com vantagens e desvantagens como o modelo em cascata que apresenta um fluxo linear e considera as atividades de formas separadas em: especificação de requisitos, projeto, implementação e teste.

Figura 1: Tipos de Fluxo de Processo.



Fonte: PRESSMAN (2011), p 54.

2.3 Conceitos e Tecnologias Importantes de Banco de Dados Relacional

Segundo ELMASRI et al. (2005) um banco de dados é uma coleção de dados com significados implícitos, como nomes, datas, valores. Com o devido relacionamento destes dados, de acordo com os índices utilizados, o banco de dados pode ser observado como um conjunto de tabelas ligadas apresentando assim a característica de ser relacional.

2.3.1 Modelo Entidade Relacionamento

A modelagem pode ser entendida como a representação (por meio de diagramas) de determinado contexto que será trabalhado no desenvolvimento. Diversas representações podem ser elaboradas para um mesmo projeto e funcionar, no entanto, a modelagem correta e concisa dos dados é diretamente proporcional ao sucesso do projeto (BAZZI, 2013).

A modelagem entidade relacionamento é uma representação conceitual bastante popular de uma determinada base de dados e da relação entre os mesmos. Neste modelo os dados são representados por meio de entidades, atributos e seus relacionamentos. A entidade é tudo aquilo que necessita ser guardado de acordo com o contexto ou problema trabalhado. Esta entidade deve ter suas características (chamadas de atributos) e relacionam-se entre si (relacionamento), podendo ser representada por meio do modelo entidade relacionamento (MER) (ELMASRI et al., 2005).

Figura 2: Representação de Entidades, Atributos e Relacionamento.

Fonte: HEUSER (2009).

2.3.2 Modelo Físico

O modelo físico é a representação técnica da estrutura de um banco de dados considerando detalhes internos das informações relacionadas com a relação entre as tabelas ou lógicas específicas. Esse tipo de modelo não apresenta um padrão (variando de tecnologia para tecnologia) e é interessante para a equipe de desenvolvimento ou responsáveis pelo SGBD, mas não para o usuário final (ELMASRI et al., 2005, p 20).

2.3.3 Linguagem SQL

A linguagem SQL (Structured Query Language) é a mais conhecida internacionalmente para a maioria dos bancos de dados. (BAZZI, 2013) Os comandos são divididos em categorias de acordo com suas funcionalidades: DDL, DML, DCL e DTL.

A Linguagem de Definição de Dados (ou DDL, *data definition language*) consiste-se basicamente em comandos SQL com características de construção dos modelos de dados.

Tabela 1: Comandos de Definição de Dados

COMANDO	DESCRIÇÃO
CREATE	Aplicado na criação de bases de dados (CREATE DATABASE), de tabelas (CREATE TABLE), índices (CREATE INDEX), entre outros;
ALTER	Aplicado na alteração da estrutura de uma tabela (ALTER TABLE) buscando, por exemplo, adicionar, modificar ou remover colunas;
DROP	Utilizado para apagar tabelas (DROP TABLE), apagar bases de dados (DROP DATABASE), entre outros.

Fonte: Elaborado pelo autor (2021).

A Linguagem de Manipulação de Dados (ou DML, *data manipulation language*) consiste-se basicamente de comandos SQL com características de manipulação dos registros das tabelas.

Tabela 2: Comandos de Manipulação de Dados

COMANDO	DESCRIÇÃO
INSERT	Aplicado para inserir registros em uma tabela (INSERT INTO);
UPDATE	Aplicado para atualizar registros de tabelas, normalmente acompanhado de comandos atribuição (SET) e filtragem (WHERE);
DELETE	Utilizado para deletar registros de tabelas, normalmente acompanhado de comandos de atribuição (SET) e filtragem (WHERE);
SELECT	Usado para visualização de registros das tabelas, acompanhado normalmente de comandos de especificação (FROM), de combinação de tabelas (LEFT JOIN, INNER JOIN, RIGHT JOIN) e de filtragem.

Fonte: Elaborado pelo autor (2021).

Os comandos da Linguagem de Controle de Dados (ou DCL, *data control language*) são utilizados para configurar as permissões de acesso ao banco de dados.

Tabela 3: Comandos de Controle de Dados

COMANDO	**DESCRIÇÃO**
GRANT	Comando para dar permissões específicas;
DENY	Comando para negar permissões;
REVOKE	Comando para remover comandos de permissões que foram previamente concedidos.

Fonte: Elaborado pelo autor (2021).

Os comandos da Linguagem de Transação de Dados (ou DTL, *data transaction language*) são utilizados para permanecer os dados do banco de dados e manipular os resultados dos comandos utilizados.

Tabela 4: Comandos de Transação de Dados

COMANDO	**DESCRIÇÃO**
COMMIT	Aplicado para gravar toda a transação no disco;
ROLLBACK	Aplicado para retornar o banco para um estado anterior ou estado não comitado.

Fonte: Elaborado pelo autor (2021).

2.3.4 Sistemas gerenciadores de Banco de Dados

Sistemas Gerenciadores de Banco de Dados, ou SGBD, como o próprio nome diz são sistemas para a administração de um banco de dados. Apresentam recursos facilitadores para a definição e manipulação dos dados armazenados no banco de dados e normalmente possuem uma interface gráfica.

2.3.5 PostgreSQL

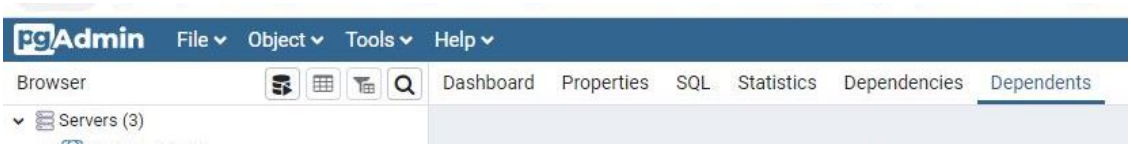
O PostgreSQL é um banco de dados relacional *Open Source* muito popular entre os desenvolvedores. Com mais de 30 anos ativo conta com uma forte comunidade mantendo a reputação de robusto, confiável e de bom desempenho. Atualmente está em sua 13ª versão (POSTGRESQL, 2021).

Foi o banco de dados escolhido para este trabalho por ser uma tecnologia de código aberto com bom desempenho e por apresentar uma boa integração com as demais tecnologias escolhidas, além de possuir uma documentação bem estruturada.

2.3.6 PgAdmin

O PgAdmin é o SGBD do PostgreSQL, uma ferramenta gráfica criada pela equipe do PostgreSQL contendo diversos recursos e funcionalidades em seu menu, como otimizações para criação de novos bancos, tabelas e conexões remotas; *query tool* para consultas SQL visualizada graficamente e mantendo histórico dos comandos; componentes gráficos para visualizar estresse do banco; facilidades para a manipulação de registros (inserções, atualizações, visualizações); *backup* e restauração, dentre outros (PGADMIN, 2021).

Figura 3: Parte da Interface do PgAdmin.



Fonte: Elaborado pelo autor (2021).

2.4 Introdução à Sistemas Web

Os sistemas para web apresentam diferenças em relação aos sistemas utilizados em outros meios como por exemplo a não necessidade de instalação, já que são acessados por navegadores, no entanto, essa característica significa também um estudo maior sobre os suportes dos navegadores para o sistema (BADALOTTI, 2014).

Normalmente um sistema web depende de um servidor web, das requisições, dos protocolos e da resposta. Outros pontos também são importantes como a segurança do sistema, criptografia das informações de login (caso o sistema tenha esta funcionalidade).

Dessa forma, basicamente temos o cliente de um lado, o servidor de outro e no meio os protocolos permitindo a comunicação, o transporte e segurança das requisições e respostas (NOLETO, 2020). Assim o servidor recebe as requisições do cliente, faz os processos lógicos codificados e devolve a resposta, essa comunicação é possível pelos protocolos que são padronizados internacionalmente.

2.5 Framework Laravel

Laravel é um framework de sistemas web com uma sintaxe objetiva e elegante, contendo um ecossistema próprio, extenso e altamente aplicável, além de possuir uma comunidade de desenvolvedores bastante ativa. Passando por constantes melhorias e atualizações, está atualmente na versão 8 (TAYLOR OTWELL, 2021).

É um framework para PHP, uma das linguagens mais utilizadas no mundo. O Laravel foi escolhido para este projeto por apresentar uma boa curva de aprendizagem e por possuir uma documentação objetiva e direta, além de um bom desempenho e mostrar-se compatível com as outras tecnologias estudadas.

2.5.1 Arquitetura Model-View-Controller

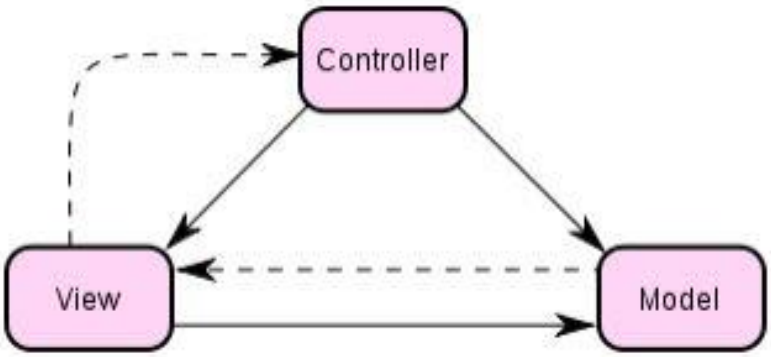
Para Luciano e Alves (2017) o padrão de arquitetura MVC (*Model-View-Controller*) apresenta uma dinâmica simples, designando um papel especifico para cada camada. Com este padrão as regras de negócio separam-se da apresentação e do gerenciamento do fluxo da aplicação permitindo uma eficiente reutilização de código.

As responsabilidades de cada camada podem ser observadas da seguinte forma:

- *Model* (ou modelo) é a camada responsável pelas regras de negócio, gerenciando o comportamento e restrição dos dados. Em uma aplicação padrão nesta camada são definidas as validações dos dados, relacionamento lógico e as especificações inerentes para cada sistema.
- *View* (ou visualização/visão) é a camada responsável pela apresentação e exibição dos dados. Com essa separação do restante da lógica do sistema a exibição dos dados fica mais simples e de fácil manutenção.
- *Controller* (ou controlador) é a camada responsável pelo gerenciamento do fluxo do sistema. É esta camada que manipula os dados e faz as chamadas das *Views* e envia as ações para o *Model*.

A figura 4 ilustra o relacionamento das 3 camadas. O controlador gerencia o fluxo tanto da visualização quanto do modelo, enviando instruções de, por exemplo, atualização (para o *model*) ou de alteração da visão atual (para a *view*) solicitadas pelo usuário. O modelo, por sua vez, fornece as informações necessárias para serem exibidas pela visão e gerenciadas pelo controlador. Já a visão é a representação dos dados presentes no modelo exibidos de acordo com os parâmetros especificados pelo controlador.

Figura 4: Arquitetura MVC.



Fonte: LUCIANO E ALVES (2017)

2.5.2 Artisan

Artisan é a interface por meio de linhas de comando do framework Laravel, contendo diversos comandos para automatizar diversos processos de software (TAYLOR OTWELL, 2021). O Artisan permite até a criação de novos comandos, por exemplo, para automatizar o envio de e-mail ou criar rotinas específicas.

Por meio do comando '*php artisan list'* é possível obter uma lista dos comandos artisan habilitados, na seguinte tabela são definidos os principais comandos.

Tabela 5: Comandos Artisan.

COMANDO	**DESCRIÇÃO**
php artisan list	Lista os comandos habilitados;
php artisan migrate	Executa as migrações criadas (pode ser complementada com outros comandos);
php artisan serve	Executa o aplicativo no servidor de desenvolvimento PHP;

php artisan cache	Limpa o cache do complemento informado após o comando;
php artisan config	Grava ou Limpa a configuração atual no cache (dependendo do complemento informado);
php artisan db	Comandos relacionados com o banco de dados (*db:seed* executa o arquivo para popular o banco, *db:wipe* dropa todas as tabelas);
php artisan key:generate	Seta uma chave para o aplicativo, está chave é fundamental para o correto funcionamento do sistema e para a segurança das conexões;
php artisan make	Comandos para criar novos arquivos no projeto (dependendo do complemento). *make:controller* cria uma nova classe controladora, *make:model* cria uma nova classe modelo, *make:test* cria uma nova classe de teste, *make:migration* cria um novo arquivo de migração, etc..

Fonte: Elaborado pelo autor (2021).

2.5.3 Proteção CSRF

As falsificações de solicitação entre sites ou CSRF (do inglês, *cross-site request forgeries*) consistem na execução de comandos não autorizados através de um usuário autenticado.

Esta vulnerabilidade é melhor entendida através de um exemplo: supondo que um sistema possui a rota “usuário/email” que aceita uma solicitação POST para alterar o e-mail do usuário logado (TAYLOR OTWELL, 2021). O intruso malicioso pode criar um formulário direcionado para essa rota, mas contendo o próprio e-mail do intruso, como observado na figura 5.

Figura 5: Formulário malicioso.

```
<form action="https://your-application.com/user/email" method="POST">
    <input type="email" value="malicious-email@example.com">
</form>

<script>
    document.forms[0].submit();
</script>
```

Fonte: TAYLOR OTWELL (2021).

Para contornar essa vulnerabilidade o Laravel cria um token CSRF para cada sessão de usuário ativa, inclusive em solicitações que não apresentam usuários logados. Formulários HTML também fazem uso dessa proteção (figura 6). Este token não pode ser acessado por um intruso pois tem validade e muda a cada atualização da página.

Figura 6: Formulário com proteção CSRF.

```
<form method="POST" action="/profile">
    @csrf

    <!-- Equivalent to... -->
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

Fonte: TAYLOR OTWELL (2021).

2.5.4 Migrations

Com estrutura e sintaxe própria, as migrações (*migrations*) são basicamente controladoras para as versões do banco de dados. É possível definir a estrutura do banco, adicionar e remover tabelas, modificar campos em qualquer etapa do projeto sem grandes dificuldades (TAYLOR OTWELL, 2021).

As migrations automatizam os comandos DDL e DTL por meio de uma sintaxe mais simples e enxuta.

A correta implementação das migrações proporciona uma série de vantagens como os *rollbacks* que se tornam mais automatizados e a facilidade de atualizações em produção.

Com as migrations criadas, é possível gerar todo o esquema do banco de dados com apenas um comando. A estrutura padrão de uma migração pode ser observada na figura 7.

Figura 7: Estrutura padrão de uma migração.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateFlightsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('flights');
    }
}
```

Fonte: Adaptado de TAYLOR OTWELL (2021).

2.5.4.1 Seeding

O Laravel possui também um método de popular o banco de dados, o chamado *seed*. Esta função é bastante útil ao subir o banco com dados iniciais ou que devem ser preenchidos inicialmente, tudo ocorre de forma automatizada e rápida.

Figura 8: Exemplo de Classe Seed.

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeders.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name' => Str::random(10),
            'email' => Str::random(10).'@gmail.com',
            'password' => Hash::make('password'),
        ]);
    }
}
```

Fonte: Adaptado de TAYLOR OTWELL (2021).

2.5.5 Livewire

O Livewire é um framework full-stack para aplicações Laravel, que possibilita a criação de interfaces dinâmicas de forma muito mais simples mantendo a otimização do desenvolvimento com Laravel (LIVEWIRE, 2021).

O funcionamento do Livewire é bem intuitivo, basicamente quando ocorre uma interação com um componente do framework é realizada uma solicitação AJAX ao servidor com os dados atualizados. O servidor por sua vez renderiza o componente e envia a resposta com o novo HTML. Por fim, o Livewire altera o DOM de forma inteligente respeitando as regras organizadas no componente e as mudanças realizadas (LIVEWIRE, 2021).

Figura 9: Componente de Pesquisa em Tempo Real com Livewire.

App \ Http \ Livewire \ SearchUsers.php

```php
use Livewire\Component;

class SearchUsers extends Component
{
    public $search = '';

    public function render()
    {
        return view('livewire.search-users', [
            'users' => User::where('username', $this->search)->get(),
        ]);
    }
}
```

resources / views / livewire / search-users.blade.php

```blade
<div>
    <input wire:model="search" type="text" placeholder="Search users..."/>

    <ul>
        @foreach($users as $user)
            <li>{{ $user->username }}</li>
        @endforeach
    </ul>
</div>
```

Fonte: LIVEWIRE (2021).

2.5.6 PHPUnit

O Laravel fornece suporte para testes por meio do PHPUnit, o qual possibilita a criação, configuração e execução dos testes de unidade, sendo baseado na arquitetura XUnit, sendo construído com foco na a linguagem PHP (BERGMANN, 2020).

O Artisan fornece uma série de comandos para otimizar o processo de criação e execução dos testes.

Tabela 6: Comandos para Testes com PHPUnit.

COMANDO	DESCRIÇÃO
php artisan make:test ExemploTest	Cria um novo caso de teste vazio;
php artisan test	Executa os testes em sequência;
php artisan test --testsuite=Feature --stop-on-failure	Executa os testes em sequência, mas encerra caso algum teste falhe;
php artisan test --parallel	Executa os testes em paralelo (depende da quantidade de núcleos do processador);

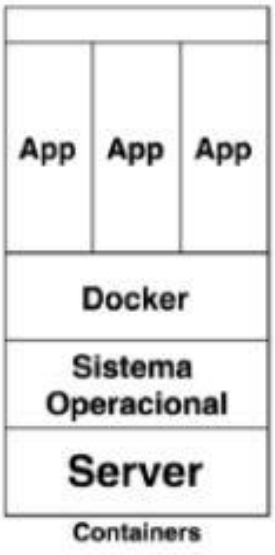
Fonte: Elaborado pelo autor (2021).

2.6 Introdução à Container

Container, no desenvolvimento de sistemas, pode ser entendido como uma unidade que agrupa o código e suas dependências isolando-os de outros elementos e permitindo um melhor desempenho. Tudo isso através da virtualização do sistema operacional no lugar da virtualização do *hardware* (DOCKER, 2021).

Para Vitalino (2016) containers são mais leves que máquinas virtuais, além de serem menos poluídos pois contém apenas os recursos necessários para o desenvolvimento e/ou funcionamento da aplicação, além de compartilhar o kernel do sistema operacional da máquina (mesmo se for VM). Um contêiner bastante utilizado atualmente é o Docker.

Figura 10: Representação de um Container.

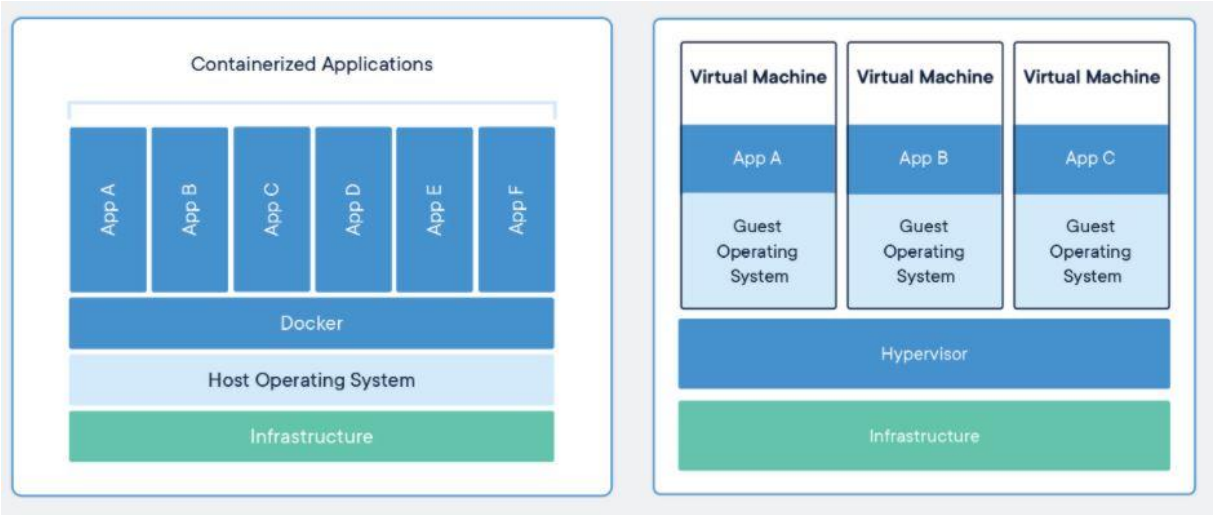


Fonte: VITALINO (2016).

2.6.1 Diferença Entre Máquina Virtual e Container

VITALINO (2016) caracteriza máquinas virtuais como sendo mais pesadas que *containers*, por emular um novo sistema operacional e todo seu hardware enquanto o container compartilha o kernel do sistema operacional do servidor.

Figura 11: Comparação entre Container e Máquina Virtual.



Fonte: DOCKER (2021).

A utilização de containers possibilita uma série de vantagens, dentre as principais pode-se listar:

- Portabilidade: o ambiente de criação e/ou configuração do container não oferece interferência no ambiente em que ele vai funcionar, já que as dependências estão no container; (DOCKER, 2021)
- Versionamento: um determinado software ou aplicação pode ter suas diversas versões executando separadamente com suas dependências específicas (que também podem estar em versões diferentes), compartilhando recursos compartilhados ou específicos. Dessa forma os testes podem ocorrer em paralelo para o *update* das versões; (POSITIVO TECNOLOGIA, 2017)
- Gerenciamento de Recursos: os recursos utilizados podem ser compartilhados entre aplicações ou permanecerem exclusivos, de acordo com o gerenciamento adotado, como alguma base de dados ou dependência. Esta característica também possibilita a otimização do armazenamento, pois são utilizados apenas

os componentes necessários implicando também em uma disponibilidade maior do sistema;

- Desempenho: como os contêineres compartilham do mesmo *kernel* do SO não existe a necessidade de um sistema por aplicação, melhorando o desempenho da aplicação; (VITALINO, 2016)
- Segurança: os isolamentos fornecem uma camada a mais de segurança.

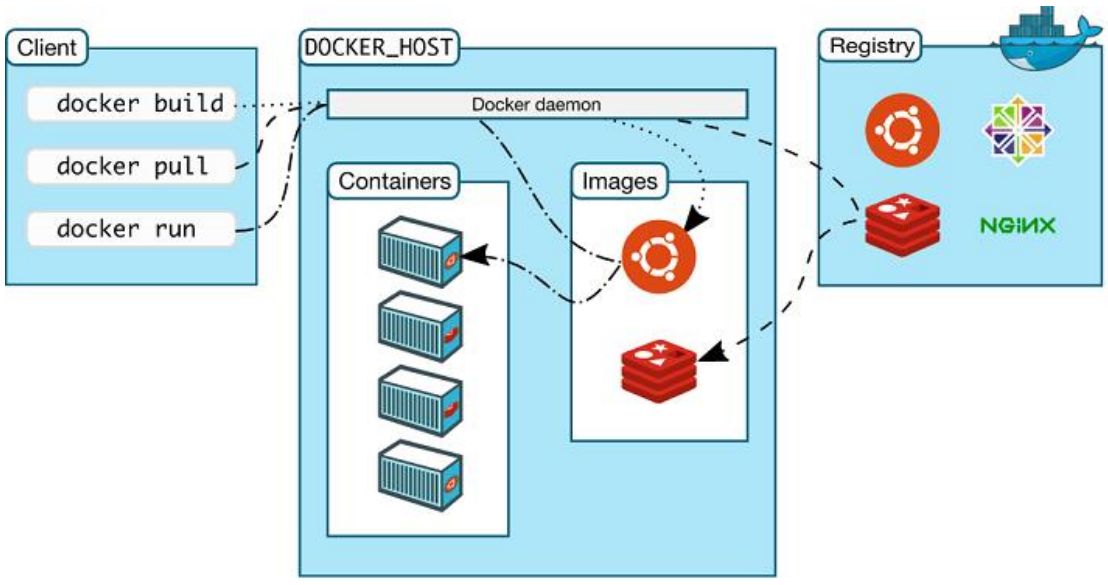
2.6.2 Docker

O Docker teve início em 2013 como uma tecnologia de conteinerização de software *open source*, possibilitando todas as vantagens características de containers e por apresentar diversas ferramentas complementares para o gerenciamento dos mesmos (DOCKER, 2021).

Para entender a arquitetura do Docker é necessário o conhecimento de alguns conceitos:

- Docker *Engine*: é um software que executa em plano de fundo auxiliando na execução e criação dos containers. É responsável pelo gerenciamento das imagens, containers, volumes e redes;
- Docker *Client*: É a interface entre o usuário e o Docker Engine;
- Docker *Registry*: Responsável por manter e distribuir a imagens (aplicação ou dependência pré configurada, podendo ser customizada para atender as necessidades).

Figura 12: Arquitetura Docker



Fonte: DOCKER DOCS (2021).

2.7 WebRTC

WebRTC é uma API de código aberto que fornece recursos para conexão em tempo real (GOOGLE INC).

Como é uma API, pode ser aplicada de diversas formas nos aplicativos, mas apresenta um fluxo padrão de “etapas” para a comunicação ser efetivada:

- Identificar os dispositivos de áudio e/ou vídeo e acessá-los com um script JS utilizando a interface “MediaDevices”;
- Abrir a conexão de pares com a interface “RTCPeerConnection”;
- Com a conexão estabelecida os dois pares devem se comunicar.

Utilizar o padrão WebRTC é consideravelmente complicado e a documentação oficial fornece alguns tutoriais simples, no entanto, a codificação requer um estudo dedicado para projetos ou funcionalidades mais complexas.

A sintaxe é na linguagem JavaScript, podendo apresentar integração com outros arquivos se necessário.

Figura 13: Trecho da Documentação do WebRTC.

```
    const listElement = document.querySelector('select#availableCameras');
    listElement.innerHTML = '';
    cameras.map(camera => {
        const cameraOption = document.createElement('option');
        cameraOption.label = camera.label;
        cameraOption.value = camera.deviceId;
    }).forEach(cameraOption => listElement.add(cameraOption));
}

// Fetch an array of devices of a certain type
async function getConnectedDevices(type) {
    const devices = await navigator.mediaDevices.enumerateDevices();
    return devices.filter(device => device.kind === type)
}

// Get the initial set of cameras connected
const videoCameras = getConnectedDevices('videoinput');
updateCameraList(videoCameras);
```

Fonte: GOOGLE INC.

A tecnologia WebRTC por vezes é confundida com WebSockets, mas são distintas. Embora ambas sejam especificadas no HTML5 os WebSockets possibilitam a comunicação

nos dois sentidos entre um cliente e um servidor, enquanto o WebRTC possibilita a comunicação em tempo real entre navegadores (BLOOGGEEK, 2019). A figura a seguir exibe o canal de dados comparativo entre WebSockets e WebRTC.

Figura 14: WebSocket e WebRTC.

Fonte: BLOGGEEK (2019).

3 TRABALHOS RELACIONADOS

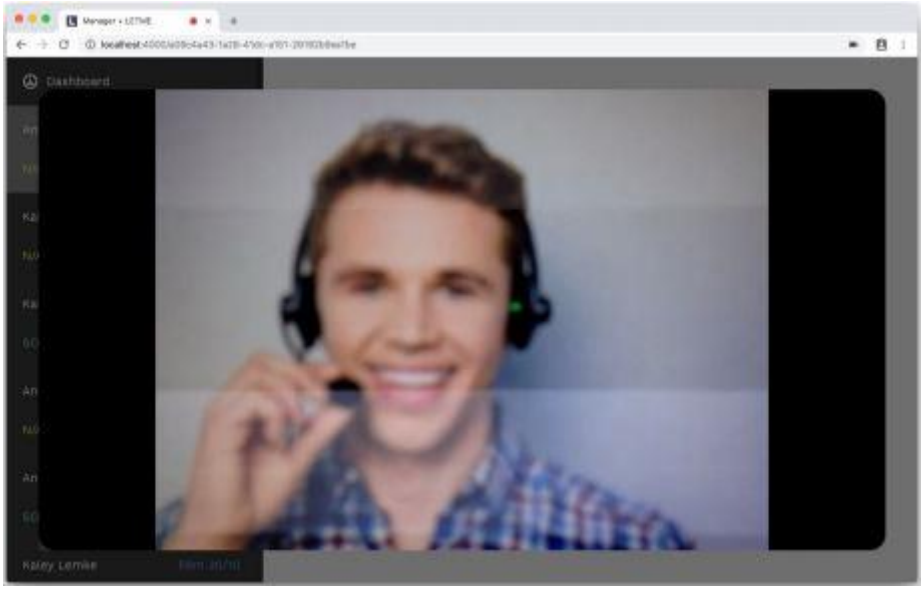
Para complementar este trabalho, foram examinadas propostas semelhantes e trabalhos relacionados com o desenvolvimento de sistemas similares.

Rinaldi et al. (2018) descreve o projeto de sistema para atendimento ao consumidor por meio de vídeo utilizando a tecnologia WebRTC. Este trabalho resolve as dificuldades da impessoalidade nos serviços de atendimento desenvolvendo um sistema de videoconferência com dados estatísticos sobre os processos.

Quando o usuário acessa o sistema é direcionado para a página de login. Após realizar a autenticação é possível acessar algumas funcionalidades. Na figura 15 é possível observar a tela de chamada do sistema, na tela tem-se o cliente em chamada com o atendente e abaixo o botão de desligar a chamada. Ao fundo à esquerda, percebe-se por trás da imagem do cliente as demais funcionalidades como de visualizar as estatísticas.

O sistema apresentado não apresenta funcionalidades de chat e nem descreve informações quanto a segurança do sistema, por exemplo a funcionalidade da verificação em duas etapas.

Figura 15: Tela do Sistema de Atendimento ao Consumidor.



Fonte: Retirado de RINALDI et al (2018), p 60.

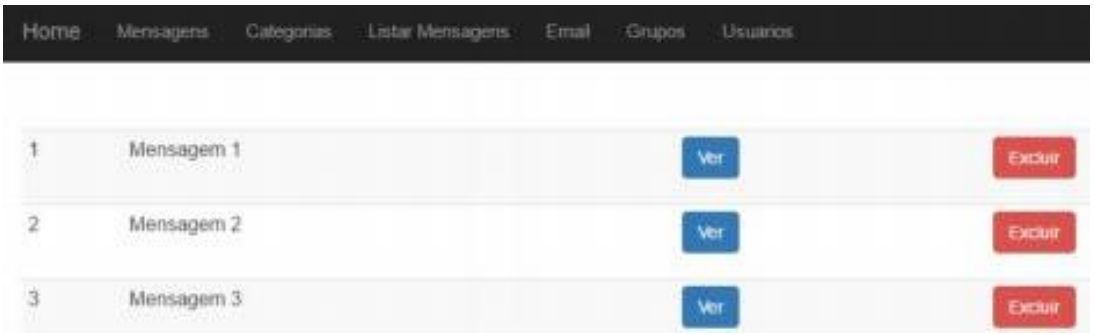
Já no trabalho de Santos (2016) é proposto o desenvolvimento de um sistema de comunicação por meio de transmissão de mensagens para que as instituições de ensino

melhorando a relação da instituição com seus receptores (servidores, comunidade externa, entre outros).

Neste sistema quando o usuário acessa o sistema é direcionado para a tela de login e após o login pode visualizar as demais funcionalidades do menu. Na figura 16 é possível observar a tela das mensagens criadas. Na parte superior é possível observar o menu com as funcionalidades do sistema. Na parte central a lista de mensagens da categoria, junto com as opções de visualizar e deletar.

O trabalho apresentado tem funcionalidades básicas no qual o usuário envia a mensagem pelo sistema e recebe a notificação ou resposta por e-mail. O sistema não fornece segurança no armazenamento da senha de autenticação apresentando um risco para a instituição.

Figura 16: Tela do Sistema de Comunicação Interna.



Fonte: Retirado de SANTOS (2016), p 61.

4 LEVANTAMENTOS DE REQUISITOS E ARQUITETURA PROPOSTA

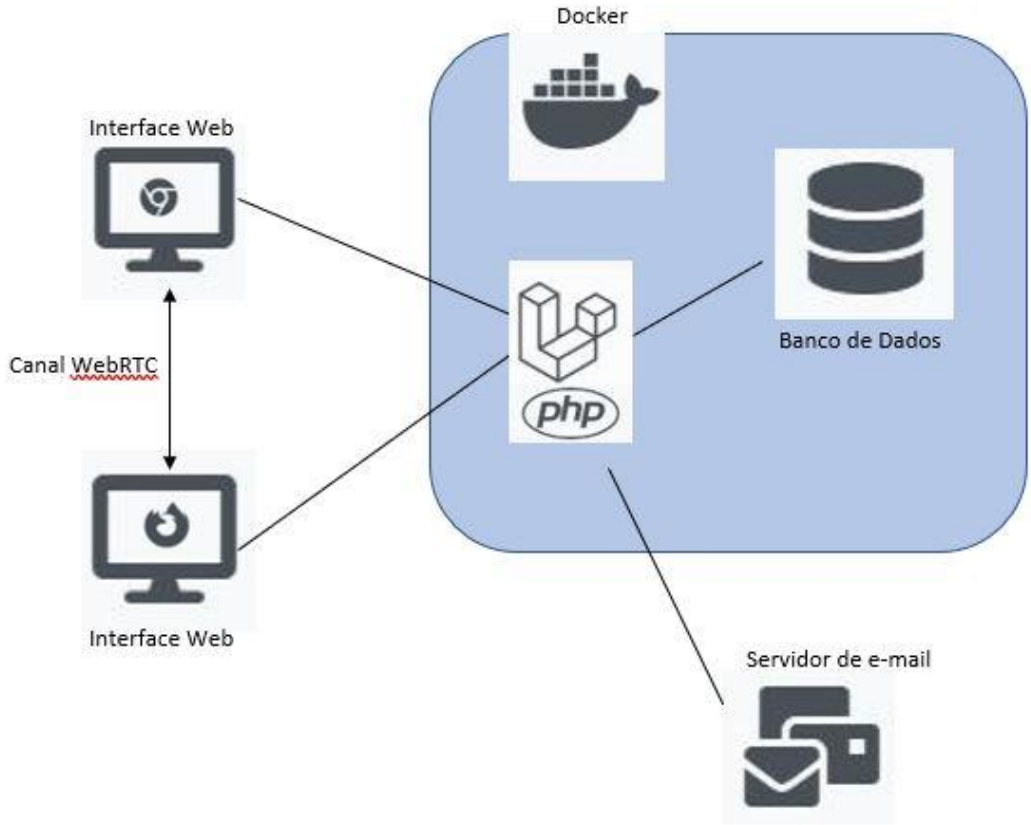
O sistema proposto foi desenvolvido com os termos e tecnologias conceituados na fundamentação teórica. Nesta seção, são apresentados os requisitos e arquitetura proposta.

4.1 Arquitetura Proposta

O objetivo do protótipo é possibilitar a comunicação remota entre os usuários por meio de mensagens de texto e chamadas de áudio e vídeo.

A ferramenta descrita neste trabalho executa sobre o Docker com suas tecnologias separadas em containers e utilizando a tecnologia WebRTC para as chamadas em tempo real. Realiza a comunicação com o servidor de e-mail permitindo o envio de mensagens e notificações para os e-mails cadastrados. A figura 17 descreve a interação do usuário com o sistema por meio da interface web.

Figura 17: Arquitetura Proposta.



Fonte: Elaborado pelo autor (2021).

4.2 Levantamento de Requisitos

Nesta etapa coletaram-se os requisitos. O sistema proposto neste trabalho de conclusão de curso é um software para comunicação pela internet através de conexões de voz, vídeo e permitindo o envio de mensagens de texto.

O sistema deve permitir que os usuários possam ter acesso às seguintes funções:

- Cadastrar, editar ou desativar sua conta;
- Buscar determinado usuário ativo;
- Enviar mensagens de texto para outro usuário ativo;
- Realizar chamada de voz com outo usuário ativo;
- Realizar chamada de vídeo com outro usuário ativo.

4.2.1 Requisitos Funcionais

De acordo com a proposta do sistema, elaborou-se a seguinte tabela com os requisitos funcionais do sistema:

Tabela 7: Requisitos Funcionais

ID	DESCRIÇÃO DO REQUISITO FUNCIONAL
RF001	Cadastrar usuário
RF002	Editar usuário
RF003	Visualizar usuário
RF004	Desativar usuário
RF005	Buscar usuário
RF006	Adicionar usuário na lista de contatos
RF007	Enviar mensagem de texto
RF008	Visualizar mensagens de texto
RF009	Realizar verificação de duas etapas
RF010	Realizar chamada de áudio
RF011	Realizar chamada de vídeo

Fonte: Elaborado pelo autor (2021).

4.2.2 Requisitos Não-Funcionais

Os requisitos não-funcionais podem dizer respeito ao desempenho, confidencialidade, integridade, disponibilidade e outras restrições arquiteturais internas ao sistema que não retratam diretamente as funções do sistema. (PRESSMAN 2011, p 213)

De acordo com a proposta do sistema elaborou-se a seguinte tabela com os requisitos não-funcionais:

Tabela 8: Requisitos Não-Funcionais

ID	DESCRIÇÃO DO REQUISITO NÃO-FUNCIONAL
RNF001	Design Responsivo
RNF002	Utilizar elementos dinâmicos nas interfaces
RNF003	Manter os dados de login criptografados no banco
RNF004	Utilizar o framework Laravel
RNF005	Utilizar o banco de dados PostgreSQL
RNF006	Utilizar o container Docker
RNF007	Compatibilidade com os principais navegadores
RNF008	Verificação em duas etapas

Fonte: Elaborado pelo autor (2021).

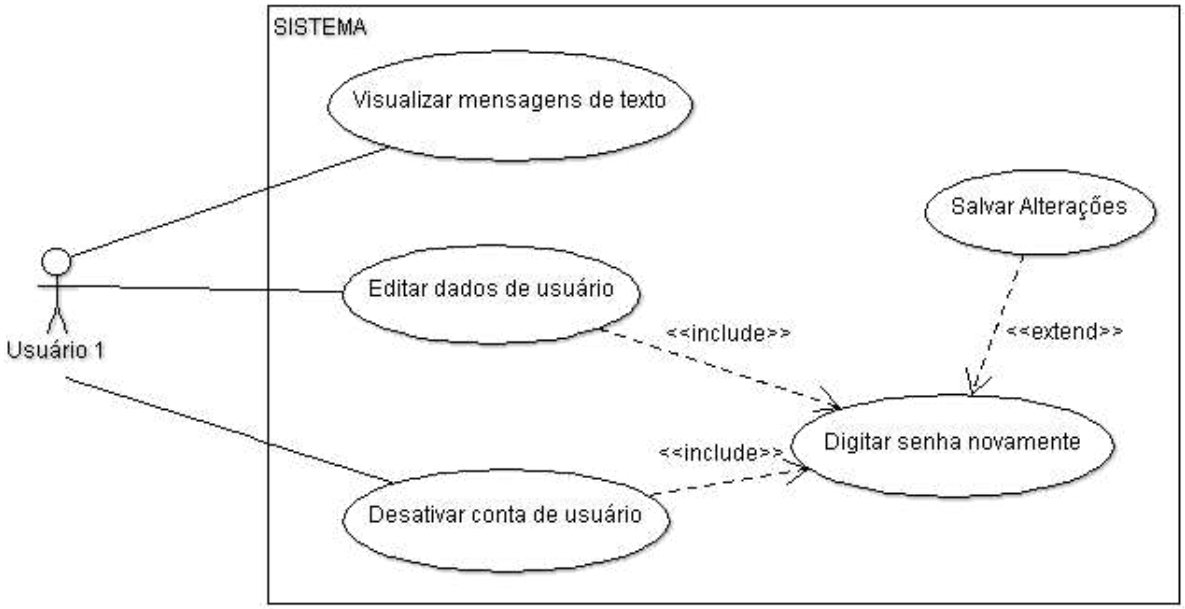
4.3 Análise de Requisitos

Nesta etapa analisaram-se os requisitos coletados e desenvolveu-se alguns diagramas necessários.

4.3.1 Diagrama de Caso de Uso

Pressman (2011, p 155) caracteriza caso de uso como a definição de como um ator interage com algum ponto do sistema para alcançar algum objetivo. A figura a seguir representa um caso de uso inicial para o sistema.

Figura 18: Exemplo de Caso de Uso.



Fonte: Elaborado pelo autor (2021).

Juntamente com este caso de uso foi desenvolvida uma pequena narrativa para apresentar a sequências de ações possíveis do usuário:

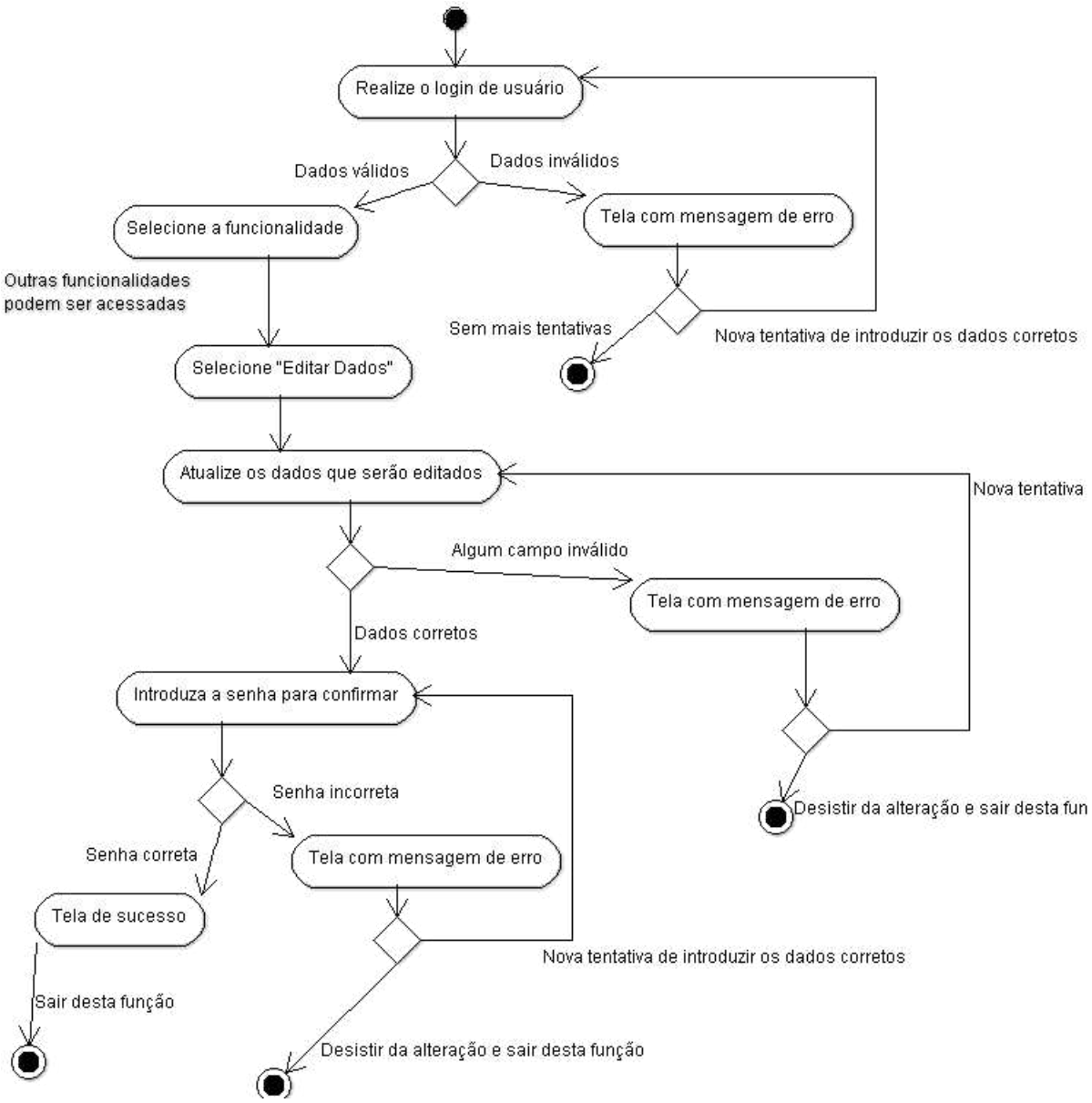
- O usuário faz o login no sistema;
- O usuário visualiza as mensagens de texto;
- O usuário clica em “editar dados”;
- O sistema exibe a tela de edição de dados;
- O usuário altera alguns dos dados de suas contas;
- O sistema pede a senha para confirmar a ação;
- O usuário digita a senha e confirma;
- O sistema valida e salva as alterações;
- O usuário clica em “desativar conta”;
- O sistema pede a senha para confirmar a ação;
- O usuário digita a senha e confirma;
- O sistema valida e desativa a conta;
- O sistema exibe a tela inicial com o usuário deslogado.

É importante ressaltar que esta narrativa não exibe os fluxos alternativos, como em caso de digitação da senha errada ou campos preenchidos inadequadamente (o sistema exibe notificações nesses casos).

4.3.2 Diagrama de Atividade

A representação gráfica do fluxo complementa o caso e uso, tal representação pode ser feita pelo diagrama de atividade. A figura a seguir exibe um diagrama de atividade para a edição de dados do usuário.

Figura 19: Exemplo de Diagrama de Atividade.



Fonte: Elaborado pelo autor (2021).

Com o auxílio do diagrama de atividade os retornos são facilmente entendidos, como por exemplo, os retornos por dados preenchidos de forma errônea.

4.4 Modelagem

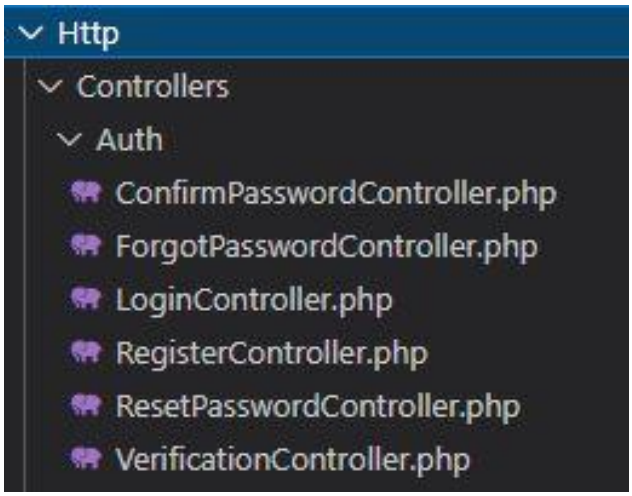
Nesta etapa foi construída a modelagem do sistema, contando com o apoio de alguns modelos, pois, como já foi mencionado o Laravel trabalha com o padrão de arquitetura MVC, além disso frameworks têm a característica de serem facilitadores do trabalho de desenvolvimento, dessa forma na estrutura do projeto diversas classes, em especial *controllers*, são necessárias para a correta utilização deste framework e de suas possibilidades.

Com esta observação esclarecida, dividiu-se os modelos observando apenas as principais classes ou conjuntos de classes que merecem comentários para otimizar o processo explicativo.

4.4.1 Registro e Autenticação de Usuários

Todo o conjunto de ações relacionadas com a autenticação de um usuário, desde o registro até o *login* e *logout* envolve diversas classes e métodos. Os principais controladores foram agrupados em um diretório de caminho "Http\Controllers\Auth" e são fundamentais para esta funcionalidade do sistema.

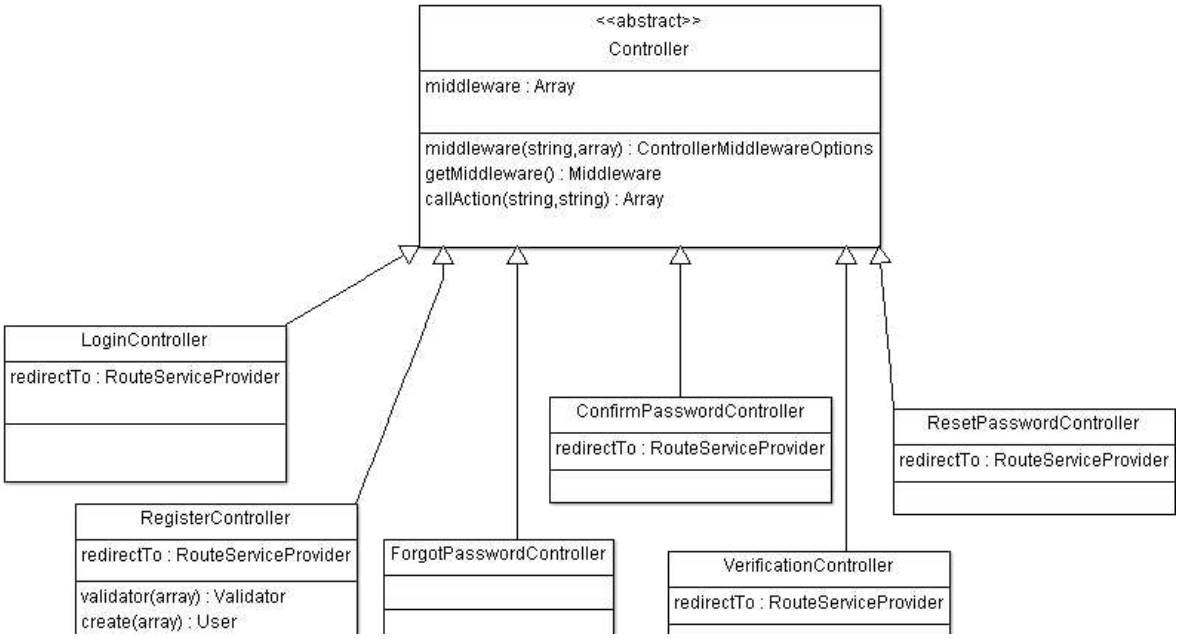
Figura 20: Diretório Auth



Fonte: Elaborado pelo autor (2021).

Todos eles relacionam-se internamente em alguns métodos. São fundamentais para a segurança do sistema. Na seguinte figura é exibido o diagrama de classe correspondente. Note que as demais classes do diagrama foram omitidas para manter a visualização menos poluída.

Figura 21: Parte do Diagrama de Classes Relacionada com a Autenticação



Fonte: Elaborado pelo autor (2021).

4.4.2 Modelo Físico

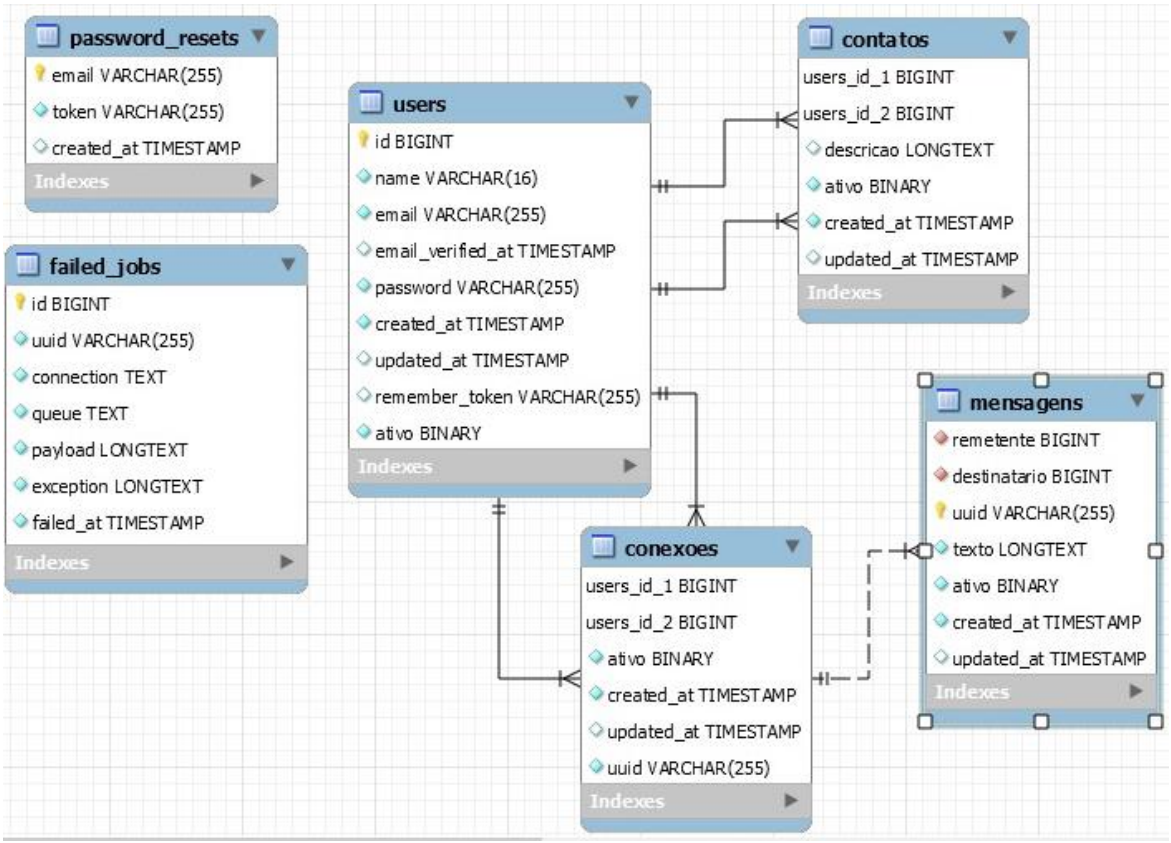
O protótipo do sistema apresenta suas regras de negócio características e apesar das tabelas do banco de dados serem construídas por meio dos comandos *artisan*, faz-se necessária a construção do modelo físico do banco de dados proposto.

Em geral, os projetos com o framework Laravel fornecem três tabelas de exemplo: “users”, “failed_jobs” e “password_resets”, relacionadas com os dados dos usuários, autenticação e recuperação de senhas.

Quando as *migrations* são executadas, uma tabela adicional de nome “migrations” é criada no banco de dados. Essa tabela armazena informações internas para a execução de *rollbacks*, *commits*, *drops*, além de informações para que novas migrações possam ocorrer em um ambiente de produção.

Existem diversas formas de representar uma mesma regra de negócio ou situação em que uma base de dados é necessária. De acordo com as regras de negócio e requisitos foi pensado e elaborado uma modelagem para representar as tabelas do banco.

Figura 22: Modelo Físico.



Fonte: Elaborado pelo autor (2021).

Nesse momento, é ideal que ocorra a representação de alguma busca no banco de dados para verificar se a modelagem está definida de forma que atenda aos requisitos. Por exemplo, o retorno da lista de contatos de determinado usuário seria com a seguinte seleção:

Figura 23: Select para Teste da Modelagem.

```
$aux = DB::select('select users.id, users.name, users.email
                   from users inner join contatos
                   on users.id = contatos.user_id_2
                   where contatos.user_id_1 = '.Auth::user()->id.
                   '');
dd($aux);
```

Fonte: Elaborado pelo autor (2021).

Tendo como resultado desta busca o seguinte JSON (resultado correto, que estava sendo esperado):

Figura 24: Resultado da Busca em JSON.

Fonte: Elaborado pelo autor (2021).

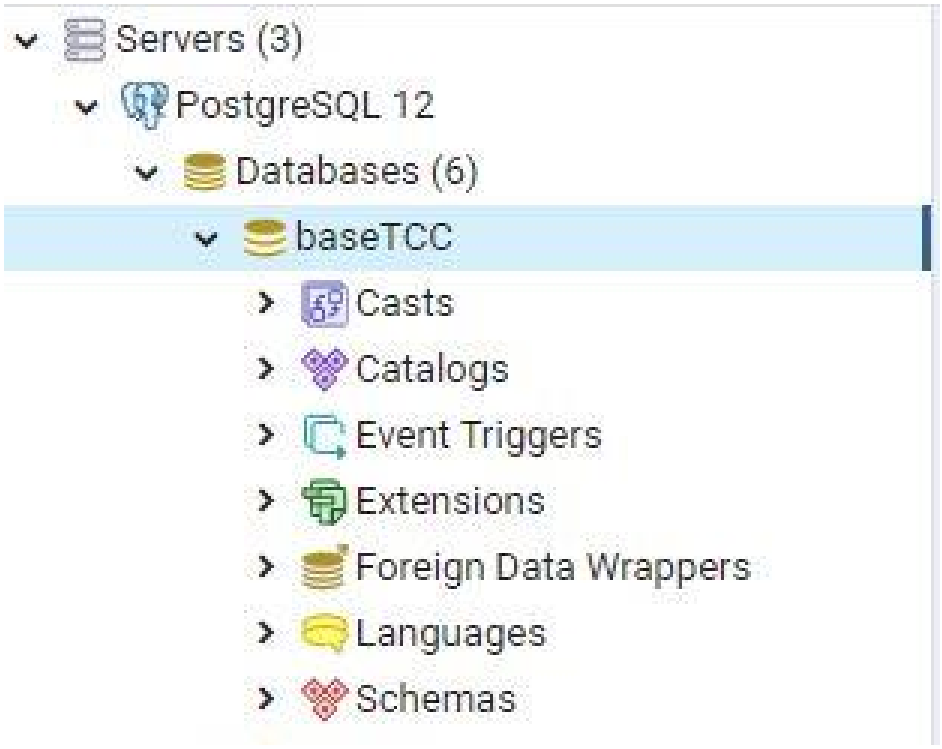
5 DESENVOLVIMENTO E TESTES

Nesta etapa descreve-se a implementação e teste do sistema proposto, assim como os resultados obtidos.

Inicialmente, configurou-se o ambiente de desenvolvimento, pois, projetos que utilizam o framework Laravel, assim como outras tecnologias, necessitam da configuração inicial do ambiente do sistema. O Laravel possibilita a configuração da conexão com o banco de dados, do servidor de e-mail, dentre outros detalhes através do arquivo ENV (Este arquivo contém senhas importantes e por isso será exibido em partes neste tópico).

Antes das execuções dos comandos é necessário criar uma base de dados no postqres. As tabelas serão criadas futuramente através dos comandos do *Artisan*.

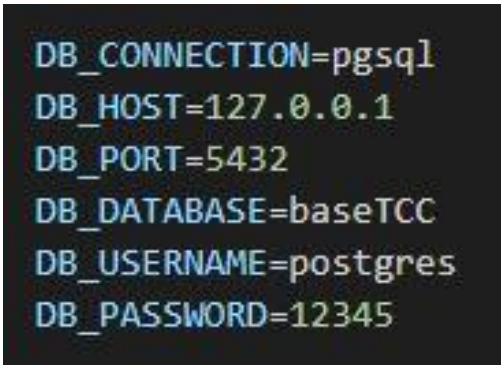
Figura 25: base de dados.



Fonte: Elaborado pelo autor (2021).

Após esta etapa o trecho do arquivo ENV pode ser configurado para a conexão e para que os comandos do *Artisan* referentes ao banco de dados funcionem.

Figura 26: Exemplo da configuração do ENV para conectar com a base.



Fonte: Elaborado pelo autor (2021).

Após a configuração da base, os arquivos referentes ao Docker foram organizados. Criou-se um arquivo de execução nomeado de “Dockerfile” com instruções que serão executadas e bibliotecas que serão instaladas.

Figura 27: Dockerfile

```
FROM php:7.4-fpm

WORKDIR /var/www/

RUN apt-get update && apt-get install -y \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    libpq-dev \
    cron

#RUN docker-php-ext-install pdo pdo_mysql
RUN docker-php-ext-install pgsql pdo pdo_pgsql

ADD . /var/www

RUN chown -R www-data:www-data /var/www
```

Fonte: Elaborado pelo autor (2021).

Com o Dockerfile configurado, criaram-se os arquivos “docker-compose” e “docker-compose_prod” de extensão “yml”, nos quais são definidos os serviços que compõem o sistema.

Em geral os arquivos criados apresentam configurações semelhantes, no entanto, algumas aplicações foram excluídas dos serviços que irão funcionar em um ambiente de produção em comparação com o ambiente de desenvolvimento, como o PgAdmin, que está apenas no arquivo “docker-compose”.

Figura 28: Trecho do arquivo docker-compose

```
#npm
npm:
  image: node:13.7
  container_name: TCC-v1_npm
  volumes:
    - ./:/var/www/
  working_dir: /var/www/
  entrypoint: ['npm']

#artisan
artisan:
  build:
    context: .
    dockerfile: Dockerfile
  container_name: TCC-v1_artisan
  volumes:
    - ./:/var/www
  depends_on:
    - postgres
  working_dir: /var/www/
  entrypoint: ['php', '/var/www/artisan']
  networks:
    - app-network
```

Fonte: Elaborado pelo autor (2021).

O processo de desenvolvimento no Laravel é semelhante para cada parte do sistema. Considerando o desenvolvimento do módulo de autenticação por exemplo: cria-se a *migration* que armazenará os dados do usuário, cria-se o *model* do usuário, codificam-se as *controllers* relacionadas, codificam-se os métodos das *controllers*, configuram-se os arquivos de rotas e no caso, deste sistema configura-se o servidor de e-mail para realizar a verificação em duas etapas.

As demais lógicas do sistema seguem o mesmo fluxo, mas com as suas peculiaridades e dependências especificas.

Figura 29: Trecho do Modelo User.

```
protected $fillable = [
    'name',
    'email',
    'password',
];

/**
 * The attributes that should be hidden for arrays.
 *
 * @var array
 */
protected $hidden = [
    'password',
    'remember_token',
];

/**
 * The attributes that should be cast to native types.
 *
 * @var array
 */
protected $casts = [
    'email_verified_at' => 'datetime',
];
```

Fonte: Elaborado pelo autor (2021).

O Laravel permite a configuração do servidor de e-mail pelo arquivo ENV como observado na seguinte imagem (a senha foi omitida do print).

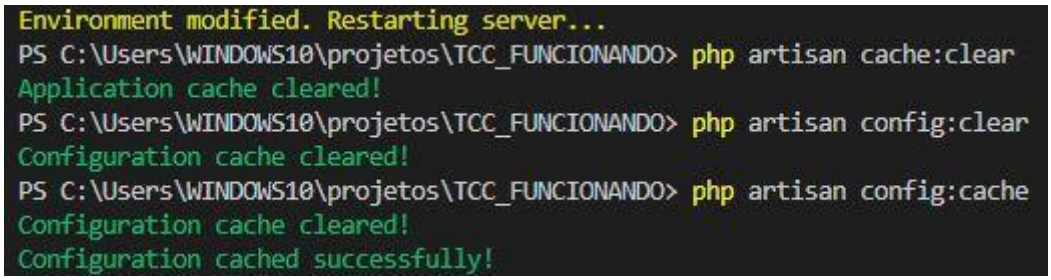
Figura 30: Configuração do Servidor de E-mail.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=TccServidorEmail@gmail.com
MAIL_PASSWORD=*********
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"
```

Fonte: Elaborado pelo autor (2021).

Sempre que a configuração presente no arquivo ENV de um projeto Laravel sofre alteração é necessário limpar o cache da aplicação.

Figura 31: Comandos para Limpeza do Cache.



Fonte: Elaborado pelo autor (2021).

Com o servidor de e-mail configurado notificações e mensagens podem ser enviadas para os e-mails cadastrados. Criou-se então a lógica da criação do código e do envio da mensagem.

A lógica desenvolvida considera a data e o horário atual na geração do código, dessa forma garantindo uma maior aleatoriedade. O envio é feito com base na configuração do arquivo ENV utilizando a biblioteca “Mail”. Para o layout da mensagem foi criado um arquivo “blade” para otimizar o processo. A criptografia utilizada foi a padrão (TLS).

Figura 32: Codificação para o envio do e-mail.

```
public function emailVerifica(array $dados){

    $this->email_temp = $dados['email'];
    $a = Mail::send('email', $dados, function ($m)  {
        $m->from('TccServidorEmail@gmail.com', 'SISTEMA');
        $m->to($this->email_temp);
        $m->subject('Verificação em Duas Etapas');
    });

}

public function verificacaoDuasEtapas(){

    $codigo = $this->generateCod();

    $dados = [
        'codigo' => $codigo,
        'email' => Auth::user()->email,
        'nome' => Auth::user()->name
    ];
    $this->emailVerifica($dados);

    return view('user.confirm', compact('codigo'));

}

public function generateCod($qtyCaraceters = 6)
{
    $smallLetters = str_shuffle('');//sem letras minusculas
    $capitalLetters = str_shuffle('');//sem letras maiusculas
    $numbers = (((date('Ymd') / 12) * 24) + mt_rand(800, 9999));
    $numbers .= 1234567890;
    $specialCharacters = str_shuffle('');//sem caracteres especiais
    $characters = $capitalLetters.$smallLetters.$numbers.$specialCharacters;
```

Fonte: Elaborado pelo autor (2021).

Como o e-mail utilizado no cadastro do usuário realmente existe, a mensagem foi recebida e a verificação foi concluída posteriormente. Este recurso consiste em um código, em geral de 6 dígitos, que pode ser utilizado para verificar a identidade do usuário.

Figura 33: Mensagem Recebida.

Fonte: Elaborado pelo autor (2021).

Com a parte de autenticação finalizada iniciou-se o mesmo fluxo de desenvolvimento para a lista de contatos e o envio de mensagens entre os usuários. O *controller* dos Contatos ficou responsável pelos métodos relacionados com a lista de contatos, por exemplo, exibir contatos e adicionar um contato na lista. O controlador das mensagens ficou responsável pelos métodos relacionados com a criação e envio de mensagens. O controlador das conexões ficou responsável pelas conexões entre os usuários.

Figura 34: Trecho do ContatoController.

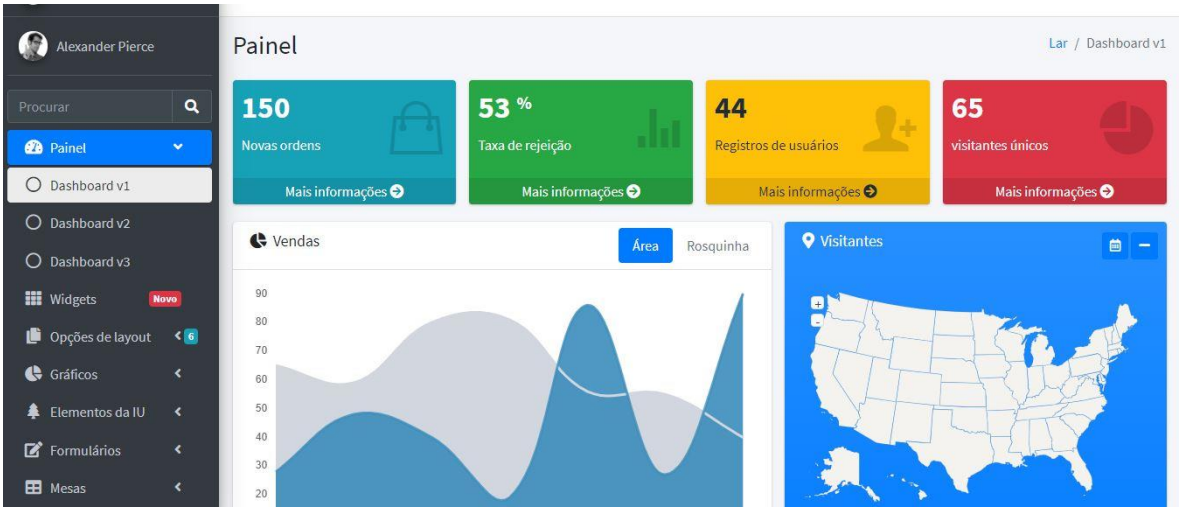
```
class ContatoController extends Controller
{
    protected $contatoRepository;

    public function __construct(ContatoRepository $contatoRepository) {
        $this->contatoRepository = $contatoRepository;
    }

    public function getIndex()
    {


        $contatos = DB::table('users')
                ->join('contatos', 'users.id', '=', 'contatos.user_id_2')
                ->where('contatos.user_id_1', '=', Auth::user()->id)
                ->select('users.id', 'users.name', 'users.email')
                ->get();
        return view('contato.index', compact('contatos'));;
    }

    public function add(Request $request)
    {
        $aux = $this->contatoRepository->create($request->all());
        if($aux) {
```

Fonte: Elaborado pelo autor (2021).

A implementação das *views* no Laravel ocorre com o auxílio da sintaxe “blade”, que possibilita diversas facilidades como aplicar a estrutura de repetição “foreach” no *front* de forma simplista.

O template escolhido foi o AdminLTE 3 (ADMINLTE, 2019). Template gratuito com diversos modelos, opções de layouts, elementos e componentes responsivos, milhares de ícones e apresentando compatibilidade com o Laravel (instalado com poucos comandos).

Figura 35: Site do AdminLTE.



Fonte: Elaborado pelo autor (2021).

Para adicionar os arquivos do AdminLTE em um projeto Laravel é necessário utilizar os seguintes comados: “composer require jeroennoten/laravel-adminlte” para instalar e “artisan adminlte:install” para configurar. Alguns outros comandos podem ser necessários, mas já é possível utilizar os recursos do template.

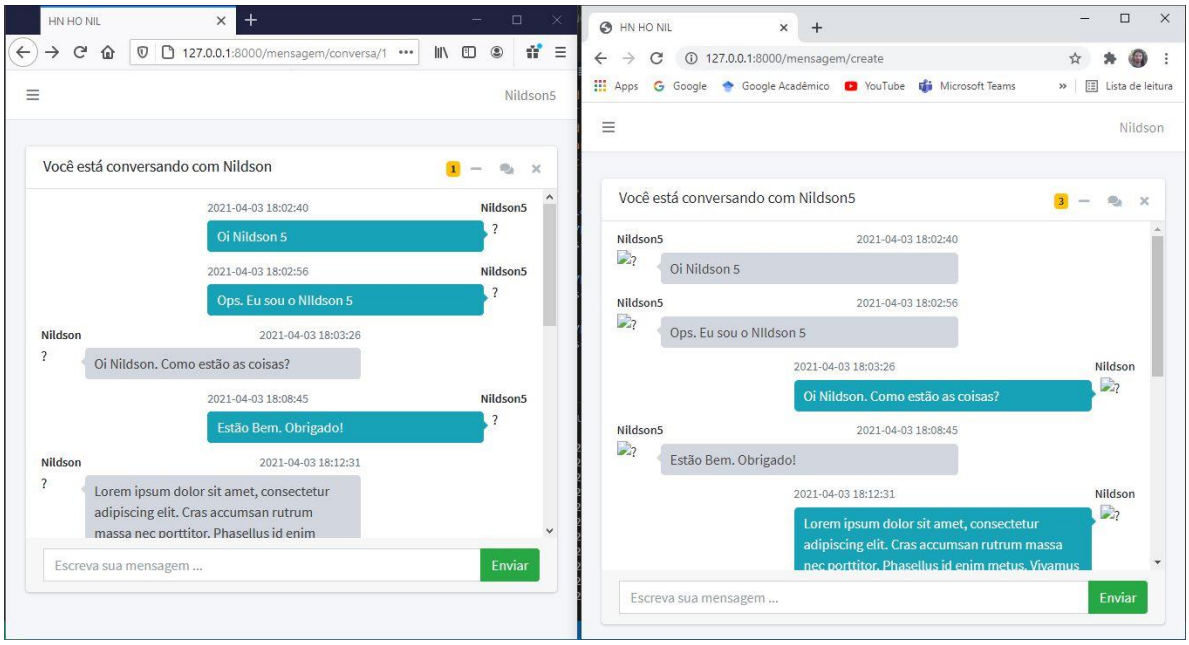
Figura 36: Blade da tela da conversa.

```
@isset($mensagens)
        @foreach ($mensagens as $msg )
                @if ($msg->user_id_1 == $user->id)
                        <div class="direct-chat-msg col-sm-8">
                                <div class="direct-chat-infos clearfix">
                                        <span class="direct-chat-name float-left">{{$user->name}}</span>
                                        <span class="direct-chat-timestamp float-right">{{$msg->created_at}}</span>
                                </div>
                                <img class="direct-chat-img" src="{{$user->img ?? ''}}" alt="?">
                                <div class="direct-chat-text">
                                        {{$msg->texto}}
                                </div>
                        </div>
                @else
                        <div class="direct-chat-msg right offset-sm-4 col-sm-8">
                                <div class="direct-chat-infos clearfix">
                                        <span class="direct-chat-name float-right">{{Auth::user()->name}}</span>
                                        <span class="direct-chat-timestamp float-left">{{$msg->created_at}}</span>
                                </div>
                                <img class="direct-chat-img" src="{{Auth::user()->img ?? ''}}" alt="?">
                                <div class="direct-chat-text">
                                        {{$msg->texto}}
                                </div>
                        </div>
                @endif
        @endforeach
@endisset
```

Fonte: Elaborado pelo autor (2021).

A figura a seguir exemplifica a tela dos pontos de vista de dois usuários conversando por mensagens de texto. Observa-se a responsividade do sistema, que com o ajuste do tamanho da tela ocultou o menu lateral para facilitar a visualização das mensagens, também adicionou uma barra de rolagem permitindo visualizar as mensagens anteriores sem deformar os campos de texto.

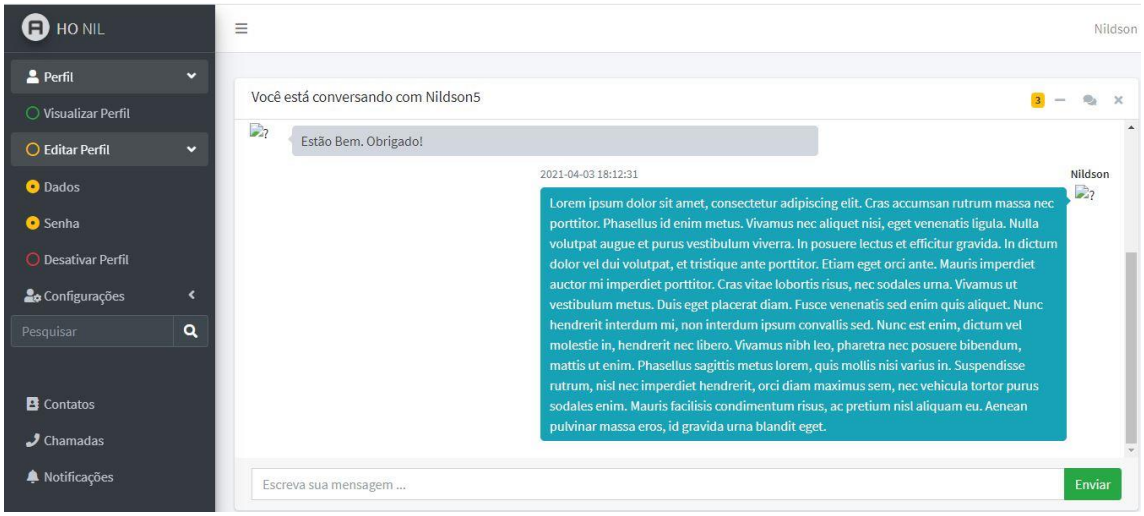
Figura 37: Conversa Entre Dois Usuários.



Fonte: Elaborado pelo autor (2021).

Percebe-se que a responsividade está funcionando corretamente para os diversos tamanhos das telas dos navegadores. Quando a janela apresenta o seguinte formato observado o menu lateral surge exibindo as opções disponíveis.

Figura 38: Tela do Sistema.



Fonte: Elaborado pelo autor (2021).

Para as chamadas, independentemente do tipo, é necessária a captura do áudio e/ou vídeo. Para isso utilizou-se a API WebRTC (GOOGLE INC) para acessar a webcam e o microfone do dispositivo.

Figura 39: Identificação dos Dispositivos de Mídia.

```
const constraints = {
    'video': true,
    'audio': true
}
function startVideoFromCamera(){
    navigator.mediaDevices.getUserMedia(constraints).then(stream=>{

        const videoElement = document.querySelector("#video")
        videoElement.srcObject = stream
    }).catch(error=>{console.log(error)})
}
window.addEventListener("DOMContentLoaded", startVideoFromCamera)
```

Fonte: Elaborado pelo autor (2021).

Dentro da API tem-se uma interface denominada de “RTCPeerConnection” que possibilita a conexão entre o dispositivo local e o par remoto.

Inicialmente cria-se uma oferta SDP descrevendo a conexão e com o sucesso cria-se o ponto local da conexão. Conecta-se o ponto local ao remoto e se essa ação for bem sucedida o ponto remoto conhece a conexão e agora responde um resumo SDP.

Tendo a resposta criada ela é passada, estabelecendo assim o ponto remoto (lembrando que estas ações também estão sendo realizadas no outro ponto da conexão, mas este passar a ser o ponto local e assim por diante). Ao fim as descrições estão configuradas. Este código por si só não gera a conexão, mas descreve bem a interação observada.

Figura 40: Estabelecendo Conexão RTC.

```
PLocal.createOffer()
    .then(offer => PLocal.setLocalDescription(new RTCSessionDescription(offer)))
    .then(() => PRemoto.setRemoteDescription(PLocal.localDescription))
    .then(() => PRemoto.createAnswer())
    .then(asnwer => PRemoto.setLocalDescription(new RTCSessionDescription(answer)))
    .then(() => PLocal.setRemoteDescription(PRemoto.localDescription));
```

Fonte: Elaborado pelo autor (2021).

Algumas informações de sessões podem ser observadas no navegador Mozilla acessando o “about:webrtc”.

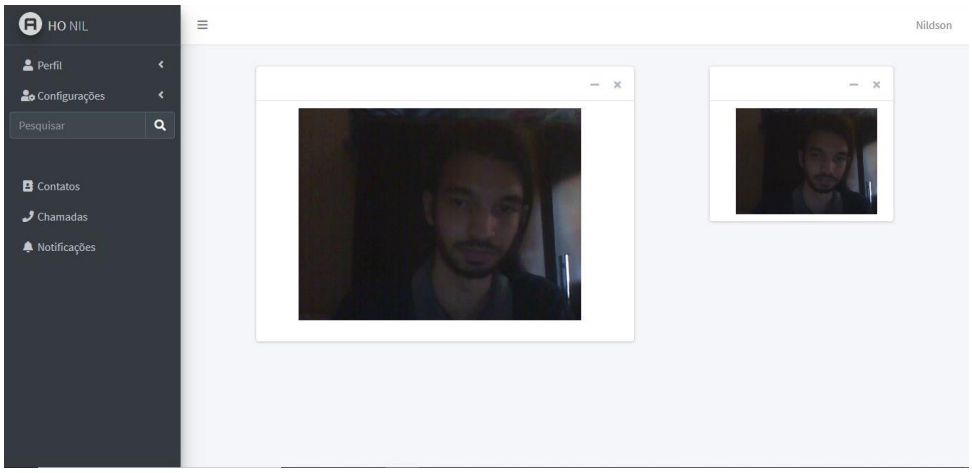
Figura 41: Trecho do SDP Remoto do About WebRTC.

```
a=fmtp:97 profile-level-id=42e01f;level-asymmetry-allowed=1
a=fmtp:120 max-fs=12288;max-fr=60
a=fmtp:124 apt=120
a=fmtp:121 max-fs=12288;max-fr=60
a=fmtp:125 apt=121
a=fmtp:127 apt=126
a=fmtp:98 apt=97
a=ice-pwd:12bb4d5d1d553871fb6c6d097939f01c
a=ice-ufrag:7c6534d8
a=mid:0
a=msid:{2ecf4f88-95c2-417c-8446-dcd0e4b493bc} {a78a949a-f07b-46a7-9
a=rtcp-fb:120 nack
a=rtcp-fb:120 nack pli
a=rtcp-fb:120 ccm fir
a=rtcp-fb:120 goog-remb
a=rtcp-fb:120 transport-cc
a=rtcp-fb:121 nack
a=rtcp-fb:121 nack pli
a=rtcp-fb:121 ccm fir
a=rtcp-fb:121 goog-remb
a=rtcp-fb:121 transport-cc
```

Fonte: Elaborado pelo autor (2021).

Com a conexão verificada os protótipos das *views* foram elaborados de forma simplista para concluir esta funcionalidade. Permitindo a ocultação das imagens ou encerramento, no menu lateral as demais funcionalidades ainda são acessáveis.

Figura 42: Protótipo da Tela para Chamada de Vídeo.



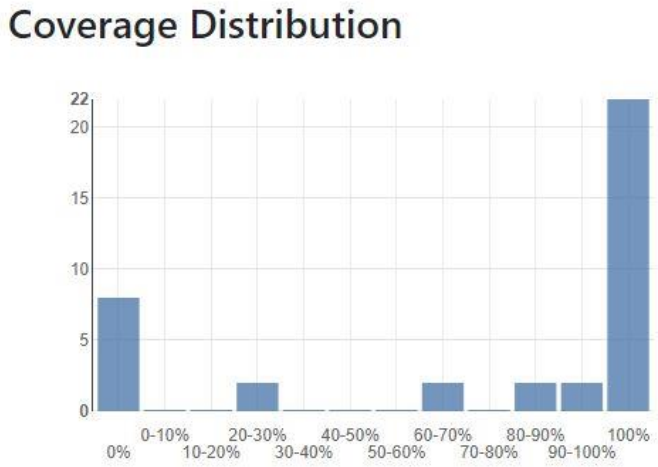
Fonte: Elaborado pelo autor (2021).

Testes com PHPUnit consistem em uma série de acessos automatizados nas rotas do sistema realizando as diversas ações do sistema.

Um fluxo de teste deve caracterizar por exemplo o usuário acessar a página de login, digitar seus dados de autenticação, acessar diversas telas, realizar diversas ações e por fim fazer o logout. Em cada uma dessas etapas existe um retorno esperado (confirmação da ação, redirecionamento, entre outros) e ao realizar os procedimentos são verificados se os retornos esperados foram alcançados. Se o retorno está correto ao esperado então a funcionalidade funcionou neste ambiente de teste.

Os casos de teste são elaborados em arquivos que estendem da classe abstrata “TestCase” que possibilita diversos métodos para auxiliar os processos. A versão atual do Laravel permite a exibição dos resultados por meio de gráficos. Com a criação e execução dos testes utilizando o PHPUnit optou-se pela representação gráfica em HTML.

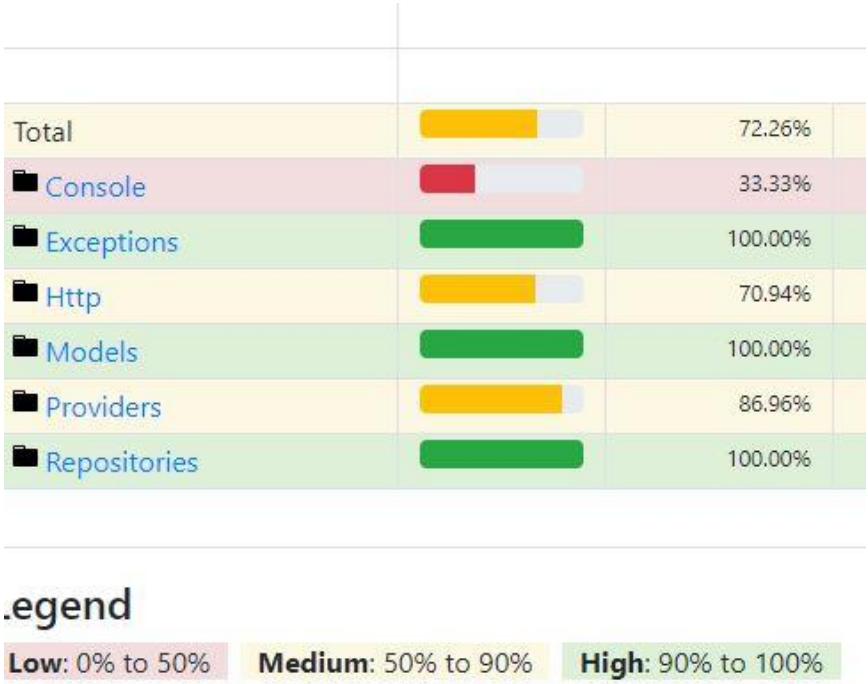
Figura 43: Gráfico de Abrangência dos Testes.



Fonte: Elaborado pelo autor (2021).

Alguns pontos ou trechos de códigos não puderam ser alcançados pelos testes, alguns por questões relacionadas com as estruturas de decisão e alguns por métodos que não podem ser chamados durante os testes.

Figura 44: Trecho do Gráfico de Cobertura dos Testes.



Fonte: Elaborado pelo autor (2021).

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O trabalho apresentou uma proposta de sistema de comunicação por chamadas e mensagens de texto. O seu foco foi desenvolver um protótipo demonstrando todas as etapas do projeto e descrevendo as tecnologias utilizadas. Para que o objetivo proposto fosse alcançado os conceitos aprendidos durante a graduação como Coleta e Análise de Requisitos, Modelagem, Desenvolvimento e Teste de Software, dentre outros, foram fundamentais.

A maioria das tecnologias utilizadas neste trabalho estão sendo consideravelmente utilizadas pelas comunidades de desenvolvedores atualmente, com destaque para o JavaScript e PHP que são algumas das linguagens mais utilizadas pelo mundo, o Docker que é um container que está ganhando cada vez mais espaço, e o Laravel que apesar de não ser um dos frameworks mais utilizados mantém uma comunidade ativa e está em constante atualização.

Em questões técnicas as principais dificuldades deste trabalho relacionam-se com a implementação das chamadas de vídeo entre os usuários. Caso um usuário não permita o acesso aos dispositivos de mídia, a chamada não é observada apesar da conexão ser estabelecida. O máximo de 300 e-mails enviados diariamente caracteriza uma limitação ao sistema também em relação ao número de usuários máximo do sistema, pois alguns usuários não conseguiriam utilizar as funcionalidades relacionadas, como a verificação em duas etapas. O sistema apresentou boas camadas de segurança com a verificação de autenticação e criptografia dos dados de login.

Em concordância com os objetivos listados neste trabalho, os requisitos não-funcionais foram alcançados e os requisitos funcionais foram elaborados. Os casos de teste foram elaborados e executados no servidor virtual na própria máquina, e executados alcançando mais de 70% do código do sistema. Apesar dos testes não alcançarem todo o código desenvolvido foram obtidos retornos satisfatórios.

A seguir estão algumas sugestões para trabalhos futuros:

- Alcançar uma taxa de mais de 95% do código testado;
- Permitir conversas em grupo;
- Aplicar suporte para novas tecnologias como chamadas holoportáticas, tecnologia recente que objetiva que modelos 3D sejam transmitidos em tempo real permitindo a comunicação remota entre as partes. Tal funcionalidade não foi possível no presente trabalho por falta de recursos tecnológicos.

REFERÊNCIAS

PRESSMAN, Roger S. **Engenharia de Software-7ª Edição**. McGraw Hill Brasil, 2011.

SOMMERVILLE, Ian. Engenharia de Software. 9ª ed. São Paulo: Pearson Prentice Hall, 2011.

PAULA FILHO, Wilson de Pádua. *Engenharia de software:* fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: LTC, 2003. 602p.

ELMASRI, Ramez et al. Sistemas de banco de dados. Vários Tradutores. São Paulo: Pearson Addison Wesley, 2005. (título original: Fundamentais of Database Systems)

BAZZI, Cláudio Leones. Introdução a Banco de Dados. 2013.

HEUSER, Carlos Alberto. **Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS.** Bookman Editora, 2009.

POSTGRESQL: The World's Most Advanced Open Source Relational Database, 2021. Página Inicial. Disponível em: <https://www.postgresql.org>. Acesso em: 10 de fevereiro de 2021.

PGADMIN – PostgreSQL Tools. 2021. Disponível em: <https://www.pgadmin.org/>. Acesso em: 10 de fevereiro de 2021.

BADALOTTI, Greisse Moser. **Introdução ao Desenvolvimento de Sistemas Web.** Indaial: Uniasselvi, 2014. 226p.

NOLETO, Cairo. **Aplicações web: entenda o que são e como funcionam!** Trybe. Março 2020. Disponível em: <https://blog.betrybe.com/desenvolvimento-web/aplicacoes-web/>. Acesso em: 7 de abril de 2021.

TAYLOR OTWELL**. LARAVEL – The PHP Framework for Web Artisans**. 2021. Disponível em: <https://laravel.com/>. Acesso em: 14 de fevereiro de 2021.

LUCIANO, Josué; ALVES, Wallison Joel Barberá. Padrão de arquitetura MVC: Model-view-controller. **EPeQ Fafibe**, v. 1, n. 3a, p. 102-107, 2017.

LIVEWIRE. Livewire Quickstart. 2021. Disponível em: <https://laravel-livewire.com/docs/2.x/quickstart>. Acesso em: 16 de fevereiro de 2021.

BERGMANN, Sebastian. **PHPUnit – The PHP Testing Framework.** 2020. Disponível em: <https://phpunit.de/>. Acesso em: 19 de fevereiro de 2021.

DOCKER. **Empowering App Development for Developers | DOCKER.** 2021**.** Disponível em: <https://www.docker.com/>. Acesso em: 4 de fevereiro de 2021.

DOCKER DOCS. **Docker overview.** 2021**.** Disponível em: <https://docs.docker.com/get-started/overview/>. Acesso em: 4 de fevereiro de 2021.

VITALINO, Jeferson Fernando Noronha; CASTRO, Marcus André Nunes. **Descomplicando o Docker**. Brasport, 2016.

POSITIVO TECNOLOGIA. **Container Docker: o que é e quais são as vantagens de usar?** Postado em 4 de Dezembro de 2017. Disponível em: <https://www.meupositivo.com.br/panoramapositivo/container-docker/>. Acesso em: 21 de fevereiro de 2021.

GOOGLE INC. **WebRTC. Real-time communication for the web.** Disponível em: <https://webrtc.org/>. Acesso em: 22 de fevereiro de 2021.

BLOGGEEK. **WebRTC vs WebSockets.** Postado em 28 de Maio de 2019. Disponível em: <https://bloggeek.me/webrtc-vs-websockets/>. Acesso em: 7 de abril de 2021.

RINALDI, Lucas et al. Sistema de atendimento ao consumidor utilizando a tecnologia WebRTC. 2018.

SANTOS, Rubens Junior. (2016). Desenvolvimento de sistema de comunicação interna no

IFSP São João da Boa Vista. Trabalho de Conclusão de Curso - Instituto Federal de São Paulo, São João da Boa Vista, 2016.

ADMINLTE. AdminLTE Bootstrap Admin Dashboard Template. 2019. Disponível em: <https://adminlte.io/>. Acesso em 16 de fevereiro de 2021.