

INSTITUTO FEDERAL DE SANTA CATARINA

JOÃO LUIS MACHADO MARTINS

SISTEMA DE AGENDAMENTOS UNIFICADO

Garopaba

2023

JOÃO LUIS MACHADO MARTINS

## SISTEMA DE AGENDAMENTOS UNIFICADO

Monografia apresentada ao  
Curso Superior de  
Tecnologia em Sistemas  
para Internet do Câmpus  
Garopaba do Instituto  
Federal de Santa Catarina  
para a obtenção do diploma  
de Tecnólogo em Sistemas  
Para Internet

Orientador: Edjandir Correa  
Costa

Garopaba

2023

Catálogo na fonte pelo Instituto Federal de Educação, Ciência e  
Tecnologia de Santa Catarina - IFSC

Martins, João Luis Machado

Sistema de Agendamentos Unificado / João Luis Machado Martins ;  
orientador, Edjandir Correa Costa, 2023.

61 p.

Trabalho de Conclusão de Curso (Graduação) - Instituto Federal  
de Educação, Ciência e Tecnologia de Santa Catarina, Curso Superior de  
Tecnologia em Sistemas Para Internet, Garopaba, 2023.

Inclui referências.

1. sistema web. 2. agendamentos. 3. layouts

I. Martins, João Luis Machado. II. Costa, Edjandir Correa. III. Instituto  
Federal de Educação, Ciência e Tecnologia de Santa Catarina – Curso  
Superior de Tecnologia em Sistemas Para Internet. IV. Título.

Ficha catalográfica elaborada pelo Bibliotecário  
David Matos Milhomens – CRB-14/1268

JOÃO LUIS MACHADO MARTINS

## SISTEMA DE AGENDAMENTOS UNIFICADO

Este trabalho foi julgado adequado para obtenção do título em Tecnólogo em Sistemas Para Internet, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

Garopaba, 06 de dezembro, 2023.

---

Prof. Edjandir Correa Costa, Msc.

Orientador

Instituto Federal de Santa Catarina

---

Prof. Luiz Antonio Schalata Pacheco, Dr.

Instituto Federal de Santa Catarina

---

Prof. Ricardo de Moura Rivaldo, Msc.

Instituto Federal de Santa Catarina

## **RESUMO**

Este trabalho apresenta o desenvolvimento de um sistema web de agendamentos que unifica algumas áreas onde há necessidade de gerenciar eventos ou pessoas de maneira organizada e que atenda as especificidades de cada área. Os layouts foram escolhidos para as seguintes áreas: clínica médica, clínica veterinária, escritório de advocacia e entrevistas de emprego. O sistema terá layouts pré-definidos para cada área citada, podendo cadastrar, editar ou excluir consultas e/ou eventos, assim atendendo as necessidades mínimas de cada área. Conterá também com um banco de dados onde será possível visualizar pacientes, animais, clientes e candidatos de acordo com a área selecionada. Conterá com uma interface simples e intuitiva, um sistema de login e permissões, a fim de providenciar um gerenciamento mais seguro das informações. As tecnologias utilizadas no desenvolvimento do sistema são Java e Spring Boot para o backend, Angular 16 para o frontend e PostgreSQL para o banco de dados.

Palavras-chave: sistema web; agendamentos; layouts

## **ABSTRACT**

This paper presents the development of a web scheduling system that unifies several areas where there is a need to manage events or people in an organized manner and that meets the specific needs of each area. Layouts were chosen for the following areas: medical clinic, veterinary clinic, law firm, and job interviews. The system will have predefined layouts for each of the mentioned areas, allowing users to create, edit, or delete appointments and/or events, thus meeting the minimum needs of each area. It will also have a database where it will be possible to view patients, animals, clients, and candidates according to the selected area. It will have a simple and intuitive interface, a login and permission system, in order to provide a more secure information management. The technologies used in the development of the system are Java and Spring Boot for the backend, Angular 16 for the frontend, and PostgreSQL for the database.

Keywords: web system; scheduling; layouts

## LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo MVC(Model, View e Controller).

Figura 2 – Representação de uma estrutura básica de um projeto Angular

Figura 3 – Representação da estrutura básica de um componente Angular

Figura 4 – Representação de uma variação da estrutura básica de um componente Angular

Figura 5 – Representação de uma interpolação no HTML

Figura 6 – Representação de uma configuração de rotas no Angular

Figura 7 – Representação da definição do caminho de um componente

Figura 8 – Diagrama de Casos de Uso

Figura 9 – Estrutura de tabelas no banco de dados

Figura 10 – Estrutura que representa um agendamento para escritório advocatício, no banco de dados.

Figura 11 – Estrutura que representa um agendamento no banco de dados.

Figura 12 – Estrutura que representa um cliente no banco de dados.

Figura 13 – Estrutura que representa uma empresa no banco de dados.

Figura 14 – Diagrama de Entidade Relacionamento do banco de dados.

Figura 15 – Representação da estrutura de pastas e arquivos

Figura 16 – Representação do pacote usuario e os pacotes contidos nele

Figura 17 – Representação parcial do UsuarioController.java

Figura 18 – Representação parcial do UsuarioOut.java

Figura 19 – Representação parcial do Usuario.java

Figura 20 – Representação parcial do UsuarioRepository.java

Figura 21 – Representação parcial do UsuarioService.java

Figura 22 – Representação parcial do UsuarioOut.java

Figura 23 – Representação das rotas responsáveis por um agendamento

Figura 24 – Representação da rota de criação de agendamentos

Figura 25 – Dependências do projeto contidas no arquivo package.json

Figura 26 – Representação da Estrutura do Projeto

Figura 27 – Imagem da tela de login do sistema

Figura 28 – Imagem da Tela Inicial do sistema (usuário não cadastrado)

Figura 29 – Imagem representando o modal de cadastro de uma nova empresa

Figura 30 – Imagem representando a tela principal para usuário logado

Figura 31 – Imagem representando o nível de acesso de cada permissão

Figura 32 – Imagem representando a lista de pacientes

Figura 33 – Imagem representando a lista de funcionários

Figura 34 – Imagem representando a lista de consultas

Figura 35 – Cadastro de um novo agendamento

Figura 36 – Estrutura de dados a ser enviada ao Backend

Figura 37 – Estrutura de dados retornada do Backend

Figura 38 – Sucesso ao cadastrar um agendamento

Figura 39 – Erro ao cadastrar um agendamento

Figura 40 – Representação da estrutura de erros retornados pelo Backend

Figura 41 – Modal de criação de agendamento em escritório de advocacia

Figura 42 – Diferença entre formulários de clínica veterinária e clínica médica respectivamente

Figura 43 – Modal de atendimento de um evento



## LISTA DE ABREVIATURAS E SIGLAS

IoC – *Inversion of Control* (Inversão de Controle)

JPA – Java *Persistence* API

CRUD – *Create, Read, Update, Delete* (Criar, Ler, Atualizar, Deletar)

DTO – *Data Transfer Object* (Objeto de Transferência de Dados)

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>12</b>
1.1 Objetivo Geral.....	12
1.2 Objetivos Específicos.....	13
1.3 Metodologia.....	13
<b>2 FUNDAMENTAÇÃO TECNOLÓGICA.....</b>	<b>14</b>
2.1 Aplicação Web.....	14
2.2 Arquitetura Web (MVC).....	15
2.2.1 Model.....	15
2.2.2 View.....	15
2.2.3 Controller.....	16
2.3 APIs.....	16
2.3.1 REST.....	16
2.4 JAVA.....	17
2.4.1 JVM.....	18
2.5 Spring Boot.....	18
2.5.1 Anotações.....	19
2.5.2 Bean.....	19
2.6 Angular.....	20
2.6.1 Roteamento Angular.....	23
2.7 PostgreSQL.....	24
<b>3 SOLUÇÃO PROPOSTA.....</b>	<b>25</b>
3.1 Requisitos da Aplicação.....	25
3.1.1 Requisitos Funcionais.....	25
3.1.2 Requisitos Complementares.....	30
3.2 Casos de Uso.....	30
<b>4 IMPLEMENTAÇÃO DA SOLUÇÃO.....</b>	<b>36</b>
4.1 Modelagem de Dados.....	36
4.2 Camada de Aplicativo (backend).....	40

4.2.1 Estrutura de Pastas e Arquivos.....	40
4.2.2 Desenvolvimento da API.....	42
4.2.3 Definição das Rotas.....	48
4.2.4 Validação dos Dados.....	50
<b>4.3 Camada de Apresentação (frontend).....</b>	<b>53</b>
4.3.1 Telas do Sistema.....	54
<b>5 CONCLUSÃO.....</b>	<b>63</b>
5.1 Trabalhos Futuros.....	64
<b>REFERÊNCIAS.....</b>	<b>65</b>

## **1 INTRODUÇÃO**

O gerenciamento de agendamentos é fundamental para ramos que envolvem consultas, entrevistas ou simplesmente precisam manter os registros de seus negócios e “é um desafio enfrentado no mundo todo” (Postal, 2021). Por ser uma ferramenta essencial para uma variedade de setores, sistemas de agendamento desempenham um papel crucial na eficiência da organização. Eles oferecem uma série de benefícios que vão muito além de simplesmente marcar compromissos, mesmo quando feitos de maneira manual e física, como por exemplo em cadernetas. No entanto, fazer agendamentos de forma manual e em cadernetas pode gerar uma dificuldade no gerenciamento das tarefas, sendo assim, um sistema web de agendamentos é uma opção para aumentar a eficiência e a segurança no gerenciamento de tarefas ou eventos.

Para este trabalho então, será desenvolvido um sistema de agendamentos que atenda clínicas médicas, clínicas veterinárias, escritórios de advocacia e entrevistas de emprego chamado de AgendaWeb. Esse sistema oferecerá uma maneira eficaz de gerenciar pacientes, animais ou candidatos a emprego, facilitando o registro e a recuperação de informações detalhadas. Isso evitará a necessidade de gerenciar agendamentos manualmente e permitirá a recuperação eficiente de dados de eventos passados.

As principais características do sistema serão: um design simples e intuitivo, as opções de layout que inicialmente serão para clínica médica, clínica veterinária, escritório de advocacia e entrevista de emprego, cada uma das opções de layout terão sua tela feita e organizada pensando no ramo de trabalho. Contará com um sistema de login, onde pessoas cadastradas poderão ser convidadas a administrar o sistema, podendo cadastrar, editar, visualizar e excluir consultas de acordo com as permissões concedidas pelo dono.

### **1.1 Objetivo Geral**

Desenvolver um sistema de agendamentos que ofereça layouts pré-definidos para diferentes ramos de trabalho.

## 1.2 Objetivos Específicos

São objetivos específicos do trabalho:

- Propor uma abordagem para o agendamento de eventos e compromissos para profissionais das áreas atendidas;
- Elaborar layouts específicos para clínica médica, clínica veterinária, escritório de advocacia e entrevista de emprego visando atender às especificidades de cada área;
- Implementar um sistema de controle de permissões que permita gerenciar eventos, clientes e funcionários;
- Prover uma interface intuitiva e documentação dinâmica a fim de garantir que o usuário possa navegar com facilidade e entender todas as funções disponíveis;

## 1.3 Metodologia

Será adotada a abordagem da Engenharia de Software Waterfall, que propõe que a construção de um sistema seja feita em etapas sequenciais, sendo elas: Levantamento de Requisitos, Análise, Projeto, Codificação, Testes e Implantação (Royce, 1970).

Na etapa de Levantamento de Requisitos serão analisadas referências de sistemas de agendamentos para as áreas que serão atendidas pelo sistema. A partir dessa análise, levantar os requisitos mínimos para o desenvolvimento da aplicação.

Na etapa de Análise, os requisitos analisados serão utilizados para identificar os processos e funcionalidades necessários para atender aos requisitos.

Na etapa de Projeto, será feita a modelagem do banco de dados, definindo as tabelas e seus relacionamentos, definindo também a arquitetura e as interfaces entre os elementos do sistema.

Na etapa de Codificação, iniciar o desenvolvimento do backend, implementar os endpoints para as operações de cadastro, edição, visualização e exclusão de consultas. Após a implementação do backend, será iniciado o desenvolvimento do frontend, onde será criado o layout para cada uma das opções de ramo de trabalho

e serão desenvolvidas as telas para cada um dos layouts. Será implementado o sistema de login, para que pessoas cadastradas possam administrar o sistema de acordo com as permissões concedidas pelo dono..

Nas etapas de Testes e Implantação, será feita a integração entre o frontend e o backend, realizando testes e ajustes necessários para garantir o bom funcionamento da aplicação. Será feita a documentação do sistema, descrevendo as funcionalidades e instruções de uso.

## **2 FUNDAMENTAÇÃO TECNOLÓGICA**

Neste capítulo será abordada a base teórica dos conceitos e tecnologias utilizados na criação da solução proposta neste projeto. Serão abordados os seguintes temas: Aplicação Web, Arquitetura Web (MVC), REST, Spring Boot, Java, Angular e PostgreSQL.

### **2.1 Aplicação Web**

Uma aplicação web, como definido por Ingalls (2021) refere-se a um software ou site capaz de receber requisições dinâmicas de conteúdo e entregar recursos pela internet para clientes ou usuários, ou seja, é um software que é acessado pela internet e não depende se um sistema operacional. Uma de suas principais vantagens é o acesso e utilização do sistema por qualquer dispositivo com acesso à internet, sem a necessidade de instalação do software localmente. Também elimina a preocupação com o hardware do usuário, permitindo o acesso por meio de computadores, celulares e tablets através do navegador web do dispositivo.

Outra vantagem é a parte de manutenção e atualizações da aplicação web, permitindo que o usuário não precise se preocupar em baixar novas atualizações da aplicação, sendo feita de maneira automática.

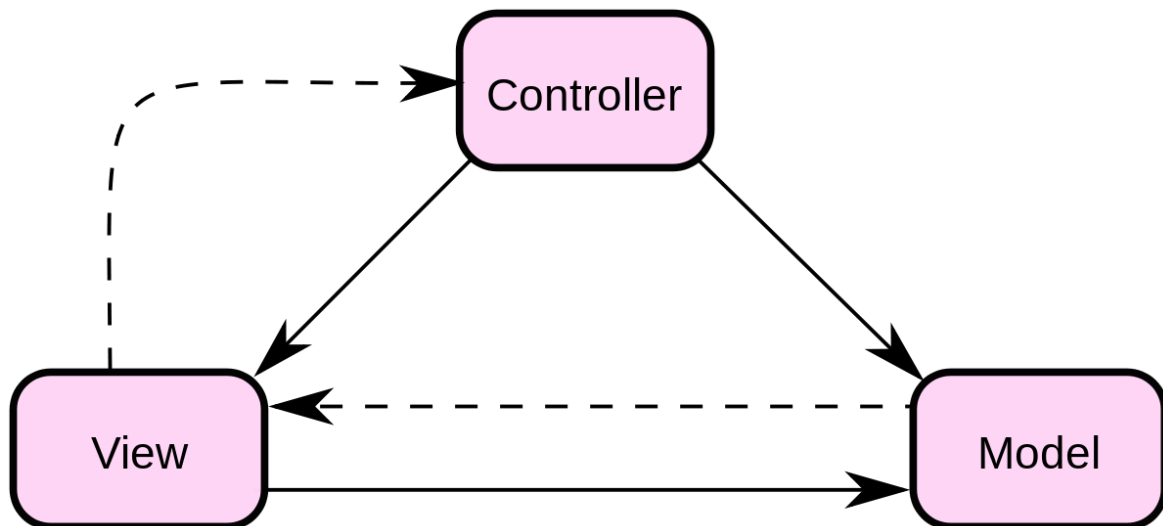
O funcionamento de aplicações Web é dividido em dois componentes: scripts do lado do cliente e scripts do lado do servidor. O lado do cliente é responsável por todas as funcionalidades da interface e suas interações. O carregamento do script do usuário acontece ao acessar o link da aplicação Web pelo navegador e com isso os elementos gráficos são renderizados para que o usuário possa interagir com a aplicação (AWS, 2023a).

Os scripts do lado do servidor são responsáveis por lidar com o processamento dos dados. O servidor processa as requisições e retorna uma resposta.

## 2.2 Arquitetura Web (MVC)

De acordo com Lemos (2013a) a arquitetura MVC é um padrão de projeto baseado em três camadas: Model, View e Controller, ou também chamadas respectivamente de camada de dados, apresentação e aplicativo. Sendo um dos modelos mais utilizados no desenvolvimento de aplicações web.

Figura 1 – Modelo MVC(Model, View e Controller).



Fonte: WIKIPEDIA (2023)

### 2.2.1 Model

Model, ou camada de dados, é responsável pelo banco de dados, onde os dados da aplicação são armazenados por meio de algum Sistema de Gerenciamento de Banco de Dados (SGBD), como o H2, MySQL, PostgreSQL, entre outros (Lemos, 2013b).

### 2.2.2 View

View, ou camada de apresentação é a camada que é apresentada ao usuário

através do navegador web para que o usuário interaja com o sistema. É normalmente composta por três partes: o conteúdo, que é criado utilizando HTML, a aparência do conteúdo que é personalizada utilizando CSS e a interação para com o conteúdo que é feita através do Javascript (Lemos, 2013c).

### 2.2.3 Controller

Controller, ou camada de aplicativo, é responsável pela implementação das regras de negócio, interação com o banco de dados e acesso a APIs. O desenvolvimento dessa camada pode ser feito utilizando diversas linguagens de programação, como por exemplo, Java, Python, PHP, entre outras (Lemos, 2013d).

## 2.3 APIs

API, do inglês Application Programming Interface, é uma interface de programação de aplicações. Estas interfaces são conjuntos de ferramentas, definições e protocolos para a criação de aplicações de software. As APIs funcionam como se fossem contratos, com documentações que representam um acordo entre as partes interessadas (Red Hat, 2023). Resumindo, através de APIs é possível liberar o acesso aos recursos criados mantendo o controle e a segurança da aplicação.

### 2.3.1 REST

Representational State Transfer, ou REST, é uma arquitetura utilizada para fornecer padrões entre sistemas web, facilitando a comunicação entre eles. A implementação desta arquitetura gera a independência entre o cliente e o servidor e que um não precise conhecer o outro, assim permitindo alterações no código do lado do cliente sem que precise de alterações no lado do servidor e vice-versa. Sistemas que utilizam a arquitetura REST são chamados de sistemas RESTful (Souza, 2020a).

Um dos conceitos importantes para o REST é o protocolo HTTP, que define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para um dado recurso (Mozzila, 2023). Segundo Mariano (2022) esses



métodos, ou comumente chamados de Verbos HTTP, são :

- GET: responsável por solicitar a representação de um recurso específico e deve retornar apenas dados.
- POST: utilizado para enviar dados a um determinado recurso. É convencional utilizar este verbo para criação de novos recursos.
- HEAD: parecido com o GET, porém não necessita do corpo da resposta.
- PUT: por convenção utiliza-se este verbo para substituir ou atualizar um recurso com novos dados.
- DELETE: apaga um recurso;
- CONNECT: estabelece um túnel de conexão para um servidor com base em um recurso.
- OPTIONS: Utilizado para definir as opções para comunicação com um determinado recurso.
- TRACE: Realiza um teste de loopback para verificar se uma mensagem consegue chegar a um determinado destino.
- PATCH: Aplica modificações parciais em um determinado recurso.

## **2.4 JAVA**

Java é uma linguagem de programação extremamente consolidada no mercado e segundo uma enquete do Stack Overflow (2023), mesmo depois de 20 anos no mercado, Java continua sendo uma das linguagens de programação mais utilizadas profissionalmente e também para pessoas que estão aprendendo a programar. Segundo Java (2023), a linguagem de programação e plataforma liberada em 1995 pela Sun Microsystems, oferece a plataforma no qual diversos serviços e aplicativos são desenvolvidos.

De acordo com AWS (2023b), Java é uma linguagem multiplataforma, orientada a objetos, rápida, segura e confiável. A linguagem é popular por ter sido projetada para ser executada em qualquer ambiente ou sistema operacional que possua compatibilidade com a JVM (Java Virtual Machine), permitindo assim que as aplicações não precisem ser recompiladas.

### **2.4.1 JVM**

Como o Java foi a linguagem de programação escolhida para ser utilizada,

cabe apresentar o conceito de JVM (Java Virtual Machine).

Segundo IBM:

A JVM é um mecanismo de cálculo interpretativo responsável pela execução dos códigos de byte em um programa Java compilado. O JVM converte os códigos de byte Java nas instruções nativas da máquina host. O servidor de aplicativos, sendo um processo Java, requer uma JVM para ser executado e para suportar os aplicativos Java nele executados. As definições de JVM fazem parte da configuração de um servidor de aplicativos (IBM, 2023).

A JVM nos permite desenvolver uma aplicação e executá-la em qualquer sistema operacional que possua uma JVM instalada.

## 2.5 Spring Boot

Segundo a IBM (2023) Java Spring Framework é um framework de nível empresarial utilizado no desenvolvimento de aplicativos que são executados na Java Virtual Machine (JVM), sendo então, compatíveis com qualquer sistema operacional que tenha suporte a JVM. Enquanto o Spring Boot é uma ferramenta desenvolvida para facilitar e agilizar a criação de aplicativos web a partir de três principais recursos: autoconfiguração, abordagem opinativa e a capacidade de criar aplicativos independentes.

Ainda segundo a IBM (2023), autoconfiguração é um recurso que permite que as dependências e configurações necessárias para um aplicativo sejam definidas automaticamente e evita a necessidade de configurá-las manualmente. Na abordagem opinativa, é possível definir as dependências de iniciador, ou *Spring Starters*, tornando o Spring Boot mais flexível e adaptável a diferentes necessidades. Existe uma ferramenta web chamada de Spring Boot Initializr, que através de um formulário é possível definir diversas configurações para o projeto, incluindo as dependências (bibliotecas de terceiros) iniciais.

A escolha do Spring Boot como ferramenta para o desenvolvimento do Backend está relacionada à inúmeras funcionalidades disponibilizadas através de anotações (Annotations), sem a necessidade de implementações complementares.

### 2.5.1 Anotações

As anotações são um recurso utilizado pelo Spring Boot e são utilizadas para

anotar (identificar) métodos, classes e campos para que o compilador faça o tratamento que a anotação define (Manzano, 2020). Existem diversas anotações para diversas situações, por exemplo (Souza, 2016):

Tabela 1 - Descrição de algumas anotações Spring Boot

Anotações com escopo de classe	
@Configuration	Indica que a classe é uma fonte de criação e/ou configuração de Beans
@Controller	Associada às classes que possuem métodos que processam requests numa aplicação web. Num projeto, normalmente ficam dentro da pasta Controller. O mesmo vale para Repository e Service
@Repository	Associada a classes que manipulam os dados através de métodos
@Service	Associada com classes que representam algum fluxo de negócio da sua aplicação
Anotações com escopo de métodos	
@Bean	Utilizada acima de métodos de classes marcadas com a anotação @Configuration, indica ao Spring que ele deve invocar o método e gerenciar os objetos retornados por eles

Fonte: Elaborado pelo Autor, 2023

Existem diversas outras, que serão abordadas à medida que aparecerem no desenvolvimento da aplicação.

### 2.5.2 Bean

Para definir o que é um Bean, primeiro é necessário falar sobre outros dois conceitos, a Injeção de Dependências e Inversão de Controle. Sobre a injeção de dependências, segundo Rosa (2022): “transferir a tarefa de criação do objeto a outra entidade e usar diretamente a dependência é chamado de injeção de dependência”

Com relação a inversão de controle, de acordo com Ramos:

Inversão de controle é algo muito utilizado por frameworks de injeção de dependência e isso se dá pela forma como esses frameworks são feitos. Esses frameworks geralmente trabalham criando uma espécie de "container" que é responsável por armazenar as instâncias das dependências das aplicações. Após instanciar todas as funções e objetos

necessários, a aplicação é, de fato, executada pelo próprio framework. (Ramos, 2021).

Após compreender estes dois conceitos, fica mais fácil de entender o que é um Bean, que, de acordo com Almeida (2023) “é um objeto que é instanciado, montado e gerenciado por um Spring IoC container”.

Um exemplo de Bean, seria uma classe anotada com `@Service`, que possui uma regra de negócio. Essa classe (Bean) é criada e gerenciada pelo container e pode ser “injetada” em outra classe que também seja gerenciada pelo container. Por exemplo, uma classe anotada com `@RestController`, poderia receber a dependência da classe de serviço, no seu construtor ou por outra forma de injeção de dependência, sem a necessidade de saber como a classe de serviço é criada.

## 2.6 Angular

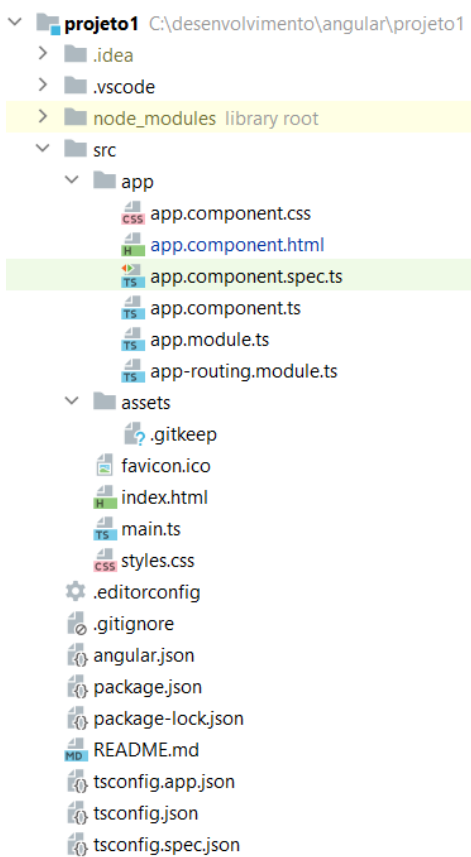
Angular é um framework JavaScript utilizado para criação de SPA (Single Page Applications, ou Aplicações de Página Única) e tem como finalidade proporcionar as ferramentas necessárias para criação de aplicações. Se utiliza da estrutura MVC através de outra estrutura chamada Two-Way Data Binding, permitindo a atualização automática da view sempre que um model for alterado (Guedes, 2023a).

A ferramenta é baseada em JavaScript através do TypeScript, é multi plataforma, podendo ser utilizada para criação de aplicações web, mobile e desktop. Segundo Guedes (2023b), sua alta produtividade se dá através de uma “API simples, bem estruturada e documentada”.

Aplicações construídas em Angular podem ser divididas em componentes, permitindo a separação de funcionalidades e reutilização de componentes. É possível também separar o HTML do Javascript, deixando o código mais legível.

A estrutura básica de um projeto Angular pode ser visto na Figura 2.

Figura 2 – Representação de uma estrutura básica de um projeto Angular



Fonte: Elaborado pelo Autor, 2023

No arquivo `app.module.ts` está a classe do módulo principal do Angular, que é o responsável por inicializar o componente principal da aplicação, localizado no arquivo `app.component.ts`. A partir do componente principal, é possível fazer a injeção de novos componentes.

Um componente básico em Angular pode ser visto na figura 3:

Figura 3 – Representação da estrutura básica de um componente Angular

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Fonte: Angular (2022)

Neste exemplo, produzido por Angular (2022), o html é introduzido no código javascript, dentro da variável 'template'. Essa abordagem é interessante quando a quantidade de código HTML é pequena. Havendo a necessidade de um código HTML mais complexo, recomenda-se a separação do código HTML do código Javascript. Para isso, muda-se a estrutura da anotação @Component, para utilizar a variável templateUrl.

Figura 4 – Representação de uma variação da estrutura básica de um componente Angular

```
import { Component } from '@angular/core';

@Component ({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-interpolation.component.html'
})
export class HelloWorldInterpolationComponent {
  message = 'Hello, World!';
}
```

Fonte: Angular (2022)

Na variável templateUrl, apresentado na Figura 4, indica-se o arquivo HTML que será renderizado pelo Angular ao carregar a página.

O conteúdo do arquivo "hello-world-interpolation.component.html" pode ser visto na Figura 5.

Figura 5 – Representação de uma interpolação no HTML

```
<p>{{ message }}</p>
```



Fonte: Angular (2022)

Quando a aplicação carrega o componente, o Angular fará a interpolação do conteúdo da variável javascript “message” na variável html de mesmo nome, que está indicada pela expressão “{{}}” (chaves duplas), dentro do arquivo HTML. Assim como o HTML pode ser separado do arquivo Javascript, é possível também separar o arquivo de estilos (CSS), permitindo a utilização de classes css para estilização da interface, de forma separada.

### 2.6.1 Roteamento Angular

Em um aplicativo de página única, você altera o que o usuário vê, mostrando ou ocultando partes da exibição que correspondem a componentes específicos, em vez de ir ao servidor para obter uma nova página. À medida que os usuários executam tarefas de aplicativo, eles precisam se mover entre as diferentes visualizações definidas.

Para lidar com a navegação de uma visualização para a próxima, você usa o Angular Router. O roteador permite a navegação interpretando a URL do navegador como uma instrução para alterar a visualização.

As rotas podem ser definidas no arquivo `app-routing.module.ts` (`AppRoutingModule`).

Figura 6 – Representação de uma configuração de rotas no Angular

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router'; // CLI imports router

const routes: Routes = []; // Sets up routes constant where you define your routes

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Fonte: Angular (2022)

Figura 7 – Representação da definição do caminho de um componente

```
// Sets up routes constant where you define your routes
const routes: Routes = [{
  path: 'first-component', component: FirstComponent
}, {
  path: 'second-component', component: SecondComponent
}];
```

Fonte: Angular (2022)

A Figura 7 apresenta a definição de duas rotas, que permitem a navegação para dois componentes diferentes.

A forma como o Angular utiliza para fazer a navegação entre componentes, é a inclusão do valor atribuído à variável “path”, na URL do browser. No exemplo da Figura 7, pode-se navegar para o componente FirstComponent, digitando na URL do browser “http://localhost:4200/first-component” e pressionando “Enter”.

## 2.7 PostgreSQL

O PostgreSQL é uma ferramenta de gerenciamento de banco de dados relacionais. Tem como papel gerenciar os dados de maneira organizada e eficiente, garantindo clareza na recuperação dos dados gravados. Permite que sejam feitas consultas sem precisar acessar diretamente o banco de dados, fazendo com que as consultas de dados sejam mais simples (Souza, 2020b).

Segundo Souza (2020c), um de seus pontos principais é sua adequação para padrões de conformidade, ajudando a otimizar o banco de dados e permitindo armazenar informações de maneira segura.



### 3 SOLUÇÃO PROPOSTA

A solução proposta neste trabalho será o desenvolvimento de uma aplicação web, com o nome de AgendaWeb. Ela permitirá o gerenciamento de tarefas e eventos a partir de um calendário, cadastro de clientes e funcionários e o “atendimento” de um evento.

#### 3.1 Requisitos da Aplicação

Nesta seção serão apresentados requisitos funcionais e não funcionais da aplicação. Assim, sendo possível seguir o desenvolvimento de todas as funcionalidades e necessidades da aplicação.

##### 3.1.1 Requisitos Funcionais

Tabela 2 - Descrição do Requisito Funcional - Cadastrar Empresa

RF1 - Cadastrar Empresa		
Descrição	O sistema deve permitir que um usuário realize o cadastro de uma empresa.	
Usuário	Usuário não autenticado	
Entrada	Os dados gerais da empresa e do usuário administrador	
Saída	Após o processo de cadastro, o usuário informado deverá ser automaticamente definido como administrador da empresa.	
Requisitos Não Funcionais		Categoria
RNF1.1	O usuário administrador terá privilégios para gerenciar a empresa, incluindo a adição, remoção e edição de outros usuários, agendamentos, clientes e consultas, associados a essa empresa	Especificação

Fonte: Elaborado pelo Autor, 2023

Tabela 3 - Descrição do Requisito Funcional - Autenticação

RF2 - Autenticação		
Descrição	O sistema deve possuir uma forma de autenticação do usuário.	
Usuário	Usuário não autenticado	
Entrada	CPF e senha	
Saída	Acesso ao sistema	
Requisitos Não Funcionais		Categoria
RNF2.1	Utilizar o conceito de token para fazer a autenticação do usuário	Segurança
RNF2.2	Após 24 dias o Token expira, exigindo novamente o login do usuário.	Segurança
RNF2.3	O sistema deve identificar o ramo de atividade a partir do usuário logado para redirecioná-lo para a tela referente ao ramo de atividade da empresa do usuário	Especificação

Fonte: Elaborado pelo Autor, 2023

Tabela 4 - Descrição do Requisito Funcional - Cadastrar Funcionários

RF3 - Cadastrar Funcionários		
Descrição	O usuário com perfil de administrador poderá cadastrar novos funcionários e delegar suas devidas permissões (recepcionista e operacional).	
Usuário	Usuário com permissão de administrador	
Entrada	Dados do novo funcionário	
Saída	Não há saída	
Requisitos Não Funcionais		Categoria
RNF3.1	Mensagem de confirmação	Interface

Fonte: Elaborado pelo Autor, 2023

Tabela 5 - Descrição do Requisito Funcional - Definição de Senha

RF4 - Definição de Senha	
Descrição	Ao cadastrar um novo funcionário, o sistema deve enviar a senha para o email do funcionário
Usuário	Usuário com permissão de administrador

Entrada	Não há entrada	
Saída	Não há saída	
Requisitos Não Funcionais		Categoria
RNF4.1	O serviço de email estar disponível	Especificação

Fonte: Elaborado pelo Autor, 2023

Tabela 6 - Descrição do Requisito Funcional - Acessar o Sistema

RF5 - Acessar o Sistema		
Descrição	Os usuários cadastrados devem conseguir acessar o sistema através do CPF e uma senha.	
Usuário	Usuário cadastrado no sistema	
Entrada	CPF e senha	
Saída	Acesso ao sistema	
Requisitos Não Funcionais		Categoria
RNF5.1	Redirecionar para tela principal do sistema	Interface

Fonte: Elaborado pelo Autor, 2023

Os requisitos a seguir estão associados à Empresa do usuário logado. O usuário administrador poderá executar todos os requisitos abaixo, mesmo que na descrição do requisito, a permissão não esteja descrita.

Tabela 7 - Descrição do Requisito Funcional - Cadastrar Eventos no Calendário

RF6 - Cadastrar Eventos no Calendário		
Descrição	O usuário poderá cadastrar novos eventos na agenda do sistema (dashboard) e visualizá-los.	
Usuário	Usuário com permissão de “recepcionista”.	
Entrada	Nome do cliente, nome do profissional que irá atendê-lo, hora de início e de fim, com duração mínima de 20 minutos	
Saída	Evento marcado na dashboard	
Requisitos Não Funcionais		Categoria

RNF6.1	Mensagem de confirmação	Interface
RNF6.2	O sistema deve garantir que não haverá conflito de horários (início e fim) entre agendamentos com o mesmo “Profissional”	Especificação

Fonte: Elaborado pelo Autor, 2023

Tabela 8 - Descrição do Requisito Funcional - Cadastrar Clientes

RF7 - Cadastrar Clientes		
Descrição	O usuário com permissão “recepcionista” poderá cadastrar novos clientes.	
Usuário	Usuário com permissão de “recepcionista”	
Entrada	Dados do novo cliente	
Saída	Não há saída	
Requisitos Não Funcionais		Categoria
RNF7.1	Mensagem de confirmação	Interface

Fonte: Elaborado pelo Autor, 2023

Tabela 9 - Descrição do Requisito Funcional - Visualizar e Editar Clientes Cadastrados

RF8 - Visualizar e Editar Clientes Cadastrados		
Descrição	O usuário poderá ver e editar os clientes cadastrados.	
Usuário	Usuário com permissão de “recepcionista”	
Entrada	Novos dados do cliente	
Saída	Cliente atualizado	
Requisitos Não Funcionais		Categoria
RNF8.1	Mensagem de confirmação	Interface
RNF8.2	Não é possível editar o CPF do cliente	Segurança

Fonte: Elaborado pelo Autor, 2023

Tabela 10 - Descrição do Requisito Funcional - Visualizar e Editar os Dados dos Atendimentos

RF9 - Visualizar e Editar os Dados dos Atendimentos		
Descrição	O usuário com permissão “recepcionista” poderá ver e editar os eventos cadastrados na dashboard.	
Usuário	Usuário com permissão de “recepcionista”	
Entrada	Novos dados do evento	
Saída	Evento atualizado	
Requisitos Não Funcionais		Categoria
RNF9.1	Mensagem de confirmação	Interface
RNF9.2	Cada evento tem uma duração mínima de 20 minutos.	Especificação

Fonte: Elaborado pelo Autor, 2023

Tabela 11 - Descrição do Requisito Funcional - Observações Sobre o Evento

RF10 - Atendimento de Evento		
Descrição	O usuário com permissão "Operacional" poderá atender a um evento, colocando observações sobre ele.	
Usuário	Usuário com permissão de “operacional”	
Entrada	Observações do evento	
Saída	Não há saída	
Requisitos Não Funcionais		Categoria
RNF10.1	Para clientes com eventos anteriores, o sistema deve trazer o histórico de observações.	Especificação

Fonte: Elaborado pelo Autor, 2023

Tabela 12 - Descrição do Requisito Funcional - Horário do Agendamento

RF11 - Horário do Agendamento	
Descrição	O agendamento só pode acontecer dentro do horário comercial (08:00 às 18:00).
Usuário	Usuário com permissão de “recepcionista”
Entrada	Horários entre 08:00 e 18:00
Saída	Não há saída

Requisitos Não Funcionais		Categoria
RNF11.1	Mensagem de erro caso fora do horário estipulado	Interface

Fonte: Elaborado pelo Autor, 2023

Tabela 13 - Descrição do Requisito Funcional - Conflitos no Agendamento

RF12 - Conflitos no Agendamento		
Descrição	O agendamento só pode acontecer caso não haja conflitos de horário envolvendo o mesmo funcionário.	
Usuário	Usuário com permissão de “recepcionista”	
Entrada	Horários entre 08:00 e 18:00	
Saída	Não há saída	
Requisitos Não Funcionais		Categoria
RNF12.1	Mensagem de erro	Interface

### 3.1.2 Requisitos Complementares

Tabela 14 - Descrição dos Requisitos Não Funcionais

RC1 - Segurança	
Descrição	Os dados de agendamento e informações pessoais dos usuários devem ser protegidos com criptografia e medidas de segurança adequadas
RC2 - Usabilidade	
Descrição	A interface do usuário deve ser intuitiva e de fácil navegação, de modo que os usuários possam agendar eventos com facilidade

Fonte: Elaborado pelo Autor, 2023

## 3.2 Casos de Uso

De acordo com os levantamentos dos requisitos funcionais, foram identificados os casos de uso da ferramenta. Estes serão apresentados juntamente com os atores envolvidos, que foram identificados como o Usuário Logado (funcionário) e o Administrador. Os casos de uso apresentados foram descritos com apenas um dos ramos de trabalho em mente, porém funcionam da mesma maneira para todos. A descrição de todos os casos de uso estão relacionadas à clínicas

médicas. O termo “consultas” vai variar de acordo com o ramo de atividade, podendo ser “entrevistas”, “reuniões”, etc..

Tabela 15 - Descrição do Caso de Uso - Criação de uma nova conta

Caso de Uso 1 - Criação de uma nova conta	
Atores	Usuário não cadastrado
Fluxo Principal	1 - O usuário clica no botão “Não tenho cadastro” 2 - O usuário clica no botão “Cadastrar empresa” 3 - O usuário preenche todos os dados necessários 4 - O usuário clica em salvar 5 - O sistema valida as informações e confirma o cadastro
Tratamento de exceções	4a - Erro no salvamento 4a1 - O sistema apresenta o erro 4a2 - O usuário corrige o erro e salva 4a3 - Volta ao passo 5

Fonte: Elaborado pelo Autor, 2023

Tabela 16 - Descrição do Caso de Uso - Cadastro de Cliente

Caso de Uso 2 - Cadastro de Cliente	
Atores	Usuário com permissão de “Administrador” ou “Recepcionista”
Fluxo Principal	1 - O usuário clica no menu lateral “Clientes” 2 - O sistema apresenta a lista de clientes cadastrados 3 - O usuário clica no botão “Novo Cliente” 4 - O usuário preenche todos os dados necessários 5 - O usuário clica em salvar 6 - O sistema valida as informações e confirma o cadastro
Tratamento de exceções	5a - Erro no salvamento 5a1 - O sistema apresenta o erro 5a2 - O usuário corrige o erro e salva 5a3 - Volta ao passo 6

Fonte: Elaborado pelo Autor, 2023

Tabela 17 - Descrição do Caso de Uso - Edição de Cliente

Caso de Uso 3 - Edição de Cliente	
Atores	Usuário com permissão de “Administrador” ou “Recepcionista”
Fluxo Principal	1 - O usuário clica no menu lateral “Clientes” 2 - O sistema apresenta a lista de clientes cadastrados 3 - O usuário clica no botão de edição representado por um botão verde com ícone de lápis, na coluna “Ações”

	4 - O sistema apresenta um modal com os dados do cliente selecionado 5 - O usuário adiciona ou edita os dados no modal 6 - O usuário clica em salvar 7 - O sistema valida as informações e confirma a edição
Tratamento de exceções	6a - Erro no salvamento 6a1 - O sistema apresenta o erro 6a2 - O usuário corrige o erro e salva 6a3 - Volta ao passo 7

Fonte: Elaborado pelo Autor, 2023

Tabela 18 - Descrição do Caso de Uso - Ativação e Inativação de Cliente

Caso de Uso 4 - Ativação e Inativação de Cliente	
Atores	Usuário com permissão de "Administrador"
Fluxo Principal	1 - O usuário clica no menu lateral "Clientes" 2 - O sistema apresenta a lista de clientes cadastrados 3 - O usuário clica no botão de ativação/inativação representado por um botão com ícone de cadeado, na coluna "Ações" 4 - O sistema apresenta modal de confirmação 5 - O usuário clica no botão "Sim" 6 - O sistema ativa/inativa o cliente, alterando o ícone conforme a situação do cliente
Fluxo alternativo	5 - O usuário clica no botão "Não" 5.1 - O sistema fecha o modal de confirmação 5.2 - O sistema fecha o modal com os dados do agendamento 5.3 - Volta ao passo 2

Fonte: Elaborado pelo Autor, 2023

Tabela 19 - Descrição do Caso de Uso - Cadastro de Funcionário

Caso de Uso 5 - Cadastro de Funcionário	
Atores	Usuário com permissão de "Administrador"
Fluxo Principal	1 - O usuário clica no menu lateral "Funcionários" 2 - O sistema apresenta a lista de funcionários cadastrados 3 - O usuário clica no botão "Novo Funcionário" 4 - O usuário preenche todos os dados necessários 5 - O usuário clica em salvar 6 - O sistema valida as informações e confirma o cadastro 7 - O sistema envia a senha para o email do funcionário
Tratamento de exceções	5a - Erro no salvamento 5a1 - O sistema apresenta o erro 5a2 - O usuário corrige o erro e salva 5a3 - Volta ao passo 6



Fonte: Elaborado pelo Autor, 2023

Tabela 20 - Descrição do Caso de Uso - Edição de Funcionário

Caso de Uso 6 - Edição de Funcionário	
Atores	Usuário com permissão de “Administrador”
Fluxo Principal	1 - O usuário clica no menu lateral “Funcionários” 2 - O sistema apresenta a lista de funcionários cadastrados 3 - O usuário clica no botão de edição representado por um botão verde com ícone de lápis, na coluna “Ações” 4 - O sistema apresenta um modal com os dados do funcionário selecionado 5 - O usuário adiciona ou edita os dados no modal 6 - O usuário clica em salvar 7 - O sistema valida as informações e confirma a edição
Tratamento de exceções	6a - Erro no salvamento 6a1 - O sistema apresenta o erro 6a2 - O usuário corrige o erro e salva 6a3 - Volta ao passo 7

Fonte: Elaborado pelo Autor, 2023

Tabela 21 - Descrição do Caso de Uso - Ativação e Inativação de Funcionário

Caso de Uso 7 - Ativação e Inativação de Funcionário	
Atores	Usuário com permissão de “Administrador”
Fluxo Principal	1 - O usuário clica no menu lateral “Funcionários” 2 - O sistema apresenta a lista de funcionários cadastrados 3 - O usuário clica no botão de ativação/inativação representado por um botão com ícone de cadeado, na coluna “Ações” 4 - O sistema apresenta modal de confirmação 5 - O usuário clica no botão “Sim” 6 - O sistema ativa/inativa o funcionário, alterando o ícone conforme a situação do cliente
Fluxo alternativo	5 - O usuário clica no botão “Não” 5.1 - O sistema fecha o modal de confirmação 5.2 - O sistema fecha o modal com os dados do agendamento 5.3 - Volta ao passo 2

Fonte: Elaborado pelo Autor, 2023

Tabela 22 - Descrição do Caso de Uso - Cadastro de Evento

Caso de Uso 8 - Cadastro de Evento	
Atores	Usuário com permissão de “Administrador” ou “Recepcionista”
Fluxo Principal	1 - O usuário clica no menu lateral “Dashboard”

	2 - O sistema apresenta o calendário com os agendamentos do mês 3 - O usuário clica em um dos dias presentes no calendário 4 - O sistema apresenta um modal para preenchimento 5 - O usuário preenche os dados necessários 6 - O usuário clica em salvar 7 - O sistema valida as informações e confirma o cadastro
Fluxo alternativo	5 - O cliente não existe 5.1 - O usuário clica no botão de inclusão de novo cliente, representado por um botão com ícone "+" 5.1 - Ponto de extensão: CDU-2 "Cadastro de cliente" 5.2 - Volta ao passo 6
Tratamento de exceções	6a - Erro no salvamento 6a1 - O sistema apresenta o erro 6a2 - O usuário corrige o erro e salva 6a3 - Volta ao passo

Fonte: Elaborado pelo Autor, 2023

Tabela 23 - Descrição do Caso de Uso - Cancelamento de Agendamento

Caso de Uso 9 - Cancelamento de Agendamento	
Atores	Usuário com permissão de "Administrador" ou "Recepcionista"
Fluxo Principal	1 - O usuário clica no menu lateral "Dashboard" 2 - O sistema apresenta o calendário com os agendamentos do mês 3 - O usuário clica sobre o agendamento 4 - O sistema apresenta um modal com os dados do agendamento 5 - O usuário clica no botão "Excluir" 4 - O sistema apresenta modal de confirmação 5 - O usuário clica no botão "Sim" 6 - O sistema exclui o agendamento
Fluxo alternativo	5 - O usuário clica no botão "Não" 5.1 - O sistema fecha o modal de confirmação 5.2 - O sistema fecha o modal com os dados do agendamento 5.3 - Volta ao passo 2
Tratamento de exceções	6 - Dia do agendamento anterior a data atual 6.1 - O sistema apresenta mensagem, informando que não é permitido excluir consultas já passadas 6.2 Volta ao passo 2

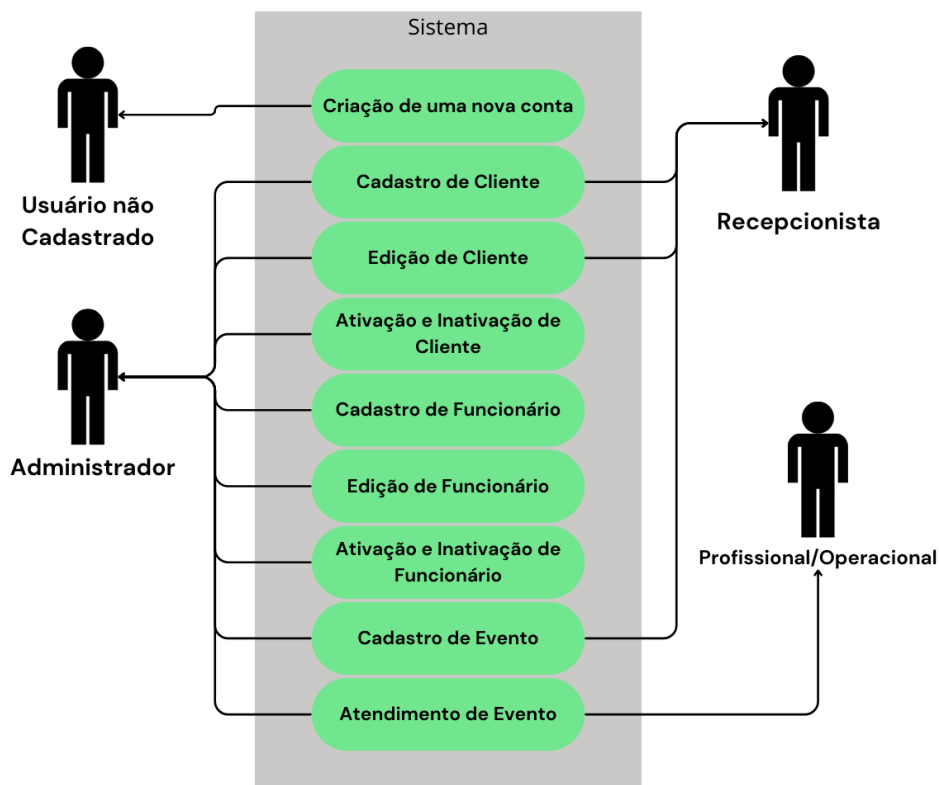
Fonte: Elaborado pelo Autor, 2023

Tabela 24 - Descrição do Caso de Uso - Atendimento de Evento

Caso de Uso 10 - Atendimento de Evento	
Atores	Usuário com permissão de “Administrador” e “Operacional”
Fluxo Principal	1 - O usuário clica no menu lateral “Consultas” 2 - O sistema apresenta a lista de consultas cadastradas 3 - O usuário clica no botão de edição representado por um botão verde com ícone de lápis, na coluna “Ações” 4 - O usuário preenche as informações necessárias no campo “Observações” 5 - O usuário clica em salvar 6 - O sistema valida as informações e confirma o cadastro
Tratamento de exceções	5a - Erro no salvamento 5a1 - O sistema apresenta o erro 5a2 - O usuário corrige o erro e salva 5a3 - Volta ao passo 4

Fonte: Elaborado pelo Autor, 2023

Figura 8 – Diagrama de Casos de Uso



Fonte: Elaborado pelo Autor, 2023










## 4 IMPLEMENTAÇÃO DA SOLUÇÃO

A implementação da solução está dividida em 3 partes: a modelagem de dados, onde será demonstrada a estrutura do banco de dados e suas tabelas; o desenvolvimento da API REST (backend), onde será apresentada a estrutura principal da aplicação; e, por fim, o desenvolvimento da interface do sistema (frontend).

### 4.1 Modelagem de Dados

Utilizando a ferramenta de banco de dados PostgreSQL, foi criada a estrutura de dados apresentada na Figura 9:

Figura 9 – Estrutura de tabelas no banco de dados.

```
>  agendamento
>  agendamento_advocaticio
>  cliente
>  cliente_veterinaria
>  empresa
>  especie_cliente_veterinaria
>  historico_consulta
>  tutor
>  usuario
```

Fonte: Elaborado pelo Autor, 2023

- agendamento: Tabela que armazena os dados de um agendamento. Sua estrutura é apresentada na Figura 11;
- agendamento\_advocaticio: Esta tabela é a complementação da tabela agendamento. Ela armazena os dados adicionais, relacionados a um escritório de advocacia. A Figura 10, apresenta a sua estrutura;
- cliente: Tabela que armazena os dados de um cliente e está relacionada a um paciente de um clínica médica, um cliente de escritório de advocacia, etc.. A sua estrutura é apresentada na Figura 12;
- cliente\_veterinaria: Assim como a tabela agendamento\_advocaticio, esta tabela é a complementação da tabela cliente, com os dados adicionais de um cliente de clínica veterinária;

- empresa: Tabela que armazena os dados de uma empresa. A sua estrutura pode ser vista na Figura 13;
- especie\_cliente\_veterinaria: Tabela utilizada para armazenar a lista de espécies de um cliente de clínica veterinária, como: cachorro, gato, ave, etc..
- historico\_consulta: Esta tabela é utilizada para armazenar o histórico de consultas/atendimentos dos pacientes/clientes;
- tutor: representa um tutor cadastrado no sistema e está relacionado ao cliente de uma clínica veterinária.
- usuario: representa um usuário cadastrado no sistema;

A abordagem utilizada para armazenar as diferentes características entre os ramos de atividades que o projeto se propõe a fazer, foi a criação de tabelas genéricas para os dados comuns entre os ramos de atividades e a criação de tabelas específicas, para armazenar os dados que são específicos para cada ramo. Um exemplo dessa abordagem é a tabela `agendamento_advocaticio` (Figura 10), que é uma especialização da tabela `agendamento` (Figura 11). Desta forma, diminui-se a necessidade de criação de campos que ficarão nulos, para determinados ramos de atividade.

Figura 10 – Estrutura que representa um agendamento para escritório advocatício, no banco de dados.

```
CREATE TABLE public.agendamento_advocaticio (
    comarca varchar(150) NULL,
    numero_processo varchar(100) NULL,
    tipo_compromisso varchar(25) NOT NULL,
    id int8 NOT NULL,
    CONSTRAINT agendamento_advocaticio_pkey PRIMARY KEY (id),
    CONSTRAINT agendamento_advocaticio_tipo_compromisso_check
        CHECK (((tipo_compromisso)::text = ANY ((ARRAY['AUDIENCIA'::character varying,
            'ANALISE_PROCESSO'::character varying, 'DILIGENCIA'::character varying,
            'REUNIAO'::character varying])::text[]))),
    CONSTRAINT fkmdsbdw3j9w02xof82ua2paoh6 FOREIGN KEY (id) REFERENCES public.agendamento(id)
);
```

Fonte: Elaborado pelo Autor, 2023

Figura 11 – Estrutura que representa um agendamento no banco de dados.

```
CREATE TABLE public.agendamento (
  id int8 NOT NULL,
  data_final timestamp(6) NULL,
  data_inicial timestamp(6) NOT NULL,
  observacao text NULL,
  situacao varchar(25) NULL,
  cliente_id int8 NOT NULL,
  empresa_id int8 NOT NULL,
  especialista_id int8 NOT NULL,
  CONSTRAINT agendamento_pkey PRIMARY KEY (id),
  CONSTRAINT agendamento_situacao_check CHECK (((situacao)::text = ANY
  (ARRAY['ATIVO'::character_varying, 'INATIVO'::character_varying])::text[])),
  CONSTRAINT fk7li2ygoege5sw546pmttx416f FOREIGN KEY (especialista_id) REFERENCES public.usuario(id),
  CONSTRAINT fkg8d6k5uh7vufobjel30xtr7p FOREIGN KEY (empresa_id) REFERENCES public.empresa(id),
  CONSTRAINT fksqdo4l8yts964f089m6ujyuef FOREIGN KEY (cliente_id) REFERENCES public.cliente(id)
);
```

Fonte: Elaborado pelo Autor, 2023

A seguir, são apresentadas algumas figuras com exemplos de outras estruturas de dados criadas para armazenamento de dados do sistema.

Figura 12 – Estrutura que representa um cliente no banco de dados.

```
CREATE TABLE public.cliente (
  id int8 NOT NULL,
  bairro varchar(80) NULL,
  celular varchar(11) NULL,
  cep varchar(8) NULL,
  cidade varchar(60) NULL,
  complemento varchar(100) NULL,
  cpf varchar(11) NOT NULL,
  data_cadastro timestamp(6) NOT NULL,
  data_nascimento date NULL,
  email text NULL,
  st_inativo bool NULL,
  nome varchar(250) NOT NULL,
  numero varchar(40) NULL,
  rua varchar(120) NULL,
  empresa_id int8 NOT NULL,
  CONSTRAINT cliente_pkey PRIMARY KEY (id),
  CONSTRAINT fkkbui05oidjdj4nb0283u4t319 FOREIGN KEY (empresa_id) REFERENCES public.empresa(id)
);
```

Fonte: Elaborado pelo Autor, 2023

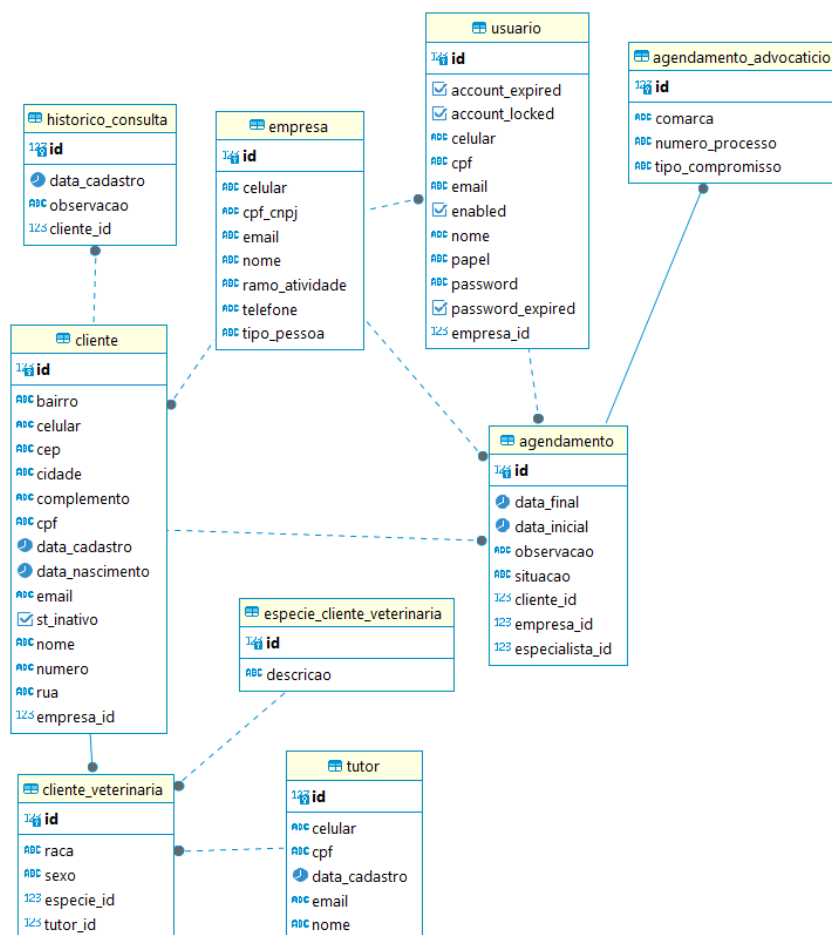
Figura 13 – Estrutura que representa uma empresa no banco de dados.

```
CREATE TABLE public.empresa (
  id int8 NOT NULL,
  celular varchar(15) NULL,
  cpf_cnpj varchar(14) NOT NULL,
  email text NOT NULL,
  nome varchar(200) NOT NULL,
  ramo_atividade varchar(30) NOT NULL,
  telefone varchar(15) NULL,
  tipo_pessoa varchar(20) NOT NULL,
  CONSTRAINT empresa_pkey PRIMARY KEY (id),
  CONSTRAINT empresa_ramo_atividade_check CHECK (((ramo_atividade)::text = ANY (ARRAY['CLINICA_MEDICA'::character_varying,
  'CLINICA_VETERINARIA'::character_varying, 'ESCRITORIO_ADVOCATICIO'::character_varying,
  'ENTREVISTA_EMPREGO'::character_varying])::text[])),
  CONSTRAINT empresa_tipo_pessoa_check CHECK (((tipo_pessoa)::text = ANY
  (ARRAY['PESSOA_FISICA'::character_varying, 'PESSOA_JURIDICA'::character_varying])::text[])),
  CONSTRAINT uk_d57lflm15ko3h7tr0jl876l8b7 UNIQUE (cpf_cnpj)
);
```

Fonte: Elaborado pelo Autor, 2023

Na Figura 14, é possível ver o diagrama de Entidade Relacionamento, com as tabelas utilizadas pelo sistema e os seus relacionamentos.

Figura 14 – Diagrama de Entidade Relacionamento do banco de dados.



Fonte: Elaborado pelo Autor, 2023

A utilização de uma única tabela de clientes para os diferentes ramos apresenta alguns prós e contras, dentre eles destacam-se:

- Um aspecto positivo é a facilidade de manutenção das informações de clientes que são comuns, ou seja, presentes em qualquer ramo de trabalho.
- Por outro lado, para atender diferentes ramos de trabalho, pode ser necessário a criação de uma tabela adicional com as informações específicas de clientes. Neste caso, para se obter o conjunto de informações de um cliente será necessário a execução de *joins*, o que pode tornar, em algumas situações, o sistema mais lento.
- Outro cenário com um possível problema seria a concorrência entre as diversas empresas, independente do ramo de atividade, nas interações com a

base de dados, por exemplo, se existem duas empresas com 1000 clientes cada, para a empresa 1 mesmo tendo somente mil clientes, suas interações com o banco de dados serão sobre os dois mil clientes.

## **4.2 Camada de Aplicativo (backend)**

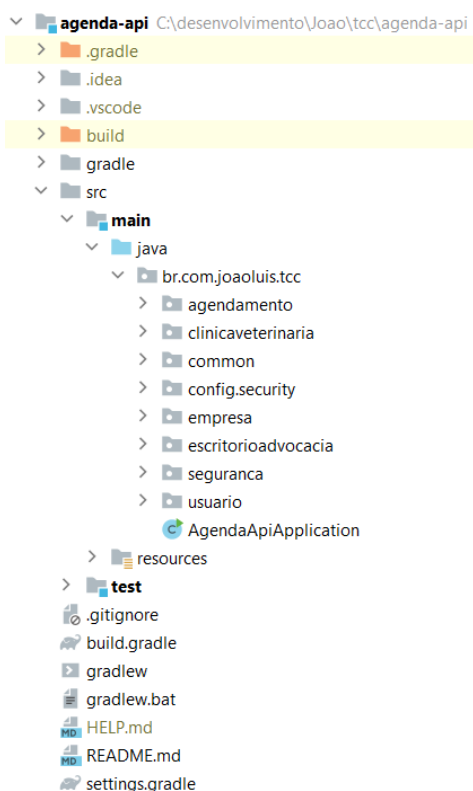
Nesta seção será apresentado o desenvolvimento do backend do sistema. Foram utilizados o Spring Boot, um framework Java, anotações das bibliotecas Lombok, Bean Validation, Spring Framework e JPA.

### **4.2.1 Estrutura de Pastas e Arquivos**

A estrutura de arquivos representada na Figura 15, foi feita com o objetivo de facilitar a organização e a manutenção do código.



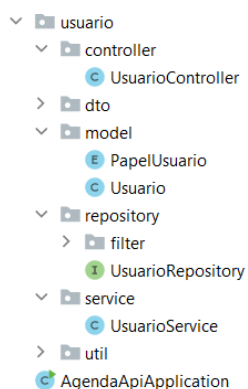
Figura 15 – Representa a estrutura de pastas e arquivos



Fonte: Elaborado pelo Autor, 2023

A classe principal da aplicação, `AgendaApiApplication`, contém a função de inicializar a aplicação através do método `main`. Ela é responsável por carregar as classes de configuração e a criação dos Beans utilizados pela aplicação, armazenando-os no container do Spring.

Figura 16 – Representa o pacote usuario e os pacotes contidos nele



Fonte: Elaborado pelo Autor, 2023

O pacote `usuario`, representado na Figura 16, engloba todas as funcionalidades relacionadas à criação, edição, exclusão e consulta de um usuário. Dentro dele, estão os pacotes **controller**, **dto**, **model**, **repository** e **service**, cada

um responsável por uma função específica no sistema. Essa estrutura se repete em todos os módulos da aplicação.

#### 4.2.2 Desenvolvimento da API

Como o objetivo do projeto é a criação de uma aplicação para atender vários clientes, para cada chamada ao Backend, os dados do usuário logado serão recuperados para a identificação da empresa ao qual o usuário está associado e assim, poder fazer a busca apenas das informações ligadas à empresa do usuário logado.

O pacote **controller** contém as classes responsáveis por receber e mapear requisições HTTP para seus métodos apropriados, e executar os serviços que irão processar a lógica de negócio. Na Figura 17, é apresentada a estrutura mínima de uma classe controller. Nela, é possível verificar como é feita a configuração de uma classe responsável por requisições HTTP e quais anotações são utilizadas, por exemplo, para receber (POST) os dados de um usuário, fazer a conversão em objetos Java e executar as regras de negócio associadas a um usuário.

Figura 17 – Representação parcial do **UsuarioController.java**

```
@RequiredArgsConstructor
@CrossOrigin({"http://localhost:4200"})
@RequestMapping("/api/usuarios")
@Controller
public class UsuarioController {

    private final UsuarioService service;

    @PostMapping
    public void save(@RequestBody @Valid UsuarioIn usuario) { this.service.save(usuario); }

    @PutMapping("/{id}")
    public void update(@PathVariable("id") Long id, @RequestBody @Valid UsuarioEdit usuario) {
        this.service.update(id, usuario);
    }

    @GetMapping
    public ResponseEntity findAll(UsuarioFilter filter, Pageable pageable) {
        if (filter.getNome() != null && filter.getNome().length() < 3) {
            List<Error> erros = Arrays.asList(Error.builder().message("A parte do nome deve pos
            return ResponseEntity.badRequest().body(erros);
        }
        return ResponseEntity.ok(this.service.findAll(filter, pageable));
    }
}
```

Fonte: Elaborado pelo Autor, 2023

A Figura 17 apresenta a definição de três métodos: save, update e findAll. O método save é um ponto de entrada, que lida com requisições do tipo POST. Após validar a requisição com a anotação @Valid, o método chama a classe de serviço (UsuarioService), que converte o objeto UsuarioIn (DTO) em uma Entidade Usuario, que por sua vez, chama a classe de repositórios (UsuarioRepository) que salva o objeto 'Usuario' no banco de dados.

O método update é responsável por lidar com requisições do tipo PUT e atualizar um recurso que é identificado por um 'id'. A anotação @PutMapping está atribuindo entre parênteses o valor "{id}" e indica que a rota ou URL que vai ser mapeada possui um parâmetro no caminho ('id') que serve para indicar qual recurso será atualizado.

Já o método findAll, é responsável por receber um determinado número de parâmetros, dentro da classe UsuarioFilter, mais os parâmetros de paginação (interface Pageable) e passar esses parâmetros para a classe de serviços, responsável por montar os filtros de pesquisas de usuários. O resultado dessa pesquisa, é retornado no formato de uma página.

Por exemplo, a chamada ao método findAll, com os parâmetros: `http://localhost:8080/api/usuarios?page=0&size=10&nome=João`, retornará a estrutura apresentada na Figura XX.

Figura 18 – Representação parcial do **UsuarioOut.java**

```
{
  "content": [{
    "id": 4,
    "nome": "Admin clínica veterinária",
    "cpf": "92991635029",
    "email": null,
    "celular": null,
    "inativo": false,
    "papel": "ROLE_ADMINISTRADOR"
  }],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 10,
    "sort": {"empty": false, "unsorted": false, "sorted": true},
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "last": true,
  "totalElements": 1,
  "totalPages": 1,
  "size": 10,
  "number": 0,
  "sort": {"empty": false, "unsorted": false, "sorted": true},
  "first": true,
  "numberOfElements": 3,
  "empty": false
}
```

Fonte: Elaborado pelo Autor, 2023

As anotações definidas na Tabela 25 são utilizadas para configurar classes responsáveis pelo recebimento das requisições HTTP e criação de construtores de objetos de maneira automática.

Tabela 25 - Definição das Anotações

Anotação	Descrição	Biblioteca
@RequiredArgsConstructor	Gera construtores para todos os campos "final" e não nulos	Lombok
@CrossOrigin	Habilita o compartilhamento de requisições vindas somente do domínio especificado na anotação.	Spring Framework
@RequestMapping	Mapeia requisições web para métodos Spring Controller.	Spring Framework
@RestController	Simplifica a criação de APIs no padrão REST.	Spring Framework
@PostMapping	Mapeia requisições HTTP do tipo POST para métodos de tratamento.	Spring Framework
@PutMapping	Similar ao @PostMapping, porém utilizado para alteração de dados de um determinado recurso.	Spring Framework
@Valid	Marca uma propriedade, parâmetro de método ou tipo de retorno de método para validação em cascata. As restrições definidas no objeto e suas propriedades são validadas quando a propriedade, parâmetro do método ou tipo de retorno do método é validado.	Bean Validation

Fonte: Elaborado pelo Autor, 2023

O pacote **model**, apresentado na Figura 18, contém as classes que representam os objetos da aplicação que mapeiam as tabelas no banco de dados.

Figura 19 – Representa uma parte do **Usuario.java**

```
@Table(name="usuario", uniqueConstraints = { @UniqueConstraint(columnNames = { "cpf" }) })
public class Usuario {

    @Id
    @SequenceGenerator(sequenceName="usuario_id_seq", name="usuario_id_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "usuario_id_seq")
    private Long id;
    @Column(length = 250, nullable = false)
    private String nome;
    @Column(length = 11, unique = true, nullable = false)
    private String cpf;

    @ManyToOne
    @JoinColumn(name = "empresa_id", referencedColumnName = "id", nullable = false)
    private Empresa empresa;

    @Enumerated(EnumType.STRING)
    @Column(length = 30, nullable = false)
    private PapelUsuario papel;
```

Fonte: Elaborado pelo Autor, 2023

A classe **Usuario** mapeia as colunas da tabela usuario do banco de dados, onde cada atributo da classe representa um campo na tabela. A partir desta classe, o JPA tem possibilidade de montar as cláusulas SQL capazes de consultar, alterar, excluir e incluir dados na tabela. O mesmo está sendo feito para todas as classes que envolvem definições de atributos no banco de dados.

As anotações na Tabela 26 representam as configurações de uma tabela no banco de dados.

Tabela 26 - Definição das Anotações

Anotação	Descrição	Biblioteca
@Entity	Define uma classe como uma representação de uma tabela no banco de dados.	JPA
@Table	Representa as definições de uma tabela no banco de dados, como nome da tabela, chaves primárias, etc..	JPA
@Id	Indica que o campo abaixo é a chave primária da entidade atual.	JPA
@SequenceGenerator	Define um gerador de chave primária que pode ser referenciado por nome quando um elemento gerador é especificado para a anotação @GeneratedValue.	JPA
@GeneratedValue	Pode ser aplicada a uma propriedade ou campo de chave primária de uma entidade ou superclasse mapeada em conjunto com a anotação @Id.	JPA
@Column	Permite adição de certas propriedades	JPA

	em uma coluna específica. Algumas propriedades permitidas: tamanho, se é única, se pode ser null ou não e o nome da coluna.	
@ManyToOne	Define um relacionamento de muitos para um. Por exemplo, na classe <b>Usuario</b> podemos verificar que a tabela usuario possui um campo chamado empresa_id que é uma chave estrangeira para tabela empresa ligada ao campo id desta tabela.	JPA
@OneToMany	Define um relacionamento de um para muitos	JPA

Fonte: Elaborado pelo Autor, 2023

Para este trabalho, utilizando as ferramentas do Spring Boot, foi definido que o pacote **repository** irá conter as interfaces que estendem a interface JpaRepository, que fornece os métodos CRUD, entre outros. Essa interface permite criar repositórios de dados personalizados ao estendê-la e definir métodos de consulta personalizados.

Figura 20 – Representação parcial do **UsuarioRepository.java**

```
public interface UsuarioRepository extends JpaRepository<Usuario, Long>, JpaSpecificationExecutor<Usuario> {

    Optional<Usuario> findByCpf(String cpf);

    boolean existsByCpf(String cpf);

    boolean existsByCpfAndIdNot(String cpf, Long usuarioId);
}
```

Fonte: Elaborado pelo Autor, 2023

A interface **UsuarioRepository** ao estender a interface JpaRepository recebe por padrão os métodos save (insert into...), findById (select \* from ... where id = ?), exists (select count(\*) from ...), entre outros.

O pacote **service** é responsável por agrupar classes que executam as regras de negócio da aplicação.

Figura 21 – Representação parcial do **UsuarioService.java**

```

@Service
public class UsuarioService {

    private final UsuarioRepository repository;
    private final EmpresaRepository empresaRepository;
    private final PasswordUtil passwordUtil;

    @Transactional
    public long save(UsuarioIn usuario) {
        this.validateNewUserData(usuario);
        Usuario novoUsuario = usuario.toEntity();
        novoUsuario.setPassword(passwordUtil.encode(usuario.getPassword()));
        this.repository.save(novoUsuario);
        return novoUsuario.getId();
    }

    public void validateNewUserData(UsuarioIn usuario) {
        if (!this.empresaRepository.existsById(usuario.getEmpresa().getId())) {
            throw new ResourceNotFoundException(String.format("Empresa com ID %d não encontrada", usuario.getEmpresa().getId()));
        }
        if (this.repository.existsByCpf(usuario.getCpf())) {
            throw new RegraNegocioException(String.format("Já existe um usuário com o CPF %s", usuario.getCpf()));
        }
    }
}

```

Fonte: Elaborado pelo Autor, 2023

A classe **UsuarioService**, representada na Figura 21, é responsável por executar regras de negócio relacionadas a usuários. No método **save**, podemos observar a anotação **@Transactional**, que inicia uma transação no banco de dados para garantir que a execução do método efetue a gravação dos dados no banco em caso de sucesso ou desfça qualquer alteração em caso de erro.

Um exemplo de regra de negócios pode ser visto na Figura 21. Ao fazer o cadastro de um novo usuário, é necessário verificar se já existe um usuário com o CPF informado. Caso já exista, o sistema lançará uma exceção, informando ao Frontend, essa situação. Outro exemplo, é a verificação da existência da empresa ligada ao novo usuário.

O pacote **DTO** é utilizado para agrupar classes de transferência de dados, de forma resumida, evitando assim expor a estrutura interna dos dados .

Figura 22 – Representação parcial do **UsuarioOut.java**

```

@Setter
@Getter
public class UsuarioOut implements Serializable {

    private Long id;
    @NotBlank(message = "O nome é obrigatório")
    private String nome;
    @Email(message = "O e-mail deve ser válido")
    @NotBlank(message = "O e-mail é obrigatório")
    private String email;
    @NotBlank(message = "O CPF é obrigatório")
    @CPF(message = "O CPF é obrigatório ser válido")
    private String cpf;
    @Valid
    @NotNull(message = "A empresa é obrigatória")
    private EmpresaId empresa;
    @NotNull(message = "O papel do usuário é obrigatório")
    private PapelUsuario papel;
}

```

Fonte: Elaborado pelo Autor, 2023

O DTO da Figura 22, está sendo utilizado para enviar somente as informações necessárias de um usuário para o Frontend, neste exemplo, a senha não está sendo enviada.

Assim como é possível a extração e determinados atributos de uma classe, que será enviada ao cliente do Backend, é possível também a junção de atributos, formando uma informação só, como por exemplo, um endereço, que poderia ser a junção de logradouro, rua e número, formando assim, uma única informação.

#### 4.2.3 Definição das Rotas

Aqui serão definidas as rotas da API que serão utilizadas. O projeto conta com 9 entidades mapeadas, onde na Figura 23 estão representados os verbos HTTP relacionados à entidade agendamentos. Este padrão de rotas se repete para todas as outras entidades, sendo este um exemplo contemplando todas funcionalidades de um agendamento.



Figura 23 – Representação das rotas responsáveis por um agendamento

agendamentos		^
GET	/api/agendamentos/{id}	v
PUT	/api/agendamentos/{id}	v
DELETE	/api/agendamentos/{id}	v
GET	/api/agendamentos	v
POST	/api/agendamentos	v
GET	/api/agendamentos/veterinaria/consultas-do-dia	v
GET	/api/agendamentos/dashboard	v
GET	/api/agendamentos/consultas-do-dia	v

Fonte: Elaborado pelo Autor, 2023

Na Figura 24, pode-se observar a rota, a estrutura de envio e a estrutura de retorno da criação de um agendamento. O processo de criação segue igual para as demais entidades.

Figura 24 – Representação da rota de criação de agendamentos

POST

/api/agendamentos

Cria um agendamento

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "id": 0,
  "cliente": {
    "id": 0
  },
  "especialista": {
    "id": 0
  },
  "observacoes": "string",
  "dataInicial": "2023-11-07T12:24:30.358Z",
  "dataFinal": "2023-11-07T12:24:30.358Z"
}
```

Responses

Code	Description	Links
201	<div>Created</div> <div>Media type</div> <div>*/*</div> <div>Controls Accept header.</div> <div>Example Value   Schema</div> <div> <pre>{   "id": 0 }</pre> </div>	No links

Fonte: Elaborado pelo Autor, 2023

#### 4.2.4 Validação dos Dados

Para garantir a qualidade dos dados inseridos na aplicação, uma ação necessária para isso, é fazer a validação dos mesmos. Como muitas das informações precisam ser validados, ficaria muito complexo criar essas validações manualmente.

Para facilitar esse processo, foi utilizada, no backend, a ferramenta

disponibilizada pelo Java, chamada Bean Validation.

De acordo com a documentação do spring, a Spring (2023), Bean Validation fornece uma forma comum de validação por meio de declaração de restrição e metadados para aplicativos Java. Para usá-lo, você anota as propriedades do modelo de domínio com restrições de validação declarativas que são então aplicadas em tempo de execução. Existem restrições integradas e você também pode definir suas próprias restrições personalizadas.

Na Tabela 27 são apresentadas algumas das validações utilizadas no projeto:

Tabela 27 - Definição das Anotações de Validação

Anotações de Validação	
@NotBlank	O elemento anotado não deve ser nulo e deve conter pelo menos um caractere que não seja um espaço em branco.
@Email	A string deve ser um endereço de e-mail bem formado. A semântica exata do que constitui um endereço de e-mail válido é deixada para os provedores de validação do Jakarta Bean.
@Size	O tamanho do elemento anotado deve estar entre os limites especificados.
@NotNull	O elemento anotado não deve ser nulo.
@Valid	Marca uma propriedade, parâmetro de método ou tipo de retorno de método para validação em cascata. As restrições definidas no objeto e suas propriedades são validadas quando a propriedade, parâmetro do método ou tipo de retorno do método é validado.
@CPF	Valida um CPF.
@CNPJ	Valida um CNPJ.

Fonte: Elaborado pelo Autor, 2023

Para o cadastro de uma empresa, foram definidas algumas características das informações fornecidas, como por exemplo:

- O nome é obrigatório, não pode ser vazio e nem somente espaços em branco. Para se obter essas características, utilizamos a anotação @NotBlank
- O CNPJ/CPF é obrigatório e deve ser um CNPJ/CPF válido. Para isso, foram utilizadas as anotações @CNPJ, @CPF, @NotBlank e @Size.
- Para o atributo tipoPessoa, ficou definido que é uma informação obrigatória,

para isso, utilizou-se @NotNull.

- Caso o celular seja informado, a anotação @Size, foi configurada para receber um valor com no mínimo 10 e no máximo 11 caracteres.
- Caso o telefone seja informado, a anotação @Size, foi configurada para receber um valor com no mínimo 10 e no máximo 11 caracteres.
- Como o email é uma informação importante para que o sistema possa enviar algum tipo de mensagem aos clientes do sistema, ficou definido que o email é obrigatório e deve ser um e-mail válido. Desta forma, as anotações utilizadas foram @NotNull e @Email.
- Para os dados do responsável pela empresa, como esta informação é obrigatória e é formada por uma estrutura separada, foram utilizadas as seguintes anotações:
  - @NotNull para validar o envio desta estrutura;
  - @Valid para validar em cascata os dados que estão na estrutura (DTO) Responsavelln
- Dentro da estrutura Responsavelln, foram utilizadas:
  - @NotBlank para validar nome do usuário responsável
  - @NotNull, @CPF e @Size para validar o cpf
  - @NotNull para validar a senha

Foram utilizadas as funcionalidades do Bean Validation nas demais classes que precisam de alguma validação dos dados.

### 4.3 Camada de Apresentação (frontend)

Figura 25 – Dependências do projeto contidas no arquivo package.json

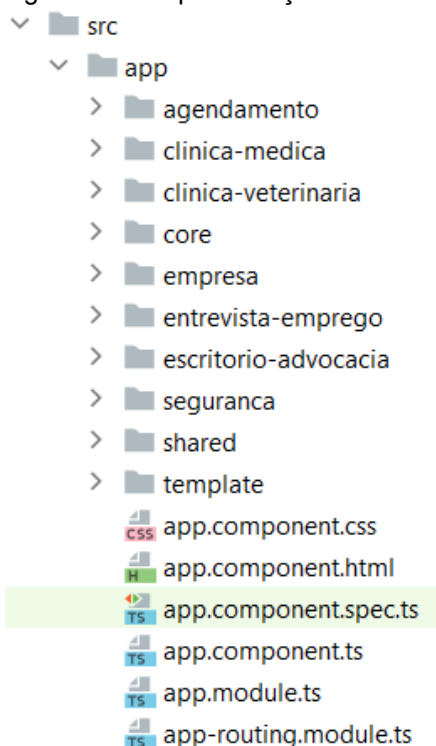
```
{
  "name": "agenda-web",
  "version": "1.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve --host 127.0.0.1 --no-live-reload",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^16.2.7",
    "@angular/common": "^16.2.7",
    "@angular/compiler": "^16.2.7",
    "@angular/core": "^16.2.7",
    "@angular/forms": "^16.2.7",
    "@angular/platform-browser": "^16.2.7",
    "@angular/platform-browser-dynamic": "^16.2.7",
    "@angular/router": "^16.2.7",
    "@fortawesome/fontawesome-free": "^6.4.2",
    "@fullcalendar/angular": "^6.1.9",
    "@fullcalendar/core": "^6.1.9",
    "@fullcalendar/daygrid": "^6.1.9",
    "@fullcalendar/interaction": "^6.1.9",
    "@fullcalendar/list": "^6.1.9",
    "@fullcalendar/timegrid": "^6.1.9",
    "@ng-bootstrap/ng-bootstrap": "^15.1.1",
    "@popperjs/core": "^2.11.6",
    "@auth0/angular-jwt": "^5.1.2",
    "crypto-js": "^4.1.1",
    "bootstrap": "^5.3.2",
    "primeflex": "^3.3.1",
    "primeicons": "^6.0.1",
    "primeng": "^16.4.1",
    "rxjs": "~7.8.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.13.0"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^16.2.4",
    "@angular/cli": "~16.2.4",
    "@angular/compiler-cli": "^16.2.7",
    "@angular/localize": "^16.2.7",
    "@types/crypto-js": "^4.1.1",
    "@types/jasmine": "~4.3.0",
    "jasmine-core": "~4.6.0",
    "karma": "~6.4.0",
    "karma-chrome-launcher": "~3.2.0",
    "karma-coverage": "~2.2.0",
    "karma-jasmine": "~5.1.0",
    "karma-jasmine-html-reporter": "~2.1.0",
    "typescript": "~5.1.3"
  }
}
```

Fonte: Elaborado pelo Autor, 2023

Para esse trabalho, optou-se por separar as funcionalidades em módulos, permitindo a reutilização de componentes que são comuns em algumas funcionalidades.

A estrutura do projeto pode ser visualizada na Figura 25:

Figura 26 – Representação da Estrutura do Projeto



Fonte: Elaborado pelo Autor, 2023

Para centralizar o gerenciamento dos módulos, foi criado o módulo Core, deixando o módulo principal apenas com a importação do módulo Core e as dependências iniciais do Angular. Todos os módulos do projeto serão importados no módulo Core.

Para classes utilitárias, foi criado o módulo Shared. Este módulo é utilizado para importação e exportação de bibliotecas (componentes) de terceiros, como por exemplo, a biblioteca PrimeNg, que nos fornece componentes HTML estilizados e com funcionalidades como: Criação de tabelas paginadas, caixas de diálogos, etc.

Para a implementação de segurança, foi criado o módulo Segurança, que é o responsável por centralizar as chamadas para a tela de login, a inclusão do Token em cada chamada HTTP realizada ao Backend, a verificação das permissões dos usuários, etc.

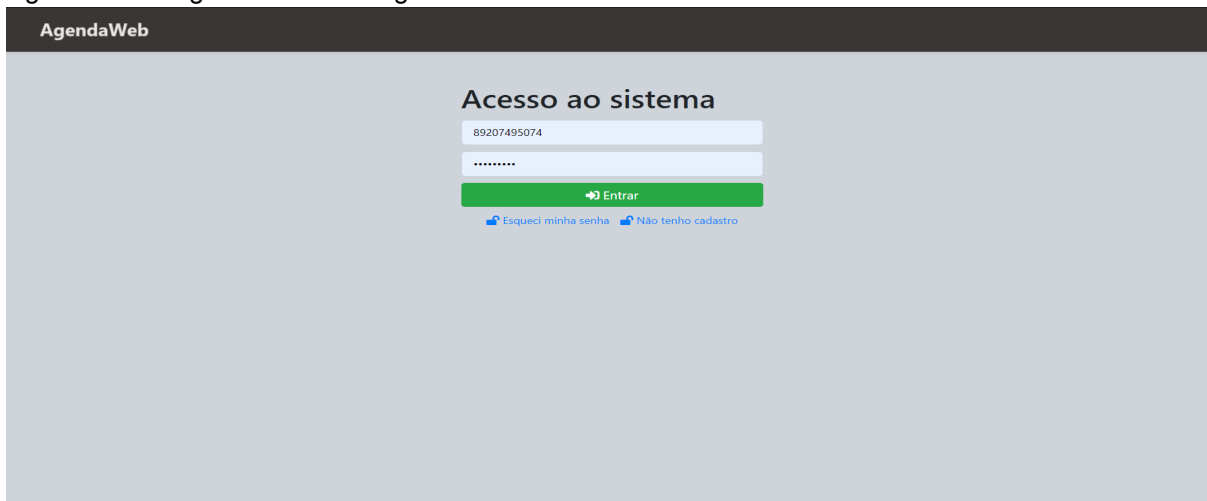
Os demais módulos são responsáveis pelas funcionalidades que o projeto se dispôs a fazer.

#### 4.3.1 Telas do Sistema

A tela de login (Figura 27) é a primeira tela do sistema, ela possui duas áreas

para inserir o CPF e a senha, o botão “Entrar”, e duas opções, a “Esqueci minha senha” e “Não tenho cadastro”.

Figura 27 – Imagem da tela de login do sistema



Fonte: Elaborado pelo Autor, 2023

A Figura 28 representa a tela principal do sistema para usuários que não possuem cadastro. Nela, é possível navegar pelos templates para cada ramo de atividade que o projeto se propôs a fazer, ou fazer o cadastro, através do botão “Cadastrar empresa”.

Ela é acessada ao clicar em “Não possuo cadastro” apresentado na Figura 27. Ao clicar no botão “Cadastrar empresa”, o modal apresentado na Figura 29 irá aparecer, permitindo assim o cadastro da empresa no sistema.

Figura 28 – Imagem da Tela Inicial do sistema (usuário não cadastrado)



Fonte: Elaborado pelo Autor, 2023

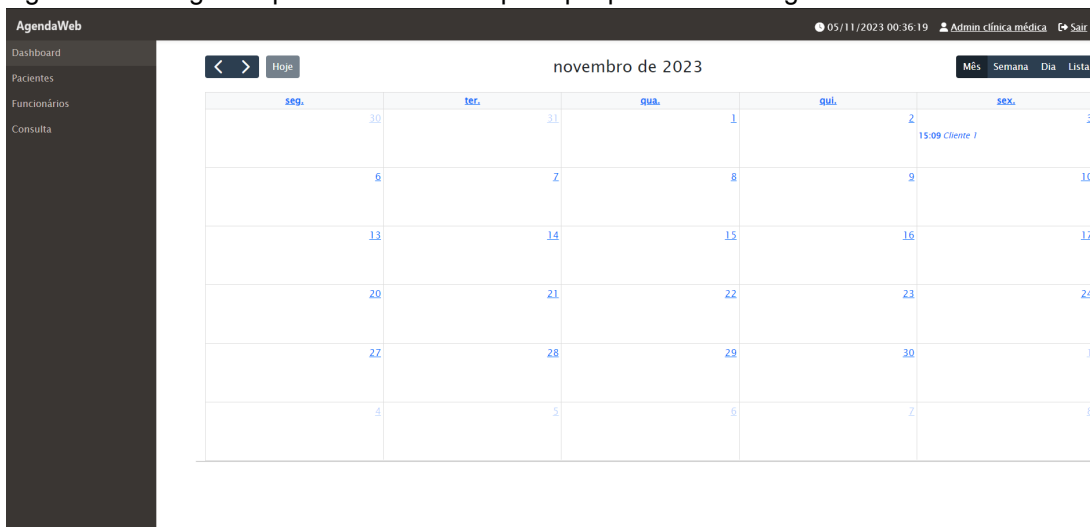
Figura 29 – Imagem representando o modal de cadastro de uma nova empresa

Fonte: Elaborado pelo Autor, 2023

Após o cadastro com sucesso, o usuário será redirecionado para tela de login (Figura 27). A partir do login, ao entrar com os dados cadastrados, o usuário é redirecionado a tela principal do sistema (Figura 30), dependendo do ramo de atividade da empresa do usuário autenticado.



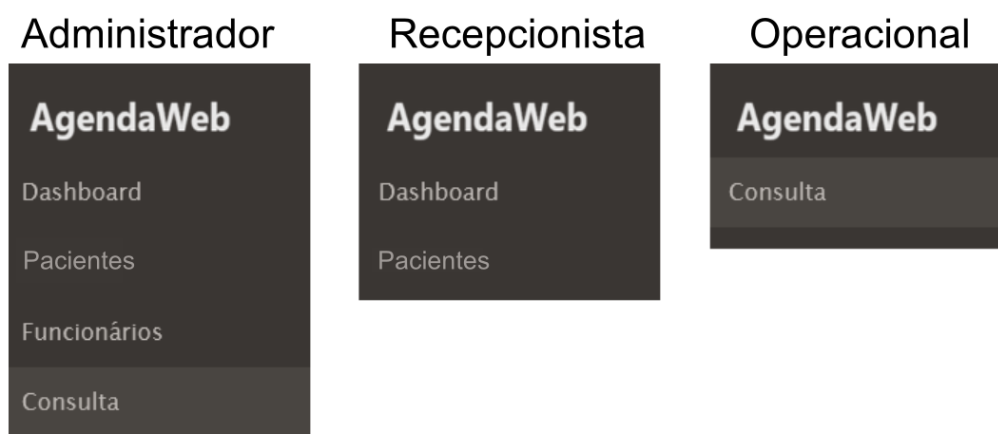
Figura 30 – Imagem representando a tela principal para usuário logado



Fonte: Elaborado pelo Autor, 2023

A Figura 30 apresenta a tela inicial de uma clínica médica, que contém como elemento principal o calendário, que será utilizado para marcar os eventos. Possui também um menu lateral, onde está disponível a Dashboard, que é o calendário, a lista de pacientes cadastrados, a lista de funcionários e as consultas marcadas na dashboard. A visibilidade das opções no menu está atrelada a permissão do usuário, como está representado na figura 31.



Figura 31 – Imagem representando o nível de acesso de cada permissão



Fonte: Elaborado pelo Autor, 2023

Estas opções de menu variam de acordo com o ramo de atividade de cada empresa.







Figura 32 – Imagem representando a lista de pacientes

Código	Nome	CPF	Data de Nascimento	Ações
1	Cliente 1	191.622.970-07		 

+ Novo cliente

Fonte: Elaborado pelo Autor, 2023


Figura 33 – Imagem representando a lista de funcionários

Código	Nome	CPF	Permissão	Situação	Ações
1	Admin clínica médica	89207495074	Administrador	Ativo	 
3	Atendente	48103659053	Recepcionista	Ativo	 
2	Médico	63879849048	Operacional	Ativo	 

+ Novo funcionário

Fonte: Elaborado pelo Autor, 2023

Figura 34 – Imagem representando a lista de consultas

Código	Nome	Data	Ações
1	Cliente 1	03/11/2023	

+ Novo consulta

Fonte: Elaborado pelo Autor, 2023

Estando o usuário logado e estando na tela “Dashboard”, a criação de um agendamento se dará pela escolha do dia que se deseja incluir a consulta. Ao selecionar o dia, o sistema apresentará a tela de cadastro (Figura 35).

Figura 35 – Cadastro de um novo agendamento

Fonte: Elaborado pelo Autor, 2023

Com todos os dados obrigatórios preenchidos, o usuário pode clicar no botão “Salvar”, que o sistema fará o envio dos dados ao Backend (Figura 36), que fará a validação dos dados e, em caso de sucesso, os dados serão gravados no banco de dados e o Backend fará o retorno do ID do agendamento criado (Figura 37).

A Figura 37, apresenta a estrutura de dados que o Backend retorna quando a gravação dos dados é feita com sucesso.

Figura 36 – Estrutura de dados a ser enviada ao Backend

```
{
  "cliente":{
    "id":1,
    "nome":"Cliente 1"
  },
  "especialista":{"id":2,"nome":"Médico"},
  "diaTodo":false,
  "dataInicial":"2023-11-07T08:00:00.000Z",
  "dataFinal":"2023-11-07T08:20:00.000Z"
}
```

Fonte: Elaborado pelo Autor, 2023

Figura 37 – Estrutura de dados retornada do Backend

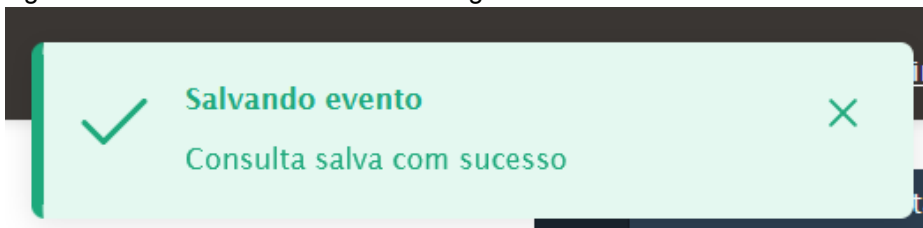
```
{
  "id":11
}
```

Fonte: Elaborado pelo Autor, 2023

Ao receber o retorno do Backend, o Frontend apresentará a mensagem de

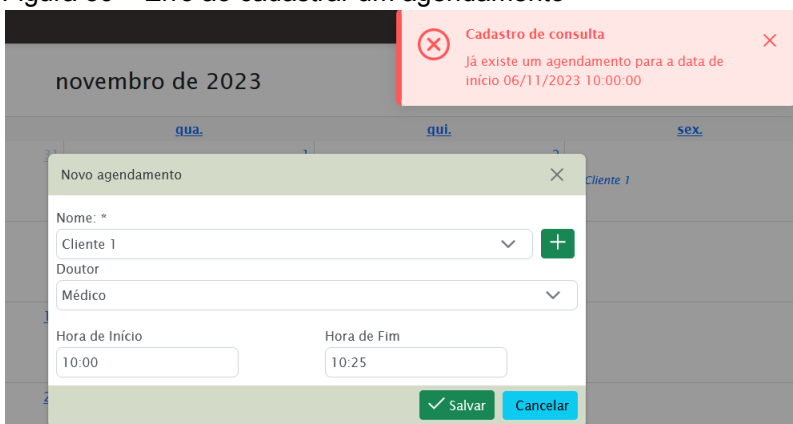
sucesso (Figura 38). No caso de haver algum erro de validação ou qualquer outro problema no Backend, o Frontend receberá uma lista de erros (Figura 39), que serão apresentados na tela.

Figura 38 – Sucesso ao cadastrar um agendamento



Fonte: Elaborado pelo Autor, 2023

Figura 39 – Erro ao cadastrar um agendamento



Fonte: Elaborado pelo Autor, 2023

Como a validação de dados pode identificar mais do que um “erro” de validação, o backend gera uma lista com esses erros e retorna ao frontend com a estrutura apresentada na Figura 40.

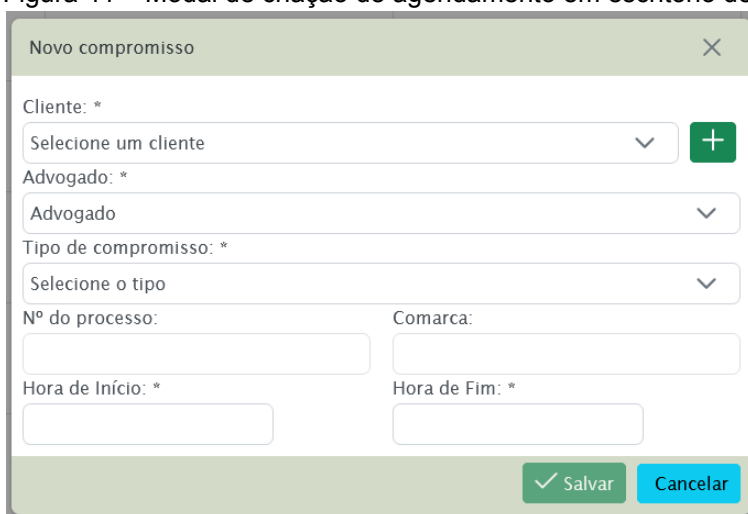
Figura 40 – Representação da estrutura de erros retornados pelo Backend

```
[{
  "message": "Já existe um agendamento para a data de início 07/11/2023 08:00:00"
}]
```

Fonte: Elaborado pelo Autor, 2023

Como um dos objetivos deste trabalho é a implementação de um sistema de agendamentos que possa ser utilizado por vários ramos de atividades, com características diferentes, a Figura 40 apresenta o agendamento de um evento, chamado de “compromisso”, para um escritório de advocacia.

Figura 41 – Modal de criação de agendamento em escritório de advocacia



Novo compromisso

Cliente: \*  
Selecione um cliente

Advogado: \*  
Advogado

Tipo de compromisso: \*  
Selecione o tipo

Nº do processo: Comarca:

Hora de Início: \* Hora de Fim: \*

✓ Salvar Cancelar

Fonte: Elaborado pelo Autor, 2023

Como pode ser visto na Figura 41, foram inseridas algumas informações que são relacionadas apenas com escritórios de advocacia.

No Frontend, foram implementadas as telas com os atributos específicos para escritórios de advocacia, sendo que essas informações são fornecidas pelo Backend, a partir da empresa ligada ao usuário logado. Para ramos de atividades que não possuem características diferentes, foi implementada uma interface comum, que compartilha as mesmas rotas. Já para ramos de atividades distintas, as interfaces são desenvolvidas separadamente, com rotas específicas, tanto no Frontend quanto no Backend.

Os passos executados para a geração de agendamentos são os mesmos para os outros ramos de atividades propostos pelo trabalho.

Figura 42 – Diferença entre formulários de clínica veterinária e clínica médica respectivamente

The image shows two side-by-side web forms for client registration. The left form is titled 'Edição de cliente' and the right is 'Novo cliente'.

**Edição de cliente:**

- Nome: \* (text input) Bichano
- Espécie: \* (dropdown menu) Gato
- Sexo: \* (dropdown menu) Feminino
- Raça: \* (text input) Raça qualquer
- Data de nascimento: (text input) 01/10/2022
- Tutor: \* (text input) João Doe
- e-mail tutor: \* (text input) email@email.com
- Celular tutor: \* (text input)
- CPF tutor: \* (text input) 331.042.720-00

**Novo cliente:**

- Nome: \* (text input)
- CPF: \* (text input)
- Data Nascimento: (text input)
- Celular: (text input)
- e-mail: \* (text input)
- Rua: (text input)
- Complemento: (text input)
- Número: (text input)
- CEP: (text input)
- Bairro: (text input)
- Cidade: (text input)

Both forms have a green 'Salvar' button and a red 'Cancelar' button at the bottom right.

Fonte: Elaborado pelo Autor, 2023

O sistema dispõe da funcionalidade de atendimento das consultas/atendimentos. Esta funcionalidade permite o registro de observações em cada consulta/atendimento, sendo possível verificar as observações feitas em consultas/atendimentos feitos anteriormente, do mesmo paciente/cliente.

Figura 43 – Modal de atendimento de um evento

The image shows a web form titled 'Consulta' for attending an event.

- Cliente: \* (text input) Livia
- Histórico: (text area)
  - obs 3
  - obs 2
  - obs 1
- Observações: (text area)

At the bottom right, there are green 'Salvar' and red 'Cancelar' buttons.

Fonte: Elaborado pelo Autor, 2023

## 5 CONCLUSÃO

Neste trabalho de conclusão de curso foi apresentado o desenvolvimento de um sistema web cujo objetivo é auxiliar o gerenciamento de agendamentos, clientes e funcionários para diferentes ramos de trabalho. Para atender esses diferentes ramos de atuação, optou-se pela criação de layouts pré-definidos. Levando-se em conta a proposta inicial e o resultado alcançado, conclui-se que o objetivo geral, que é suprir uma demanda de gerenciamento de agendamentos para perfis diferentes de profissionais, foi cumprido.

A escolha pelo uso de ferramentas de desenvolvimento web que fornecem componentes visuais de fácil utilização, reduziu a quantidade de linhas de código escrita, tornando viável a implementação do projeto dentro do prazo estabelecido. Também foi observada a contribuição das bibliotecas e frameworks no desenvolvimento do backend, com destaque para a criação das APIs.

O uso de uma biblioteca de componentes visuais possibilitou, também, o desenvolvimento de uma interface intuitiva que conta com *Tooltips*. Os *Tooltips* são elementos gráficos visíveis quando o ponteiro do mouse é posicionado sobre um elemento, fornecendo uma descrição sobre ele. Isso permite um melhor entendimento sobre a funcionalidade do elemento.

Devido a limitação de tempo, não foi possível aplicar o conceito de responsividade no sistema, tornando a utilização do mesmo prejudicada em tablets, celulares e monitores com baixa resolução. Essa limitação também influenciou na descoberta das necessidades mais específicas de cada ramo de trabalho, desta forma foram aplicadas somente as informações mais genéricas.

## 5.1 Trabalhos Futuros

Mesmo com o objetivo geral sendo alcançado, foram observadas algumas oportunidades de expansão e melhoria do sistema. Os trabalhos futuros serão essenciais para a evolução da plataforma. Algumas dessas oportunidades de melhoria seriam:

- Uma pesquisa aprofundada sobre as especificidades dos ramos de trabalho, a fim de compreender melhor as necessidades presentes em cada um desses ramos e proporcionar funcionalidades mais particulares
- Como a aplicação foi construída de forma modular, com a divisão do projeto em *client-side* e *server-side*, torna-se oportuna a implementação de um cliente *mobile* para a API REST. Uma aplicação nativa contribui para uma experiência de usuário eficiente e atraente em dispositivos móveis. Além de uma implementação *mobile*, recomenda-se alterações para permitir a responsividade do sistema possibilitando o seu uso em diversos dispositivos de maneira satisfatória.



## REFERÊNCIAS

- ALMEIDA, S. **Beans, entendendo a base fundamental do Spring Framework**. Dio, 11 janeiro 2023. Disponível em: <<https://www.dio.me/articles/beans-entendendo-a-base-fundamental-do-spring-frame-work>>. Acesso em: 23 out. 2023.
- ANGULAR. **What is Angular?** Angular, 28 fevereiro 2022. Disponível em: <<https://angular.io/guide/what-is-angular>>. Acesso em: 5 nov. 2023.
- AWS. **O que é Java?** AWS, 24 outubro 2023. Disponível em: <<https://aws.amazon.com/pt/what-is/java/>>. Acesso em: 24 out. 2023.
- AWS. **O que é uma aplicação Web?** AWS, 2023. Disponível em: <<https://aws.amazon.com/pt/what-is/web-application/>>. Acesso em: 26 set. 2023.
- GAMMA, Erich et al. **Padrões de Projeto: soluções reutilizáveis de software** Orientado a Objetos. Porto Alegre: Bookman, 2000.
- GUEDES, Marylene. **O que é o Angular e para que serve?** Treinaweb, 2020. Disponível em: <<https://www.treinaweb.com.br/blog/o-que-e-o-angular-e-para-que-serve>>. Acesso em: 1 out. 2023.
- IBM. **Configurando a JVM**. IBM, 31 setembro 2023. Disponível em: <<https://www.ibm.com/docs/pt-br/was-zos/8.5.5?topic=servers-configuring-jvm>>. Acesso em: 24 out. 2023.
- IBM. **O que é Java Spring Boot?** IBM, 27 setembro 2023. Disponível em: <<https://www.ibm.com/br-pt/topics/java-spring-boot>>. Acesso em: 27 set. 2023.
- INGALLS, Sam. **Web Application**. Webopedia, 24 maio 2021. Disponível em: <<https://www.webopedia.com/definitions/web-application/>>. Acesso em: 20 set. 2023.
- JAVA. **O que é tecnologia Java e por que preciso dela?** Java, 24 outubro 2023. Disponível em: <[https://www.java.com/pt-BR/download/help/whatis\\_java.html](https://www.java.com/pt-BR/download/help/whatis_java.html)>. Acesso em: 24 out. 2023.
- LEMONS, Maxmilian Ferreira de. et al. Aplicabilidade da arquitetura MVC em uma aplicação web (WebApps). **RE3C-Revista Eletrônica Científica de Ciência da Computação**, v. 8, n. 1, 2013.
- MANZANO, Guilherme. **Anotações do curso Spring Boot, JPA e Hibernate**. Medium, 2020. Disponível em: <<https://guilherme-manzano.medium.com/anotacoes-do-curso-spring-boot-jpa-e-hibe>>

[rnat-4be7e6a827c6](#)>. Acesso em: 7 out. 2023.

MARIANO, Diego. **Verbos HTTP**. diegomariano, 14 maio 2022. Disponível em: <<https://diegomariano.com/verbos-http/>>. Acesso em: 26 set. 2023.

MOZILLA. **Métodos de requisição HTTP**. Mozilla, 2023. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>>. Acesso em: 26 set. 2023.

POSTAL, Lucas et al. Sistema de agendamento online: uma ferramenta do PEC e-SUS APS para facilitar o acesso à Atenção Primária no Brasil, **Ciência & Saúde Coletiva**, v. 26, n. 6, p. 2023–2034, 2021.

RAMOS, Gabriel. **Inversão de Controle**. Gabrielluizramos, 12 março 2021. Disponível em: <<https://gabrielluizramos.com.br/inversao-de-controle>>. Acesso em: 11 dez. 2023.

Red Hat. **O que é API?** Red Hat, 2023. Disponível em: <<https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>>. Acesso em: 25 set. 2023.

ROSA, Daniel. **Uma rápida introdução à injeção de dependências: o que é e quando usá-la**. FreeCodeCamp.org, 29 janeiro 2022. Disponível em: <[www.freecodecamp.org/portuguese/news/uma-rapida-introducao-a-injecao-de-dependencias-o-que-e-e-quando-usa-la](http://www.freecodecamp.org/portuguese/news/uma-rapida-introducao-a-injecao-de-dependencias-o-que-e-e-quando-usa-la)>. Acesso em: 11 dez. 2023.

ROYCE, Winston W. Managing the Development of Large Software Systems (1970). *In: Ideas That Created the Future*. [s.l.]: The MIT Press, 2021, p. 321–332. Disponível em: <<http://dx.doi.org/10.7551/mitpress/12274.003.0035>>. Acesso em: 19 set. 2023.

SOUZA, Alberto. **Guia das annotations do Spring**. Domineospring, 2016. Disponível em: <<https://domineospring.wordpress.com/2016/07/13/guia-das-annotations-do-spring/>>. Acesso em: 7 out. 2023.

SOUZA, Ivan. **Postgresql: saiba o que é, para que serve e como instalar**. Rock Content, 2020. Disponível em: <<https://rockcontent.com/br/blog/postgresql/>>. Acesso em: 1 out. 2023.

SOUZA, Ivan. **Saiba o que é REST (Representational State Transfer) e como usá-lo**. Rock Content, 2020. Disponível em: <<https://rockcontent.com/br/blog/rest/>>. Acesso em: 26 set. 2023.

SPRING. **Introduction to the Spring IoC Container and Beans**. Spring Framework, 23 outubro 2023. Disponível em: <<https://docs.spring.io/spring-framework/reference/core/beans/introduction.html>>. Acesso em: 23 out. 2023.

SPRING. **Java Bean Validation**. Spring, 05 novembro 2023. Disponível em:

<<https://docs.spring.io/spring-framework/reference/core/validation/beanvalidation.html>>. Acesso em: 5 nov. 2023.

WIKIPEDIA. **MVC**. Wikipédia, 2 setembro 2023 . Disponível em:  
<<https://pt.wikipedia.org/w/index.php?title=MVC&oldid=66526285>>. Acesso em: 2 set. 2023.