Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

FERNANDO MENDES NETO

SISTEMA PARA GERENCIAMENTO DE CLÍNICAS MÉDICAS

Assis/SP
2023

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

FERNANDO MENDES NETO

SISTEMA PARA GERENCIAMENTO DE CLÍNICAS MÉDICAS

Trabalho de Conclusão de cursos apresentado ao curso de Análise e Desenvolvimento de Sistemas do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando(a): Fernando Mendes Neto
Orientador(a): Dr. Almir Rogério Camolesi

Assis/SP
2023

Mendes Neto, Fernando

M538s Sistema para gerenciamento de clínicas médicas / Fernando Mendes Neto. -- Assis, 2023.

53p. : il.

Trabalho de Conclusão de Curso (Análise e Desenvolvimento de Sistemas) -- Fundação Educacional do Município de Assis (FEMA), Instituto Municipal de Ensino Superior de Assis (IMESA), 2023.
Orientador: Prof. Dr. Almir Rogério Camolesi.

1. Aplicativos móveis. 2. Medicina. 3. Java. I Camolesi, Almir Rogério. II Título.

CDD 005.2

Elaborada por Anna Carolina Antunes de Moraes – Bibliotecária – CRB-8/10982

SISTEMA PARA GERENCIAMENTO DE CLÍNICAS MÉDICAS

FERNANDO MENDES NETO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador: Dr. Almir Rogério Camolesi

Examinador: Esp. Domingos de Carvalho Villela Junior

Assis/SP
2023

DEDICATÓRIA

Dedico este trabalho aos meus familiares e amigos, que sempre me motivaram e ajudaram a concluir essa etapa da minha vida.

AGRADECIMENTOS

RESUMO

O presente trabalho tem como foco principal a sistematização de clínicas médicas, auxiliando em grande parte das etapas para agendamento de consulta, cadastro de pacientes e médicos, armazenamento de informações clínicas em nuvem, geração de relatórios referentes ao desempenho da clínica e gerenciamento de agenda. Utilizando a linguagem Java com Spring, para desenvolvimento da API consumida pela aplicação web desenvolvida em Angular.

Palavras-chave: Clínica médica, Aplicação Web, Java, Angular, Banco de dados Postgresql.

ABSTRACT

The main focus of the present work is the systematization of medical clinics, assisting in most of the steps for scheduling appointments, registering patients and doctors, storing clinical information in the cloud, generating reports regarding the performance of the clinic and schedule management. Using the Java language with Spring, for the development of the API consumed by the web application developed in Angular.

Keywords: Medical clinic, Web Application, Java, Angular, Postgresql database.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Atualmente, as clínicas médicas ficaram cada vez mais dependentes dos sistemas de informação. Pois, estes facilitam o cotidiano da clínica, trazendo uma maior agilidade nos agendamentos de consultas, garantindo o armazenamento correto das informações dos pacientes e favorecendo a obtenção de informações com uma maior rapidez e livre de erros humanos, uma vez que, estas informações clínicas não se encontram mais em papéis.

Segundo Galvão e Barbosa (2012) “A Informação Clínica (IC) é qualquer informação produzida ou utilizada por profissionais da área de saúde (médico, enfermeiro, fisioterapeuta, nutricionista, terapeuta ocupacional, fonoaudiólogo, dentre outros) em busca de diagnosticar, avaliar, tratar ou recuperar as condições de saúde dos indivíduos que buscam assistência. A informação clínica precisa ser clara, objetiva, efetiva, ter qualidade e segurança, pois a mesma pode agravar as condições de saúde do paciente se não fidedigna”. Portanto, estas informações devem ser armazenadas, a fim de trazer um prontuário mais detalhado aos médicos durante a consulta, com a finalidade de levar a estes médicos um panorama geral do paciente, auxiliando no diagnóstico e no tratamento correto da enfermidade.

Os profissionais de saúde, o tempo todo, têm a necessidade de obter informações clínicas para melhor assistir o paciente, seja por meio do prontuário com foco na investigação individual ou pelas bases de dados científicas (CORREIA, PADILHA, VASCONCELOS, 2014). Por conseguinte, é de extrema importância que um sistema voltado para uma clínica médica, seja capaz de fornecer individualmente para cada paciente: informações referentes às últimas consultas, datas destas consultas, o diagnóstico e armazenamento dos últimos exames realizados por aquele paciente.

Por fim, a digitalização do processo de agendamento das consultas traz uma maior flexibilidade, rapidez e otimização do tempo gasto para a realização desse agendamento, isto aliado às informações que o sistema armazenará e capacidade de gerar prontuários médicos de forma individual para cada paciente, utilizando os históricos de consultas do mesmo, o presente sistema proverá todas as necessidades básicas para o desenvolvimento da clínica e fortalecimento da confiança do médico para a tomada de decisão durante as consultas realizadas.

1.1. OBJETIVOS

O presente projeto tem como objetivo o desenvolvimento de um software que auxilie na administração de clínicas médicas, desde o agendamento de consultas, pagamentos, armazenamento de informações clínicas e exames, e por fim, a facilitação do diagnóstico médico a partir de um histórico de consultas para cada paciente.

1.2. JUSTIFICATIVA

O exposto trabalho se fundamenta a partir das técnicas utilizadas por grande parte das clínicas médicas para o armazenamento e tratamento das informações que são utilizadas durante as consultas, trazendo estas para o meio digital, além de manter a integridade e segurança dessas informações, obteremos também uma maior agilidade para a obtenção das mesmas, eliminando assim, a necessidade de registros físicos dentro da clínica. Pode-se justificar também por meio da digitalização do processo de agendamento que trará maior velocidade e precisão a essa etapa, pois, os métodos utilizados atualmente podem ser imprecisos e estão mais sujeitos a falhas humanas.

1.3. MOTIVAÇÃO

De acordo com o site Insper Jr (2020), a utilização de um sistema de informação é um diferencial competitivo, pois, permite às empresas que o utilizam, explorar, controlar e organizar seus dados e informações, com isso, a empresa passa a conhecer melhor o seu público e a si mesma, auxiliando na tomada de decisão. Na área da saúde não é diferente, clínicas médicas necessitam obter informações a partir de dados gerados durante consultas passadas relativas àquele paciente que está em atendimento, para melhor assistir e curar a queixa/enfermidade a qual motivou a consulta. Por fim, a sistematização de alguns processos rotineiros dentro da clínica médica, trará mais produtividade e melhora na estrutura organizacional da mesma.

1.4. PERSPECTIVAS DE CONTRIBUIÇÃO

O trabalho em questão tem por perspectiva de contribuição, uma melhora na organização e na automação de alguns processos rotineiros dentro de uma clínica médica, trazendo maior agilidade e assertividade nos agendamentos das consultas, assegurando que não haja consultas no mesmo horário. Manter as informações e dados dos pacientes no ambiente de nuvem, ou seja, seguros e facilitando a obtenção destes quando necessário. Trazer um relatório mais detalhado sobre o paciente durante o atendimento, sem a necessidade do ajuntamento dessas informações na forma física e manualmente, garantindo maior organização e favorecendo a tomada de decisão do médico no tratamento da enfermidade.

1.5. METODOLOGIA

A metodologia para o desenvolvimento do trabalho em questão, utilizou-se os conhecimentos adquiridos durante o curso de Análise e Desenvolvimento de Sistemas, cursos na plataforma Alura referentes a arquitetura de microsserviços e Amazon Web Services. Pesquisas sobre o funcionamento de toda a administração de clínicas médicas e sobre informações utilizadas pelos médicos para a realização do diagnóstico da enfermidade. Recorreu-se também à documentação das linguagens Java e Typescript usadas para o desenvolvimento do sistema, não só, mas também à documentação das *frameworks* utilizados, Spring para o *backend* e Angular para o *frontend*.

2. TECNOLOGIAS E FERRAMENTAS UTILIZADAS

2.1. JAVA

De acordo com Paul e Harvey, Java é uma linguagem de programação mundialmente utilizada que suporta 4 paradigmas de programação, procedural, orientada a objetos, genérica e funcional. A versão utilizada no sistema em questão oferece recursos para o desenvolvimento de aplicativos desktop e servidor, aliado ao *framework* Spring o java se torna uma linguagem extremamente prática e rápida para o desenvolvimento de API´s no padrão REST

2.2. SPRING

O Spring é um framework Java que tem por finalidade facilitar e agilizar o desenvolvimento de aplicação utilizando essa linguagem. Ao utilizarmos esse framework temos a disposição alguns recursos extremamente necessários por grande parte das aplicações, dentre esses recursos podemos citar: módulos para persistência de dados (JDBC e JPA), integração, segurança (*Spring Security*), testes, desenvolvimento web e outros que nos permitem criar soluções pouco acopladas, e mais fáceis de dar manutenção.

2.3. PADRÃO MVC (MODEL-VIEW-CONTROLLER)

De acordo com o site Devmedia, o MVC é utilizado em muitos projetos devido a arquitetura que possui, possibilitando a divisão do projeto em camadas bem divididas. Cada uma dessas camadas, Model, View e o Controller, executam apenas o que lhe é atribuído e nada mais. Ao optar por um padrão MVC em seu projeto, obtemos como beneficio o isolamento da regra de negócio e das demais partes de um código, tornando assim, a manutenção mais fácil e facilitando o reuso de classes através de uma maior flexibilidade.

2.4. CLEAN CODE

O Clean Code ou Código limpo em português, são um conjunto de boas práticas que tem como finalidade melhorar a compreensão do código que estamos desenvolvendo. Ao aplicar o Clean code, de acordo com Robert Cecil Martin ou Uncle Bob, como é conhecido no meio tecnológico, temos uma série de benefícios para manter aquele código em pleno funcionamento durante muito mais tempo, que se fosse desenvolvido sem aplicar essas boas práticas do clean code. Ao melhorar a compreensão e a elegibilidade do código a partir do clean code, vamos facilitar sua manutenção a longo prazo, sem depender de consultar os desenvolvedores que criaram determinada função para poder compreender o que foi feito ali, pois, partindo do clean code, o código passa a ser autoexplicativo.

2.5. INTELLIJ

De acordo com o site da criadora da IDE, O IntelliJ IDEA é o principal IDE para o desenvolvimento em Java e Kotlin. Ele ajuda você a se manter produtivo, com um conjunto de recursos que aumentam a sua eficiência, tais como assistência inteligente à programação, refatorações confiáveis, navegação instantânea pelo código, ferramentas de desenvolvimento incorporadas, suporte ao desenvolvimento corporativo e para a Web, e muito mais.

2.6. POSTGRESQL

É um sistema de gerenciamento de banco de dados, escrito em linguagem c, esse gerenciador usa e estende o SQL, possuindo entre suas principais vantagens à economia e o alto desempenho, suportando um intenso fluxo de dados, garantindo assim, a segurança e estabilidade do sistema por um baixo custo

2.7. ANGULAR

É um *framework* que usa a arquitetura *Model-View-Controller* (MVC), sendo comumente utilizado para a construção de aplicativos web baseados em uma única página. Dentre suas principais vantagens podemos citar: Códigos mais limpos e melhor estruturados, facilitando a manutenção e economizando tempo no desenvolvimento; a

compatibilidade entre desktop e mobile, afinal o angular é executado na grande maioria dos navegadores; suporte a teste integrados, dentre outros. Este *framework* foi utilizado para a construção do *frontend* do sistema.

2.8. VISUAL STUDIO CODE

É um editor de código desenvolvido pela Microsoft, disponível pra grande maioria dos sistemas operacionais e com uma grande coleção de extensões desenvolvidas pelos próprios utilizadores da IDE, a ferramenta se tornou popular rapidamente, trazendo um ambiente de desenvolvimento mais confortável, prático e altamente produtivo. O presente sistema utilizou esta IDE para o desenvolvimento do seu *frontend* com o *framework* Angular.

2.9. LUCIDCHART

O Lucidchart, é uma plataforma que nos permite criar diversos tipos de diagramas, fluxogramas e desenhos, afim trazer maior clareza no desenvolvimento de software. Por ser um aplicativo web, O mesmo projeto pode ser compartilhado por diversas pessoas, trazendo uma rápida visualização e favorecendo a colaboração visual entre os membros da equipe.

2.10. VIACEP

O Viacep, é um webservice gratuito utilizado para consultar CEP. Sua utilização é bastante simples, basta realizar uma requisição http contendo o cep desejado, que o webservice nos retorna informações vinculadas a esse cep. Por exemplo, desejamos consultar o CEP: ‘’01001000’’, basta realizar a seguinte requisição para o ViaCep:

Figura 1 - Exemplo requisição viacep

Em azul temos a URL base para o webservice. Em laranja o CEP que desejamos obter suas informações. E por fim, na cor rosa, temos o formato do corpo da resposta devolvido pelo webservice. Com a requisição da figura 1, obtemos a seguinte resposta:


```
{
    "cep": "01001-000",
    "logradouro": "Praça da Sé",
    "complemento": "lado ímpar",
    "bairro": "Sé",
    "localidade": "São Paulo",
    "uf": "SP",
    "ibge": "3550308",
    "gia": "1004",
    "ddd": "11",
    "siafi": "7107"
}
```


2.11. GIT

O Git, é uma ferramenta que nos permite implementar o versionamento em nossos códigos, trazendo maior segurança e agilidade para o desenvolvimento, afinal, tudo aquilo que acabamos de implementar ainda não foi comitado para o repositório, garantindo assim algumas vantagens para a etapa de desenvolvimento, dentre essas vantagens podemos citar: A garantia de que podemos manter o código do repositório sempre em uma versão que funciona, o “poder” de conseguir navegar entre commits e a possibilidade de um trabalho em equipe no mesmo código e ao mesmo tempo com a função das *branches*.

2.12. LOMBOK

É uma biblioteca Java que tem como finalidade o aumento da produtividade e a redução dos códigos repetitivos, que podem acabar deixando seu código com um aspecto mais “sujo”. Por meio de anotações no nosso código fonte, o Lombok é capaz de criar por baixo dos panos métodos *Getters, Setters, Equals*, Construtores, entre outras coisas, deixando o código mais limpo e focado naquilo que realmente importa para funcionamento correto do nosso sistema.

2.13. MAVEN

Uma das ferramentas mais importantes e que acaba facilitando muito a vida dos desenvolvedores. O Maven é utilizado para facilitar o gerenciamento das “builds” do nosso projeto, com a utilização dessa ferramenta é possível criar uma padronização do ambiente de desenvolvimento, sem a necessidade de ficar procurando e baixando *Jars* manualmente e evitando o conflito de versão dessas bibliotecas para diferentes desenvolvedores. Em um único arquivo XML, podemos reunir tudo aqui que é necessário para a execução da nossa aplicação, de forma rápida e precisa.

2.14. ASTAH UML

É uma ferramenta utilizada para modelagem UML (*Unified Modeling Language*), Por nos permitir apresentar um sistema de forma padronizada, favorecendo o desenvolvimento do mesmo, este software já está bem consolidado no mercado, sendo utilizado por diversas empresas grandes atualmente. Entre suas principais formas de utilização, destaca-se: Diagrama de classes, diagramas de casos de uso e diagramas de atividades.

2.15. AMAZON WEB SERVICES

O Amazon Web Services(AWS) é uma plataforma de computação em nuvem, fornecida por uma das maiores empresas do mundo. A plataforma fornece alguns modelos de serviço em nuvem, são esses:

- Infraestrutura como Serviço (IaaS): segundo o site da google cloud, esse modelo de serviço é focado em oferecer recursos de infraestrutura sob demanda, como armazenamento, rede e virtualização.

- Plataforma como Serviço (PaaS): de acordo com o site da IBM, as PaaS buscam fornecer uma plataforma em cloud completa, flexível e com custo reduzido, para auxiliar em diversas etapas do desenvolvimento de software, sendo elas: desenvolvimento, execução e gerenciamento de softwares já em produção.

- Software como Serviço – (SaaS): é uma das formas de serviço que a AWS oferece dentro da sua plataforma, disponibilizando softwares por meio da internet como serviço, ao utilizar esse modelo, sua empresa não precisa se preocupar com a manutenção, instalação e atualização desses softwares, tornando tudo mais simples e fácil.

2.16. AMAZON EC2

De acordo com o site da própria fornecedora do serviço, o Amazon Elastic Compute Cloud ou Amazon EC2, oferece uma plataforma de computação em nuvem mais ampla e aprofundada, com diversos tipos de instâncias e opções de processadores, armazenamentos, redes e sistemas operacionais, tudo isso com uma alta escalabilidade, tornando muito mais vantajoso e acessível para pequenas e grandes empresas optarem pela utilização de seus softwares em ambiente de nuvem.

2.17. AMAZON RDS

O Amazon RDS é uma plataforma que possui uma coleção de serviços gerenciados, com a finalidade de facilitar a configuração, operação e escalabilidade de banco de dados na nuvem. Além de oferecer suporte a aplicações crescentes, com alta disponibilidade, taxa de transferência e escalabilidade de armazenamento, o RDS exclui a necessidade de um gerenciamento mais ativo de seus bancos de dados, tornando algo mais automático as operações de manutenção, que por sua vez eram um trabalho complexo e demorado, possibilitando a realocação do tempo gasto com esse tipo de atividade.

2.18. DOCKER

De acordo com a IBM, Docker é uma plataforma que permite nós, desenvolvedores, gerenciar contêineres executáveis que combinam código fonte de aplicativos com algumas estruturas de sistemas operacionais. Além de simplificar o desenvolvimento, entrega e distribuição de aplicativos, o Docker vem se destacando por facilitar o *deploy* de novos aplicativos em ambiente de cloud, tudo isso se deve a sua versatilidade e escalabilidade que essa ferramenta nos proporciona.

3. SISTEMA / APLICAÇÃO WEB

Um sistema web, nada mais é que um software hospedado na internet, mas também podemos chamar de sistema web qualquer aplicação que não necessariamente esteja na web, porém, utilize em sua base tecnologias voltadas para o desenvolvimento web, como por exemplo: HTML, CSS, JavaScript, dentre outras.

3.1. VANTAGENS DE UTILIZAR UM SISTEMA WEB

3.1.1. Aumento na produtividade

O aumento na produtividade ao utilizar um sistema web está diretamente relacionado a alta disponibilidade que esse tipo de sistema possui, uma vez que, este pode ser acessado a qualquer momento e de qualquer computador que possua um navegador web. Tornando assim, a resolução dos problemas mais ágil e aumentado a eficácia do sistema como um todo.

3.1.2. Custo / Benefício

Softwares Web possui um custo-benefício bem maior que as demais modalidades presentes no mercado atualmente. Ao optar por esse tipo de sistema, a entidade que o adquiriu se livra da necessidade de adquirir máquinas mais potentes para executar o software, uma vez que, grande parte do processamento se torna uma tarefa dos servidores, deixando sob responsabilidade das máquinas do cliente, apenas exibição das informações. Por isso, o custo-benefício de um software web é um ponto importante na hora de escolher qual plataforma irá adquirir.

3.1.3. Integração

Algumas empresas já possuem seus próprios softwares para executar tarefas do cotidiano. Portanto, conseguir uma boa integração entre esses softwares pode não ser uma tarefa tão simples, ainda sim um sistema web facilita muito os processos de integração, uma vez que, possibilita a hospedagem de tudo em um único servidor e isso acaba tornando a integração mais eficiente.

3.1.4. Segurança

Com a utilização de softwares web, o cliente tem a liberdade de adotar backups automáticos e regulares, evitando a perda de dados por falta de cuidado com os servidores locais. Outra funcionalidade importante para a segurança da sua aplicação, é o controle das permissões de cada usuário, estas podem ser administradas de acordo com a preferência de quem adquire o software, garantindo assim, total controle sobre o seu sistema e mantendo a integridade das suas informações. Quando optamos por utilizar um software web, a grande maioria das linguagens já possuem padrões e bibliotecas para auxiliar a criação de sistemas mais robustos e seguros.

3.1.5. Atualizações

Uma das grandes necessidades para uma empresa que busca um sistema nos dias atuais, é a alta disponibilidade por atualizações e melhorias. Ao optar por um software web isso deixa de ser um problema, pois, caso venha a surgir a carência por mais recursos de hardware, essa necessidade pode ser sanada por um simples suporte técnico na grande maioria dos provedores desse tipo de serviço, sem a busca por novos recursos ou substituição de componentes de forma manual.

4. ANÁLISE E PROJETO DO SISTEMA

4.1. ESCOPO

Um escopo de projeto nada mais é que um levantamento de todo o trabalho que se faz necessário, para obter um determinado produto, serviço ou resultado. Reunindo diversas informações necessárias e relevantes sobre um determinado projeto, dentre essas informações podemos dar ênfase à os objetivos esperados ao final do projeto, prazos, custos e entregas. Além dessas informações podemos citar e estabelecer os limites de um projeto e os critérios utilizados para validação e aceitação do mesmo.

De acordo com o Guia PMBOK, uma das maiores referências atualmente em gestão de projetos, o escopo do projeto deve contar apenas o necessário para que o projeto seja concluído com sucesso. Entregar mais do que o cliente pediu recebe o nome de *gold planting,* prática que é considerada arriscada e pode colocar o projeto que faz esse tipo de prática em risco.

Para o Trabalho de conclusão de Curso em questão, foi desenvolvido o seguinte Escopo:

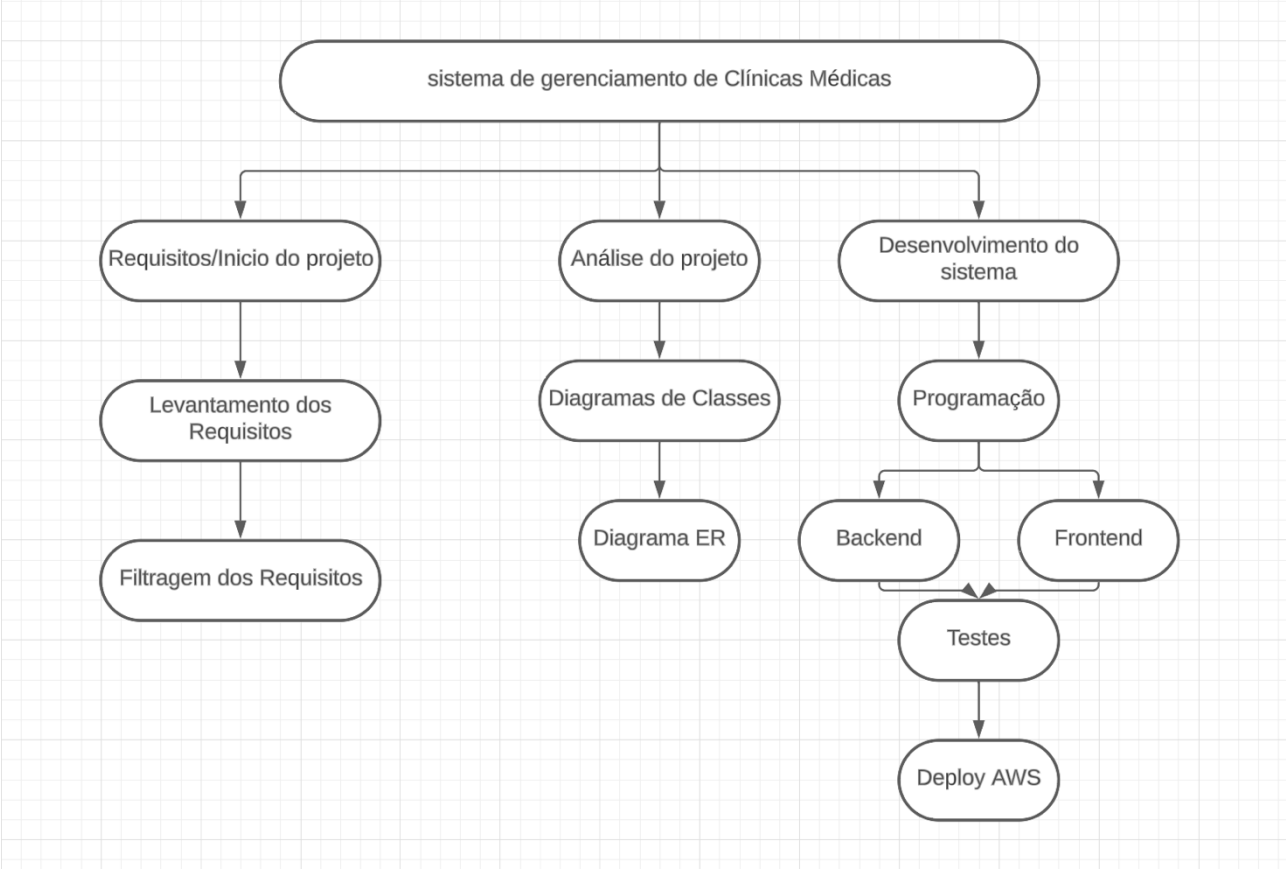


Figura 2 - Escopo do TCC

4.2. PROBLEMAS A SEREM SANADOS COM A IMPLANTAÇÃO DO SISTEMA

Com a implantação do sistema, busca-se uma melhora em todo o processo de gerenciamento de uma clínica médica. Dentre essas melhorias, podemos citar: Agendamentos mais consistentes e mais práticos que da forma convencional; Melhor organização das consultas diárias; facilitação na visualização da produção de cada médico cadastrado no sistema; agilidade na busca de informações relacionadas ao prontuário para cada paciente; etc.

4.3. REQUISITOS E FUNCIONALIDADES

1. Manter Paciente.
2. Manter Médico.
3. Agendar Consulta.
4. Atualizar dados cadastrais (Médico e Paciente).
5. Visualizar consultas do dia.
6. Visualizar consultas anteriores dos pacientes.
7. Buscar por paciente.
8. Emitir relatórios de consultas realizadas.
9. Emitir relatórios de pacientes.
10. Manter diagnóstico no momento da consulta.
11. Enviar ao paciente uma notificação do dia e horário da consulta.
12. Gestão de agenda.

4.4. DIAGRAMA DE CLASSES

Diagrama de classes, nada mais é do que uma representação das entidades presentes no nosso sistema, nele podemos identificar os atributos presentes em cada classe e seus respectivos modificadores de acesso, os métodos e os relacionamentos entre estas entidades. Para o desenvolvimento do sistema em questão foi criado o diagrama abaixo (Figura 1) para auxiliar na implementação das classes presentes.

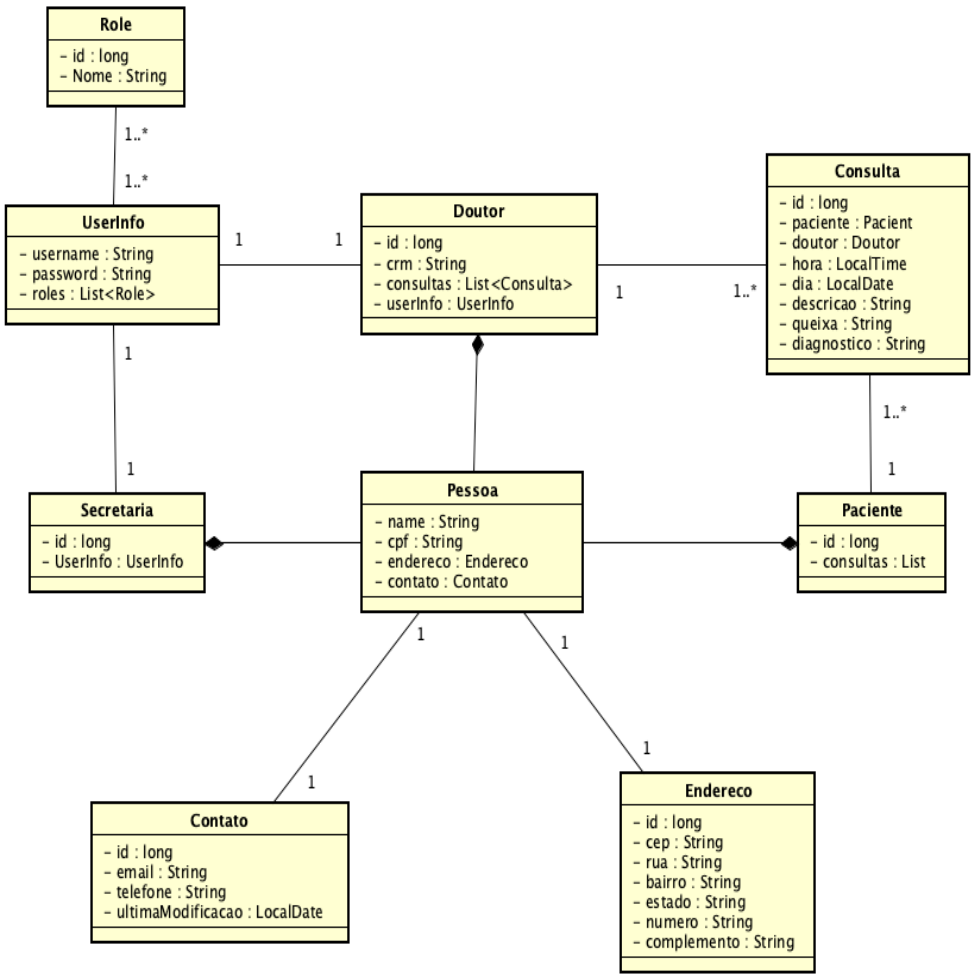


Figura 3 - Diagrama de Classes

4.5. DIAGRAMA DE CASOS DE USO

O diagrama de casos de uso é utilizado para expressar todas as funcionalidades desejadas para um sistema. De acordo com o site Devmedia, a grande parte dos problemas encontrados em sistemas orientados a objetos vem desde a construção problemática do modelo. Afim de se evitar problemas na construção do sistema em questão, foi desenvolvido o seguinte diagrama (Figura 2) para auxiliar.

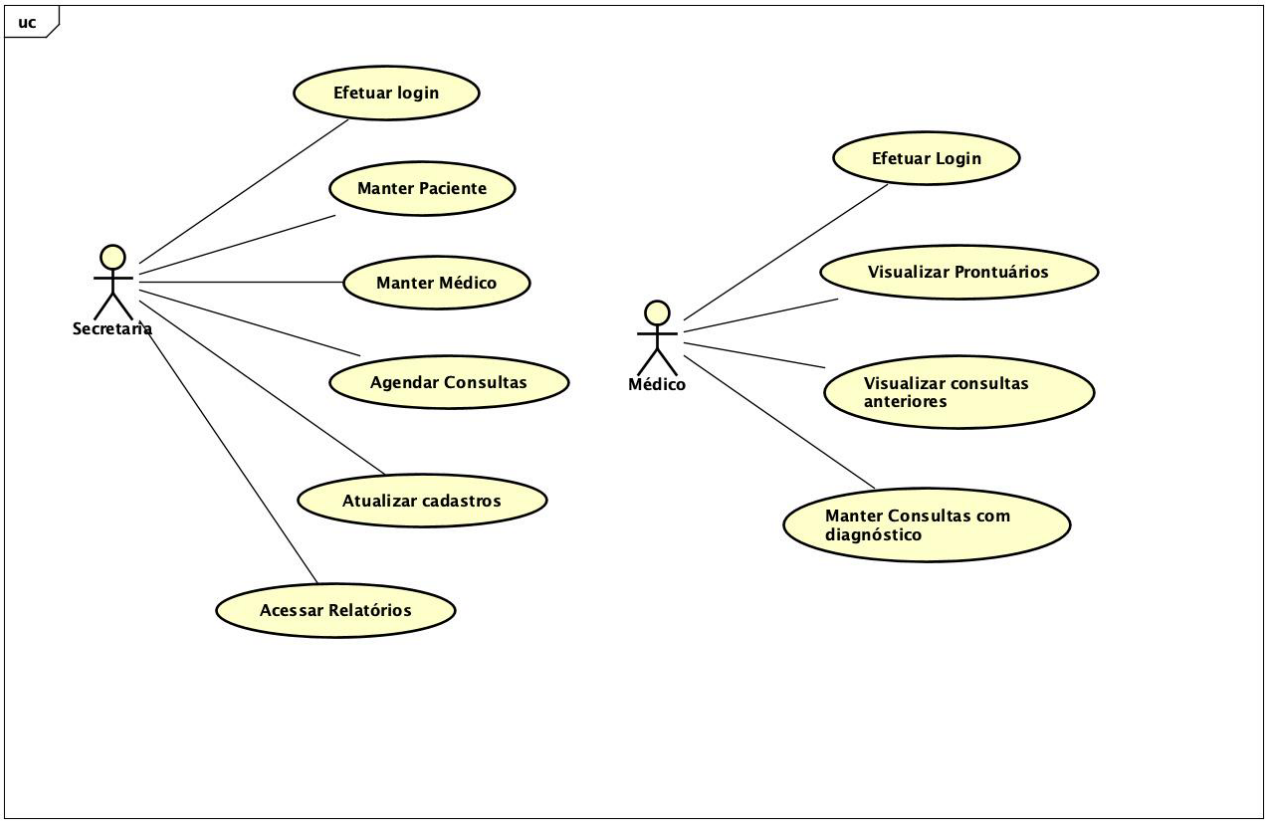


Figura 4 - Diagrama Casos de uso Geral

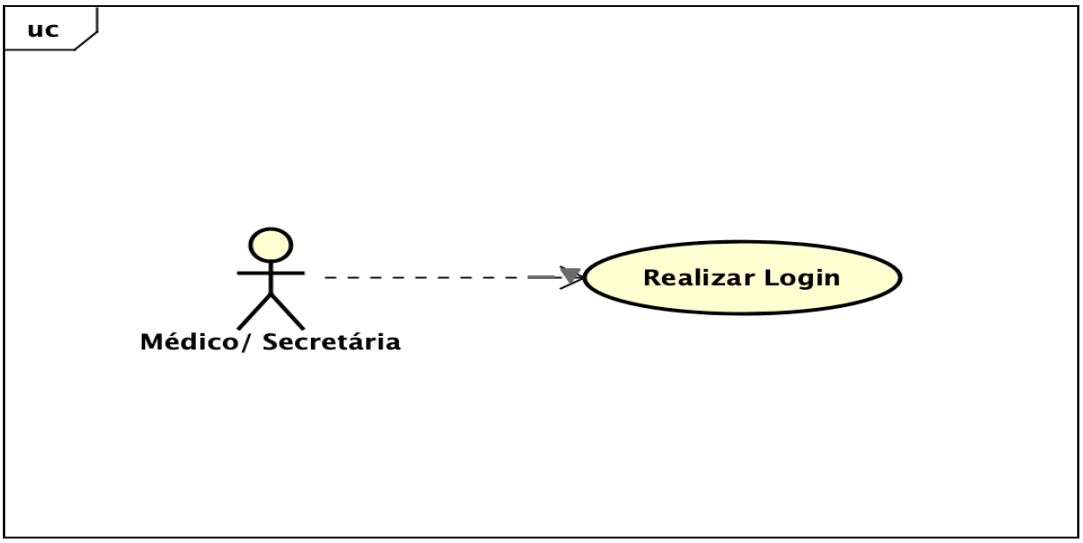


Figura 5 - Caso de uso: Login no sistema

1. **Finalidade:** Realizar o acesso ao sistema.
2. **Ator:** Médico e/ou Secretária.
3. **Evento Inicial:** O Ator preenche os campos nome de usuário e senha.
4. **Fluxo Principal:**
   a. O Ator acessa o sistema por meio de um navegador web, e não possui um token válido em seu *LocalStorage.*
   b. O sistema exibe a tela de login.
   c. O Ator preenche os campos usuário e senha e clica em ‘’ENTRAR’’.
   d. O usuário consegue entrar no sistema e tem uma notificação informando que o login foi realizado com sucesso.
5. **Fluxo Alternativo:**

   A1. O médico não possui cadastro no sistema.
   a. O sistema informa a partir de uma notificação que as credenciais então inválidas ou o médico não possui cadastro no sistema.
   b. O médico deve solicitar a uma secretária para que realize seu cadastro no sistema.
   c. Após o cadastro médico realizado, O mesmo deve realizar o login no sistema a partir das credenciais informadas no momento do cadastro.

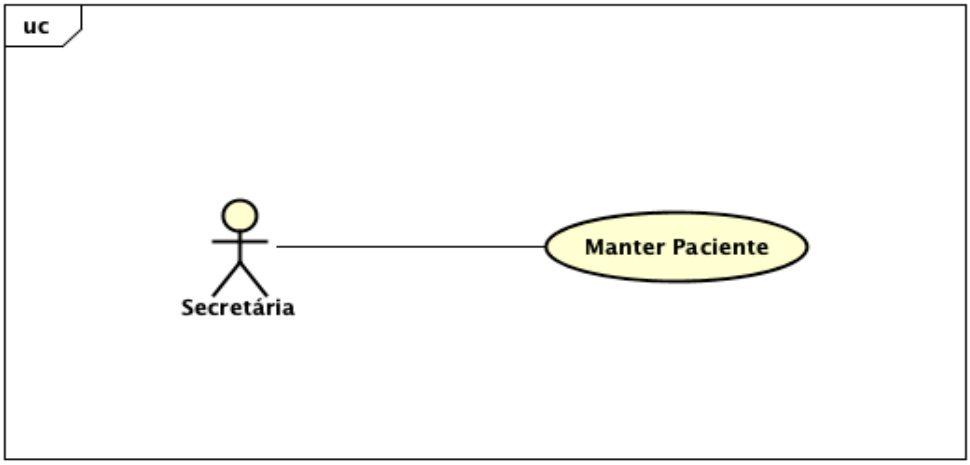


Figura 6 - Caso de uso: Manter Paciente

1. **Finalidade:** Realizar cadastro de um novo paciente no sistema.
2. **Ator:** Secretária.
3. **Evento Inicial:** O ator seleciona a opção paciente no menu lateral do sistema. Após a seleção no menu lateral o ator deve clicar em qual ação deve realizar, para o caso de uso em questão, o mesmo deve clicar na opção ‘’cadastrar paciente’’.
4. **Fluxo principal:**
    a. Após o passo a passo do evento inicial, o sistema irá exibir a tela de cadastro de pacientes.
    b. O Ator deve preencher todos os campos com as informações do paciente.
    c. Depois do fornecer todas informações, clicar em ‘’cadastrar’’.
    d. O sistema irá exibir uma notificação se o cadastro foi realizado com sucesso ou não.
5. **Fluxo Alternativo:**

    A1. A ator deseja desistir de realizar o cadastro.
    a. Basta clicar no botão ‘’voltar’’, no canto superior esquerdo da tela de cadastro

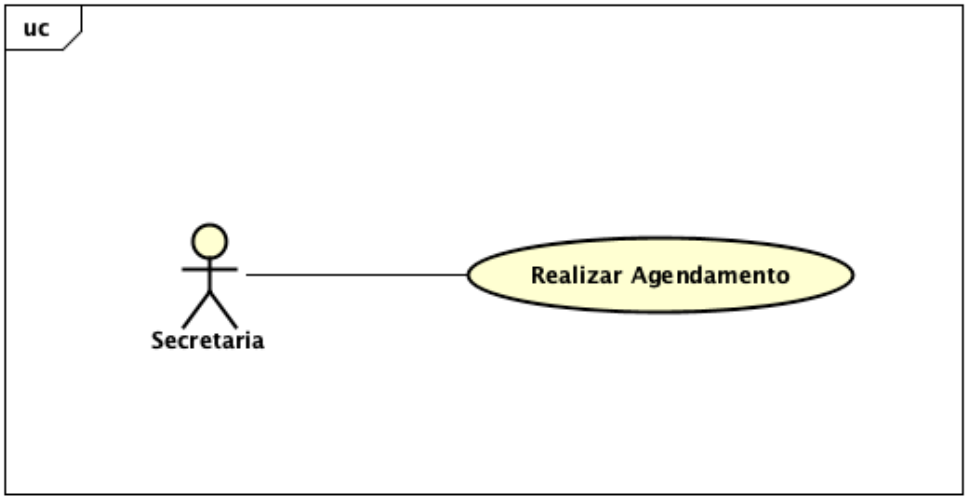


Figura 7 - Caso de uso: Realizar Agendamento

1. **Finalidade:** agendar um horário para o paciente.
2. **Ator:** Secretária.
3. **Evento inicial:** O ator clica na opção ‘’agendamentos’’ no menu lateral do sistema e dentro da tela de agendamentos, deverá selecionar a opção novo agendamento.
4. **Fluxo Principal:**
    a. Após realizar o evento inicial, o sistema irá exibir a tela de agendamentos.
    b. Na tela de agendamentos o ator de selecionar a data desejada no calendário situado a esquerda ao centro da tela.
    c. Depois de selecionar a data para o agendamento, o ator deve selecionar o médico.
    d. Com a seleção do médico e data, o sistema vai exibir uma tabela, contendo os horários vagos para o dia.
    e. Basta clicar no horário que desejar e preencher a ficha do paciente.
    f. Por fim, com o preenchimento da ficha, o ator deverá clicar em “concluir”.
5. **Fluxo Alternativo:**

    A1 – O Ator deseja cancelar a ficha de lançamento do agendamento:

    a. O ator deve apenas clicar no botão cancelar no *dialog* da ficha.

4.6. DIAGRAMA ENTIDADE RELACIONAMENTO

Segundo o próprio site da ferramenta (Lucidchart), a qual utilizei para desenvolver este diagrama (Figura 3), o diagrama ER trata-se de uma espécie de fluxograma, utilizado para ilustrar as entidades que se relacionam entre si em um sistema, comumente utilizado para projetar um banco de dados relacional, o diagrama ER que representa o banco de dados do sistema em questão está ilustrado abaixo.

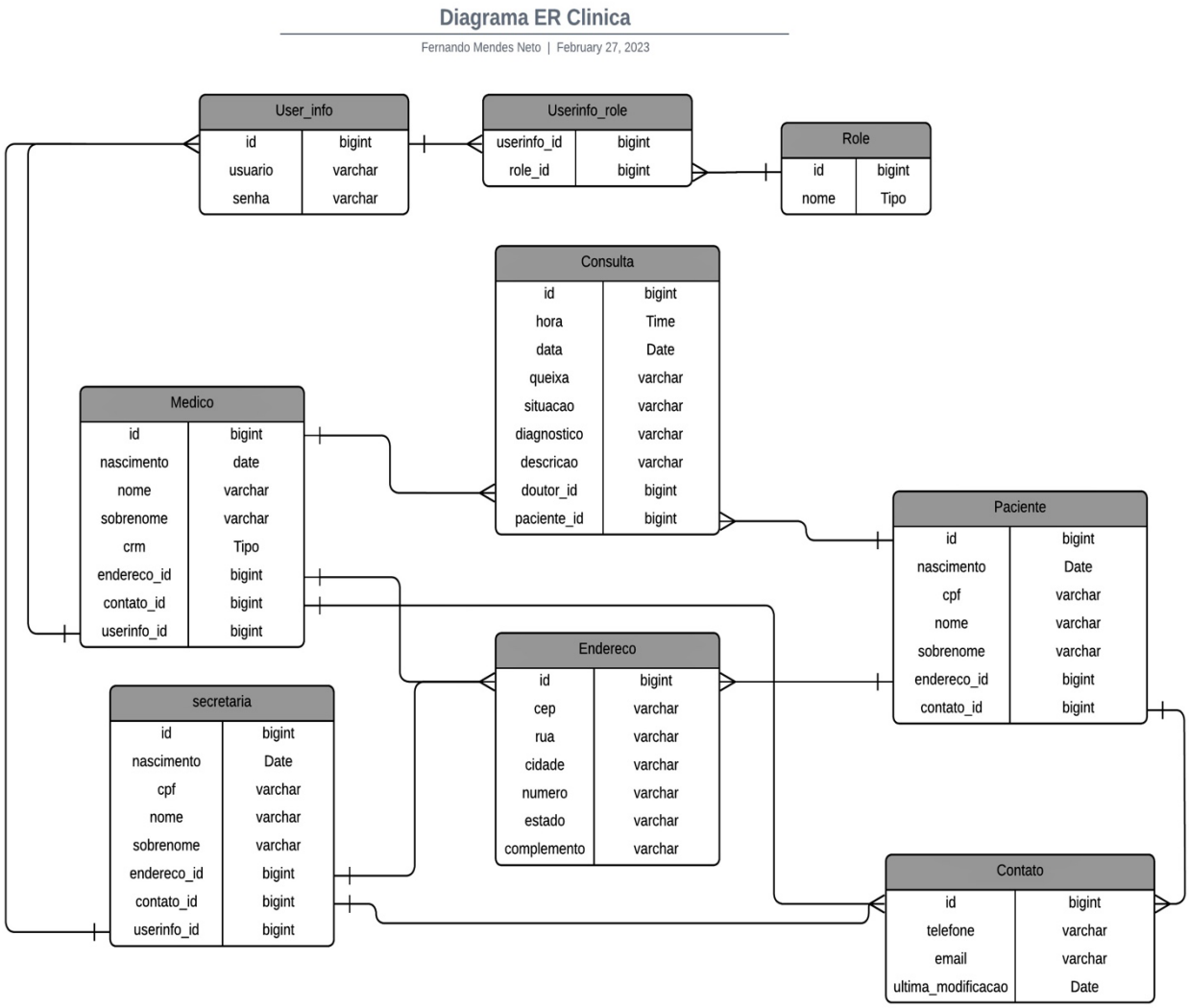


Figura 8 - Diagrama Entidade Relacionamento

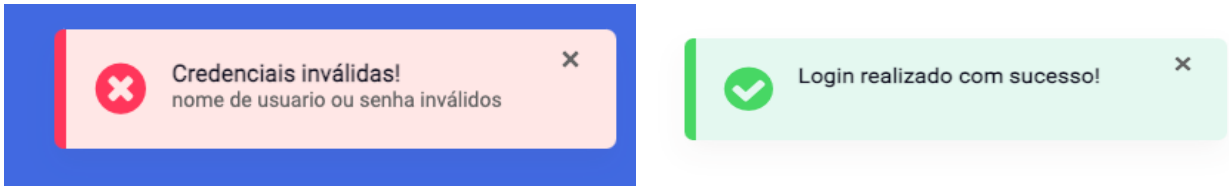
5. INTERFACES DO SISTEMA

Este capitulo tem a finalidade não só de apresentar grande parte das interfaces do sistema, mas também expor um pouco das funcionalidades e funcionamento destas. Todas as interfaces foram desenvolvidas com HTML, CSS, Typescript e alguns componentes prontos do Angular Material, para uma maior destreza no desenvolvimento do *Frontend*.

5.1. LOGIN

Figura 9 - Interface de Login

A interface de login foi desenvolvida em *HTML*, *CSS* e *TS,* com o auxilio do *framework* Angular, para a geração dos componentes. Ao clicar em “ENTRAR”, com os campos preenchidos de acordo com as validações necessárias, o *Frontend* faz uma requisição ao *backend,* caso possua algum usuário no banco de dados com essas informações, o *backend* retorna para ao *frontend* um token, o qual será utilizado para realizar requisições dentro do sistema de forma *stateless.* Se algum dos dados estiver incorreto ou não possuir registros compatíveis, o usuário será alertado através de uma notificação.



5.2. HOME

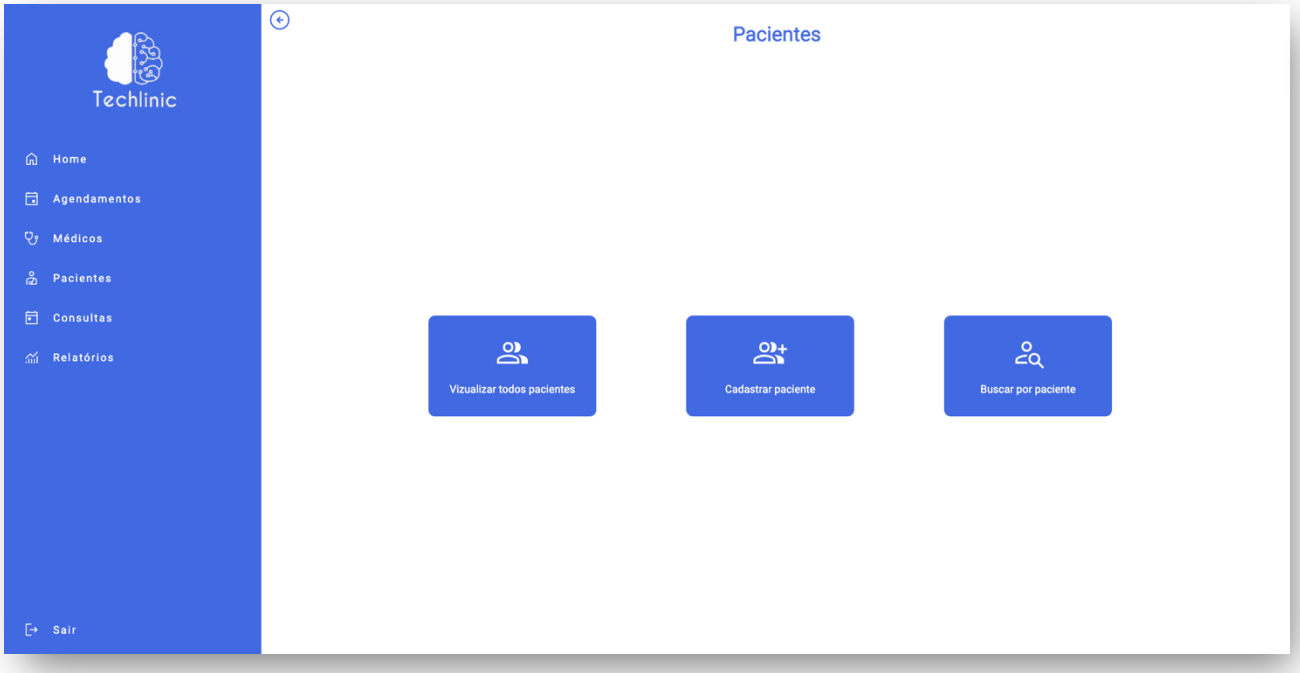


Figura 10 - Interface Inicial do sistema

Do lado esquerdo está o menu responsivo, com todas as categorias presentes no sistema. Ao lado direto, os quadrados representam as opções para cada categoria e mudam de acordo com a categoria selecionada no menu. Ao clicar em um dos quadrados, o usuário será direcionado a outra página. Por exemplo, caso o usuário, na figura 10, clique em “Cadastrar paciente” será imediatamente direcionado para a figura 11(interface de cadastro dos pacientes).

5.3. CADASTRO DE PACIENTES

Figura 11 – Interface para cadastro de pacientes

Este é o formulário de cadastro, todos no sistema seguiram esse mesmo padrão. O formulário conta com uma série de validações para que seja liberado o envio do mesmo para o *backend,* afim de manter os dados organizados e no padrão desejado. A parte de endereço utilizou o ViaCep para preenchimento automático dos campos de Cidade, rua, bairro e estado, ficando apenas para a obrigação do usuário o preenchimento do cep, o qual é utilizado para puxar os demais dados do ViaCep e do número. O campo de complemento pode ser preenchido ou não, dependendo única e exclusivamente da necessidade do usuário.

5.4. LISTAGEM DE PACIENTES

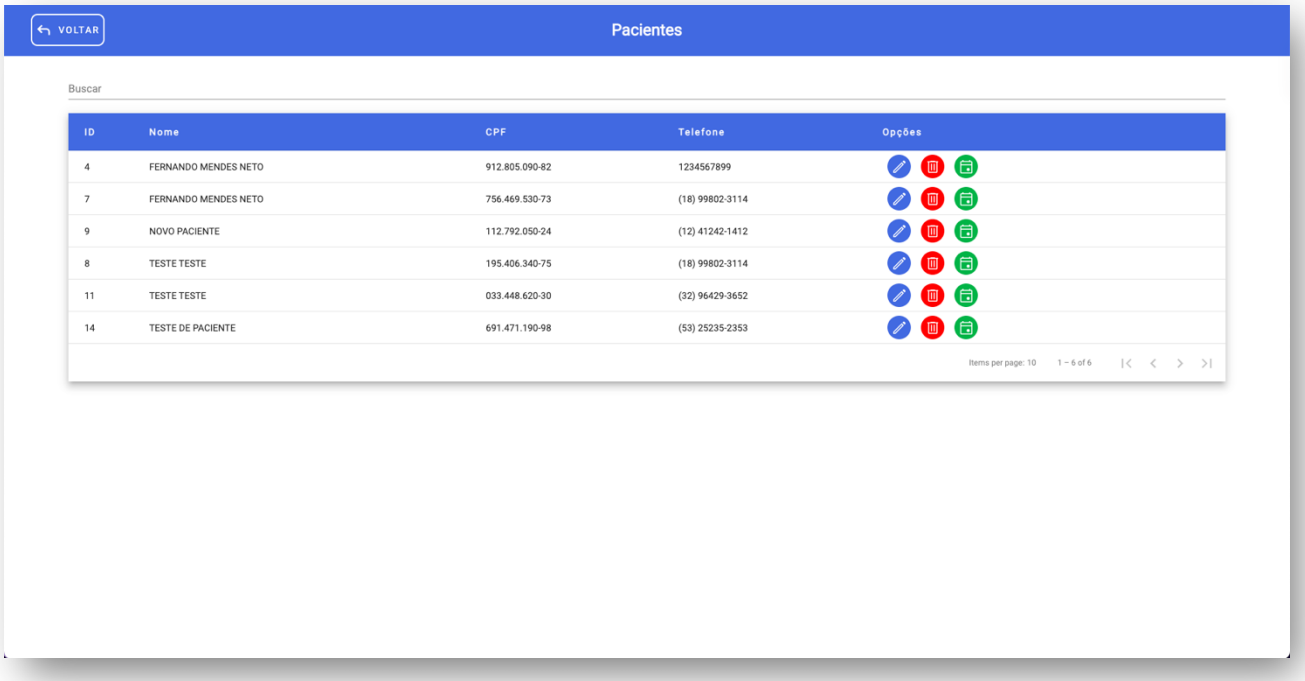


Figura 12 - Interface de Listagem dos pacientes

Esse é um exemplo de como as entidades serão listadas no sistema, possuindo uma barra de busca no topo da tabela, que filtra as entidades por id, nome, cpf e telefone, trazendo uma maior versatilidade e agilidade na hora de encontrar um paciente em especifico. A tabela possui também uma coluna de opções o botão azul será utilizado para editar os dados cadastrais, o botão vermelho para excluir e o verde para criar um agendamento.

5.5. CADASTRO DE MÉDICOS

Figura 13 – Interface para cadastro de Médicos

Este é um protótipo da interface responsável pelo cadastro dos médicos presentes dentro da clínica. Dentro dela podemos definir o intervalo de atendimento onde aquele médico irá atender, bem como suas credencias que ele utilizará para realizar *login* no sistema e o valor de suas consultas. O intervalo de atendimento é vital para o funcionamento da tela representada na figura 14, uma vez que, os horários exibidos ali surgem a partir desses dois horários cadastrados nessa tela.

5.6. INTERFACE DE AGENDAMENTO CONSULTAS

Figura 14 - Interface de agendamento

Nessa interface serão realizados os agendamentos para cada médico. O usuário primeiramente irá escolher uma data e posteriormente selecionar o médico, após a seleção do médico desejado, o sistema listará todos os horários que aquele médico tem disponível, inclusive os que já possuem pacientes agendados, será possível identificar os horários já ocupados através dos campos paciente, telefone e status já preenchidos na tabela, isso irá bloquear o horário, não permitindo demais agendamento em um mesmo horário para o médico e dia selecionado.

5.7. INTERFACE DE CONSULTAS DO DIA

Figura 15 – Interface de Consultas do dia

Aqui temos a interface de consultas do dia, está é responsável por exibir para cada médico suas consultas diárias. Basta o médico se selecionar no input, que o sistema irá listar todas as consultas do presente dia e fornecerá a opção para iniciar o atendimento do paciente desejado.

5.8. TELA DE ATENDIMENTO

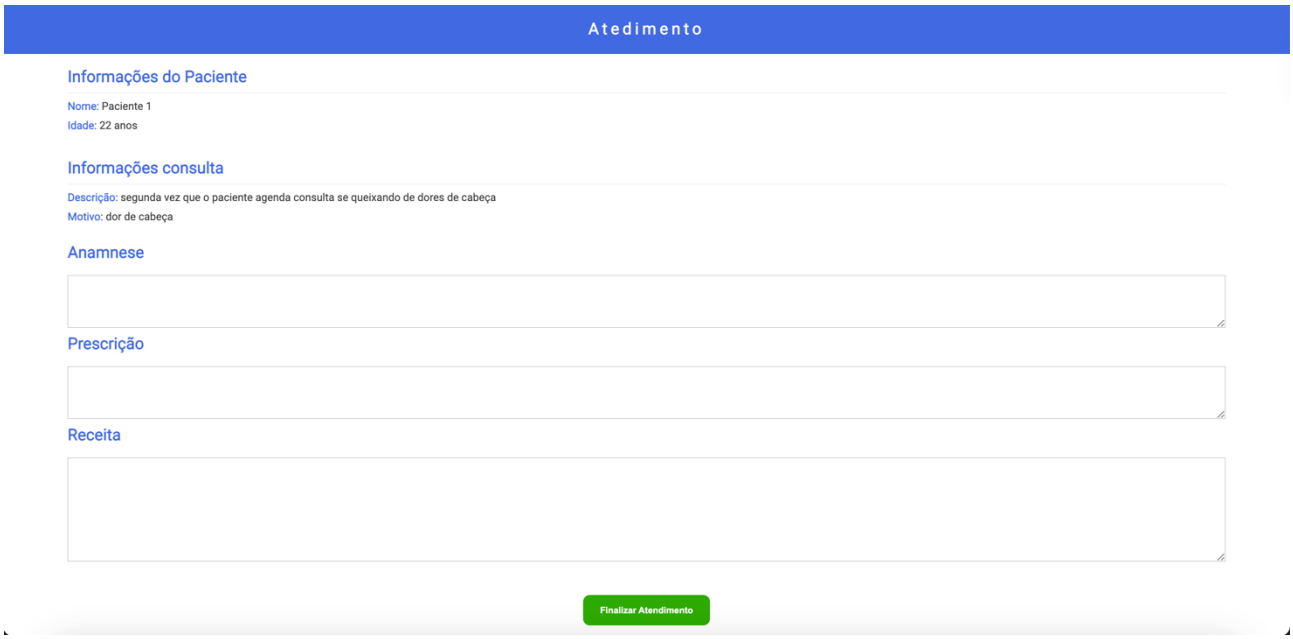


Figura 16 - Interface de Atendimento

Na interface de atendimento o médico terá acesso a informações referentes ao paciente e a consulta. Nessa tela deverá ser preenchido a anamnese, prescrição e a receita para o paciente e logo clicar no botão para finalizar o atendimento. Para acessar a tela representada na figura 16, o médico deverá clicar no botão iniciar na tela da figura 15.

6. DESENVOLVIMENTO

6.1. ESTRUTURAÇÃO DOS PACOTES

Os pacotes da aplicação tanto no *backend,* quanto no *frontend,* foi distribuído de acordo com as responsabilidades de cada classe. Essa estruturação foi aplicada para facilitar a busca e separar bem as classes, afim trazer maior agilidade no processo de desenvolvimento e futuramente as manutenções.

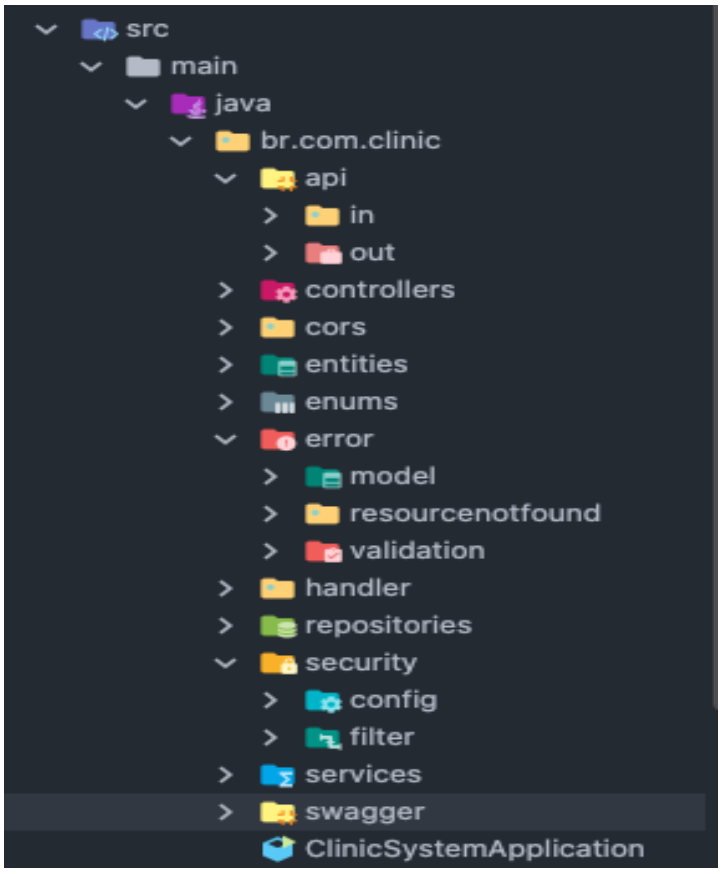


Figura 17 - Estrutura de pacotes *Backend*

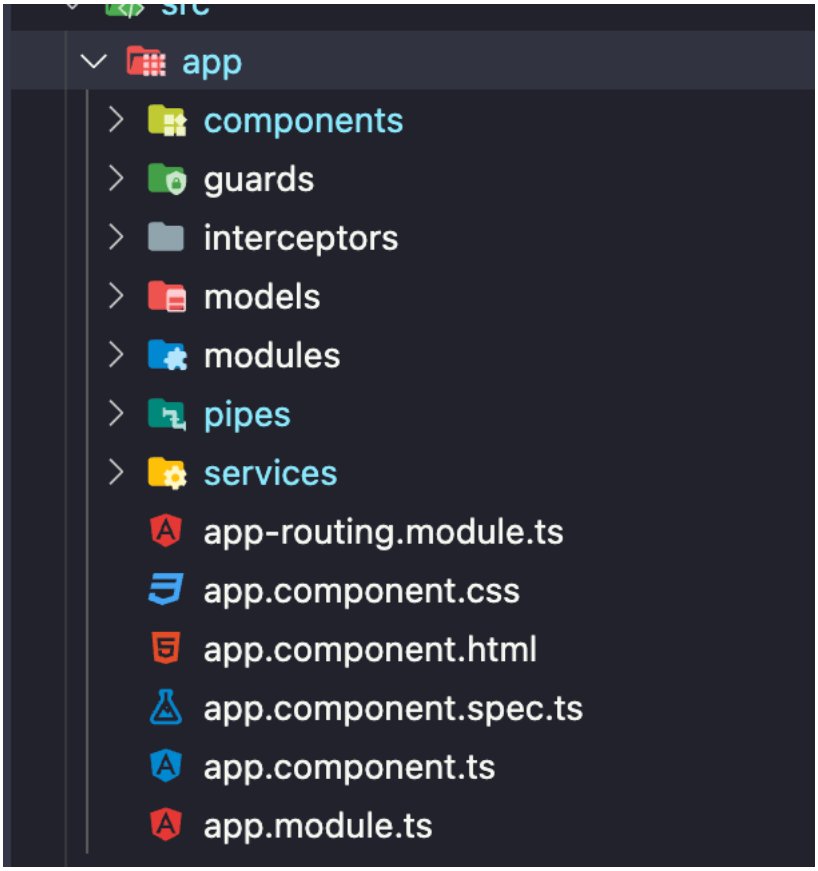


Figura 18 - Estrutura de pacotes *Frontend*

6.2. FLUXO DAS REQUISIÇÕES

O Sistema adotou uma autenticação do tipo *stateless*, o que acaba modificando o fluxo das requisições que, por sua vez, devem possuir um local para armazenar um token na aplicação *frontend* e a cada requisição este token, previamente adicionado ao “LocalStorage” do navegador, assim que o usuário faz login no sistema, deverá ser alocado ao Header “Authetication” da requisição http, a qual antes de acessar o *controller* do servidor *backend,* passará por um filtro de autenticação para a verificação deste token. Caso tudo esteja valido, o filtro seguirá com a requisição e a resposta será devolvida ao frontend para tratamento dos dados e informações.

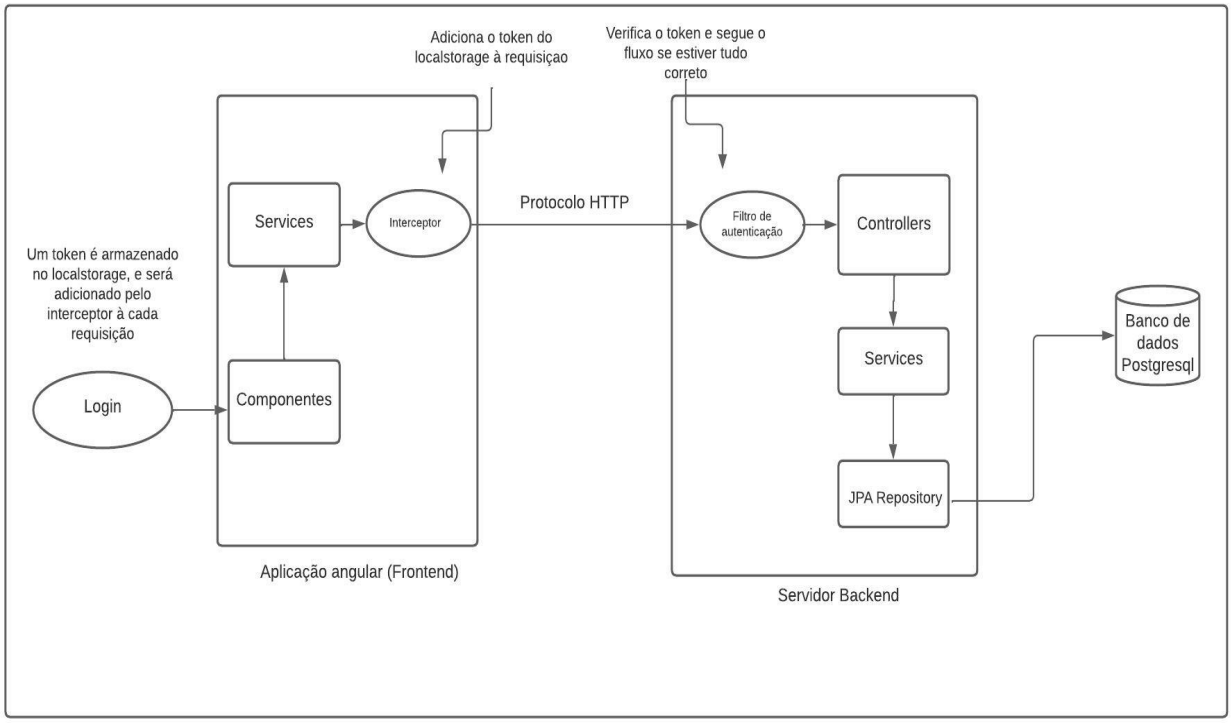


Figura 19 - Fluxo das requisições

6.3. SEGURANÇA

6.3.1. Backend

Toda requisição feita para a API do nosso sistema é verificada através do nosso filtro de requisição (Figura 13), este é responsável por verificar se essa requisição foi realmente feita por algum usuário válido do nosso sistema. O Spring Security possui uma série de ferramentas para facilitar a implantação de camadas de segurança em nossas aplicações e foi a biblioteca escolhida para auxiliar no desenvolvimento dessa etapa. Abaixo temos algumas classes responsáveis pela segurança backend do nosso sistema.

```
public class AutheticationTokenFilter extends OncePerRequestFilter {

    private final TokenService tokenService;
    private final UserInfoRepository userInfoRepository;

    @Autowired
    public AutheticationTokenFilter(TokenService tokenService, UserInfoRepository userInfoRepository) {
        this.tokenService = tokenService;
        this.userInfoRepository = userInfoRepository;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
            FilterChain filterChain) throws ServletException, IOException {

        String token = recoverToken(request);
        boolean tokenIsValid = tokenService.isTokenValid(token);

        if (tokenIsValid) {
            autheticateToken(token);
        }

        filterChain.doFilter(request, response);
    }

    private void autheticateToken(String token) {

        String username = tokenService.getUsernameToken(token);
        UserInfo userInfo = userInfoRepository.findByUsername(username);
        Authentication authetication = new  UsernamePasswordAuthenticationToken(userInfo.getUsername(),
userInfo.getPassword(), userInfo.getAuthorities());

        SecurityContextHolder.getContext().setAuthentication(authetication);
    }

    private String recoverToken(HttpServletRequest request) {
        String token = request.getHeader("Authorization");
        if (token == null || !token.startsWith("Bearer ")) {
            return null;
        }
        return token.substring(7); }
}
```

Classe de filtragem das requisições.

```
public class SecurityConfiguration {

    private final TokenService tokenService;
    private final UserInfoRepository userInfoRepository;

    @Autowired
    public SecurityConfiguration(TokenService tokenService, UserInfoRepository
userInfoRepository) {
        this.tokenService = tokenService;
        this.userInfoRepository = userInfoRepository;
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration authenticationConfiguration)
throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .cors().and()
                .csrf().disable()
                .authorizeRequests(auth -> {
                    auth.antMatchers(HttpMethod.POST).permitAll();
                    auth.antMatchers(HttpMethod.GET).permitAll();
                    auth.anyRequest().authenticated();
                })
                .addFilterBefore(new AutheticationTokenFilter(tokenService,
userInfoRepository), UsernamePasswordAuthenticationFilter.class)

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        return http.build();
    }
}
```

Classe de configuração do Spring Security.

6.3.2. Frontend

Assim como o backend, o frontend também exige uma camada de segurança para proteger nossa API. Dentre algumas responsabilidades do frontend, voltadas para a segurança, podemos citar: Armazenamento do token, para que este seja utilizado em futuras requisições dentro do sistema; tratamento das requisições a partir de um filtro para adicionar ao Header da requisição, o token armazenado no momento do login, com informações referentes ao usuário que está efetuando a requisição ao backend; exibir a tela de login e tratar corretamente os dados(nome de usuário e senha), para que estes não estejam vulneráveis e sejam enviados corretamente para o backend ao realizar o login. Abaixo temos alguns trechos de códigos que implementam a camada de segurança do frontend.

```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {
    intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {

        const requestUrl = request.url;
        const loginUrl = environment.baseUrl + 'login';
        const token = localStorage.getItem('token');

        if (token
            && (requestUrl !== loginUrl)
            && (requestUrl.split("/")[2] !== "viacep.com.br")) {
            const newRequest = request.clone({ setHeaders: {'Authorization':
token}});
            return next.handle(newRequest);
        } else {
            return next.handle(request);
        }
    }
}
```

Classe de intercepção das requisições.

```
export class AutheticationService {

  baseUrl = environment.baseUrl;

  token?: responseToken;

  constructor(
    private http: HttpClient,
    private tokenService: TokenService,
    private router: Router,
    private toast: NgToastService
  ) { }

  submitLogin(value: any) {

    this.http.post<responseToken>(this.baseUrl + 'login', {
      "username": value.username,
      "password": value.password
    }).subscribe({
      next: (res) => {
        this.token = res;
        this.tokenService.saveTokenInStorage(this.token);
        this.toast.success({detail: "Login realizado com sucesso!"});
        this.router.navigate(['home']);
      },
      error: (erro) => {
        if(erro.status == 400) {
          this.toast.error({detail: "Credenciais inválidas!", summary: "nome
de usuario ou senha inválidos"});
        }else {
          this.toast.error({detail: "Não foi possível conectar ao
servidor!", summary: "Tente novamente mais tarde!"});
        }
      }
    });
  }

  verifyToken() {
    return this.http.get(this.baseUrl + 'isTokenValid')
  }
```

Classe de autenticação do sistema.

```
export class TokenService {

  constructor() { }

  saveTokenInStorage(token: responseToken) {
    localStorage.setItem('token', token.requestType + " " + token.token);
  }

  getToken() {
      return localStorage.getItem('token')
  }

  logoutToken() {
    localStorage.removeItem('token');
  }

  isLoggedIn(): boolean {
    return !!localStorage.getItem('token');
  }
}
```

Classe de serviço do Token de autenticação.

7. DEPLOY DA APLICAÇÃO NA NUVEM

A aplicação final desenvolvida, foi totalmente hospedada em um provedor de serviços em nuvem. A AWS foi a provedora dos serviços em nuvem escolhida, nela utilizamos dois serviços: EC2 e RDS. Na instância do EC2, apenas fizemos a instalação do Docker para executar a nossa imagem gerada a partir do empacotamento do *backend.* Já no RDS realizamos somente uma configuração padrão, para colocar o serviço do postgresql em funcionamento e permitir a conexão do EC2. Após os dois serviços estarem conectados e funcionando, devemos apenas realizar o mapeamento das portas do grupo de segurança de cada uma das instâncias, afim de liberar a comunicação e troca de informações entre o banco de dados e o container Docker em execução no EC2 com backend.

8. CONCLUSÃO

Ao fim desse trabalho, podemos concluir a grande importância que os *softwares* possuem dentro da rotina de uma clínica médica. Estes são capazes de controlar e facilitar diversos processos ali presentes, desde de agendamento de consultas, realização de atendimentos, armazenamento seguro e correto das informações dos pacientes e médicos, etc. Portanto, é primordial que clínicas que não possuam o seu software venham a adquirir. Outro ponto notório, é a importância da criação de um ambiente de nuvem para hospedar nosso sistema, de modo que ofereça ainda mais segurança às informações médicas ali mantidas, escalabilidade e disponibilidade.

9. REFERÊNCIAS

CORREIA, L. O. dos S.; PADILHA, B. M.; VASCONCELOS, S. M. L. **Métodos para avaliar a completitude dos dados dos sistemas de informação em saúde do Brasil: uma revisão sistemática. Ciênc. saúde coletiva,** Rio de Janeiro , v. 19, n. 11, Nov. 2014 Disponível em: <http://www.scielosp.org/scielo.php?script=sci_arttext&pid=S1413-81232014001104467&lng=en&nrm=iso>. Acesso em: 20 Out. 2022.

DEVMEDIA. **O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML.** 2012 Disponível em: <https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408> Acesso em: 20 Fev. 2023.

GALVAO, M. C. B.; RICARTE, I.L.M. **Prontuário do paciente**. Rio de Janeiro: Guanabara Koogan, 2012.

GOOGLE CLOUD. O que é IaaS?. Disponível em: <https://cloud.google.com/learn/what-is-iaas?hl=pt-br#:~:text=A%20IaaS%20em%20computa%C3%A7%C3%A3o%20em,recursos%20de%20rede%20e%20armazenamento.>

IBM. O que é Paas? (Platform-as-a-service). Disponível em: https://www.ibm.com/br-pt/topics/paas#:~:text=IBM-,O%20que%20%C3%A9%20PaaS%20(Platform%2Das%2Da%2DService,execu%C3%A7%C3%A3o%20e%20gerenciamento%20de%20aplicativos.

IBM. O que é Docker?. Disponível em: https://www.ibm.com/br-pt/topics/docker

INSPER JR. **A SUA EMPRESA PRECISA DE SISTEMAS DE INFORMAÇÃO?**. 2020 Disponível em: <https://insperjr.com.br/sistemas-de-informacao-importancia/> Acesso em: 23 Out. 2022.

LUCIDCHART. **O que é um diagrama entidade e relacionamento?**. Disponível em: <https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408> Acesso em: 23 Fev. 2023.

PAUL DEITEL & HARVEY DEITEL. **Java como Programar – 10° Edição**. 2016.

ROSA, C. D. P.; ROVAI, R. L.; MATHIAS, D. **Sistemas de Informação na Área da Saúde: A INFORMAÇÃO CLÍNICA COMO INSTRUMENTO DE TRABALHO PARA OS PROFISSIONAIS DE SAÚDE.** 10, Jun. 2015 Disponível em: <https://www.e-publicacoes.uerj.br/index.php/polemica/article/view/17839/13289>. Acesso em: 22 Out. 2022.

ANEXO A – CRONOGRAMA

A fim de se concluir os objetivos desejados para o presente trabalho, foi desenvolvido um cronograma (Tabela 1). De acordo com o PMBOK, o Cronograma é uma ferramenta de planejamento e controle das atividades a serem executadas durante o período estimado para um projeto. O cronograma detalha as atividades do projeto frente aos prazos de execução e permite que se tenha um controle sobre o andamento das atividades elencadas para ele.

Atividade/Mês	novembro	dezembro	janeiro	fevereiro	abril	maio	junho	julho	agosto
Levantamento de bibliografia	X								
Aprimoramento de linguagem Java / Spring	X								
Curso arquitetura de microsserviços		X							
Curso Amazon Web Services		X							
Desenvolvimento diagrama de classes			X						
Desenvolvimento entidade e relacionamento			X						
Casos de uso			X						
Diagrama casos de uso				X					
Narrativas casos de uso				X					
Desenvolvimento do Backend				X	X	X			
Desenvolvimento do Frontend					X	X	X		
Testes							X		
Deploy da aplicação na AWS							X		
Elaboração documentação final								X	
Defesa									X

Figura 20 - Cronograma