CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA
PAULA SOUZA

Faculdade de Tecnologia da Baixada Santista
Rubens Lara

Curso Superior de Tecnologia em
Sistemas para Internet

CAROLINA OLIVEIRA FERREIRA FREITAS
LUIZ FERNANDO DIAS SORBELLO
MIGUEL ARCANJO RAMOS

EPHEMERIS:
aplicativo mobile de agendamento de especialistas online

Santos, SP
2021

CAROLINA OLIVEIRA FERREIRA FREITAS

LUIZ FERNANDO DIAS SORBELLO

MIGUEL ARCANJO RAMOS

EPHEMERIS:
aplicativo mobile de agendamento de especialistas online

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia da Baixada Santista Rubens Lara, como exigência para obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Esp. Jorge Luiz Chiara

Santos, SP

2021

RESUMO

Com avanço da tecnologia, o mundo está vivenciando uma melhor qualidade de vida, dentro deste contexto estão os especialistas (médicos e terapeutas) acompanhando este progresso, e a relação médico / paciente através do agendamento das consultas pode fazer parte desta evolução. Novas tecnologias estão sendo criadas e aperfeiçoadas a cada dia, facilitando o desenvolvimento de softwares na forma de aplicativos. Este projeto visa atender a uma parcela do mercado dos especialistas da área de saúde, sejam eles médicos ou terapeutas com relação ao agendamento das consultas, beneficiando desta forma os profissionais e pacientes. O sistema interage com dois tipos de usuários, o paciente (cliente) e o especialista (médico ou terapeuta responsáveis pelo controle da agenda), como usuário administrador é possível cadastrar profissionais, disponibilizar horários, ver informações sobre os pacientes, por parte dos médicos e verificar a agenda de consultas. O paciente, poderá acessar os especialistas e marcar um horário para consultar-se e/ou cancelar uma consulta se caso necessário.

Palavras-chave: Agendamentos. Consultas. Pacientes. Especialistas. Médicos e Terapeutas.

ABSTRACT

With the advancement of technology, the world is experiencing a better quality of life, within this context are the experts following this evolution, and the doctor / patient relationship through the appointment can be part of this evolution. Technologies are being created every day, making it easier to create and develop software in the form of applications. This project aims to serve a portion of the market of specialists, whether doctors or therapists regarding the scheduling of appointments, thus benefiting patients. The system interacts with two types of users, the client (patient) and the specialist (responsible for controlling the schedule). As an expert user you can register professionals, provide schedules, view patient information, and check the history of appointments. The patient will be able to access the specialists and make an appointment to consult and / or cancel an appointment if necessary.

Keywords: Schedules. Consultations. Specialists and Therapists.

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As tecnologias estão sendo criadas a cada dia, facilitando o desenvolvimento de *softwares* na forma de aplicativos para *smartfones*. Este projeto visa atender a uma parcela do mercado dos especialistas sejam eles médicos ou terapeutas com relação ao agendamento das consultas, beneficiando desta forma os pacientes.

Ao mesmo tempo que as tecnologias de *Softwares*, os *Smartfones* evoluíram na mesma proporção, para Steve (2014, p. 141) “Telefones, com o passar dos anos, foram ficando gradualmente mais inteligentes, sugados para dentro das gavetas, cada vez mais parecidos entre si. Mas foi somente com o Grande Salto Adiante[1] que eles finalmente alcançaram a consciência”.

1.1 JUSTIFICATIVA DO TEMA

Desenvolver e aplicar as técnicas de programação no ambiente mobile utilizando a linguagem de programação *Dart* para o ambiente mobile e um *webservice* com a linguagem de programação *Java* conectado a um banco de dados relacional, o acesso com *Login* e Senha, sendo o aplicativo *mobile* direcionado às plataformas *Android* e *iOS*, o público-alvo: os especialistas de medicina e terapeutas e os usuários destes serviços.

1.2 HIPÓTESE

Um paciente acompanhado de um especialista da área médica ou terapeuta tem como uma ferramenta de controle de agendamentos dos pacientes, o Aplicativo de Agendamento Especialistas *Online*, o aplicativo disporá de diversos recursos que facilitem o controle de agendamento e cancelamento por parte do Paciente e do controle por parte do Especialista.

1.3 PROBLEMAS

Observa-se a possibilidade de haver um descontrole pois, temos a figura do Paciente, do Especialista e de uma possível Secretária, caso não haja um controle centralizado

1 Lançamento do iPhone, em junho de 2007.

e com algumas funções automatizadas podem causar uma ineficiência. Como este Aplicativo, contribui para evitar retrabalho, reclamações por parte dos Pacientes, melhora assim o atendimento como um todo. A prestação destes atendimentos poderá até ser avaliadas de forma cômoda, através deste Aplicativo pelos Pacientes atendidos. Fica facultativo a solicitação de uso deste Aplicativo por parte do Especialista, dependendo do tipo de atendimento e/ou do tempo de tratamento.

1.4 OBJETIVO

Elaborar um Software que organize e facilite o agendamento das consultas, e permita o acompanhamento por parte do paciente e do especialista.

1.4.1 Objetivo geral

O paciente terá a chance de visualizar e interagir de forma ativa na Agenda, solicitar e cancelar horários disponíveis dentro de regras preestabelecidas pelos especialistas.

Terá com o Aplicativo, a comodidade de agendar uma consulta de qualquer lugar, sem a necessidade de ir até o consultório conveniado.

1.4.2 Objetivos específicos

- Fazer o levantamento dos Requisitos Normais e Alternativos;
- Desenvolver estudo dos Casos de Uso;
- Desenvolver os Fluxogramas de Atividades;
- Criar o Mapa de Navegação;
- Criar o Protótipo para teste de usabilidade;
- Criar o diagrama de classes e seus relacionamentos.
- Desenvolver o Servidor *REST* e sua base de dados.
- Desenvolver o *App mobile*.
- Realizar testes do consumo do serviço *REST* pelo *App*;
- Resultado das atividades.

1.5 ESTADO DA ARTE

Neste capítulo são expostos Aplicativos *Mobile* de Agendamento Médico de mercado e suas características principais, na sequência feito comparação de recursos entre eles.

1.5.1 Aplicativos de mercado e suas características

Na sequência foram pesquisados quatro Aplicativos de Mercado mostrando suas principais características quando a instalação, armazenamento e recursos principais. Os Aplicativos são:

1. Consultório Me;
2. Doutore;
3. Consultório Live;
4. Estela;

1.5.1.1 Consultório me

Consultorio.me (Administração no Site: “https://consultorio.me/’. Aplicativo disponível na Play Story: “https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR”. Acesso em: 02/março/2020).

Na tabela 1, mostra as características do Consultório Me:

Tabela 01 – Características do Consultório Me

Item	*Característica*	*Descrição*
1	Instalação.	Download pela Play Store.
2	Período de teste.	Tempo de teste ilimitado.
3	Armazenamento de dados.	Na nuvem.
4	Administração.	No Site.
5	Permissões de acesso (papéis).	Sim.
6	Agenda de Pacientes/Clientes.	Sim. Com vários tipos de consulta na versão premium.

7	Envio de mensagem.	De forma manual.
8	Informações de Prontuário.	Sim.
9	Controle Contábil.	Sim. Controle de despesas.
10	Forma de pagamento.	Mensal, de acordo com o número de profissionais (PagSeguro / PayPal).
11	Pacientes agendam consulta.	Não.

Fonte: Autores

Na sequência são mostradas as telas capturadas do Consultório Me:

Ilustração 01 – App 1 Tela Inicial

Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

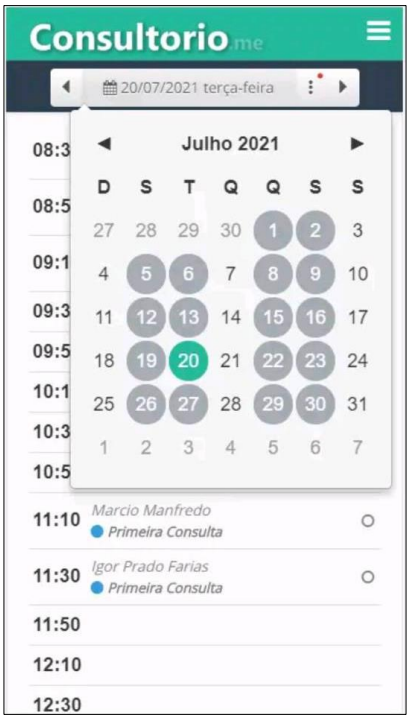
Ilustração 02 – App 1 Tela Cadastro

Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

Iustração 03 – App 1 Tela Agenda

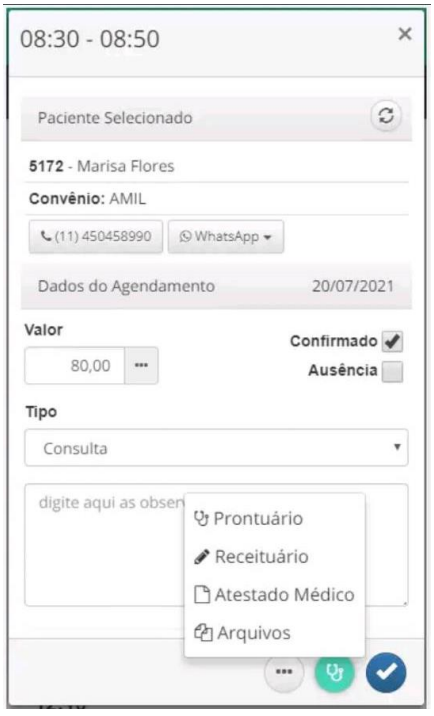
Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 04 – App 1 Tela Agendar



Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 05 – App 1 Tela Agendamento



Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 06 – App1 Tela Dados Paciente

Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 07 – App1 Tela Lista Pacientes

Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 08 – App Tela Prontuário

Fonte:
https://play.google.com/store/apps/details?id=me.consultorio&hl=pt_BR.
Acesso em: 02/março/2020.

1.5.1.2 Doutore

Doutore (Administração no Site: https://doutore.com /. Aplicativo disponível na Play Story: https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl=pt_BR. Acesso em: 02/março/2020).

Na tabela 2, mostra as características do Doutore:

Tabela 02 – Características do Doutore

Item	*Característica*	*Descrição*
1	Instalação.	Download pela Play Store.
2	Período de teste.	15 dias.
3	Armazenamento de dados.	Na nuvem.
4	Administração.	No Site.

5	Permissões de acesso (papéis).	Sim.
6	Agenda de Pacientes/Clientes.	Sim.
7	Envio de mensagem.	Automática por SMS.
8	Informações de Prontuário.	Sim.
9	Controle Contábil.	Sim, controle de faturamento.
10	Forma de pagamento.	Mensal, com três níveis de recursos.
11	Pacientes agendam consulta.	Não.

Fonte: Autores

Na sequência são mostradas as telas capturadas do Doutore:

Ilustração 09 – App 2 Tela Inicial

Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

Ilustração 10 – App 2 Tela Cadastro

Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

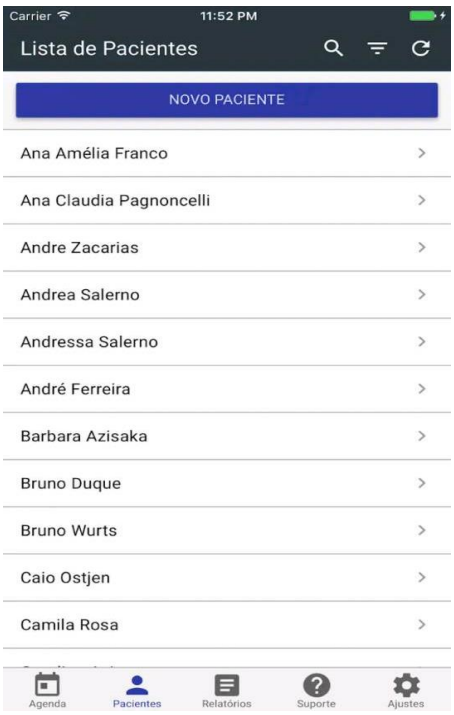
Ilustração 11 – App 2 Tela Login

Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

Ilustração 12 – App 2 Tela Agenda

Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

Ilustração 13 – App 2 Tela Pacientes



Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

Ilustração 14 – App 2 Tela Cadastro Paciente

Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

Ilustração 15– App 2 Tela Pagamentos

Fonte:
https://play.google.com/store/apps/details?id=io.gonative.ios.dyeqa&hl =pt_BR.
Acesso em: 02/março/2020.

Ilustração 16 – App 2 Tela Total Pagamentos

Fonte:
ttps://play.google.com/store/appsdetailsId=io.gonative.ios.dyeqa&hl = pt_BR
Acesso em: 02/março/2020.

1.5.1.3 Consultório Live

Consultório Live (Administração no Site: “https://www.vbbsoftware.com/consultorio-live/”. Aplicativo disponível na Play Story: “https://play.google.com/store/apps/details?id=com.vbbsoftware.consultoriolive&hl=pt_BR”. Acesso em: 02/março/2020).

Na tabela 3, mostra as características do Consultório Live:

Tabela 03 – Características do Consultório Live

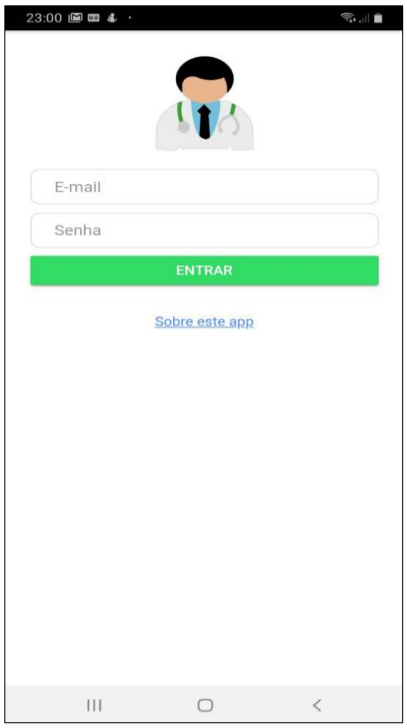
Item	*Característica*	*Descrição*
1	Instalação.	Download pela Play Store.
2	Período de teste.	Pagamento (licença) por 1 ano/por pessoa.
3	Armazenamento de dados.	Na nuvem.
4	Administração.	No Site.

5	Permissões de acesso (papéis).	Somente o médico tem acesso ao prontuário.
6	Agenda de Pacientes/Clientes.	Sim.
7	Envio de mensagem.	Automática por SMS ou e-mail.
8	Informações de Prontuário.	Sim.
9	Controle Contábil.	Fluxo de caixa e contas a pagar e a receber, pagamentos de clientes, controle de estoque.
10	Forma de pagamento.	PagSeguro - cartões, boleto. Baseado no número de usuários (1, 2, 4, 8). O pagamento por 1 ano.
11	Pacientes agendam consulta.	Sim.

Fonte: Autores

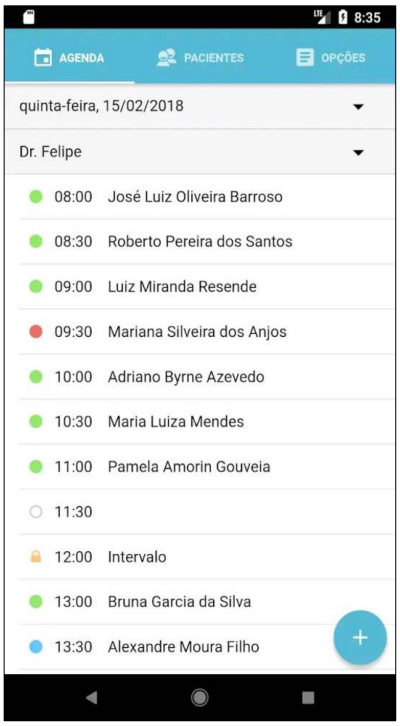
Na sequência são mostradas as telas capturadas do Consultório Live.

Ilustração 17 – App 3 Tela Login



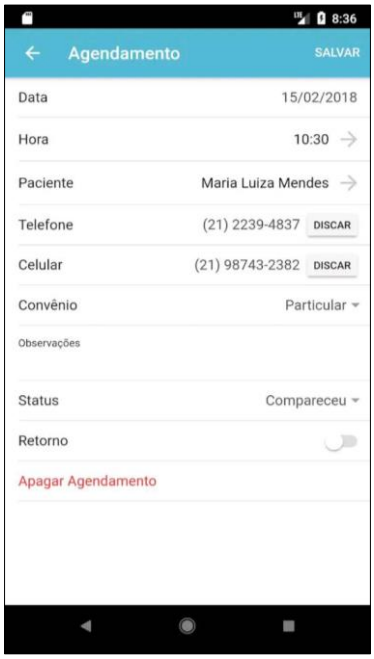
Fonte:
https://play.google.com/store/apps/detailsid=com.vbbsoftware.consultoriolive&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 18 – App 3 Tela Agenda



Fonte:
https://play.google.com/store/apps/details?id=com.vbbsoftware.consultoriolive&hl=pt_BR.
Acesso em: 02/março/2020

Ilustração 19 – App 3 Tela Agendar

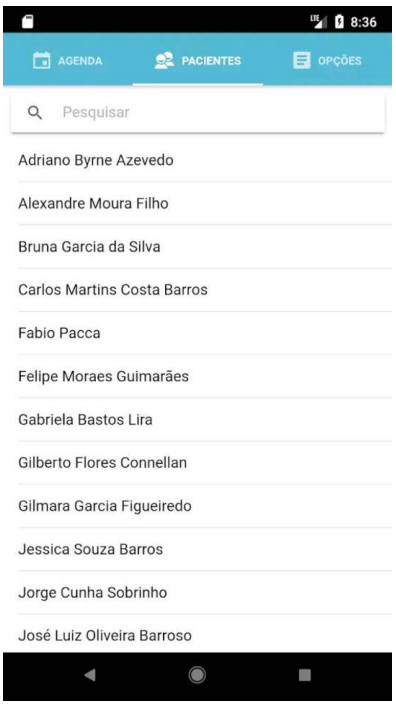


Fonte:
https://play.google.com/store/apps/details?id=com.vbbsoftware.consultoriolive&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 20 – App 3 Tela Cadastro Paciente

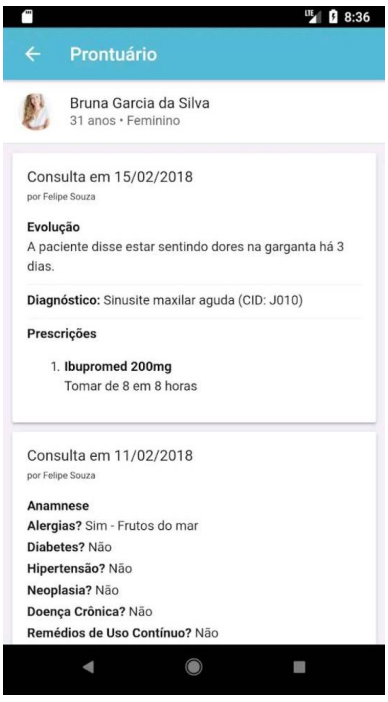
Fonte:
https://play.google.com/store/apps/details?id=com.vbbsoftware.consultoriolive&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 21 – App 3 Tela Pesquisa Paciente



Fonte:
https://play.google.com/store/apps/details?id=com.vbbsoftware.consultoriolive&hl=pt_BR.
Acesso em: 02/março/2020.

Ilustração 22 – App 3 Tela Prontuário



Fonte:
https://play.google.com/store/apps/details?id=com.vbbsoftware.consultoriolive&hl=pt_BR.
Acesso em: 02/março/2020.

1.5.1.4 Estela

Estela (Administração no Site: “https://www.estela.io/” Aplicativo disponível na Play Story: “https://play.google.com/store/apps/details?id=com.estelapro”. Acesso em: 02/março/2020).

Na tabela 4, mostra as características do Estela:

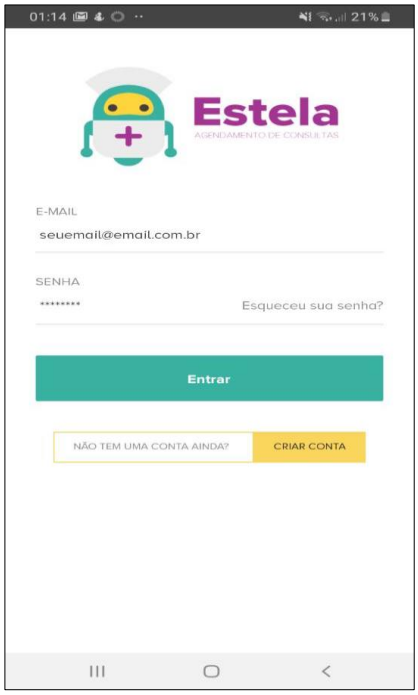
Tabela 04 – Características do Estela

Item	*Característica*	*Descrição*
1	Instalação.	Download pela Play Store.
2	Período de teste.	Na versão Free é ilimitado.
3	Armazenamento de dados.	Na nuvem.
4	Administração.	No Site.
5	Permissões de acesso (papéis).	Todos têm o mesmo tipo de acesso com o mesmo e-mail e senha.
6	Agenda de Pacientes/Clientes.	Sim.
7	Envio de mensagem.	Automática por SMS.
8	Informações de Prontuário.	Sim.
9	Controle Contábil.	Não.
10	Forma de pagamento.	A versão Premium está em desenvolvimento e será de pagamento mensal.
11	Pacientes agendam consulta.	Não.

Fonte: Autores

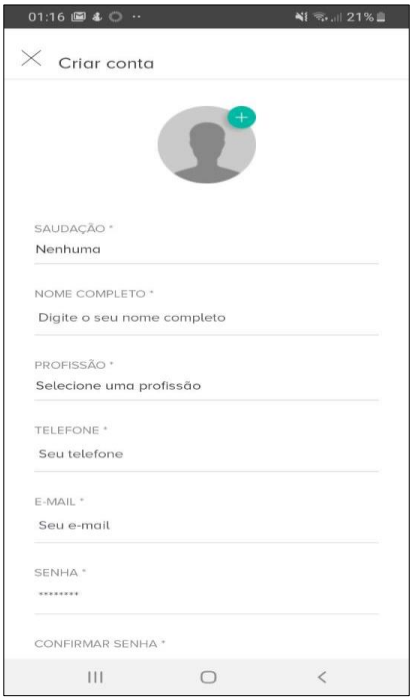
Na sequência são mostradas as telas capturadas do Aplicativo Estela. Na sequência é feita uma comparação entre os aplicativos de mercado, destacando os principais itens.

Ilustração 23 – App 4 Tela Inicial



Fonte:
https://play.google.com/store/apps/details?id=com.estelapro
Acesso em: 02/março/2020

Ilustração 24 – App 4 Tela Cadastro

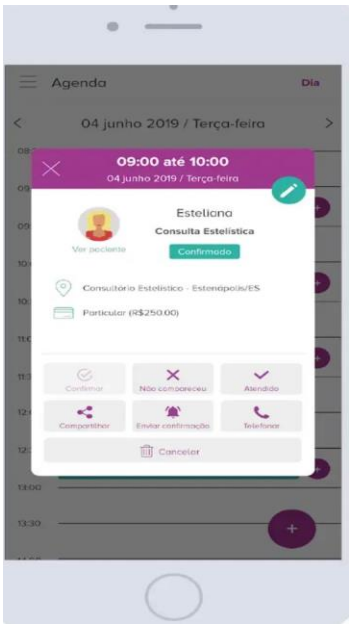


Fonte:
https://play.google.com/store/apps/details?id=com.estelapro
Acesso em: 02/março/2020

Ilustração 25 – App 4 Tela Agenda

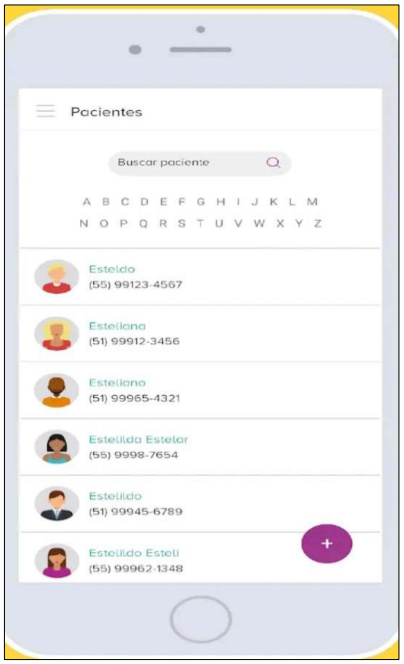
Fonte:
https://play.google.com/store/apps/details?id=com.estelapro
Acesso em: 02/março/2020

Ilustração 26 – App 4 Tela Agendar



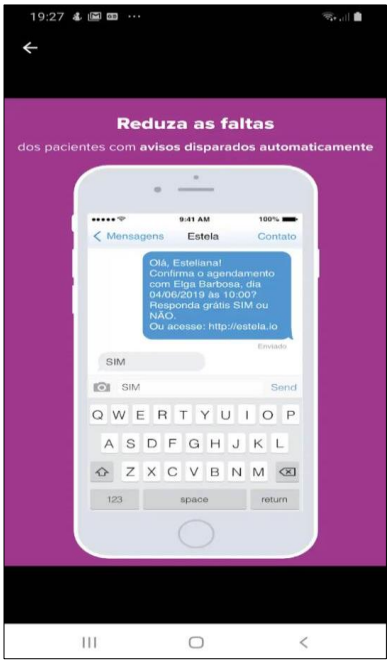
Fonte:
https://play.google.com/store/apps/details?id=com.estelapro
Acesso em: 02/março/2020.

Ilustração 27 – App 4 Tela Pacientes



Fonte: https://play.google.com/store/apps/details?id=com.estelapro
Acesso em: 02/março/2020.

Ilustração 28 – App 4 Tela Aviso



Fonte:
https://play.google.com/store/apps/details?id=com.estelapro
Acesso em: 02/março/2020.

1.6 COMPARAÇÃO ENTRE APLICATIVOS DE MERCADO

Com base nas tabelas 1, 2, 3 e 4 de cada Aplicativo podemos destacar os seguintes itens:

a) Todos têm a mesma forma de instalação fazendo do download pela *Play Store*.
b) Quanto a experimentação dos *Softwares*, existem diferenças quanto a liberação dos recursos, a diferença vai de usar a totalidade dos recursos por um período de dias, sendo obrigatoriamente pagar para continuar usando; no caso de não pagamento, corta completamente o funcionamento e até com a perda total dos dados inseridos no período de teste, em outros casos se libera uma parte pequena de recursos essenciais gratuitamente por tempo indeterminado, com a reserva de uma versão "*premium*" através de pagamentos mensais sem o prejuízo da perda das informações inseridas no Aplicativo durante o período de teste. Existe uma modalidade de pagamento em que o Paciente faz o pagamento para o Médico e este repassa uma comissão para os desenvolvedores do Aplicativo.

c) Quanto ao armazenamento, todos os Aplicativos pesquisados têm como local de armazenamento a nuvem, com isso as informações são vistas praticamente em todos os locais.
d) Quanto a entrada no Aplicativo, todos fazem um *Login* onde é criado o acesso por e-mail e uma senha, em alguns casos é dado uma chave pelo fabricante do software para se ter acesso ao Aplicativo no primeiro acesso.
e) Quanto à administração, os aplicativos pesquisados têm diferenças que vão da necessidade de um site para o cadastramento dos Médicos, preferências etc., e casos em que a administração é feita no próprio Aplicativo.
f) Aplicativos com relação a administração e aplicação no site, as informações são criptografadas e com papéis de permissão de leitura/escrita na questão do prontuário dos pacientes.
g) Na administração de papéis de acesso, cadastro de médicos/terapeutas, secretárias e/ou recepcionistas, vai depender da forma e quantidade de recursos que o aplicativo possui e geralmente estão em *websites* de administração do aplicativo. No caso de ter a administração no *website*, o aplicativo do médico faz esta função.
h) Todos os Aplicativos têm como padrão a Agenda de Pacientes/Clientes.
i) Como recurso adicional existe a possibilidade de um gerenciamento contábil das despesas dos profissionais médicos/terapeutas, mostrando de forma gráfica, estatísticas de rendimento dos atendimentos com gráficos mensais/semanais, fluxo de caixa e registro do histórico dos pacientes (Prontuário).

1.7 O QUE SE PRETENDE

Desenvolver um Aplicativo no ambiente Windows, sendo previsto basicamente duas partes principais, a primeira parte, para que possa funcionar trocando informações será construído um servidor *REST*, sendo disponibilizado um serviço com as informações de um Banco de Dados. A segunda é o Aplicativo *Mobile*, que será desenvolvido com a *Integrated Development Environment* (*IDE*) *Android Studio* (*Flutter*).

Com relação aos recursos principais, o servidor *REST* funcionará com *Login* e Senha, cadastro de pacientes com histórico de consultas, cadastro de médicos/terapeutas e a agenda de consultas.

2 DESENVOLVIMENTO

Neste capítulo demonstra as fases implementação para chegar ao resultado do aplicativo *mobile* de agendamento de especialistas *online*.

O que se pretende é que um lado, o médico e/ou secretária consiga visualizar os cadastros de pacientes feito pelos mesmos, disponibilizar os horários de consulta e que não gere conflito com os atendimentos de retorno. Disponibilizar visões que possibilitem o gerenciamento dos atendimentos.

2.1 ANÁLISE DO SISTEMA

Para análise e desenvolvimento deste sistema, optou-se pela *UML* (Unified Modeling Language) por facilitar, de forma visual, seu entendimento. Eduardo (2015, p. 14) a *UML* define uma notação padrão que pode ser utilizada por desenvolvedores de *software* orientado a objetos. Sem dúvida o domínio dessa notação é importante para qualquer desenvolvedor que queira aproveitar todas as capacidades que a *UML* fornece. A modelagem de sistemas de *software* consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se várias perspectivas diferentes e complementares.

> Finalmente, a *UML* foi adotada, em 1997, pela *OMG* (Object Management Group), como uma linguagem-padrão de modelagem. A versão 2.0 da linguagem foi oficialmente lançada em julho de 2005. Atualmente a encontra-se na versão 2.5. Essa linguagem foi lançada com o objetivo de simplificar a estrutura da linguagem. A documentação oficial da *UML* pode ser encontrada no site da *OMG* em www.omg.org ou mais exatamente em www.uml.org (Gilleanes, 2018, p. 21).

2.1.1 Análise de Requisitos

A atividade de levantamento de requisitos (também conhecida como elicitação de requisitos) corresponde à etapa de compreensão do problema aplicada ao desenvolvimento de *software*. O principal objetivo do levantamento de requisitos é que usuários e desenvolvedores tenham a mesma visão do problema a ser resolvido Eduardo (2015, p. 31).

Como ponto de partida, foram realizados os Requisitos Normais (Funcionais) e Requisitos Alternativos (Não funcionais). Para Gilieanes (2018), os requisitos funcionais correspondem ao que o cliente quer que o sistema realize, ou seja, as

funcionalidades do *software*. Já os requisitos não funcionais correspondem a restrições, condições, consistências e validações que devem ser levadas a efeito sobre os requisitos funcionais.

Nesta etapa foram levantadas as necessidades dos Clientes em narrativa informal e descreve as tarefas principais do aplicativo.

RF01*:* **Cadastrar Usuário**.
Fluxo Normal:
1 Preencher o Formulário com dados pessoais;
2 Enviar cadastro;
3 Receber a confirmação de Sucesso.
Fluxo Alternativo:
2.1 Preenchimento indevido do(s) campo(s);
2.1.1 Retornar no Item 1 (Fluxo Normal);
3.1 Não recebeu confirmação de Sucesso;
3.1.1 Retornar no Passo 1 (Fluxo Normal).

RF02: **Fazer Login**.
Fluxo Normal:
1 Inserir Usuário;
2 Inserir Senha;
3 Confirmar.
Fluxo Alternativo:
3.1 Erro de Usuário;
3.1.1 Solicitar Cadastro;
3.1.2 Retornar no Passo 1 (Fluxo Normal);
3.2 Erro de Senha;
3.2.1 Inserir Senha novamente;
3.2.2 Retornar Passo 2 (Fluxo Normal);
Não conseguindo,
solicitar nova Senha.
Verificar Senha no e-mail.
Retornar Passo 1.

RF3: **Agendar Consulta.**

Fluxo Normal:

1 Selecionar a Especialidade;

2 Selecionar o Médico;

3 Selecionar data e hora;

4 Agendar.

Fluxo Alternativo:

1.1 Não ter a Especialidade;

1.1.1 Fechar Aplicativo;

2.1 Não ter o Especialista;

2.1.1 Fechar Aplicativo

3.1 Não ter disponibilidade de data e/ou horário;

3.1.1 Retornar passo 3.

RF4: **Visualizar Consulta.**

Fluxo Normal:

1 Mostrar a Agenda.

Fluxo Alternativo:

Não há.

RF5: **Cancelar Consulta.**

Fluxo Normal:

1 Cancelar na Agenda.

Fluxo Alternativo:

Não há.

RF6: **Controlar Agenda.**

Fluxo Normal:

1 Disponibilizar dias e horários;

2 Visualizar Agendamentos efetuados;

3 Cancelar Agendamento (pelo Paciente).

Fluxo Alternativo:

3.1 Mensagem de Cancelamento;

3.2 Fazer novo Agendamento.

RF7*:* **Confirmar Agendamento.**

Fluxo Normal:

1 Automático – Aplicativo confirma Agendamento;

Fluxo Alternativo:

2.1 Não ser possível a Confirmação para o Paciente;

RF8: **Visualizar Agenda.**

Fluxo Normal:

1 Visualizar todos os Agendamentos (Médicos).

Fluxo Alternativo:

Não há.

Requisito 9: **Manter Agenda.**

Fluxo Normal:

1 Cadastrar;

2 Alterar;

3 Consultar;

4 Excluir.

Fluxo Alternativo:

1.1 Erro ao Cadastrar;

1.1.1 Ver mensagem de erro;

2.1 Erro ao Alterar;

2.1.1 Ver mensagem de erro;

3.1 Erro ao Consultar;

3.1.1 Ver mensagem de erro;

4.1 Erro ao Excluir;

4.1.1 Ver mensagem de erro.

Requisito 10: **Manter Cadastro Usuário.**

Fluxo Normal:

1 Cadastrar;

2 Alterar;

3 Consultar;

4 Excluir.

Fluxo Alternativo:

1.1 Erro ao Cadastrar;

1.1.1 Ver mensagem de erro;

2.1 Erro ao Alterar;

2.1.1 Ver mensagem de erro;

3.1 Erro ao Consultar;

3.1.1 Ver mensagem de erro;

4.1 Erro ao Excluir;

4.1.1 Ver mensagem de erro.

2.1.2 Diagrama de Caso de Usos

Conforme Eduardo (2015, p. 14) a visão de Caso de Uso descreve o sistema de um ponto de vista externo como um conjunto de interações entre o sistema e os agentes externos ao sistema. Esta visão é criada em um estágio inicial e direciona o desenvolvimento das outras visões do sistema.

Os Diagramas de Caso de Uso *UC01 UC02* são apresentados no Apêndice A, mostram as principais ações que o sistema oferece, sendo representadas pelos atores usuário, operador e especialista com o aplicativo. Cada Diagrama de Caso de Usos tem como pré-condição o *Login* do Usuário, o Usuário poderá ser o paciente, o operador ou o especialista (administrador). No caso destes atores devem ter seus cadastros previamente feitos para utilização do sistema. Uma vez logados, cada ator terá seus papeis (suas permissões) de usuário para ver relatórios, inserir, alterar, excluir os dados dos cadastros de operadores, usuários, históricos etc.

2.1.3 Diagrama de Atividades

No Apêndice B estão representados os diagramas de atividades (processos) específicas relacionadas aos atores Usuário e Especialista, relacionado ao retorno do aplicativo.

2.1.4 Mapa de Navegação

O mapa de navegação representa os *links* (menu) de acesso aos cadastros, formulários e relatórios dos usuários. Os principais *links* são mostrados na ilustração dos Apêndice C.

2.2 TECNOLOGIA E FERRAMENTAS

Nesta parte do capítulo, diante do objetivo esperado que é o desenvolver de um webservice com banco de dados e através de seus *endpoints* se consiga consumir/manipular as informações por meio de uma aplicação *mobile* de agendamento de especialistas *online*, serão abordados quais linguagens de programação e as ferramentas utilizadas.

A relação das ferramentas utilizadas , encontra-se no Apêndice E.

2.2.1 Começando pelo back-end

Dentro deste contexto, o *back-end* será desenvolvido com a construção de um servidor *REST*, utilizando a linguagem *Java* e com o *Framework Spring Boot*.

2.2.1.1 Linguagem de programação Java

"*Java* é uma linguagem de programação orientada a objetos, uma plataforma computacional lançada pela primeira vez pela *Sun Microsystems* em 1995. Em 2008 o Java foi adquirido pela empresa Oracle Corporation.

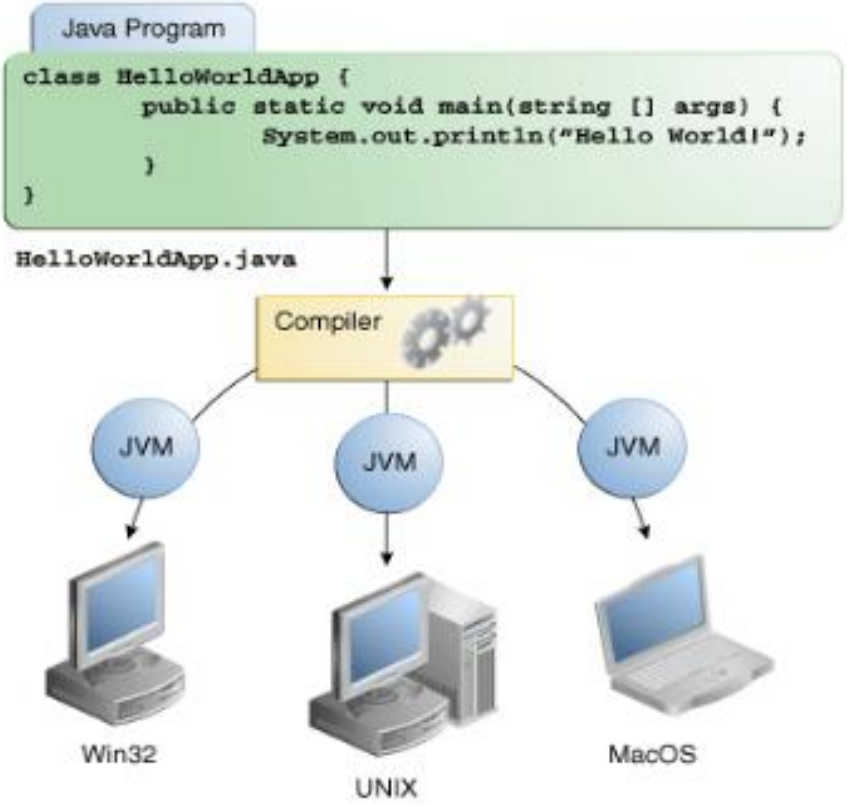
Existem muitas aplicações e sites que não funcionarão, a menos que você tenha o *Java* instalado, e mais desses são criados todos os dias. O *Java* é rápido, seguro e confiável. De *laptops* a *datacenters*, console de *games* a supercomputadores científicos, telefones celulares à *internet*, o *Java* está em todos os lugares! *Java* é o termo geral usado para denotar o *software* e seus componentes, que incluem *JRE* (*Java Runtime Environment*), *JVM* (*Java Virtual Machine*) especificamente.", conforme o site *Java*. (https://www.java.com/pt-BR/download/help/index.html).

"Usada por mais de 10 milhões de desenvolvedores e rodando em 56 bilhões de dispositivos globalmente, a plataforma Java realmente move o mundo para a frente".

Uma relação de tutoriais de Java pode ser encontrada o *link*: https://dev.java/learn/

Na ilustração 31, exemplo de programa em *Java*.

Ilustração 31 – Exemplo de programa Java



Fonte: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

2.2.1.2 Servidor REST - Teoria

Ao servidor *REST* (*Represetational State Transfer*) é uma técnica de desenvolvimento de *webservice* fortemente baseada nos métodos do protocolo *HTTP* (*GET, POST, PUT, DELETE*), cabe ao servidor *REST* a responsabilidade da integração e comunicação e garantir a integridade das informações para os aplicativos *mobiles* dos clientes (médicos e pacientes).

Conforme (Ricardo, 2015), “uma das grandes vantagens na construção de *webservices* é que eles permitem acessar os serviços de uma forma padronizada e independente de linguagem de programação. Por exemplo, é possível escrever um webservice com a linguagem Java, mas consumi-lo em qualquer outra linguagem, como *C#, PHP, Python* etc.”.

A comunicação dos *webservices* está fortemente acoplada aos conceitos do protocolo *HTTP*, em termos gerais, existem quatro métodos nos quais o servidor opera conforme a tabela 06, abaixo:

Tabela 06 – Métodos do protocolo *HTTP (GET, POST, PUT, DELETE)*

Método	*Descrição*
GET	Método mais comum de requisição. Deve ser utilizado para páginas que retornam resultados e consultas. Preferencialmente, devem ser passados poucos parâmetros na *URL*. Sempre que uma endereço de uma página é digitado no browser uma requisição o tipo *GET* é feita.
POST	O *POST* deve ser utilizado para enviar dados para o servidor, como, por exemplo, um formulário de login ou um cadastro de usuários. No REST, o *POST* é utilizado para enviar dados ao webservice com objetivo de inserir informações no banco de dados.
PUT	No *REST*, o método *PUT* deve ser utilizado para atualizar registros.
DELETE	No *REST*, o método *DELETE* deve ser utilizado para excluí um registro.

Fonte: Ricardo R. Lecheta, 2015 – Livro "*Web Services RESTfull*"

2.2.1.3 Visão panorâmica do webservice

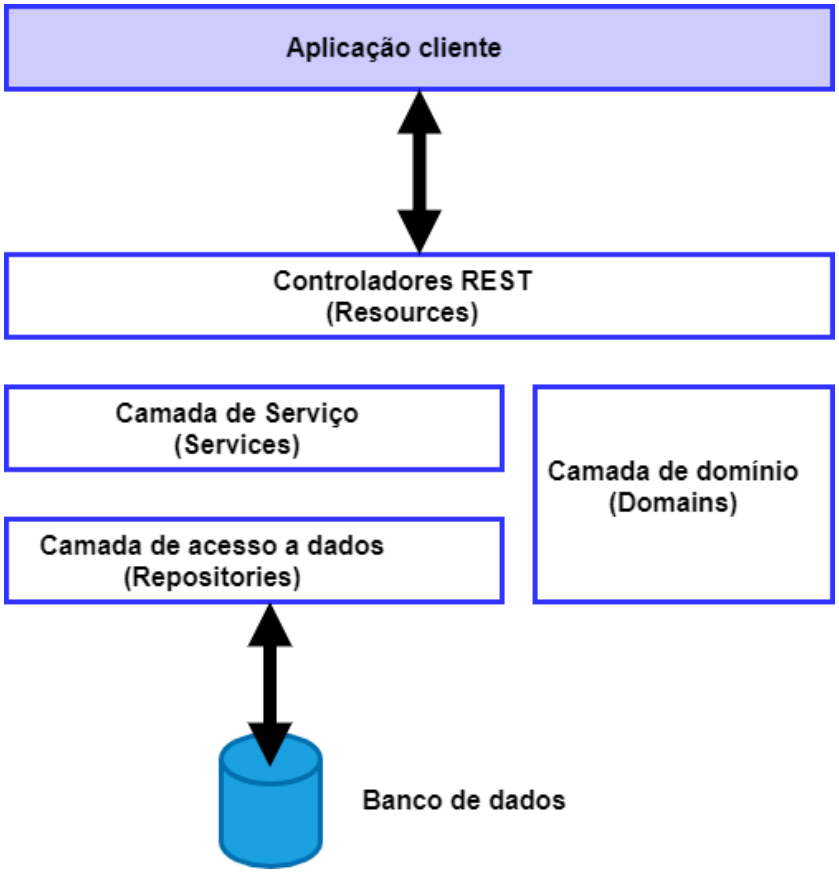
Na ilustração 32, mostra uma visão panorâmica da estrutura do *webservice* em camadas e suas atribuições. Para esta missão foi escolhido o *framework Spring Boot*, este tem a vantagem de que grande parte da programação fica automatizada.

Conforme Ricardo (2015), O *Spring* é um conjunto de bibliotecas que auxilia na tarefa de injeção de dependências *DI* (*Dependency Injection*), no controle de transações de banco de dados, de segurança etc.

Antes de iniciar o projeto, tem uma ferramenta auxiliar chamada *Maven* que facilita no processo de compilação e gerenciamento de dependência do projeto. "Quanto maior o projeto, mais dependências. Isto é uma relação quase certa.

Neste contexto, temos a necessidade de organizar os artefatos do projeto de forma simples e eficiente, e é exatamente aqui que o *Maven* pode ajudar" (Ricardo, 2015).

Ilustração 32 - Estrutura das camadas do webservice



Fonte: Autores

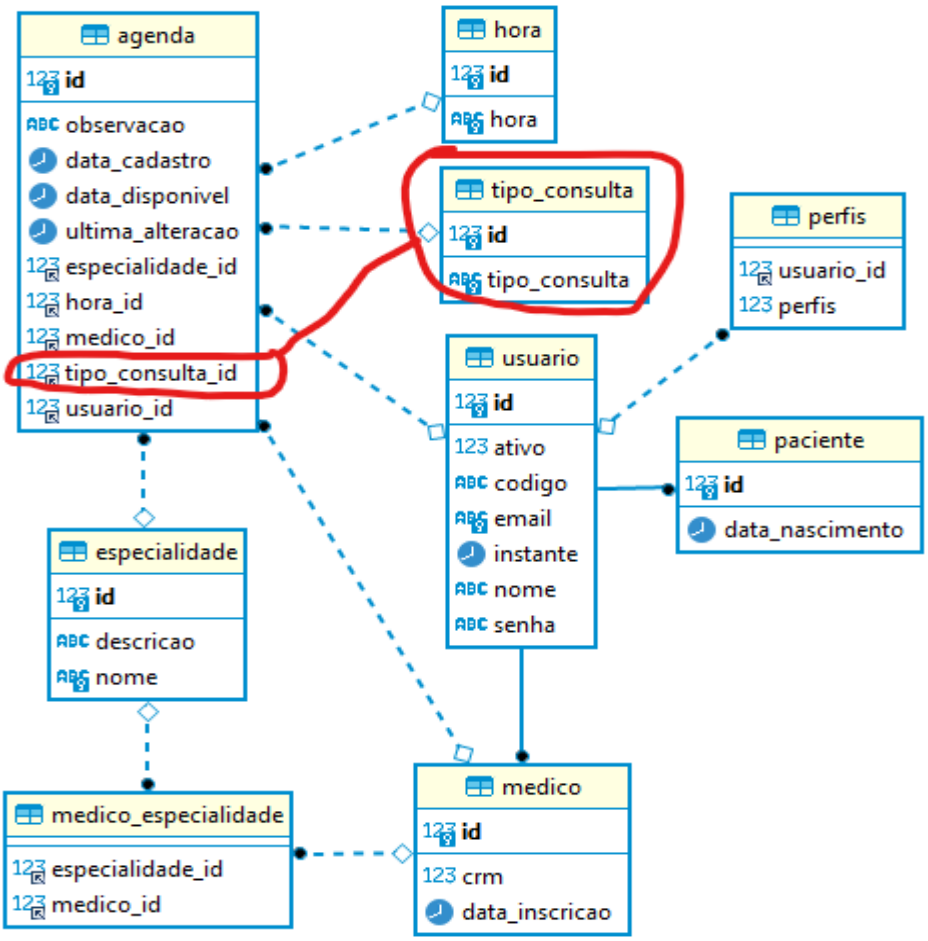
2.2.2 Banco de dados

É importante dizer que o *Java* tem especificações e implementações. Conforme o site *stackoverflow* (“https://pt.stackoverflow.com/questions/111284/especifica%C3%A7%C3%A3o-e-implementa%C3%A7%C3%A3o”, 2021) especificação é algo abstrato. É algum “documento” que estabelece regras a serem seguidas, o que se espera das possíveis implementações. Essas regras podem exigir algumas coisas e proibir outras. É uma formalização de como algo deve proceder. É a lei.

A implementação é algo concreto. É como é feito de verdade. É o que existe para ser usado. É a aplicação da lei. Uma especificação amplamente aplicada é a *JPA* (*Java Persistence API*) cuja implementação é através do *Hibernate*, esta implementação é a responsável pela persistência dos dados e construção das tabelas no banco de dados. O *JPA* é colocado como uma dependência no *POM.xml* (*Project Object Model*) para fazer parte do projeto.

Dito isto, o ponto forte relacionado com as tabelas do banco de dados é fazer um modelo conceitual de classes que represente o banco de dados, pois, a implementação *JPA* citada acima cuidará de transformar o modelo de classes em tabelas no banco de dados relacional. Na próxima ilustração 33, mostra o Diagrama de Entidade Relacional como resultado da associação de classes do Framework Spring Boot, neste modelo gerado temos os relacionamentos entre tabela.

Ilustração 33 – DER (Diagrama Entidade Relacional)



Fonte: Autores

O Banco de Dados na visão do (HEUSER, 2009) é “A solução para evitar a redundância não controlada de informações, é o compartilhamento de dados. Nesta forma de processamento, cada informação é armazenada uma única vez, sendo acessada por vários sistemas que dela necessitam”. Antes da construção do banco de dados, obtém-se a identificação das entidades, seus atributos e seus relacionamentos.

2.2.3 Camada de negócio

A camada de Negócio é manipulada pelo *framework* sendo equivalente a camada de serviço, mostrada na penúltima ilustração 32.

Conforme Evandro (2012), as regras de negócio são regras que ditam o comportamento, as restrições e as validações existentes num sistema de informação.

Estas regras são programadas na camada *services* (serviço), esta camada recebe os pedidos da camada *resources* (recurso), faz as devidas restrições e validações e retransmite para a camada *repositories* (repositório) que atua diretamente no banco de dados fazendo sua persistência), executando as ordens de inserção, alteração, exclusão e visualização dos dados.

No servidor em questão a camada de serviço usa a *annotation @Service* na sua declaração e é convencionado a representação do nome de domínio / entity que representa, na ilustração 35 está a classe *EspecialidadeService* que o serviço manipula, ela recebe as ordens da camada *resources* (recursos) onde ficam os controladores *REST - @RestController*.

Nas próximas ilustrações 34 mostram o trajeto percorrido desde a requisição até a persistência no banco de dados fazendo uma busca de especialidade pelo seu Id.

A camada dos Recursos, mostrada como exemplo na figura 34 mostra onde fica configurado o caminho da *URL* que faz parte das requisições externas ao servidor *webservice*, a forma de acessar este recurso é por uma *URL*, poderia ser assim: http://localhost:8080/especialidades/2. No exemplo, a especialidade de Id = 2 seria trazida do banco de dados.

O detalhe das anotações *@RestController* e o mapeamento *@RequestMapping*(value = “/especialidades”).

Esta solicitação vai para a camada de serviço – ilustração 35, no código em questão está a injeção de dependência na classe *EspecialidadeService* como *@Service*, é para avisar o framework *Spring Bo*ot para processar como serviço automaticamente, coloca-se a anotação *@Autowired.*

O estudo das anotações torna-se relevante pois, o Spring Boot automatiza diversas etapas, minimizando o tempo de programação e contribuindo ainda mais na reutilização de códigos.

Ilustração 34 – Controlador REST da classe Especialidade

```
import java.net.URI;

@RestController
@RequestMapping(value = "/especialidades")
public class EspecialidadeResource {

    @Autowired
    private EspecialidadeService service;

    @ApiOperation(value="Busca por id")
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public ResponseEntity<Especialidade> find(@PathVariable Integer id) {
        Especialidade obj = service.find(id);
        return ResponseEntity.ok().body(obj);
    }
```

Fonte: Autores

Chegando na camada de serviço figura 35, temos a classe marcada com a anotação *@Service*, esta classe depende e precisa da camada *Repositories* sendo injetada como *EspecialidadeRepository* atribuindo no nome de repo, a anotação é *@Autowired*. A consulta ao *repository* é executada pela expressão: `Optional<Especialidade> obj = repo.findById(id);.`

Ilustração 35 – Camada Serviço da classe Especialidade

```
import java.util.List;

@Service
public class EspecialidadeService {

    @Autowired
    private EspecialidadeRepository repo;

    public Especialidade find(Integer id) {
        Optional<Especialidade> obj = repo.findById(id);
        return obj.orElseThrow(() -> new ObjectNotFoundException(
                "Objeto não encontrado! Id: " + ", Tipo: " + Especialidade.class.getName()));
    }
```

Fonte: Autores

Observar que a classe no *Repository* da ilustração 36, que não tem a função *findById()*, isto acontece devido que esta classe estende da classe

jpaRepository<Especialidade, Integer> do *framework Spring Boot*, nesta classe existe várias funções disponíveis para usar. Ver que tem nela a anotação *@Repository* que faz com que o framework reconheça de forma automática e execute as ordens no banco de dados.

Ilustração 36 – Camada Repository da classe Especialidade

```
2
3  import org.springframework.data.jpa.repository.JpaRepository;
7
8  @Repository
9  public interface EspecialidadeRepository extends JpaRepository<Especialidade, Integer> {
10
11 }
12
```

Fonte: Autores

2.2.4 Camada de apresentação

Um aplicativo *mobile* para o consumo do *webservice*. A ideia é utilizar o *framework Flutter* produzido pela *Google* através da *Ide Android Studio*.

Conforme Frank (2020), Graças aos talentosos engenheiros do *Google*, o *Flutter* é uma plataforma que fornece um meio de escrevermos uma única (mais ou menos) base de código que funcione igualmente bem no *Android* e no *iOS* e distribua ao mesmo tempo desempenho e recursos nativos.

A linguagem de programação que a *Google* adotou para o *framework Flutter* é a *Dart* no ano 2011.

> “Inicialmente ele foi revelado na conferência GOTO em Aarhus, Dinamarca. A versão inicial, 1.0, saiu em novembro de 2013, aproximadamente dois anos antes de o Flutter ser lançado. A propósito, temos de agradecer a Lars Bak (também desenvolvedor do engine JavaScript V8 que suporta o Chrome e o Node.js) e a Kasper Lund pela existência do Dart!”
> Zammetti, Frank. Flutter na prática (p. 21). Novatec Editora. Edição do Kindle.

No *Flutter*, tudo é um *widget*. *Widgets* são blocos de códigos da *UI* (*User Interface*), na ilustração 37 representa *widgets*. Todo aplicativo desenvolvido em *Flutter* é construído por associações de *widgets* simples.

Ilustração 37 – Exemplo de widgets

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Sample App"),
    ),
    body: Center(
      child: ElevatedButton(
        style: ElevatedButton.styleFrom(
          padding: EdgeInsets.only(left: 20.0, right: 30.0),
        ),
        onPressed: () {},
        child: Text('Hello'),
      ),
    ),
  );
}
```

Fonte: https://flutter.dev/docs/get-started/flutter-for/android-devs.

Na ilustração 37 acima, observa-se uma árvore de widgets.

1. Começamos com a construção de um layout (*Scaffold*);
2. Aplicamos uma Barra com o texto: Sample App (*Appbar*);
3. No corpo do layout tem um widget de centralização (*Center*);
4. Dentro do layout centralizado um Botão, com o estilo espaçado nas laterais, na esquerda de 20.0 e direita 30.0 (*padding*);
5. Tem a disposição uma função vazia que pode executar uma classe ou método;
6. Neste Botão tem um filho (*child*) *Text* que está escrito: '*Hello*';

Este é um exemplo de codificação em *Flutter*. Desta forma se finaliza a parte teórica, indo para o desenvolvimento propriamente dito, pôr em prática e mostrar os resultados.

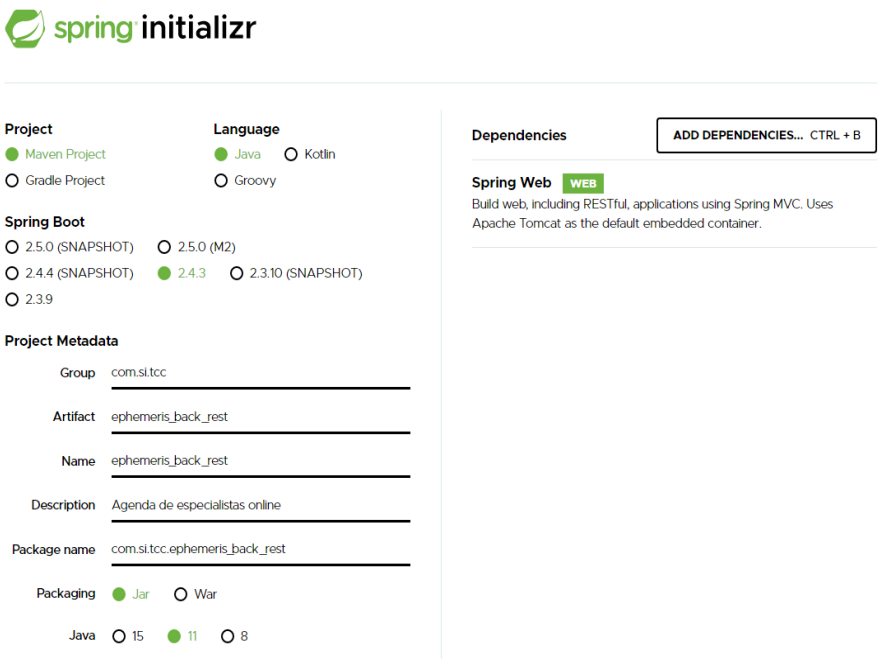
2.3 DESENVOLVENDO O *BACK-END* (*FRAMEWORK SPRING BOOT*)

Conforme Ricardo (2015), O *Spring* é um conjunto de bibliotecas que auxilia na tarefa de injeção de dependências *DI* (*Dependency Injection*), no controle de transações de banco de dados, de segurança etc. Antes de iniciar o projeto, tem uma

ferramenta chamada *Maven* que auxilia no processo de compilação e gerenciamento de dependência do projeto. “Quanto maior o projeto, mais dependências. Isto é uma relação quase certa. Neste contexto, temos a necessidade de organizar os artefatos do projeto de forma simples e eficiente, e é exatamente aqui que o *Maven* pode ajudar” (Ricardo, 2015).

O modo de configurar o projeto inicial no link: https://start.spring.io/ na ilustração abaixo mostra o formulário que quando clicado em “*GENERATE*”, é criado um arquivo compactado *zip* do projeto inicial em *Spring Boot*. De posse deste arquivo, descompactar o mesmo na pasta do projeto. O nome do arquivo gerado é “*POM.xml*” ("*Project Object Model*"). A configuração do projeto vai ser toda importada deste arquivo *xml*. Este arquivo é onde inserimos as dependências do projeto tais como: bancos de dados, componentes de segurança, *e-mail*, persistência de banco de dados etc. Esta facilidade de construção da estrutura básica de projetos através da importação do arquivo *POM.xml* é mantida pela *Apache Maven Project*, o link para mais detalhes: https://maven.apache.org/pom/maven/. Na ilustração 38 a configuração criada no site mencionado anteriormente, o arquivo final de configuração “*POM.xlm*” do projeto está no Apêndice F.

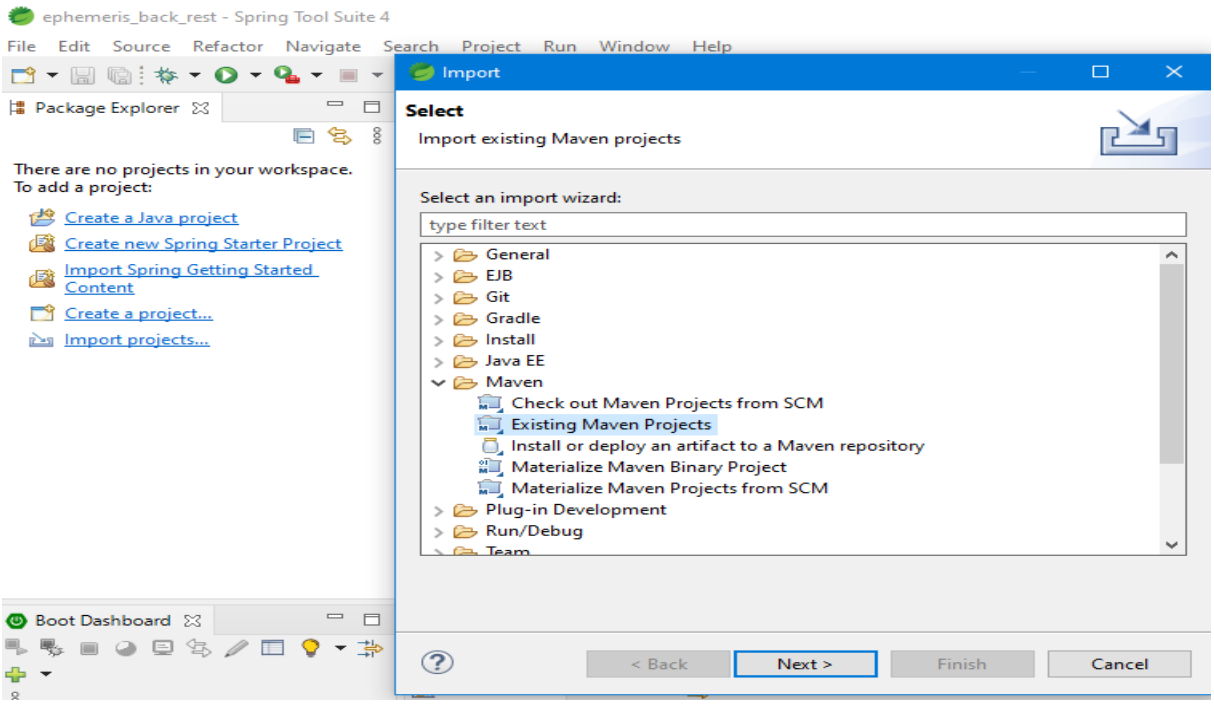
Ilustração 38 - Início do projeto Spring Boot



Fonte: Autores

Após abrir a *Ide ST4* (*Spring Tools 4*), importar o projeto através do arquivo *POM.xml* descompactado na pasta do projeto, conforme ilustração 39.

Ilustração 39 – Importação do projeto *Maven*

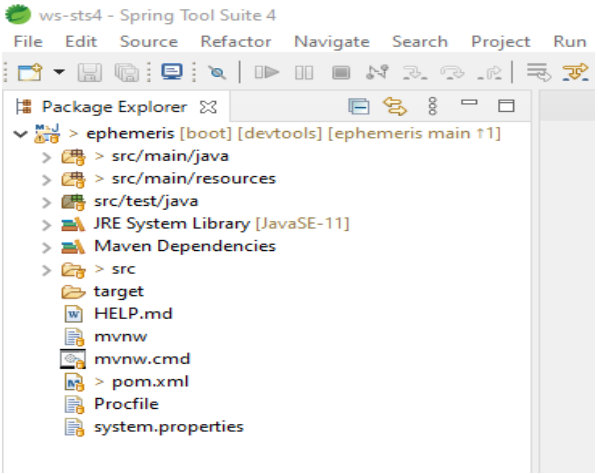


Fonte: Autores

Ao importar o projeto e salvar, serão baixadas todas as dependências para o projeto funcionar a partir do arquivo *POM.xml*.

Ao ser baixadas as dependências, toda a estrutura básica de pastas é organizada, sendo a partir daí dado sequência na programação do projeto. Na ilustração 40, mostra a estrutura básica principal.

Ilustração 40 – Estrutura típica do projeto

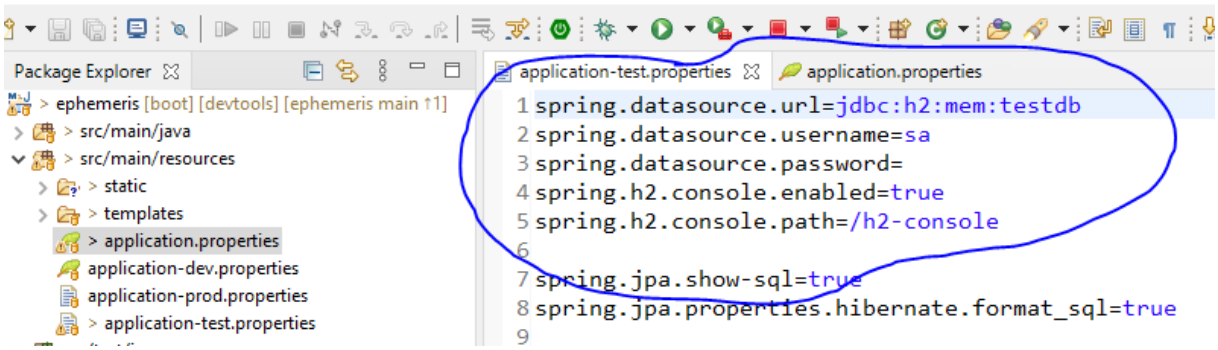


Fonte: Autores

Uma vez que a estrutura do projeto está montada, é chegada a hora de configurar os ambientes de trabalho, ou seja, teste, desenvolvimento e produção.

No ambiente de teste o banco de dados é o H2, este componente está nas dependências do arquivo *POM.xml* e configurado no arquivo "*application-test.properties*, conforme mostra na ilustração 41.

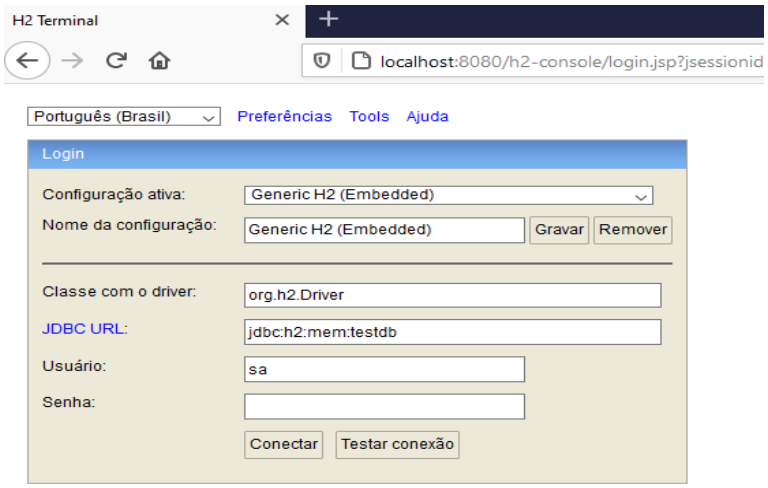
Ilustração 41 - Configuração do perfil de teste (*application.properties*)



Fonte: Autores

Começamos pelo ambiente de teste. Neste ambiente, o banco de dados é o H2 é prático pois, as tabelas são criadas na memória do computador, toda vez que der start no projeto, o banco é recriado com mais rapidez. Este banco de dados é relacional faz parte da estrutura das tabelas e das consultas SQL na sua *Interface* de conexão. Na ilustração 42, mostra a interface de conexão.

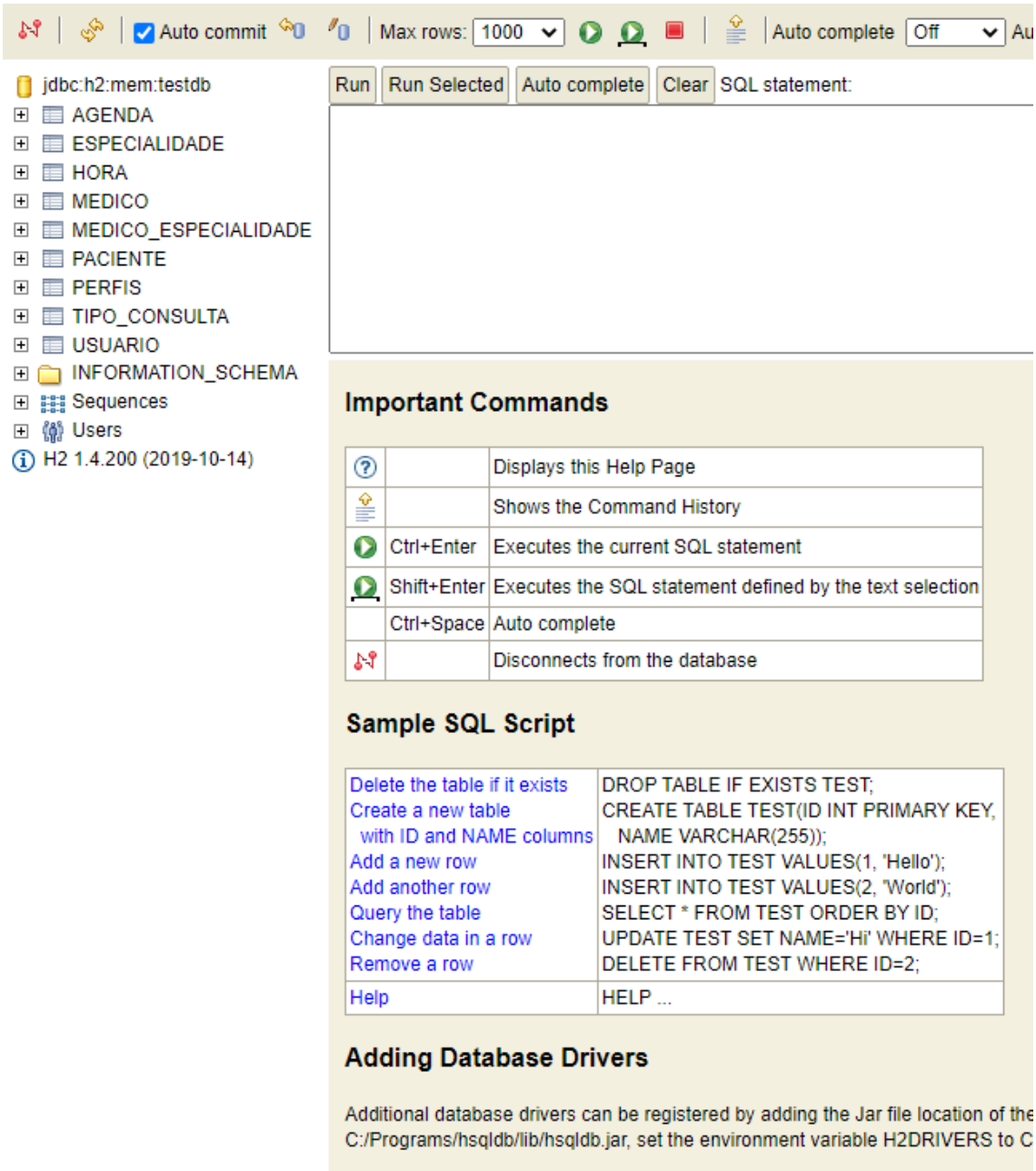
Ilustração 42 – Interface de conexão do Banco H2



Fonte: Autores

Após conectado, o resultado na ilustração 43, mostra as tabelas do banco de dados. Neste caso são mostradas as nove tabelas projetadas para o *webservice*. As tabelas foram construídas pelo *framework Spring Boot* através das classes e seus relacionamentos, para que o framework identifique os relacionamentos entre as classes para relacionar as tabelas, são colocadas as chamadas *@annotations*. Exemplo, para que a classe se transforme numa tabela, se põe na assinatura da classe a anotação *@Entity*.

Ilustração 43 – Tabelas do Bando de Dados



Fonte: Autores

No *framework Spring Boot*, a separação dos perfis: teste, desenvolvedor e produção, se dá por arquivos com o préfixo padrão "*application*" seguidos de: *-test*, -d*ev* e *-prod* somados à extensão "*.properties*", este arquivo de configuração facilita o desenvolvimento dos projetos nas várias fases de construção e testes.

a. *application.properties*;
b. *application-test.properties*;
c. application-dev.properties;
d. application-prod.properties.

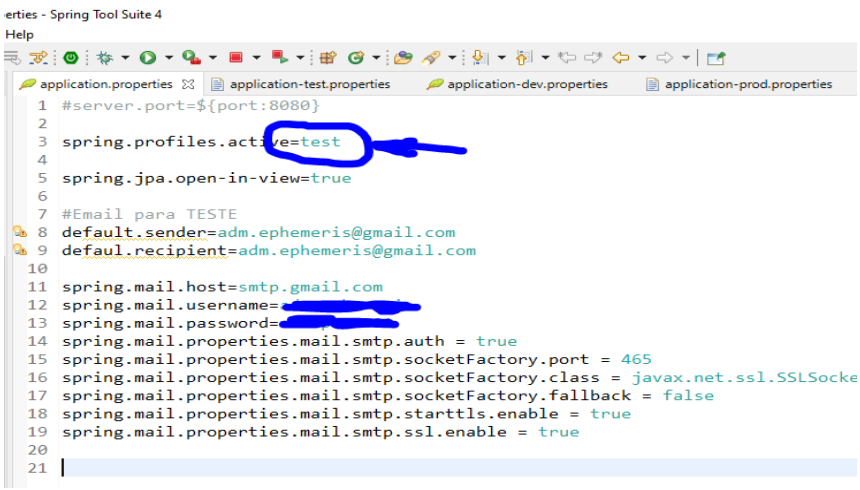
A mudança de perfil de desenvolvimento começando pelo "*test*", faz sentido, pois no início as mudanças são inúmeras, principalmente referente às classes que correspondem as tabelas do banco de dados, nesta fase, então, se configura o banco H2 na opção (b) acima.

Após esta fase já se pode pensar em ter o banco de dados definitivo, fazer o *backup* das tabelas com a exportação do arquivo.*sql* e o diagrama de relacionamentos entre tabelas. No caso em questão foi instalado o banco de dados *PostgreSQL* localmente no desktop. A configuração estaria no arquivo de configuração da opção (c).

Por último o perfil de produção, já direcionado na configuração para o banco de dados na nuvem, na opção (d).

A forma de como de mudar de um perfil para o outro é só alterar a propriedade "`spring.profiles.active=test`" do arquivo padrão "*application.properties*", mostrado na ilustração 44.

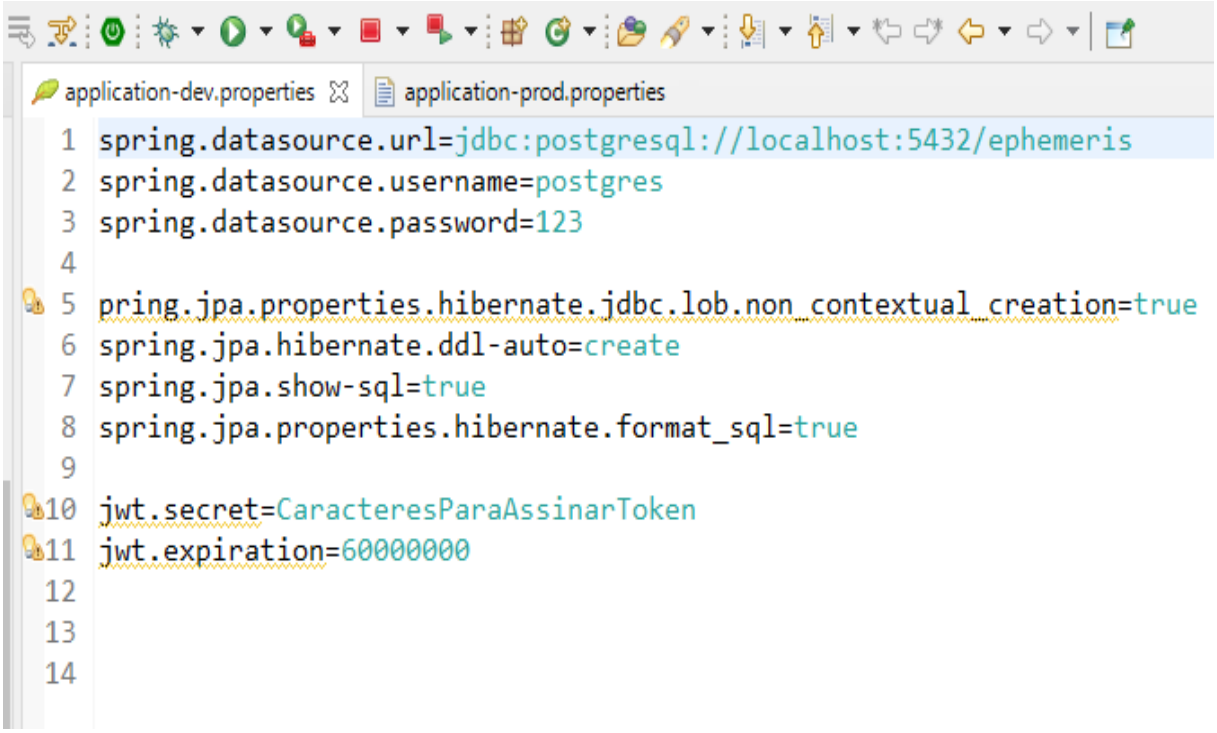
Ilustração 44 – Configurações do *application.properties*



Fonte: Autores

A ilustração 45, mostra o arquivo “*application-dev.properties*” com as configurações padrões para usar o banco de dados *PostgreSQL* localmente, ao trocar para este perfil, o banco em memória H2 deixa de funcionar. O parâmetro `spring.jpa.hibernate.ddl-auto=create` está configurado como “*create*”, significa que o banco de dados será recriado toda vez que o projeto for reiniciado, “*update*” só atualiza as mudanças e “*none*”, não faz nada.

Ilustração 45 – Configurações do *application-dev.properties*

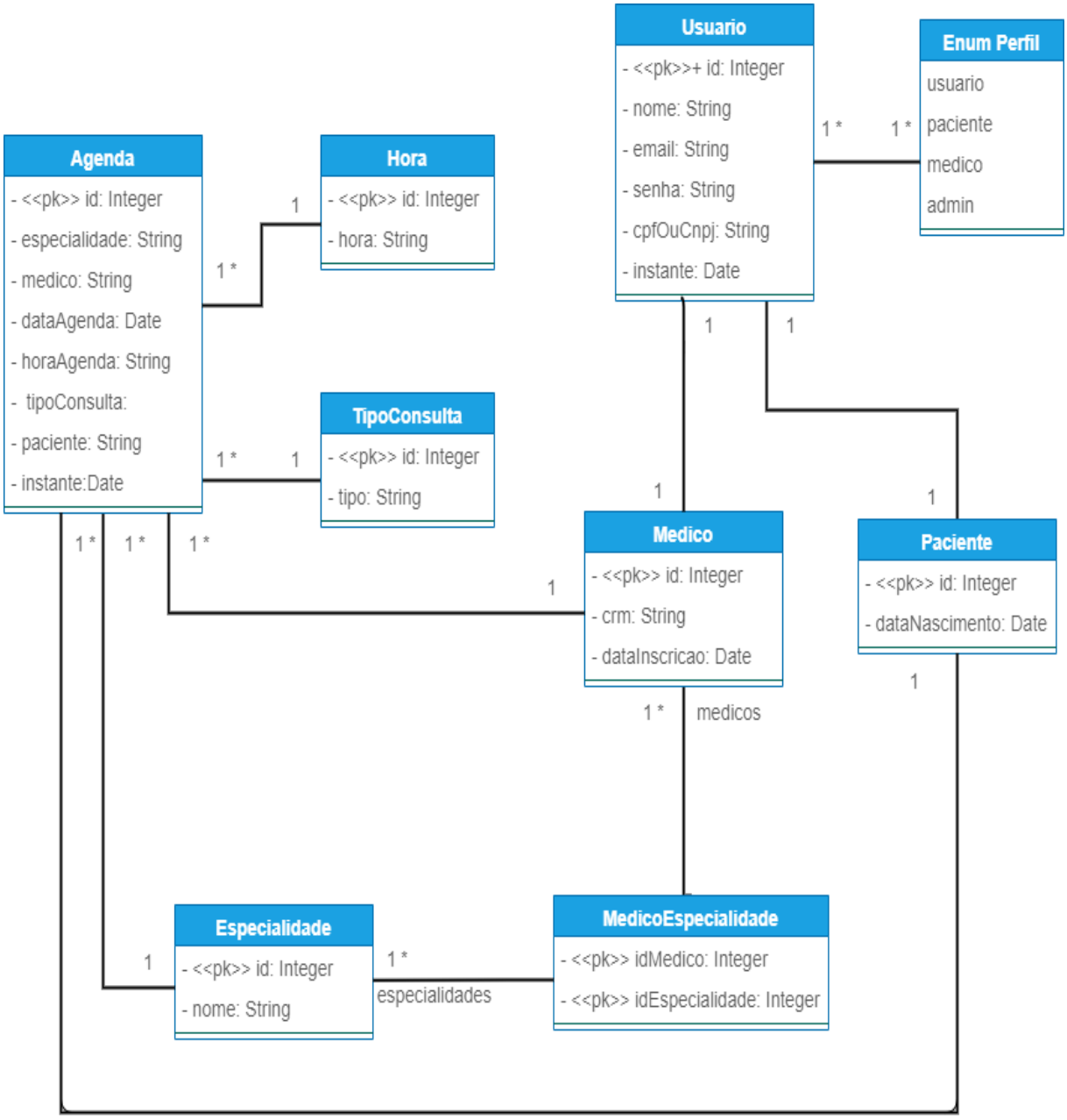


Fonte: Autores

O banco de dados é intrinsicamente lidado ao *back-end*, através da implementação da *JPA* que é colocada como uma dependência no arquivo *POM.xml* para fazer parte do projeto.

Dito isto, o ponto forte relacionado com as tabelas do banco de dados é fazer um modelo conceitual de classes que represente o banco de dados, pois, a implementação *JPA* citada acima cuidará de transformar o modelo de classes em tabelas no banco de dados relacional. Na ilustração 46, mostra este diagrama de classes.

Ilustração 46 – Diagrama conceitual de classe



Fonte: Autores

Exemplificando como os relacionamento entre tabelas são construídos, nas duas ilustrações 47 e 48 a seguir, mostram classes e o uso de anotações *@Entity* entre outras, ficando a responsabilidade da implementação *Hibernate*, cumprir a especificação *JPA*.

Ilustração 47 – Domínio Agenda

```java
package br.com.signote.agenda.domain;

import java.io.Serializable;

@Entity
public class Agenda implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne
    @JoinColumn(name="especialidade_id")
    private Especialidade especialidade;

    @ManyToOne
    @JoinColumn(name="medico_id")
    private Medico medico;

    @DateTimeFormat(iso = ISO.DATE)
    private LocalDate dataDisponivel;

    @ManyToOne
    @JoinColumn(name="hora_id")
    private Hora hora;

    @ManyToOne
    @JoinColumn(name="tipoConsulta_id")
    private TipoConsulta tipoConsulta;

```

Fonte: Autores

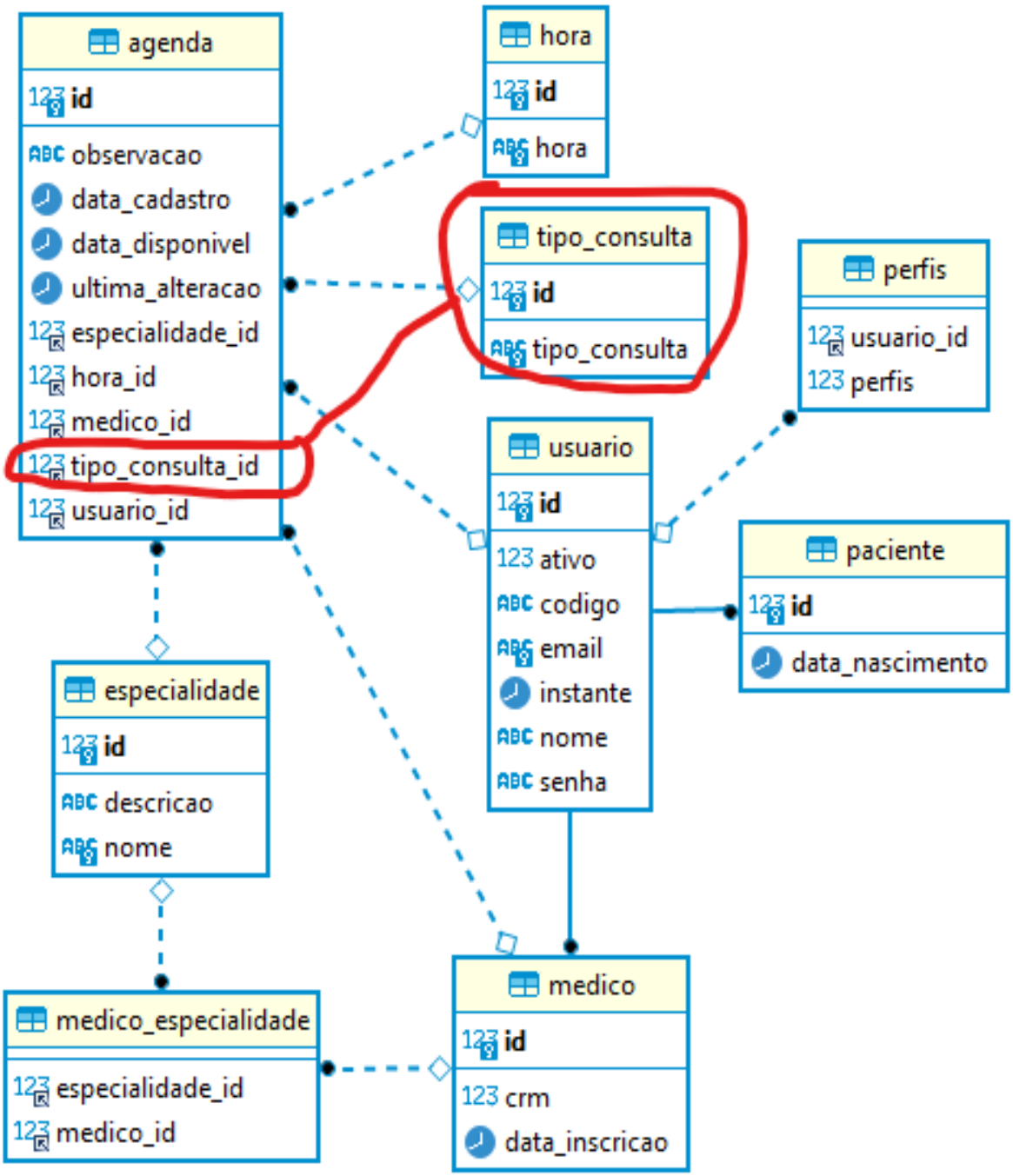
Ilustração 48 – Domínio TipoConsulta

```java
package br.com.signote.agenda.domain;

import java.io.Serializable;

@Entity
public class TipoConsulta implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(unique = true, length = 120, nullable = false)
    private String tipoConsulta;

    @JsonIgnore
    @OneToMany(mappedBy = "tipoConsulta")
    private List<Agenda> agenda = new ArrayList<>();

    public TipoConsulta() {
    }

    public TipoConsulta(Integer id, String tipoConsulta) {
        super();
        this.id = id;
        this.tipoConsulta = tipoConsulta;
    }
```

Fonte: Autores

Como resultado da programação das classes (domínios), o DER (diagrama de entidade relacional) extraído do banco de dados é mostrado na ilustração 49. Nesta ilustração confirma a relação entre as tabelas “tipo_consulta” e “agenda”.

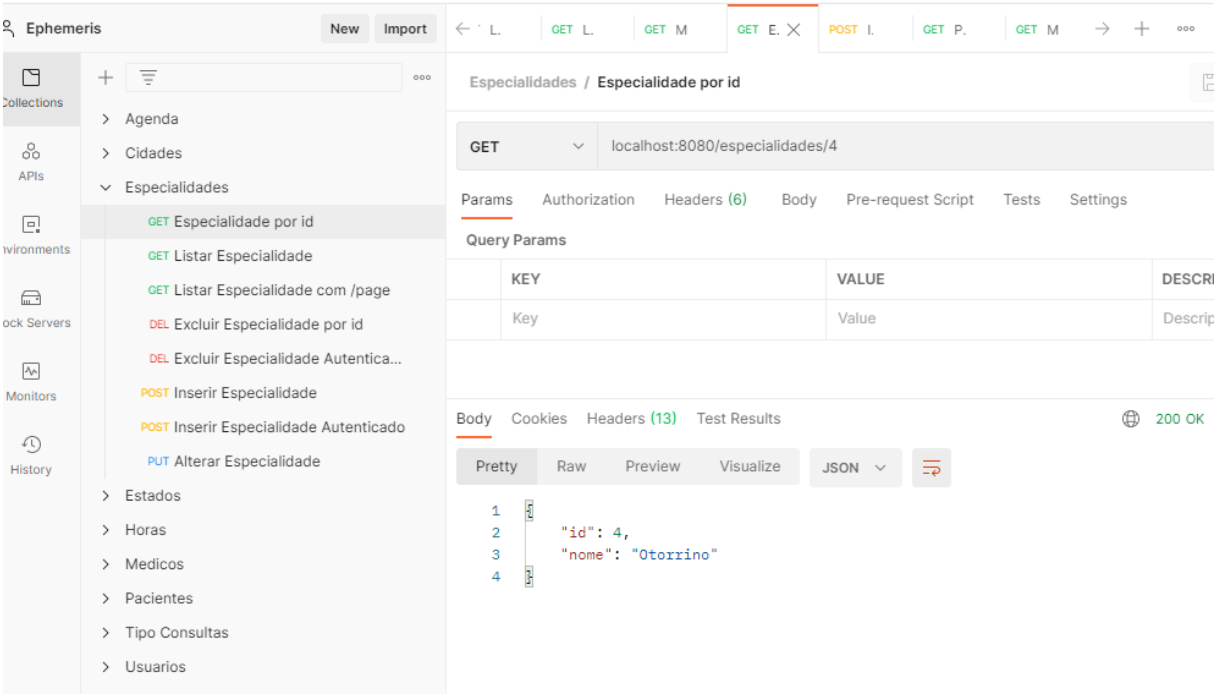
Ilustração 49 – DER (Diagrama de Entidade Relacional)



Fonte: Autores

A ferramenta *POSTMAN*, facilita mostrar resultados do servidor web service, exemplificando na ilustração 50.

Ilustração 50 – Requisição de especialidade por id



Fonte: Autores

O resultado da requisição da especialidade de id = 4, foi “Otorrino”, finaliza aqui a parte do *back-end*.

2.4 DESENVOLVENDO O *FRONT-END (FRAMEWORK FLUTTER)*

No desenvolvimento *mobile* considerado *front-end*, a parte que julgamos mais importante é a conexão com servidor. Pois a conexão envolve obter o *JWT* (*JSON Web Token*), que é a autorização para o acesso ao *webservice* e seus *endpoints*.

Conforme informações obtidas no próprio site https://jwt.io/, trata-se de um método de código aberto padrão RFC7519 para representar reivindicações de forma segura entre duas partes. O padrão é composto de três partes:

- *Header* (cabeçalho)
- *Payload* (carga útil)
- *Sign* (assinatura)

As três partes citadas anteriormente são separadas por (.) ponto.

O padrão adotado para o cabeçalho é codificado em *Base64Uri* para formar a primeira parte do *JWT*.

{

“*alg*” : “HS256”,

“*typ*” : “JWT”

}

Codificado fica assim: eyJhbGciOiJIUzUxMiJ9

A segunda parte do *JWT* chamada de *Pyload* (carga útil), contém as informações mínimas necessárias para acessar o *webservice*, no caso deste *app mobile* é o *e-mail* e a senha do usuário e codificada em *Base64Uri*.

{
"*email*": "p.ephemeris@gmail.com",
"senha": "123"
}

Codificado fica assim:
eyJzdWIiOiJwLmVwaGVtZXJpc0BnbWFpbC5jb20iLCJleHAiOjE2MzExNDM2MjB9

Por último, a assinatura que une as partes *header* e *payload* codificadas em *base64* somadas a uma palavra secreta que está no *webservice* que pode ser criptografada.

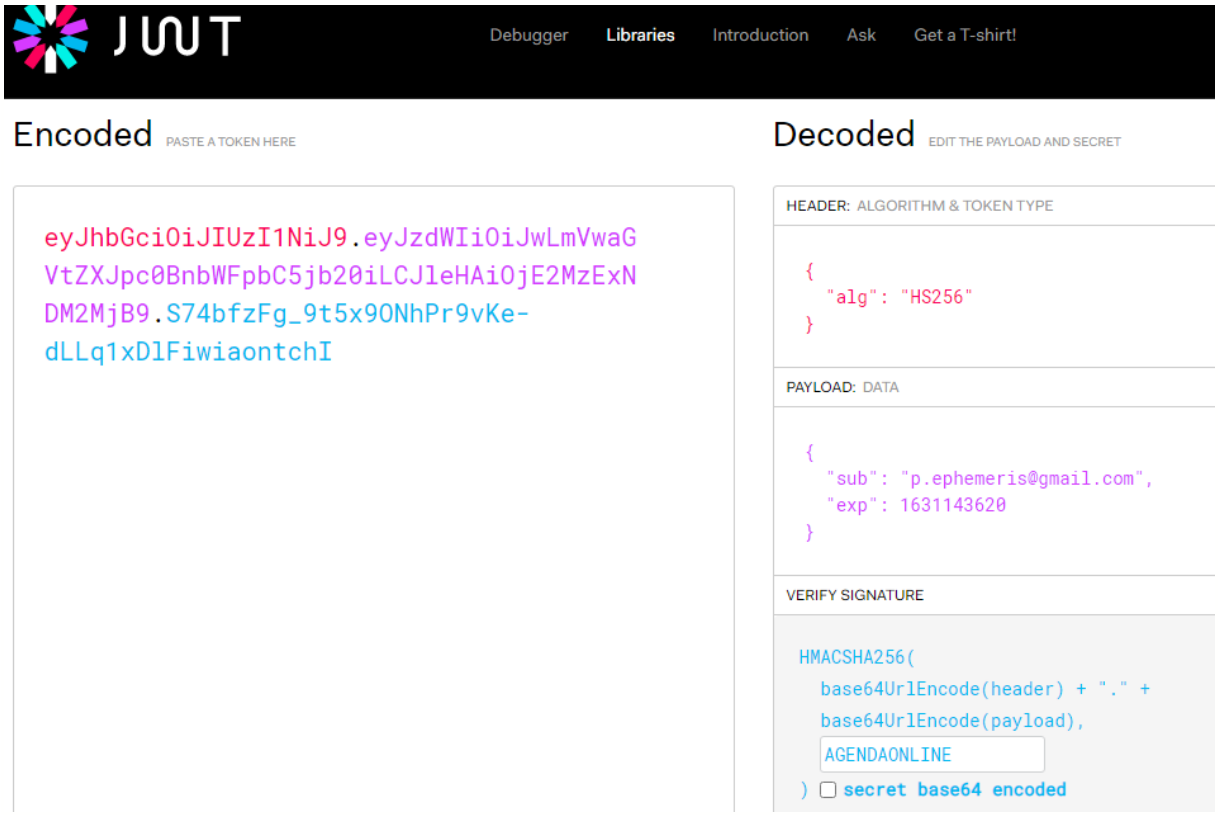
O *Token* completo enviado pelo *webservice* fica então formado pelas três partes e enviada para o aplicativo *mobile*, que este usa este *Token Authorization* (Autorização) para o acesso aos demais *endpoints* que necessitam de autenticação. Qualquer mudança deste *Token* durante o processo de conexão, bloqueia o acesso aos dados do *webservice*.

No site https://jwt.io/, pode ser testado o token recebido do *webservice*, conforme ilustração 51.

O *Token* completo fica assim:

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwLmVwaGVtZXJpc0BnbWFpbC5jb20iLCJleHAiOjE2MzExNDM2MjB9.S74bfzFg_9t5x9ONhPr9vKe-dLLq1xDlFiwiaontchI

Ilustração 51 – *Encoded / Decoded JWT*



Fonte: https://jwt.io/

A classe responsável pela conexão chamada de *LoginApi.dart* está reproduzida em seguida.

LoginApi.dart

```
import 'dart:convert';
import 'package:agenda_front_flutter/pages/login/user.dart';
import 'package:agenda_front_flutter/utils/prefs.dart';
import 'package:http/http.dart' as http;
import 'package:shared_preferences/shared_preferences.dart';

class LoginApi{

  static Future<User> login(String user, String password) async {

    String _email = user;
    String _senha = password;
    String _token = '';

    //var url = Uri.parse('https://agenda-online-tcc.herokuapp.com/login');
    var url = Uri.parse('http://192.168.15.15:8080/login');
    var header = {"Content-Type": "application/json", "Accept-Charset": "utf-8"};
```

```
    Map params = {
      "email" : user,
      "senha" : password
    };

    var prefs = await SharedPreferences.getInstance();

    var _body = json.encode(params);
    print("json enviado: $_body");

       var response = await http.post(url, headers:header, body: _body);
    print('Response status: ${response.statusCode}');

    Map mapResponse = json.decode(response.body);

      _token = (mapResponse["Authorization"] == null) ? '' : mapResponse["Auth
orization"];

 // print("Dados da API: token $_token, email: $_email,  senha: $_senha");

  Map<String, String> mapUser = {
    "email": _email,
    "senha": _senha,
    "token": _token
  };
      final usser = User.fromJson(mapUser);

    if(response.statusCode == 200){
      prefs.setString("tokenjwt", _token);
      prefs.setString("mail", _email);
      usser.save();
    }

    return usser;
  }
}
```

Para fazer o *Login* o usuário não tem o *token* (autorização) e o *webservice* permite a conexão, após a busca no banco de dados e encontra o usuário correspondente, o *webservice* emite um *token*, e a partir deste, todo consumo dos *endpoints* deve ser incluído no cabeçalho da mensagem, esta autorização recebida no *Login*.

As Telas do app são mostradas a seguir:

Ao lançar o app, a tela de boas-vindas vide ilustração 52.

Se o usuário tem cadastro no sistema, clica no botão “Entrar” e a tela da ilustração 53 aparece.

Ilustração 52 - Tela de Boas-vindas

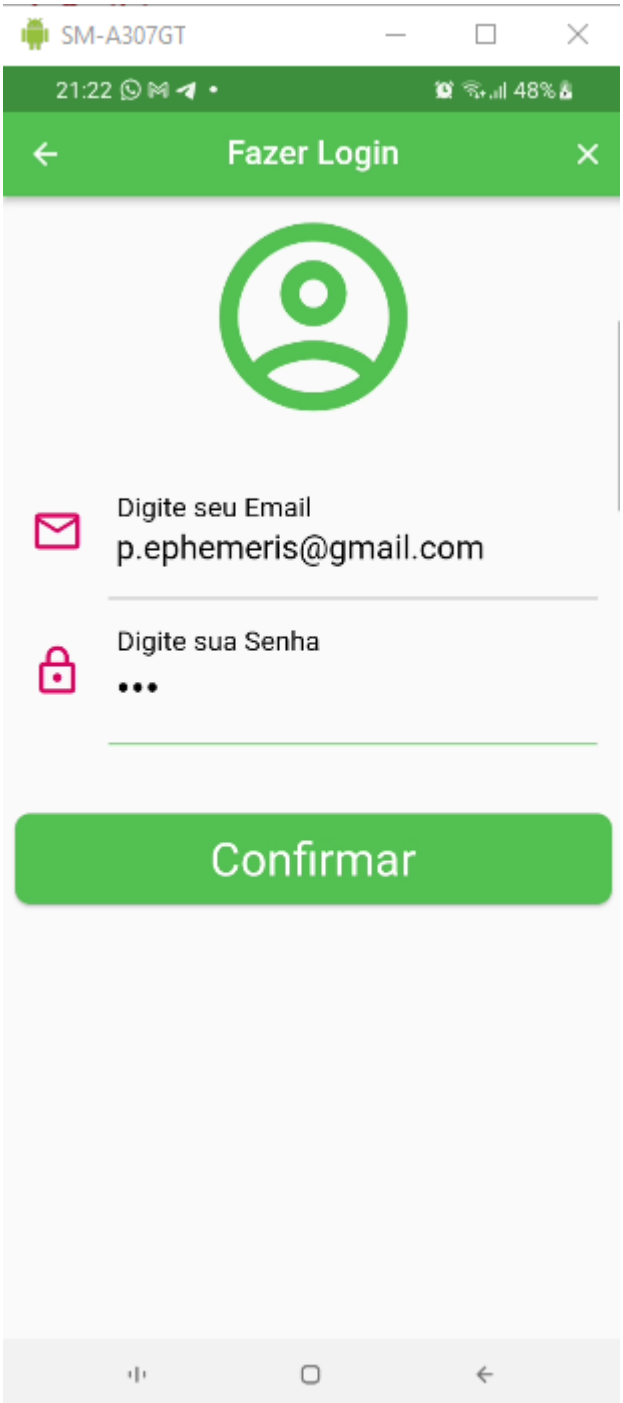
Fonte: Os autores

Ao preencher os campos de *E-mail* e Senha, o usuário clica em “Confirmar”.

Se estiverem corretos, o usuário acessará a tela do menu de opções correspondente ao seu perfil, caso seja um paciente, médico ou administrador (ilustração 55).

Para cada Perfil tem um menu adequado.

Ilustração 53 - Tela de Login



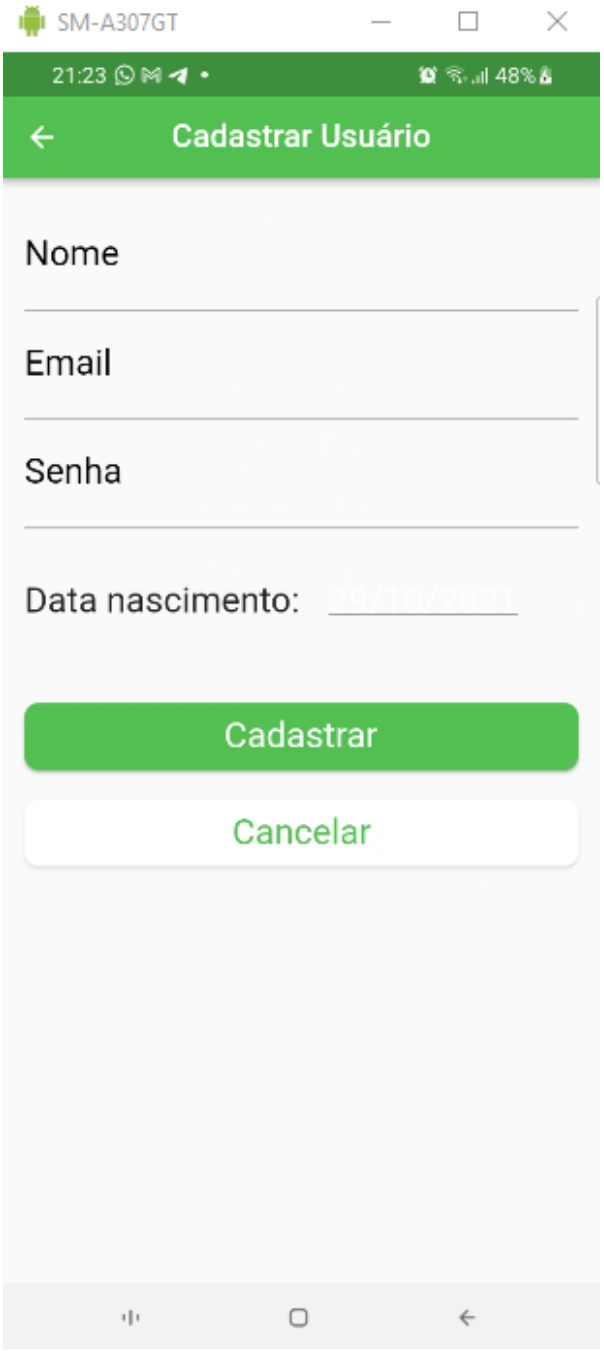
Fonte: Os autores

No caso de o usuário não ter cadastro, deve ser preenchido o formulário de cadastro da ilustração 54.

Em caso de desistir do cadastro, o usuário clica em “Cancelar”, seguindo com o cadastro, clica em “Cadastrar”.

A partir do cadastro feito, o usuário é direcionado para a tela de Boas-vindas (ilustração 52).

Ilustração 54 - Tela de Cadastro de Usuário



Fonte: Os autores

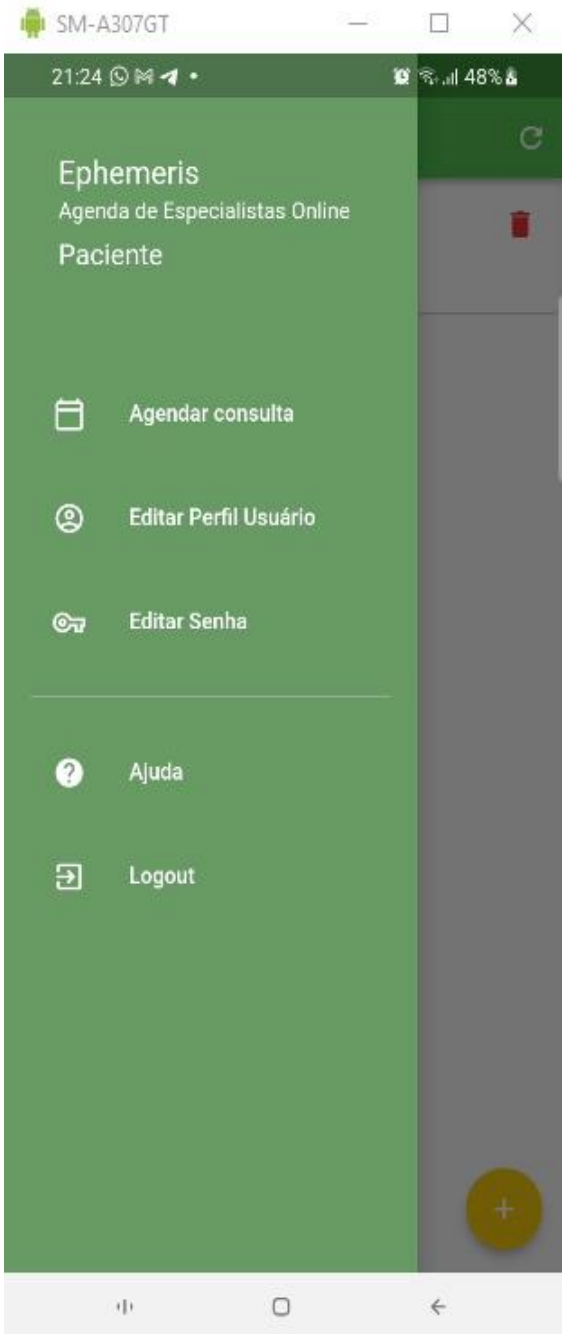
Uma vez o usuário cadastrado no sistema e feito o *Login* corretamente, a tela do menu principal aparece. Este menu é diferente se o paciente, médico ou administrador.

Na ilustração 55, temos a tela do menu do usuário (paciente).

No item “Editar Perfil Usuário”, o usuário terá uma tela semelhante a tela de cadastro (ilustração 54) mas, desta vez é para editar o seu perfil.

Para agendar uma consulta, o usuário clica em “Agendar Consulta” e a tela de agendamento aparece (ilustração 57).

Ilustração 55 - Tela de Menu *Drawer*



Fonte: Os autores

Caso o usuário queira trocar sua senha, clica em “Editar Senha” e a tela Editar Senha ilustração 56 aparece.

Ilustração 56 - Tela de Edição de Senha

Fonte: Os autores

Na ilustração 57, temos a tela de agendamento de consulta.

Primeiro se verifica se tem a especialidade desejada, caso não tenha clica em “Cancelar”.

Se escolher uma especialidade, o próximo passo é escolher qual o médico.

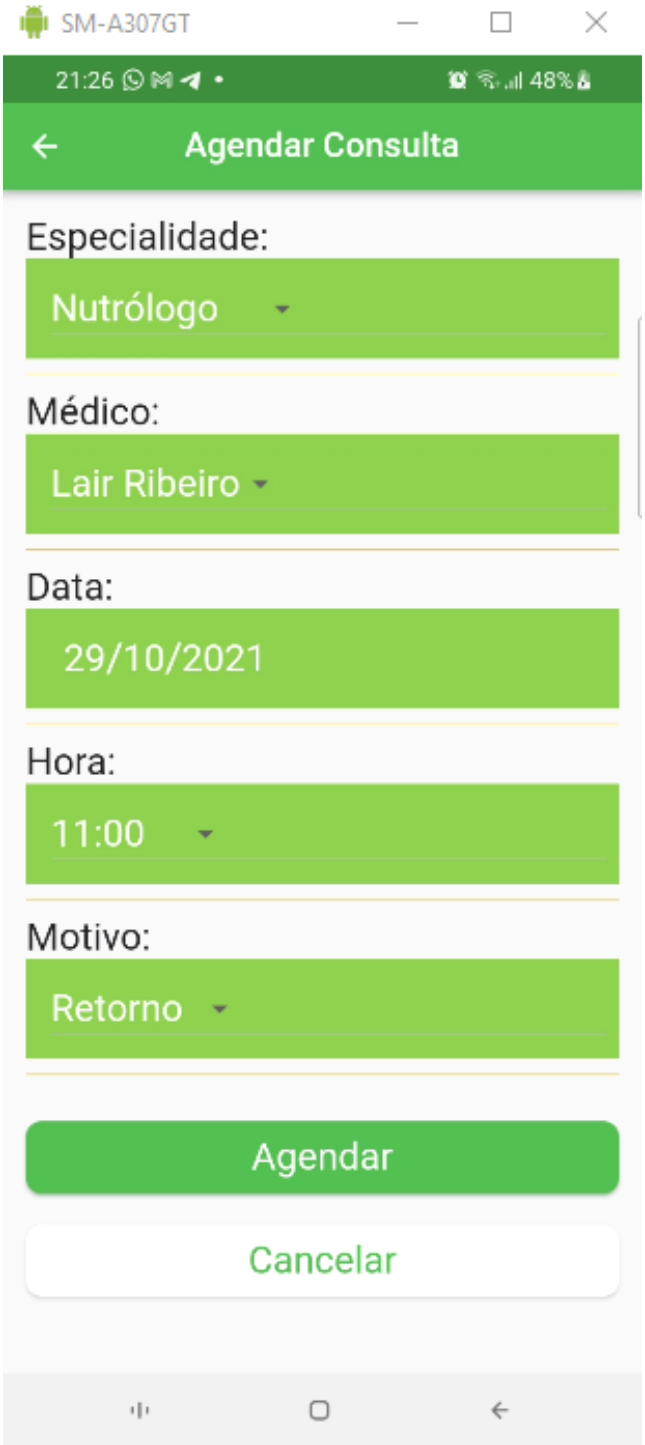
Escolhendo o médico, vai aparecer as datas disponíveis, escolhe uma data.

Em seguida verifica quais os horários existem, e escolhe o horário desejado.

Por fim, qual o motivo, que pode ser “Consulta”, “Retorno” ou “Exames”.

Estando tudo conforme desejado, clica em “Agendar”. O *webservice* enviará para *e-mail* do paciente, a confirmação da consulta com os dados do agendamento.

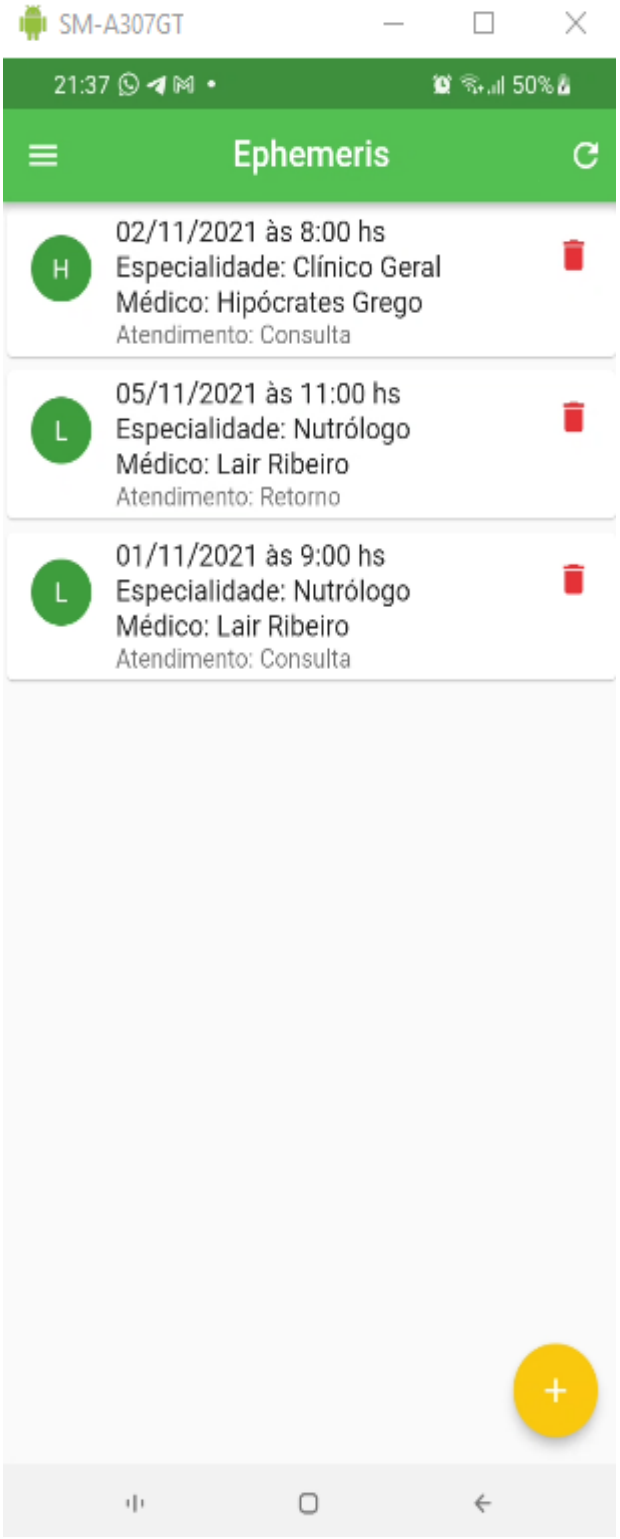
Ilustração 57 - Tela de Agendar Consulta



Fonte: Os autores

Nas ilustrações 58, caso o paciente tenha consultas agendadas, aparecerá nesta tela.

Ilustração 58 - Tela Lista Agendamentos



Fonte: Os autores

3 RESULTADO

Neste capítulo mostra um método de testes com um protótipo construído para este fim, as ilustrações do protótipo estão no Apêndice G. O objeto dos testes é perceber se existe alguma funcionalidade ou recurso que passe desapercebido ou até mesmo que não funcione adequadamente para um usuário. É saber o quanto é fácil de usar sem que precise pensar muito como executar tal função.

Segundo Krug (2014, p. 111),

> Testes de usabilidade consistem na observação de uma pessoa de cada vez tentando usar algo (seja um site, um protótipo de site ou um conceito de design) para executar tarefas comuns a fim de que se detectem e se consertem as coisas que o confundem ou o frustram.

Os testes de usabilidade foram feitos com usuários aleatórios e que esteve dentro de certos critérios:

a) EFICACIA: quanto que o sistema é bom em fazer o que se espera dele.
b) EFICIÊNCIA: quanto que o sistema auxilia o usuário na realização de tarefas.
c) SEGURANÇA: quanto que o sistema protege o usuário de ações indesejadas.
d) UTILIDADE: quanto que o sistema possui as funções necessárias.
e) APRENDIZAGEM: quanto o sistema é fácil de aprender.
f) MEMORIZAÇÃO: quanto que o sistema é fácil de lembrar depois de um tempo sem utilizá-lo.

3.1 MÉTODO PARA O TESTE DE USABILIDADE DO PROTÓTIPO

O método para este teste foi escolher seis pessoas aleatoriamente para execução de cinco tarefas de usabilidade sem tempo definido, a cada tarefa executada e concluída, o tempo foi registrado. Em seguida cada pessoa que executou o teste, respondeu um questionário com dez perguntas relacionadas ao tema usabilidade. Os resultados foram registrados para posterior análise e relatório.

3.1.1 As tarefas escolhidas

a. Fazer cadastro.
b. Fazer Login.
c. Fazer Agendamento.
d. Visualizar Agendamento.
e. Navegação em geral.

3.1.2 Questionário

Após o teste, cada usuário respondeu dez perguntas que foram as seguintes: O questionário está no Apêndice H.

3.1.3 Critério de usabilidade versus questões (medições dos tempos de execução)

Na Eficácia: tempo medido com as questões 3, 9.
Na Eficiência: tempo medido com as questões 2, 3, 6.
Na Segurança: tempo medido com as questões 5, 9.
Na Utilidade: tempo medido com a questão 1
Na Aprendizagem: tempo medido com as questões 7, 8, 10.
No critério Memorização: Este critério não se aplicou no teste, pois, os testes foram feitos apenas uma vez.

3.1.4 Resultados do teste com o protótipo

Na tabela 05, mostra o resultado das tarefas propostas aos usuários e os tempos correspondentes, sendo a última coluna representa a média dos tempos de cada usuário.

Tabela 05 – Tempos de cada tarefa

Seq.	Tarefa	Tempo de segundos						
		Usu. 1	Usu. 2	Usu. 3	Usu. 4	Usu. 5	Usu. 6	Média
1	Fazer Cadastro	30	30	55	49	29	40	38,8
2	Fazer Login	5	17	7	12	3	10	9,0
3	Fazer Agendamento	30	19	20	18	49	25	26,8
4	Visualizar Agendamento	1	3	2	3	5	7	3,5
5	Navegação Geral	66	69	84	82	86	82	78,2

Fonte: Os autores

Todas as ilustrações gráficas extraídas da tabela 05 e do questionário das dez perguntas estão no Apêndice H.

3.1.5 Situações percebidas nos testes do protótipo do aplicativo

Feito os testes de usabilidade, medidos os tempos das tarefas que foram executadas com o protótipo de aplicativo, os resultados foram extremamente relevantes, pois, assim pudemos saber onde estão os problemas e, ao mesmo tempo, por outro lado, o que foi problema para alguns usuários, para outros não foi.

Desta forma aplicando os testes de usabilidade pode corrigir situações que poderiam causar embaraço ao usuário e sua desistência ou relutância em utilizar o aplicativo, contribuindo assim para a qualidade do *software*, concomitantemente com a qualidade final do produto (aplicativo).

Obtivemos três melhorias em relação ao critério de Eficiência, foi percebido que ficaria melhor a troca das posições dos botões Salvar para o lado direito e Cancelar e/ou Voltar para o lado esquerdo, neste caso, no final houve padronização de botões que ocupam a largura da tela, outra melhoria foi em relação a tela de horário que deve ser mais “clara” e deveria ter um botão de menu, neste caso o tipo de tela foi abolido com a simplificação padronizada das telas.

3.2 CONCLUSÃO

Foi construído um *webservice* que disponibiliza vários *endpoints* (*links*/ serviços) que podem ser acessados através de um *login* e senha, utilizando o padrão *JWT* de código aberto para a proteção dos dados do servidor.

Dentro da expectativa de aprendizado de aprender as técnicas de programação de *webservice* e aplicativo *mobile*, foi muito significativo.

Na construção do *webservice* foi utilizado o recurso de perfis de trabalho como teste, de desenvolvimento e de produção que o *framework Spring Boot* possibilita e que agiliza o desenvolvimento do *software*.

Depois do *webservice*, iniciado a construção do aplicativo *mobile* com *framework Flutter*, neste momento se teve a noção da importância do planejamento detalhado, da visão geral dos recursos e da forma adequada do *webservice* em disponibilizar as informações para o consumo pelos clientes. Neste sentido, experiências mais performáticas de construção certamente virá com a prática na produção de *softwares* deste nicho.

Por outro lado, trabalhar com *framework Flutter* que é uma das tecnologias mais novas que surgiu no mercado, exigiu mais estudo, e a documentação disponibilizada pelo desenvolvedor do *framework* foi de grande valia, mesmo assim, foi desafiador na forma de organizar as ideias para formar as interfaces dos usuários, é como montar um quebra-cabeças, neste *framework* tudo são peças que encaixam dentro outras peças (*widgets*). A maior dificuldade foi trabalhar a questão dos tempos de resposta com os processos paralelos, que para contornar o problema, duas funções onde a primeira tinha que ter o resultado primeiro em relação a segunda, que pôr vezes na prática acontecia da segunda função ter o resultado primeiro que a segunda, causando erro na execução, como solução foi colocadas as duas funções dentro da mesma classe, fazendo funcionar na sequência desejada. Outra solução foi fazer a segunda função depender da primeira enviando um argumento como parâmetro da segunda para dar condição de continuidade na sequência correta.

Com a experiência do dia a dia, virá a maturidade e a cada software desenvolvido a certeza de melhores resultados.

As partes construídas estão funcionando, mas para um sistema comercial certamente vários recursos não foram contemplados. O que está feito ajudará em muito em trabalhos futuros utilizando este mesmo tema ou aproveitar ideias deste para outros modelos de negócios.

REFERÊNCIAS BIBLIOGRÁFICAS

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**; 3. Ed. Rio de Janeiro: Elsevier, 2015.

GUEDES, Gilleanes T. A. **UML2 Uma Abordagem Prática**; 3. Ed. São Paulo: Novatec, 2018.

GOOGLE, Flutter. **Flutter documentation**; Disponível em: https://flutter.dev/docs, acessado em 06/2021.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados**; 6. Ed. Porto Alegre: Bookman, 2009.

JAVA, Oracle. **Obtenha Informações sobre a Tecnologia Java**; Disponível em: https://www.java.com/pt-BR/about/, acessado em 11/2020.

KRUG, Steve. **Não me faça pensar**; Uma Abordagem de Bom Senso à Usabilidade na Web e Mobile. 3. Ed. Rio de Janeiro: Alta Books, 2014.

TERUEL, Evandro Carlos. **Arquiteturas de Sistemas para Web com Java utilizando Design Patterns e Frameworks**; 1. Ed. Rio de Janeiro: Ciência Moderna, 2012.

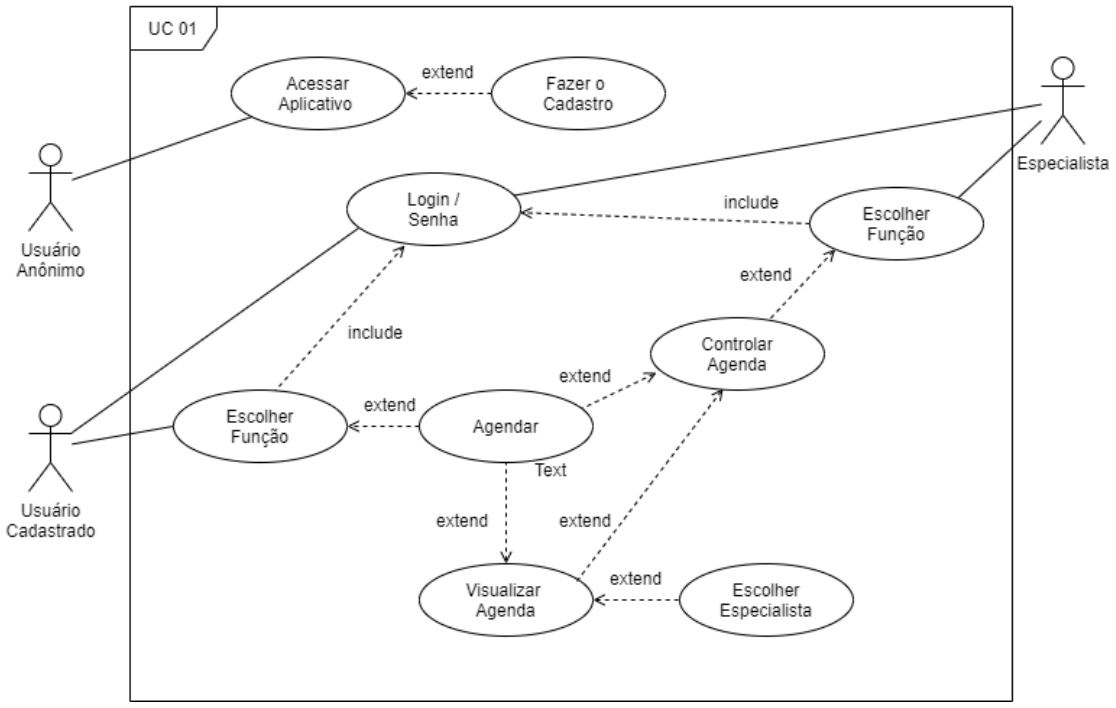
LECHETA, Ricardo R. **Web Services RESTful**; 1. Ed. São Paulo: Novatec, 2015.

STACKOVERFLOW, **Especificação e Implementação**; Disponível em: https://pt.stackoverflow.com/questions/111284/especifica%c3%a7%c3%a3o-e-implementa%c3%a7%c3%a3o, acessado em 03/2021.

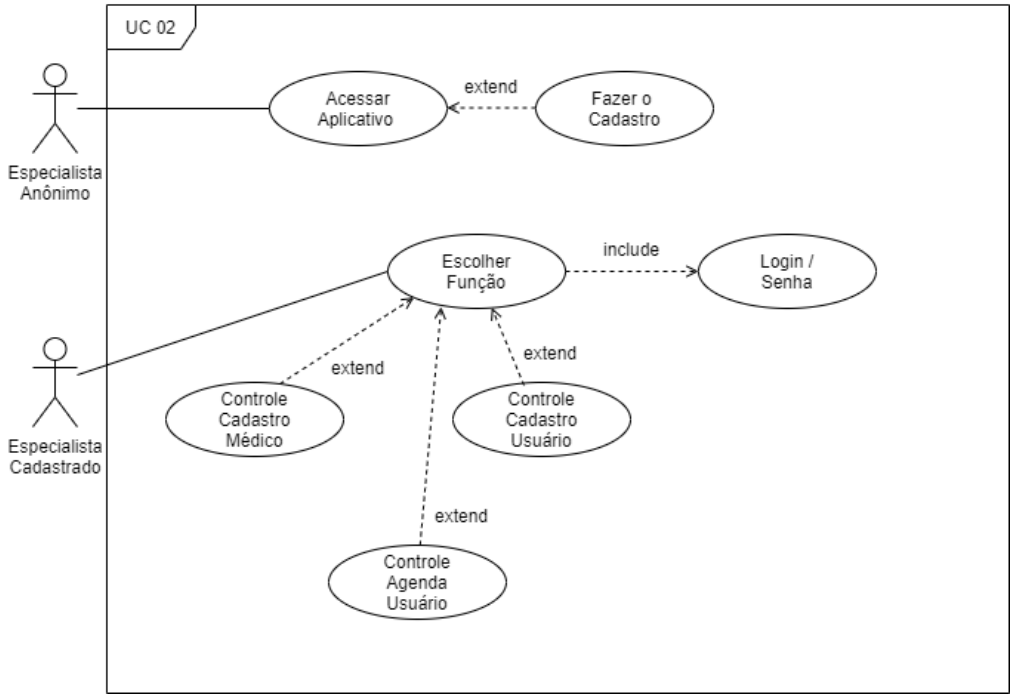
ZAMMETTI, Frank. **Flutter na prática;** 1. Ed. São Paulo: Novatec, 2020.

APÊNDICE A – DIAGRAMA DE CASO DE USOS

ATOR: USUÁRIO = Paciente , Cliente. ATOR OPERADOR = Médico

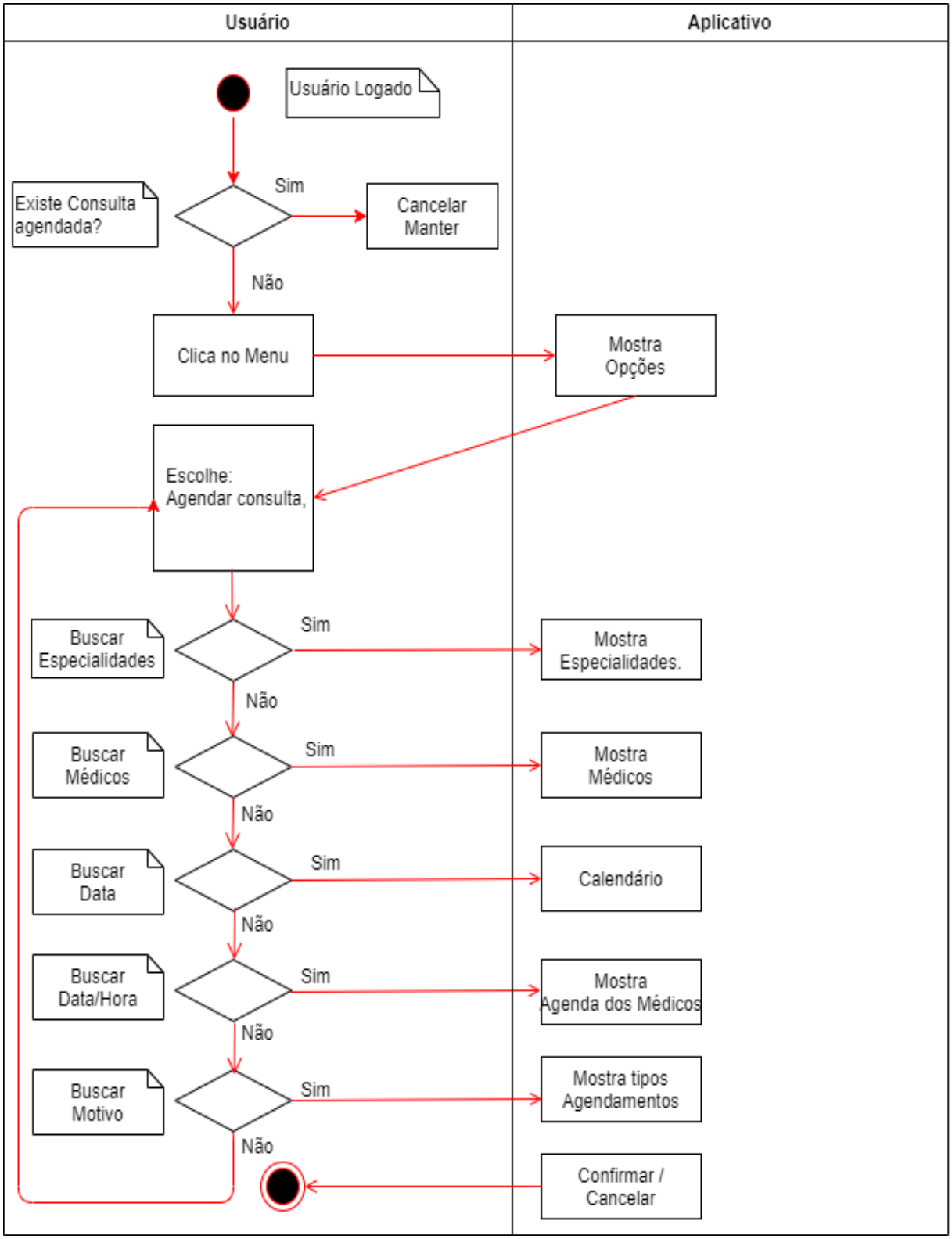


O primeiro Especialista recebe a senha de ADMINISTRADOR que dará acesso a todo o Sistema.

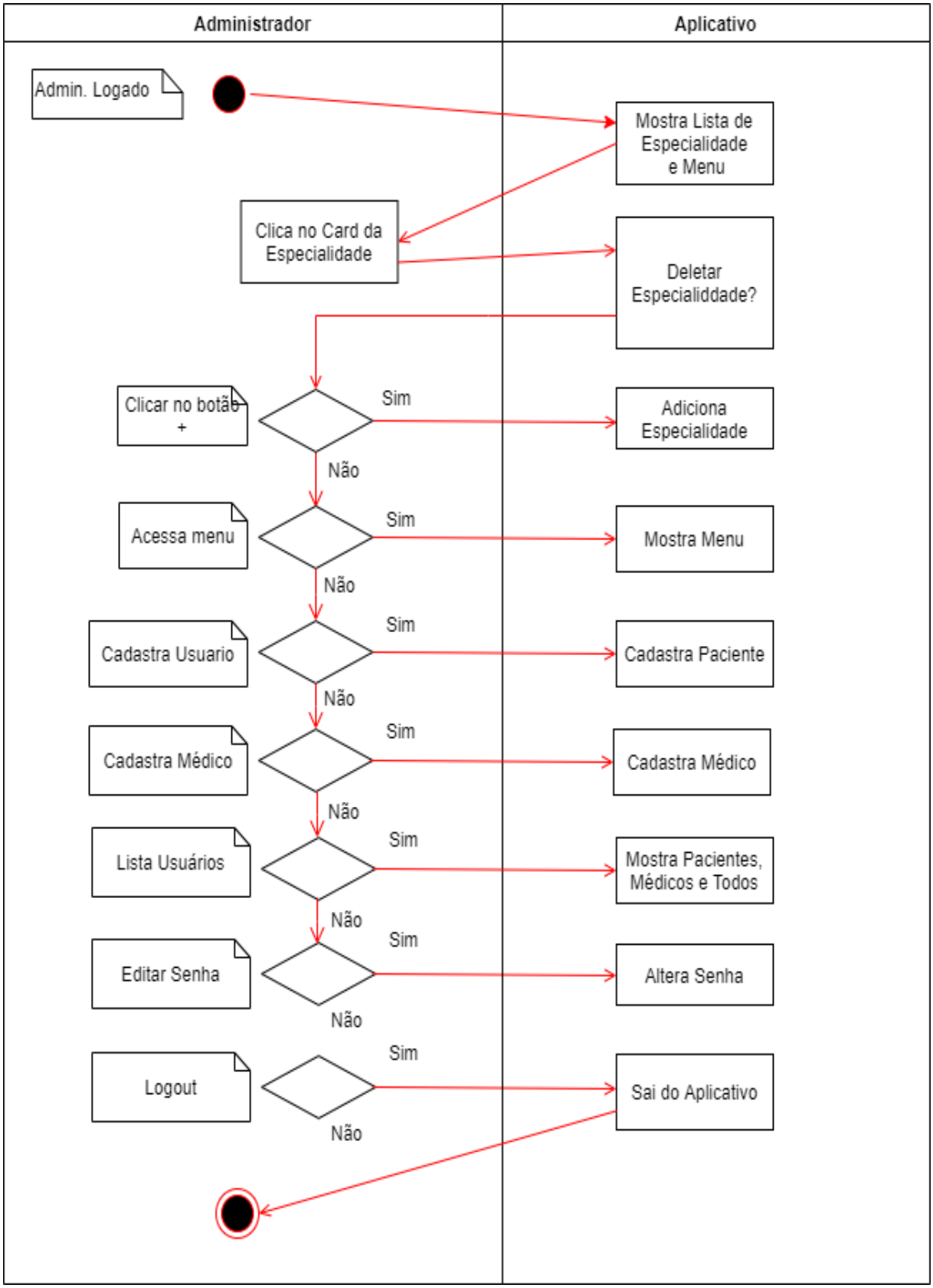


APÊNDICE B – DIAGRAMAS DE ATIVIDADE

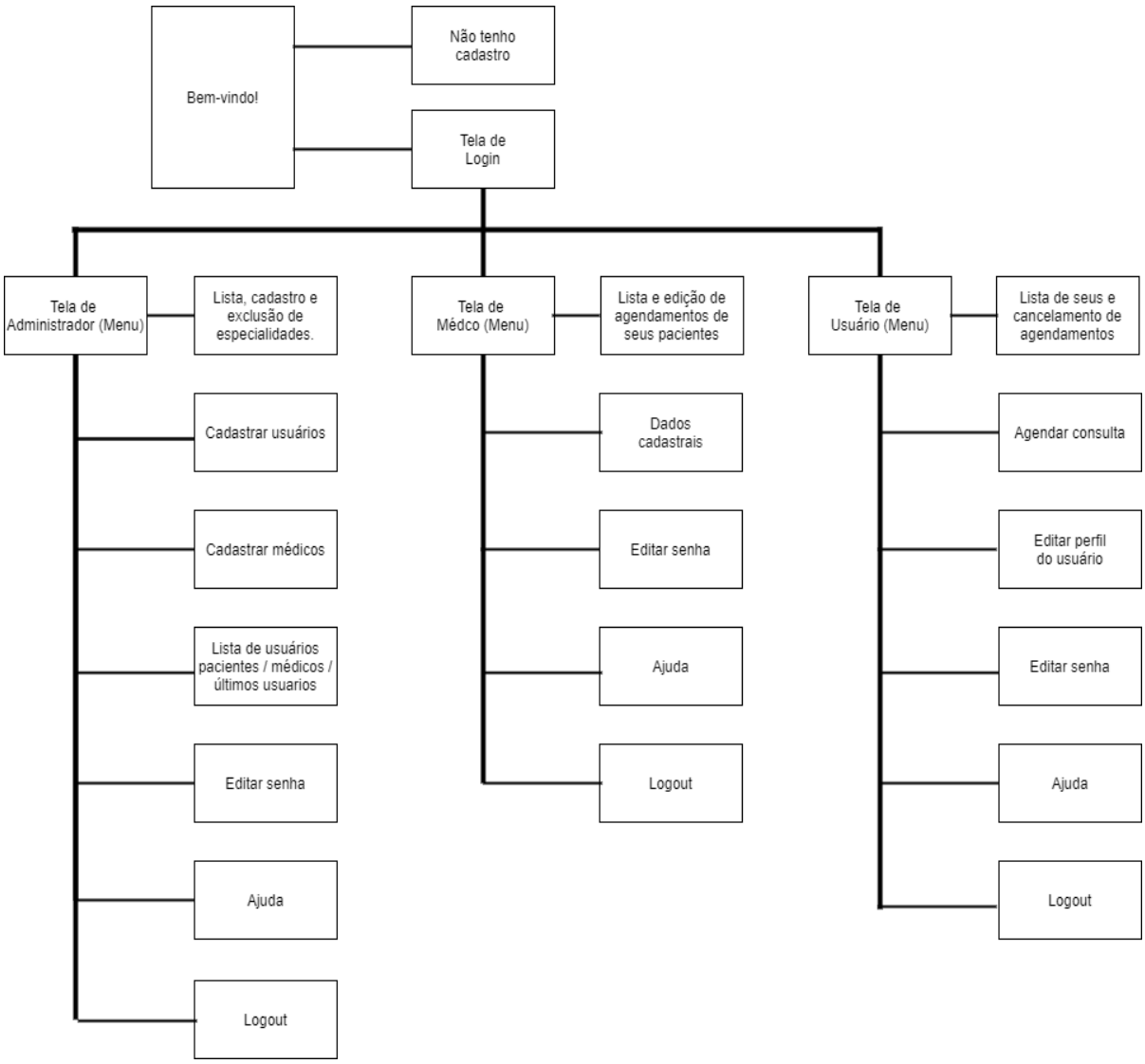
AD01 – Diagrama de atividade do usuário com o sistema.



AD02 – Diagrama de atividade do administrador com o sistema.



APÊNDICE C – MAPA DE NAVEGAÇÃO



APÊNDICE E – FERRAMENTAS (AMBIENTE DE DESENVOLVIMENTO)

a. *JDK* – (Java Development Kit) 8 ou superior– é o *plug-in* usado para desenvolvimento dos programas em *Java*, o *download* pode ser feito no *link*: https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html.
b. *ST4 – Spring Tools 4 – Ide* de programação *Java* com ferramentas direcionadas para o *framework Spring Boot*, o download pode ser feito no link: “https://spring.io/tools”.
c. *Postman* – ferramenta para testes em *webservices REST*, o *download* pode ser feito no *link*: https://www.postman.com/downloads/.
d. *PostgreSQL* – servidor de banco de dados, o *download* pode ser feito no *link*: https://www.postgresql.org/download/windows/.
e. *pgAdmin* - ferramenta de administração de banco de dados *PostgreSQL*, o download pode ser feito no *link*: https://www.pgadmin.org/download/.

APÊNDICE F – POM.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.4.2</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.siderbit</groupId>
	<artifactId>ephemeris</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>ephemeris</name>
	<description>Back-end Agendamento de Especialistas Online</description>
	<properties>
		<java.version>11</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>com.h2database</groupId>
			<artifactId>h2</artifactId>
			<scope>runtime</scope>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-validation</artifactId>
		</dependency>

		<dependency>
			<groupId>org.postgresql</groupId>
			<artifactId>postgresql</artifactId>
			<scope>runtime</scope>
```

```
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-mail</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-thymeleaf</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-security</artifactId>
		</dependency>

		<dependency>
			<groupId>io.jsonwebtoken</groupId>
			<artifactId>jjwt</artifactId>
			<version>0.7.0</version>
		</dependency>
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.7.0</version>
		</dependency>
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger-ui</artifactId>
			<version>2.7.0</version>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

APÊNDICE G – PROTÓTIPO

A Prototipagem foi construída no site: https://quant-ux.com.


telaConsulta
CONSULTAS
DIA
HORA
MÉDICO
09/05
08:00
HIPÓCRATES
09/05
11:00
HIPÓCRATES
10/05
08:00
PARACELSO
FAZER AGENDAMENTO
SAIR

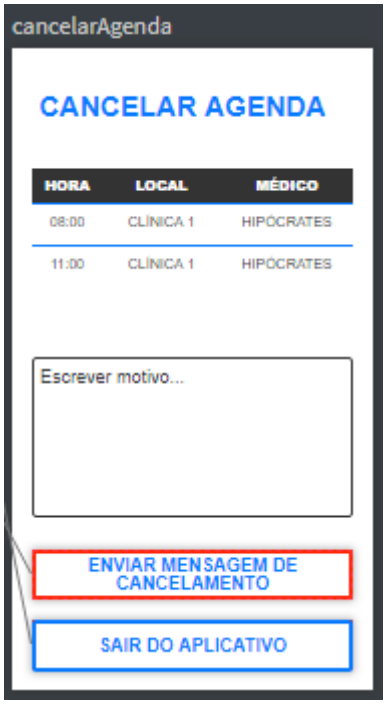

telaAdm
ADMINISTRAÇÃO
CONSULTAS
CADASTROS
AGENDAMENTOS
SAIR DO APLICATIVO


controleAdmin
CONTROLE
May 2018
Sun
Mon
Tue
Wed
Thu
Fri
Sat


confirmarAgenda
CONFIRMAR AGENDA
HORA - ATENDIMENTO
LOCAL
MÉDICO
ESPECIALIDADE
CLÍNICA 1
HIPÓCRATES
CARDIOLOGISTA
SALVAR
CANCELAR


telaConsultas
CONSULTAS
DIA
HORA
PACIENTE
MÉDICO
09/05
08:00
Fulano
HIPÓCRATES
09/05
11:00
Cicrano
HIPÓCRATES
10/05
08:00
Beltrano
PARACELSO
VOLTAR


cadastrosGerais
CADASTROS
PACIENTES
PROFISSIONAIS
CLÍNICAS
VOLTAR


cancelarAgenda
CANCELAR AGENDA
HORA
LOCAL
MÉDICO
08:00
CLÍNICA 1
HIPÓCRATES
11:00
CLÍNICA 1
HIPÓCRATES
Escrever motivo...
ENVIAR MENSAGEM DE CANCELAMENTO
SAIR DO APLICATIVO

APÊNDICE H – QUESTIONÁRIO DO TESTE

1) *Este site (aplicativo) tem muita coisa que é de interesse para mim.*
( ) com certeza ( ) sim ( ) um pouco ( ) não

2) É difícil se mover neste website *(aplicativo)*.
( ) não ( ) poucas vezes ( ) algumas vezes ( ) muitas vezes ( ) sempre

3) Eu pude encontrar facilmente o que eu queria neste site*(aplicativo)*.
( ) não ( ) poucas vezes ( ) algumas vezes ( ) muitas vezes ( ) sempre

4) Este site*(aplicativo)* me parece lógico.
( ) com certeza ( ) sim ( ) algumas ( ) poucas ( ) não

5) Este site *(aplicativo)* me ajuda a encontrar o que eu estou procurando.
( ) com certeza ( ) sim ( ) talvez ( ) um pouco ( ) não

6) Aprender a encontrar meu caminho de volta neste site *(aplicativo)* é um problema.
( ) com certeza ( ) sim ( ) talvez ( ) um pouco ( ) não

7) Usar este site *(aplicativo)* a primeira vez é fácil.
( ) com certeza ( )sim ( )talvez ( )um pouco ( )não

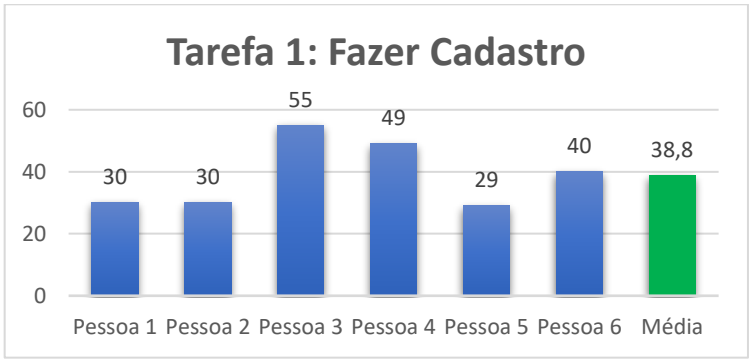
8) É difícil saber onde estou neste site *(aplicativo)*.
( ) com certeza ( ) sim ( ) talvez ( ) um pouco ( ) não

9)Eu recebo o que eu esperava quando clico neste site *(aplicativo)*.
( ) com certeza ( ) sim ( ) talvez ( ) um pouco ( ) não

10)Tudo neste site *(aplicativo)* é fácil de entender.
( ) com certeza ( ) sim ( ) talvez ( ) um pouco ( ) não

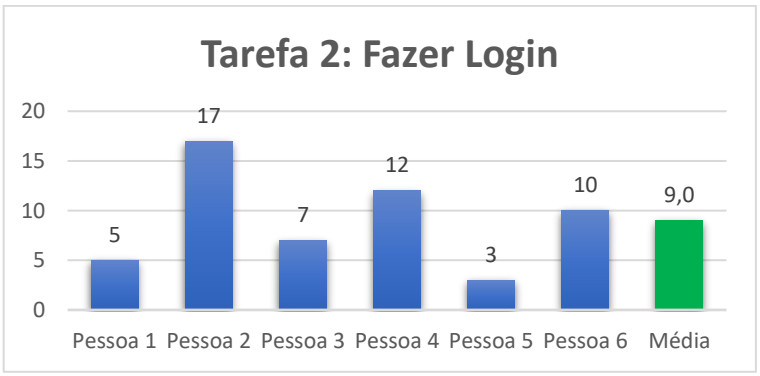
APÊNDICE I – RESULTADO DOS TESTES

Ilustração – Tempo de Cadastro



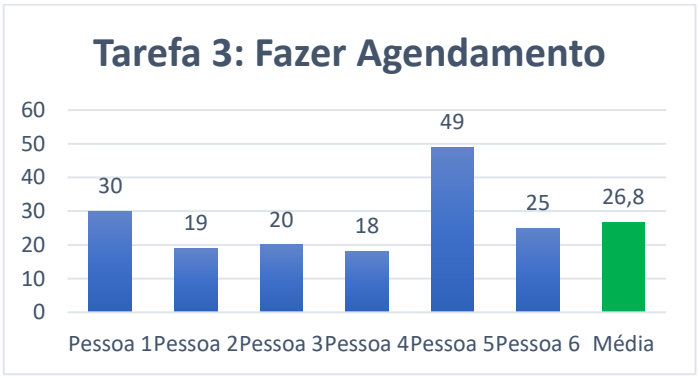
Fonte: Autores

Ilustração – Tempo de Login



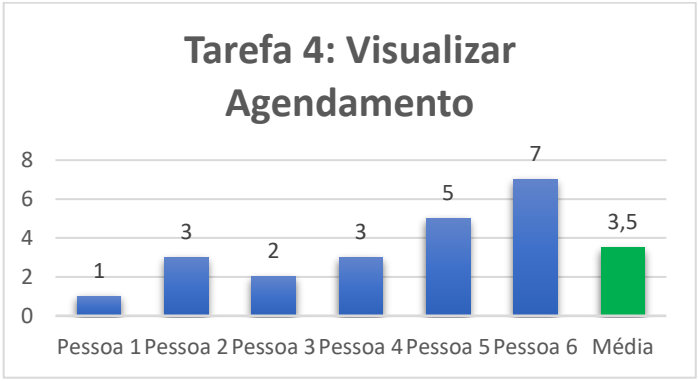
Fonte: Autores

Ilustração – Tempo de Agendamento



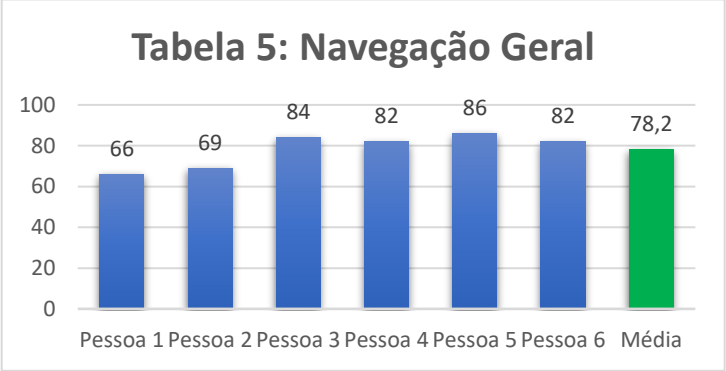
Fonte: Autores

Ilustração – Tempo de Visualizar Agendamento



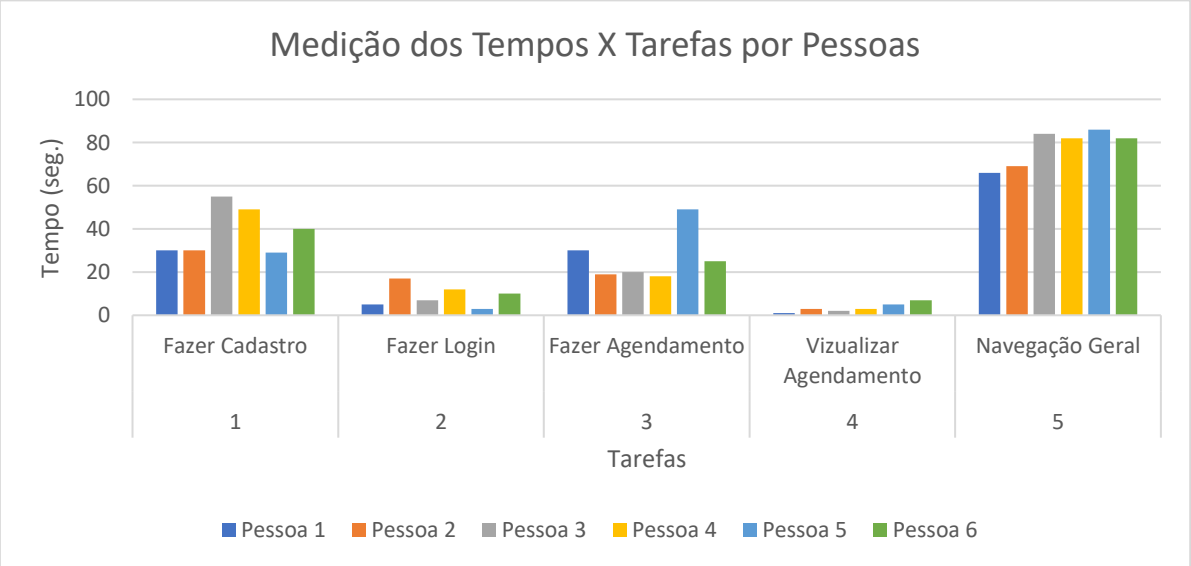
Fonte: Autores

Ilustração – Tempo de Navegação Geral



Fonte: Autores

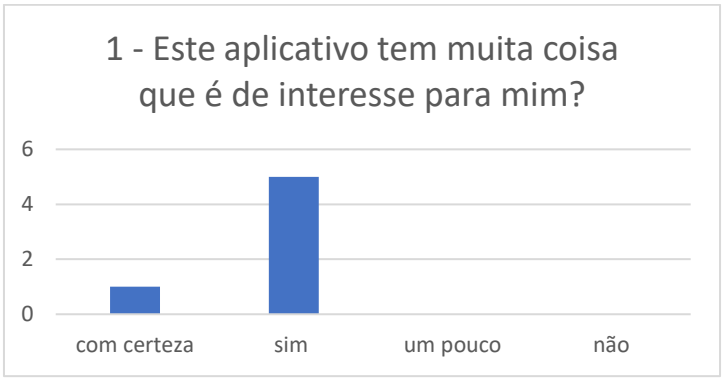
Ilustração – Comparação dos tempos das tarefas entre pessoas



Fonte: Autores

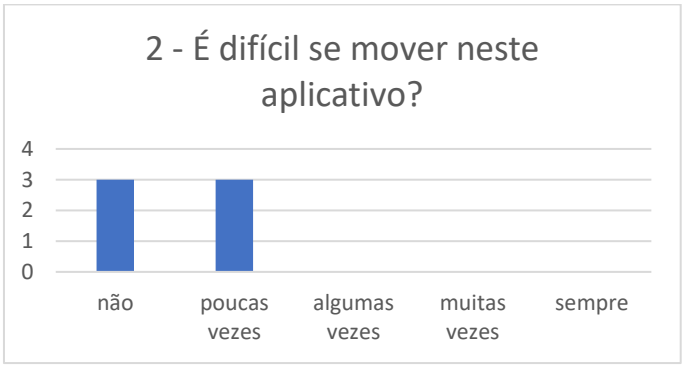
Resultados gráficos das dez questões formuladas a cada participante, que terminava os testes de usabilidade.

Ilustração – Respostas da questão 01



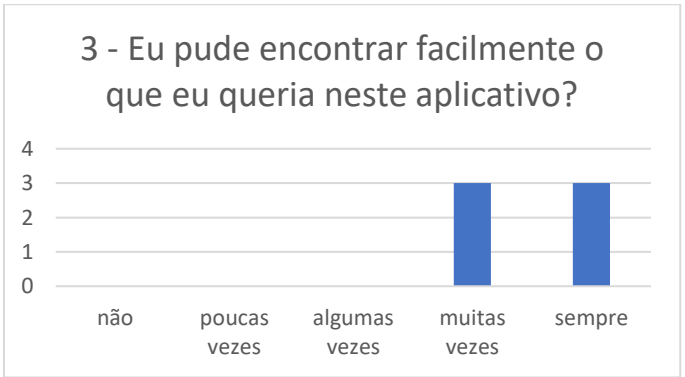
Fonte: Autores

Ilustração – Respostas da questão 02



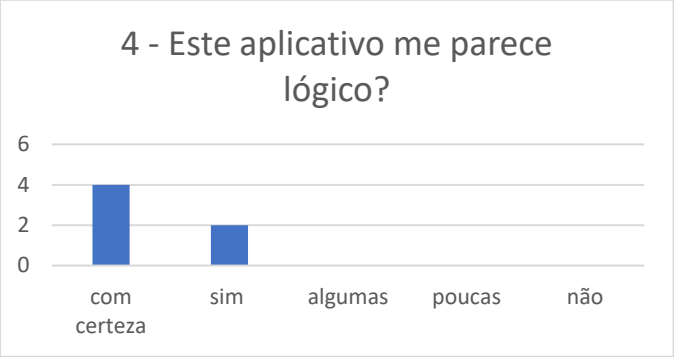
Fonte: Autores

Ilustração – Respostas da questão 03



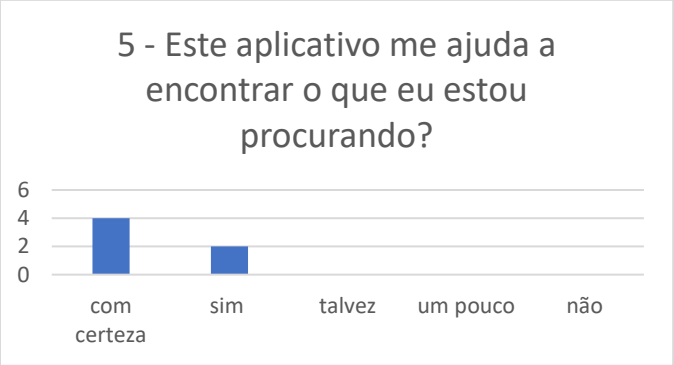
Fonte: Autores

Ilustração – Respostas da questão 04



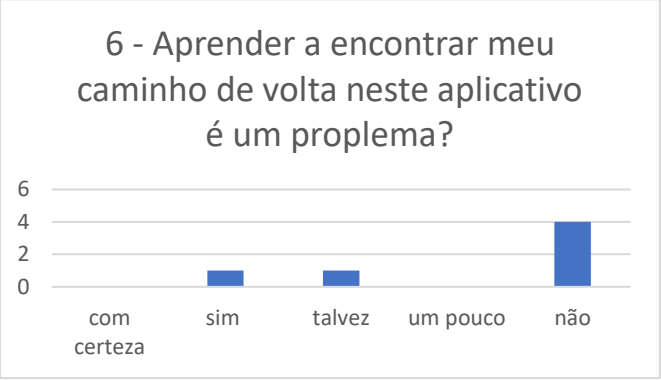
Fonte: Autores

Ilustração – Respostas da questão 05



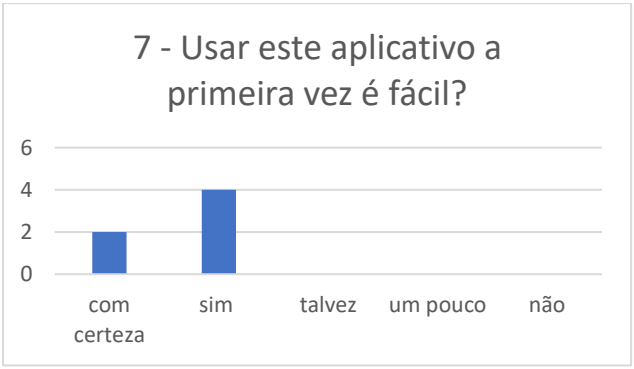
Fonte: Autores

Ilustração – Respostas da questão 06



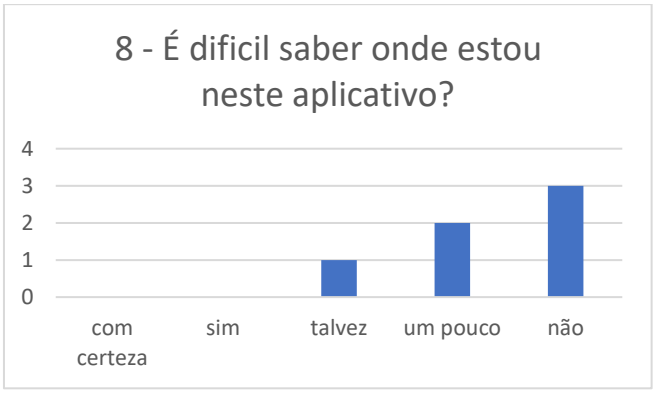
Fonte: Autores

Ilustração – Respostas da questão 07



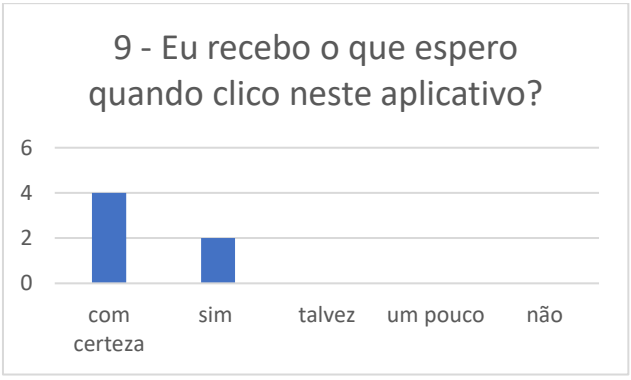
Fonte: Autores

Ilustração – Respostas da questão 08



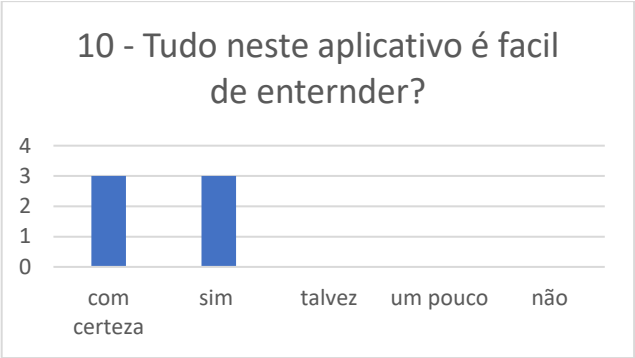
Fonte: Autores

Ilustração – Respostas da questão 09



Fonte: Autores

Ilustração – Respostas da questão 10



Fonte: Autores