UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

STÊNIO MARCOS RODRIGUES CAMARGO FILHO

APLICATIVO MOBILE PARA GESTÃO DE AGENDAMENTO, PRONTUÁRIO E RECEITUÁRIO

PONTA GROSSA

2021

STÊNIO MARCOS RODRIGUES CAMARGO FILHO

APLICATIVO MOBILE PARA GESTÃO DE AGENDAMENTO, PRONTUÁRIO E RECEITUÁRIO

Mobile app for schedule, medical and recipe management

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Ciências da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Prof.ª Dr.ª Simone de Almeida.

PONTA GROSSA

2021

STÊNIO MARCOS RODRIGUES CAMARGO FILHO

APLICATIVO MOBILE PARA GESTÃO DE AGENDAMENTO, PRONTUÁRIO E RECEITUÁRIO

Trabalho de Conclusão de Curso de Graduação/ apresentado como requisito para obtenção do título de Bacharel em Ciências da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 22 de Dezembro 2021

Prof.ª Simone de Almeida
Doutora
Universidade Tecnológica Federal do Paraná

Prof. Geraldo Ranthum
Doutor
Universidade Tecnológica Federal do Paraná

Prof. Richard Duarte Ribeiro
Doutor
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2021

Dedico este trabalho à minha família, pelos momentos de ausência.

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço ao(a) meu(minha) orientador(a) Prof.(a) Dr.(a) Simone de Almeida, pela sabedoria com que me guiou nesta trajetória.

Aos meus colegas de sala.

A Secretaria do Curso, pela cooperação.

Gostaria de deixar registrado também, o meu reconhecimento à minha família, pois acredito que sem o apoio deles seria muito difícil vencer esse desafio.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

Eu denomino meu campo de Gestão do Conhecimento, mas você não pode gerenciar conhecimento. Ninguém pode. O que você pode fazer, o que a empresa pode fazer é gerenciar o ambiente que otimize o conhecimento. (DAVENPORT; PRUSAK, 2012).

RESUMO

Este trabalho apresenta uma discussão sobre temas relacionados ao desenvolvimento de aplicativos móveis e fornece uma solução que permite facilitar a gestão de uma clínica de estética localizada na cidade de Ponta Grossa, Paraná a qual necessita de maneiras mais práticas e rápidas para agendamento de consultas e organização de receituário, prontuário e consultas de seus clientes, com o intuito de inovar e evoluir para que assim sempre fique a frente de outras clínicas. A aplicação utiliza algumas ferramentas como o Ionic framework juntamente com a linguagem Angular, além da API para acesso ao banco de dados denominada Cloud FireStore e o uso de acessibilidade web para criar um aplicativo de fácil acesso ao usuário, possuindo um bom desempenho. Como resultado foi produzido um sistema mobile para gestão de agendamento, prontuário e receituário para uma clínica de estética.

Palavras-chave: Ionic; Angular; Acessibilidade Web; Clínica de Estética.

ABSTRACT

This paper presents a discussion on some topics related to the development of mobile applications and provides a solution that allows to facilitate the management of an aesthetic clinic located in the city of Ponta Grossa, Paraná, which requires more practical and quick ways to schedule appointments and organization of prescriptions, medical records and consultations of its clients, in order to innovate and evolve so that it always gets ahead of other clinics. The application uses some tools such as the Ionic framework together with the Angular language, in addition to the API for access to the database called Cloud FireStore and the use of web accessibility to create an application that is easy to access the user, having a good performance. As a result, a mobile system was produced to manage scheduling, medical records and prescription sums of an aesthetic clinic.

Keywords: Ionic; Angular; Web accessibility; Aesthetic clinic.

LISTA DE ILUSTRAÇÕES

Figura 1 - Tela horários disponíveis para agendamento salão de beleza 23
Figura 2 - Tela de dashboard.....24
Figura 3 - Tela de Agendamento 25
Figura 4 - Tela do Perfil Profissional 25
Figura 5 - Tela de lista do Histórico Médico. (A) Tela de Histórico, (B) Tela de Exames e (C) Tela de Documentos Salvos 26
Figura 6 – Código da Tela de Cadastro de Serviços(cabeçalho) 28
Figura 7 – Código da Tela de Cadastro de Serviços (Corpo) 29
Figura 8 – Front-end Cadastro de Serviços 30
Figura 9 – Interface Serviços.....30
Figura 10 – Servico.service 31
Figura 11 – Cadastro.servicos.ts 32
Figura 12 – Cadastro.cliente.page.html(cabeçalho).....34
Figura 13 – Cadastro.cliente.page.html(corpo) 35
Figura 14 – Cadastro.cliente.page.html(corpo) 36
Figura 15 – Interface Cliente.....37
Figura 16 – Interface Consultas 37
Figura 17 – Cadastro-cliente 38
Figura 18 – Consultar-clientes 39
Figura 19 – Consultar-clientes.page.html(cabeçalho) 40
Figura 20 – Consultar-clientes.page.html (corpo).....40
Figura 21 – Clientes.service 42
Figura 22 – Consulta de Serviços 43
Figura 23 – Tela de Anamnese.....44
Figura 24 – Tela de Tratamentos.....45
Figura 25 – Home.page (A)Mês, (B)Semana, (C)Diário, (D)Menu, (E) Agendamento 46
Figura 26 – Home.page.html (Calendário).....47
Figura 27 – Home.page.html (Botões) 47
Figura 28 – Home.page.html (Agendamento) 48
Figura 29 – Home.page.ts (Cabeçalho) 49
Figura 30 – Home.page.ts(event) 49
Figura 31 – Home.page.ts (Construtor) 50
Figura 32 – Home.page.ts (funções).....51
Figura 33 – Home.page.ts (Adicionar-evento) 51
Figura 34 – Home.page.ts (Outras funções) 52
Figura 35 – Home.page.ts (onEventSelected e onTimeSelected) 53

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CLI.	Command-line Interface
CRUD	Client Read Update Delete
CSS	Cascading Style Sheets
EMAG	Modelo de Acessibilidade em Governo Eletrônico
FGV	Fundação Getúlio Vargas
GPS	Sistema de Posicionamento Global
HTML	Hyper Text Markup Language
MIT	Massachusetts Institute of Technology
SDKs	Software Development Kits
SMS	Short Message Service
SPA	Single Page Applications
TI	Tecnologia da Informação
TIC	Tecnologia da Informação e Comunicação
W3C	Word Wid Web
WAI	Iniciativa de Acessibilidade na Web
WCAG	Web Content Accessibility Guidelines

SUMÁRIO

1	INTRODUÇÃO	13
1.1	Objetivo Geral	14
1.2	Objetivos específicos	14
1.3	Justificativa	15
1.4	Estrutura do Trabalho	15
2	REFERENCIAL TEÓRICO	16
2.1	Aplicativos Móveis	16
2.2	Ionic	17
2.3	Angular	18
2.4	Cloud Fire Store	19
2.5	Acessibilidade Web	19
2.6	Trabalhos Correlatos	21
2.7	Considerações do Capítulo	27
3	IMPLEMENTAÇÃO	28
3.1	Tela de Cadastro de Serviços	28
3.2	Tela de Cadastro de Clientes	34
3.3	Tela de Consulta de Clientes (Pacientes)	38
3.4	Tela de Histórico de Pacientes	43
3.5	Tela de Consulta de Serviços	43
3.6	Tela de Anamnese	44
3.7	Tela de Tratamentos (Receituário e Prontuário)	44
3.8	Tela Home	45
4	CONSIDERAÇÕES FINAIS DO TRABALHO	54
4.1	Conclusão	54
4.2	Trabalhos Futuros	55
	REFERÊNCIAS	56

1 INTRODUÇÃO

Após a Revolução Tecnológica ocorrida no século XX, o homem passou a aperfeiçoar e desenvolver novas tecnologias de informação e comunicação (TIC) que ocasionaram significativas mudanças na sociedade em seu modo de se comunicar, interagir, pensar e de viver, principalmente quando o computador aliado com a internet passou a fazer parte da vida das pessoas. Na sociedade contemporânea existe uma variedade de dispositivos digitais disponíveis no mercado, fazendo da internet uma das principais ferramentas de comunicação e informação do mundo (TYSKA, 2018).

Percebe-se que cada vez mais a população vem incorporando os dispositivos móveis em seu cotidiano para as mais diversas tarefas como assistir filmes e séries, escutar música, realizar leituras e pesquisas, efetuar o *download* de materiais para estudo quanto para ter acesso às informações de inúmeras fontes. Nesse cenário surgem os *smartphones*, telefones celulares desenvolvidos para diversas funcionalidades, que no Brasil de acordo com o Instituto Brasileiro de Geografia e Estatística (IBGE) e sua pesquisa intitulada PNAD 2015 - Acesso à Internet e a Televisão e Posse de Telefone Móvel Celular para Uso Pessoal, vem se consolidando como o principal meio para acessar a internet como forma de diversão, comunicação, pesquisa, estudo, trabalho entre diversas outras atividades (TYSKA, 2018).

A utilização de *smartphones* no mundo não para de crescer, e sua tecnologia cada dia mais não para de avançar. Uma pesquisa realizada pela Google (2013) com mil brasileiros proprietários de *smartphones* relatou que as pessoas estão cada dia mais dependentes de seus aparelhos e que 46% dos entrevistados acessam a Internet todos os dias a partir deles. Um dos resultados desta pesquisa aponta que o comportamento do consumidor está mudando e conclui que as empresas podem se beneficiar desta modificação caso utilizem na organização algum dos recursos oferecidos por estes dispositivos (GOOGLE, 2013).

Pesquisa coloca o Brasil como líder no uso de *smartphones* entre os países emergentes, a quantidade de celulares chega a aproximadamente 234 milhões de aparelhos (FGV, 2021).

Com isto a tendência das empresas é investir nessa área, assim o investimento na área de TI (Tecnologia da Informação) tende a aumentar, principalmente por causa da migração para aos dispositivos móveis sendo antecipada por causa do isolamento, provocado pela pandemia. (FGV, 2021).

O Ensino e trabalho à distância aparentemente deixarão marcas permanentes. Com este rápido impulso na tecnologia muitas atividades do cotidiano do ser humano estão sendo automatizadas para facilitar suas vidas (FGV, 2021).

De acordo com dados obtidos da FGV (Fundação Getúlio Vargas) na sua 32ª edição da Pesquisa Anual do FGVcia pode-se verificar um crescimento de aproximadamente de 8 milhões de *smartphones* no Brasil dos anos de 2020 a 2021. (FGV ,2020; FGV, 2021)

De acordo com o cenário apresentado, este trabalho objetiva o desenvolvimento de uma aplicação mobile para gestão de agendamento, prontuário e receituário de uma clínica de estética, localizada na região de Ponta Grossa, estado do Paraná.

O fato é que aparentemente os procedimentos estéticos estão cada vez mais sendo visados pela população, com isto estas clínicas terão de buscar uma evolução tecnológica tanto nos seus tratamentos estéticos com também nos métodos atendimentos de seus clientes. Deste modo uma aplicação mobile pode ajuda-las a alcançar muito mais clientes e facilitar muito mais sua própria organização de consultas e agendamentos, trazendo um ar de inovação e evolução à clínica.

1.1 Objetivo Geral

Desenvolver um aplicativo mobile para agendamento, prontuário e receituário para uma clínica de estágio situada na região de Ponta Grossa, Paraná.

1.2 Objetivos Específicos

- Realizar a configuração da *API Cloud FireStore*;
- Criar o projeto Ionic com base na linguagem Angular;
- Definir os requisitos necessários da aplicação experimento;
- Utilizar tecnologias que permitam a expansão da estrutura do aplicativo sem comprometer seu funcionamento;
- Identificar os critérios de acessibilidade web na aplicação experimento

1.3 Justificativa

Para desfrutar da oportunidade oferecida pelos smartphones em atingir um grande número de pessoas e, também, objetivando maximizar ainda mais a vantagem competitiva e a inovação entre as empresas de prestação de serviços, mais especificamente na área de estética, este trabalho propõe o desenvolvimento de um aplicativo para *smartphones* que possibilita ao usuário consultar a agenda dos profissionais da clínica de estética da cidade de Ponta Grossa, Paraná, podendo realizar um agendamento prévio na mesma, de acordo com sua disponibilidade.

Além disso, o aplicativo possibilita o controle dos tratamentos disponibilizados pela clínica na forma de um prontuário de seus clientes, controlando os medicamentos indicados pelos profissionais da clínica, permitindo um controle mais efetivo nos tratamentos realizados, possibilitando uma análise mais criteriosa de seu quadro evolutivo, fazendo os ajustes, sempre que necessário. Assim, o resultado deste trabalho pode oferecer novos atrativos à empresa do setor de estética, trazendo mais um diferencial para as empresas desse nicho de mercado

1.4 Estrutura do Trabalho

Este trabalho está organizado em 4 capítulos, sendo o primeiro dedicado à introdução. No segundo capítulo terá um pouco sobre o que é um aplicativo móvel e depois são apontadas as ferramentas que serão utilizadas para o desenvolvimento da aplicação e também será apresenta as métricas que serão cumpridas para que seja uma aplicação de qualidade. O capítulo 3 é dedicado ao desenvolvimento do projeto, apresentando os passos necessários para utilização das ferramentas Ionic e *FireBase* e também os códigos do sistema em Angular, e seus resultados.

No capítulo 4 é destinado as considerações finais do trabalho, apresentando as conclusões acerca do trabalho e os possíveis trabalhos futuros associados a este.

2 REFERENCIAL TEÓRICO

Este capítulo está dividido em 6 seções. Na seção 2.1 são apresentados os conceitos básicos em relação a criação de Aplicativos Móveis. A seção 2.2 discorre sobre os principais aspectos técnicos do Ionic. Na seção 2.3 é apresentada a linguagem angular sendo utilizada no Ionic e alguns exemplos sobre.

A seção 2.4 explica sobre a *API* de banco de dados usada denominada de *Cloud FireStore* e a seção 2.5 apresenta sobre acessibilidade *web* e seus padrões a serem seguidos. Na seção 2.6 encontra-se trabalhos correlatos que possam colaborar com o desenvolvimento deste trabalho. A última seção destina-se às considerações do capítulo.

2.1 Aplicativos Móveis

Aplicativos móveis são programas que são utilizados em dispositivos como aparelhos celulares ou *tablets*. Fincotto (2014, p. 2) afirma que “ao iniciar um projeto de *softwares* para dispositivos móveis, é necessário realizar uma análise criteriosa e estratégica sobre a plataforma, sistemas, produtos e arquiteturas a serem utilizadas”. Isto é necessário porque existem vários sistemas operacionais para *smartphones* e estes determinam a linguagem de programação necessária para construir os aplicativos da plataforma. As diferenças entre linguagens de programação acarretam em *SDKs* (*Software Development Kits*) e paradigmas de desenvolvimento distintos entre as plataformas.

Os aplicativos desenvolvidos com o *SDK* da plataforma móvel são denominados aplicativos nativos. Um aplicativo nativo, segundo Appcelerator (2012, p. 3, tradução nossa), “é um programa específico para dispositivos projetado para ser executado em um dispositivo móvel e seu sistema operacional associado”, ou seja, são aplicações criadas para uma plataforma específica. Os aplicativos nativos “garantem a melhor usabilidade, os melhores recursos e a melhor experiência móvel global”, já que podem “acessar funcionalidades oferecidas por recursos nativos do sistema operacional, tais como Sistema de Posicionamento Global (*GPS*), banco de dados, *Short Message Service* (*SMS*), *e-mail*, gerenciador de arquivos, entre outros” (DA SILVA; SANTOS, 2014, p. 163).

Um problema enfrentado durante o desenvolvimento nativo para múltiplas plataformas é a fragmentação dos dispositivos. Segundo Amatya e Kurti (2014, p. 220,

tradução nossa), “dispositivos com diferentes capacidades de processamento, memória, comunicação e tela são exemplos de fragmentação de dispositivos”. Segundo De Mendonça et al. (2011, p. 5) “o programador precisa levar em conta esses vários fatores na hora do desenvolvimento (...) a fragmentação de um Sistema Operacional é um obstáculo durante o desenvolvimento”, pois estas características influenciam na execução do aplicativo desenvolvido.

Com o objetivo de superar o problema da fragmentação no desenvolvimento nativo e construir um aplicativo compatível com o maior número possível de sistemas operacionais móveis, uma nova abordagem ao desenvolvimento surge, esta é denominado de aplicativo *web* que utilizam o navegador do celular para funcionar, em termos leigos são *sites* adaptados aos dispositivos mobiles que oferecem controles mais simples e intuitivos para o uso no dispositivo. O fato de utilizarem o navegador significa que é necessário ter acesso a internet para funcionar.

Aplicativos híbridos são uma mistura de aplicativos *web* com nativos, ou seja, que funcionam em ambas as plataformas. A maneira como eles são desenvolvidos se assemelha com a de um aplicativo *web*. Utilizam a linguagem *HTML*, *CSS* (*Cascading Style Sheets*) e *Java script* e ainda utilizam recursos nativos do dispositivo móvel por meio de uma *API Java* script comum (DA SILVA; SANTOS, 2014, p. 163). Um aplicativo híbrido é como um navegador de internet, sem usar o aplicativo do navegador do sistema. No caso eles fazem uso de *software* de outras empresas para poder transportar a linguagem da *web* para sistemas móveis.

2.2 Ionic

Ionic[1] é um *framework Open Source* gratuito sobre a licença Massachusetts *Institute of Technology* (*MIT*)[2] para desenvolvimento de aplicações móveis híbridas. Este *framework* não foi construído com a intenção de ser executado em navegadores *desktop*, como o Google *Chrome* ou *Safari*, mas sim em navegadores de baixo nível, como o *WebView*, utilizados em aplicativos híbridos (IONIC, 2021).

Foi desenvolvido por Max Lynch, Bem Sperry e Adam Bradley da *Drifty*, sua primeira versão foi lançada em 2013, hoje se encontra na versão 4. O Ionic auxilia as equipes a criarem e distribuírem aplicativos híbridos para várias plataformas, contendo

[1] https://ionicframework.com/docs
[2] http://opensource.org/licenses/MIT

o foco na experiência do usuário ou sua interação com a aplicação. Ele possui um design "enxuto", simples e funcional. Possui um cliente de linha de comando *CLI* (*Command-line Interface*) que serve para gerenciar todo o projeto criado com Ionic. Essa é uma ferramenta que cria aplicações Ionic de forma rápida e fornece inúmeros comandos uteis para facilitar o desenvolvimento utilizando o *framework*. As vantagens de se usar o Ionic são (ANDRADE, 2021):

- **Estabilidade na criação de aplicações híbridas:** entrega de um produto altamente estável e com desempenho similar ao de aplicativos nativos.
- **Multiplataforma:** pode-se desenvolver um código que execute em mais de um sistema. Isso diminui o tempo de desenvolvimento de novas aplicações.
- **Menor tempo de desenvolvimento:** mais vantajoso do que as aplicações nativas pois funcionam em multiplataformas.
- **Menor custo:** uma vez que se poupa tempo de desenvolvimento, reduzindo os custos da aplicação, fazendo com que criar as aplicações de multiplataformas seja algo mais rentável e menos custoso.
- **Prototipação:** Criação de interface é fácil, já que possui uma ferramenta para tal finalidade *Ionic Creator* ("um simples arrasta e solta").
- **Documentação:** Existe uma grande comunidade e vasta documentação.

Ionic (2021) afirma que algumas tarefas desempenhadas pelo Ionic *Framework* requerem o Angular, portanto, é necessário entender como ele funciona para explorar todo o potencial da ferramenta.

2.3 Angular

O Angular é um *framework* estrutural baseado em *JavaScript*, mas por meio do *TypeScript* de código-fonte aberto (*open source*) mantido pela Google para a construção de *SPA (Single Page Applications)* uma criação de aplicações dinâmicas da *web* (GUEDES, 2021).

As principais características são:

- **Suporte *cross-plataform*:** Esse *framework* fornece suporte a múltiplas plataformas de desenvolvimento.
- **Integração e suporte a todas as fases de desenvolvimento:** Fornece ferramentas e suporte para todas as fases de desenvolvimento, desde o

código em si até a criação de fluxos de teste, passando pelo suporte excelente à criação de animações.

- **Produtividade aliada à performance:** Oferece suporte ao desenvolvimento rápido de aplicações por meio de uma *API* simples, bem estruturada e bem documentada, o que acaba gerando bastante produtividade.

2.4 Cloud Fire Store

O *Cloud FireStore*[3] é um banco de dados flexível e escalonável para desenvolvimento focado em dispositivos móveis, *web* e servidores a partir do *FireBase* e do Google *Cloud* (FIREBASE, 2021).

Ele tem como principais recursos:

- Flexibilidade;
- Consultas expressivas (o desempenho da consulta é proporcional ao tamanho do conjunto de resultados);
- Atualização em tempo real;
- Suporte *Off-line;*
- Projetado para escala.

Ele é um banco de dados NoSQL hospedado na nuvem para que os aplicativos do IOS, do Android e da *web* possam acessar diretamente por meio de *SDKs* nativos (FIREBASE, 2021).

2.5 Acessibilidade Web

A acessibilidade à *web* refere-se a forma de garantir acesso facilitado a qualquer pessoa, independente das condições físicas, dos meios técnicos ou dispositivos utilizados. No entanto, ela depende de vários fatores, tanto de desenvolvimento quanto de interação com o conteúdo (EMAG, 2014).

A ideia de *web* acessibilidade é dar conforto e praticidade a todos os usuários, baseando-se em documentos que tem como seu principal foco, estabelecer critérios que ajudam a tornar o conteúdo acessível a todas as pessoas. Um desses documentos que é considerado o principal internacionalmente é o *WCAG 2.0* (*Web*

[3] https://firebase.google.com/docs/firestore

Content Accessibility Guidelines), que foram desenvolvidas pelo consorcio *W3C* (*Word Wid Web*), por meio do *WAI* (*Iniciativa de Acessibilidade na Web*).

Documentos da *WAI*:

- **WCAG** - para conteúdo *web*;
- **ATAG** - *Authoring Tool Accessibility Guidelines*: para ferramentas de autoria, editores HTML, *content management systems* (CMS), *blogs*, *wikis*, etc;
- **UAAG** - *User Agent Accessibility Guidelines*: para navegadores *web*, *media players* e outros agentes de usuário;
- **WAI-ARIA** - *Accessible Rich Internet Applications Suite*: aplicações *web* ricas e acessíveis (desenvolvidas com Ajax, por exemplo).

Estrutura do documento *WCAG* 2.0 é baseado em quatro princípios, cada um contendo recomendações. As recomendações possuem critérios de sucesso que devem ser seguidos. Para seguir os critérios de sucesso, são disponibilizadas técnicas específicas.

Dos quatro princípios o primeiro é o Perceptível onde a informação e os componentes da interface do usuário têm de ser apresentados aos usuários em formas que eles possam perceber, o segundo é o Operável cujo os componentes de interface de usuário e a navegação têm de ser operáveis, terceiro é o Compreensível no caso a informação e a operação da interface de usuário têm de ser compreensíveis e por último o quarto princípio que é o da robustez onde o conteúdo tem de ser robusto o suficiente para poder ser interpretado de forma concisa por diversos agentes do usuário, incluindo recursos de tecnologia assistiva (EMAG, 2014).

Há também recomendações, para as quais existem critérios de sucesso, que são pontos específicos a serem atingidos, a classificação de critérios de sucesso são A, AA ou AAA onde A não garante um *site* altamente acessível, AA garante um *site* bastante acessível em termos o *site* será acessível para a maioria dos usuários, enquanto o AAA refere-se a situações bem específicas normalmente refinando um nível AA (EMAG, 2014).

No Brasil temos Modelo de Acessibilidade em Governo Eletrônico (eMAG) como documento de recomendações de acessibilidade, o eMAG foi construído tomando como base o *WCAG* 2.0 e outros documentos internacionais. O processo para desenvolver um *site* acessível é realizado em três passos (EMAG, 2014):

1. Seguir os padrões *web*: Para se criar um ambiente online efetivamente acessível é necessário, primeiramente, que o código esteja dentro dos padrões *web*

internacionais definidos pelo *W3C4*. A conformidade com os padrões *web* permite que qualquer sistema de acesso à informação interprete a mesma adequadamente e da mesma forma, seja por meio de navegadores, leitores de tela, dispositivos móveis (celulares, *tablets*, etc.) ou agentes de *software* (mecanismos de busca ou ferramentas de captura de conteúdo). Páginas que não possuem um código de acordo com os padrões do *W3C* apresentam comportamento imprevisível, e na maioria das vezes impedem ou pelo menos dificultam o acesso (EMAG, 2014);

2. Seguir as diretrizes ou recomendações de acessibilidade: as diretrizes ou recomendações de acessibilidade explicam como tornar o conteúdo *web* acessível a todas as pessoas, destinando-se aos criadores de conteúdo *web* (autores de páginas e criadores de sites) e aos programadores de ferramentas para criação de conteúdo. A principal documentação nessa área é a *WCAG* desenvolvida pelo consórcio *W3C* a partir da criação do *WAI*, contendo as recomendações de acessibilidade para conteúdo *web* (EMAG, 2014);

3. Realizar a avaliação de acessibilidade: após a construção do ambiente *online* de acordo com os padrões *web* e as diretrizes de acessibilidade, é necessário testá-lo para garantir sua acessibilidade. No caso dos padrões *web*, há validadores automáticos. No que diz respeito às diretrizes de acessibilidade, é necessário realizar, inicialmente, uma validação automática, que é realizada por meio de *softwares* ou serviços *online* que ajudam a determinar se um *site* respeitou ou não as recomendações de acessibilidade, gerando um relatório de erros. Uma das ferramentas que podem ser utilizadas é o *ASES*, avaliador e simulador de acessibilidade em *sites* (EMAG, 2014).

2.6 Trabalhos Correlatos

Foi efetuada uma busca para encontrar outras soluções de *software* que possuíssem funcionalidades similares e que utilizem as mesmas ferramentas tecnológicas propostas neste trabalho. Após algumas pesquisas foram encontrados 3 trabalhos que se assemelham a proposta deste trabalho e que fazem uso também do *framework* Ionic, da linguagem angular e da *API* de banco de dados o *FireBase*.

O primeiro trabalho correlato corresponde a um aplicativo híbrido desenvolvido pelo aluno Silvio Soares Santos da Universidade Tecnológica Federal do Paraná que tem como principal função o agendamento de horário em um salão de

beleza. Neste trabalho pode-se extrair um pouco sobre a forma de agendar os horários, porém com uma ideia um pouco diferente da que está sendo apresentada por este trabalho, pois no caso do salão de beleza o usuário cliente ficará encarregado em marcar sua hora e no caso do agendamento para clínicas, o usuário administrador ficara encarregado de agendar o horário (SANTOS, 2016).

Há algumas semelhanças entre eles no caso *login*, cadastro de usuário do aplicativo, cadastro de clientes, listagem de clientes, alteração de clientes e exclusão de clientes, cadastro de serviços, listagem de serviços, alteração de serviços e exclusão de serviços. E como ideia para os próximos trabalhos pode-se pensar em um modo de disponibilizar para que usuários (cliente) marquem suas consultas, e que o usuário (administrador) tenha que apenas confirmar ou não.

A Figura 1 demostra a funcionalidade responsável pela confirmação de um agendamento. Após selecionar o horário do agendamento para o serviço e profissional desejado, o sistema exibe o detalhamento do agendamento. O usuário clica no botão “Confirmar”. Ao clicar em confirmar o sistema valida se o horário não foi agendado por outro cliente e adiciona a reserva na agenda do salão (SANTOS, 2016, p. 31).

No segundo trabalho correlato “Aplicativo móvel multiplataforma para consulta e agendamento de serviços estéticos com geolocalização papum” desenvolvido pela aluna Georgea Neto, pode-se extrair algumas ideias de como fazer uso da *API FireBase* para o agendamento de consultas, além de também deixar ideias futuras como, expandir o número de estéticas e de profissionais que utilizaram o aplicativo e também conter um método de geolocalização, para que os clientes achem uma clínica próxima a eles. Levando em conta que essa aplicação tem a função principal o agendamento e a geolocalização para o usuário, porém não há nada relacionado a prontuário e receituário (NETO, 2020).

Figura 1 - Tela horários disponíveis para agendamento salão de beleza

Fonte: Santos (2016, p. 31)

A Figura 2 está a tela inicial do protótipo do aplicativo, onde é apresentado o componente de *Tabs* que permite a navegação entre as páginas, a partir dele podem ser acessadas as 60 demais funcionalidades como listagem de clientes, profissionais e serviços. No final do mapa são apresentados os cards dos profissionais, com o nome do profissional e qual a sua distância em relação a localização atual do usuário que está acessando o dispositivo. Clicando tanto nos cards quanto nos marcadores que representam os profissionais disponíveis é apresentado os detalhes do mesmo e a listagem dos seus serviços para consulta e realização do agendamento (NETO, 2020, p.60).

Figura 2 - Tela de dashboard

Fontes: Neto (2020, p. 61)

A Figura 3 apresenta a tela de agendamentos e nela são solicitadas a data e horário do mesmo, a exibição dos campos limita o atendimento validando a data e hora atual (NETO, 2020, p. 70).

Figura 3 - Tela de Agendamento

Fonte: Neto (2020, p. 70)

O terceiro trabalho correlato é o “FaleMed”5, um sistema que tem como foco armazenar documentos médicos e melhora a interação entre o profissional da saúde e seu paciente, ele foi desenvolvido pelo aluno Leandro Paiva Andrade. Deste trabalho pôde-se extrair ideias e informações de como montar um Histórico médico, exames e documentos salvos, que no caso deste trabalho foram, ficha médica, receituário e prontuário.

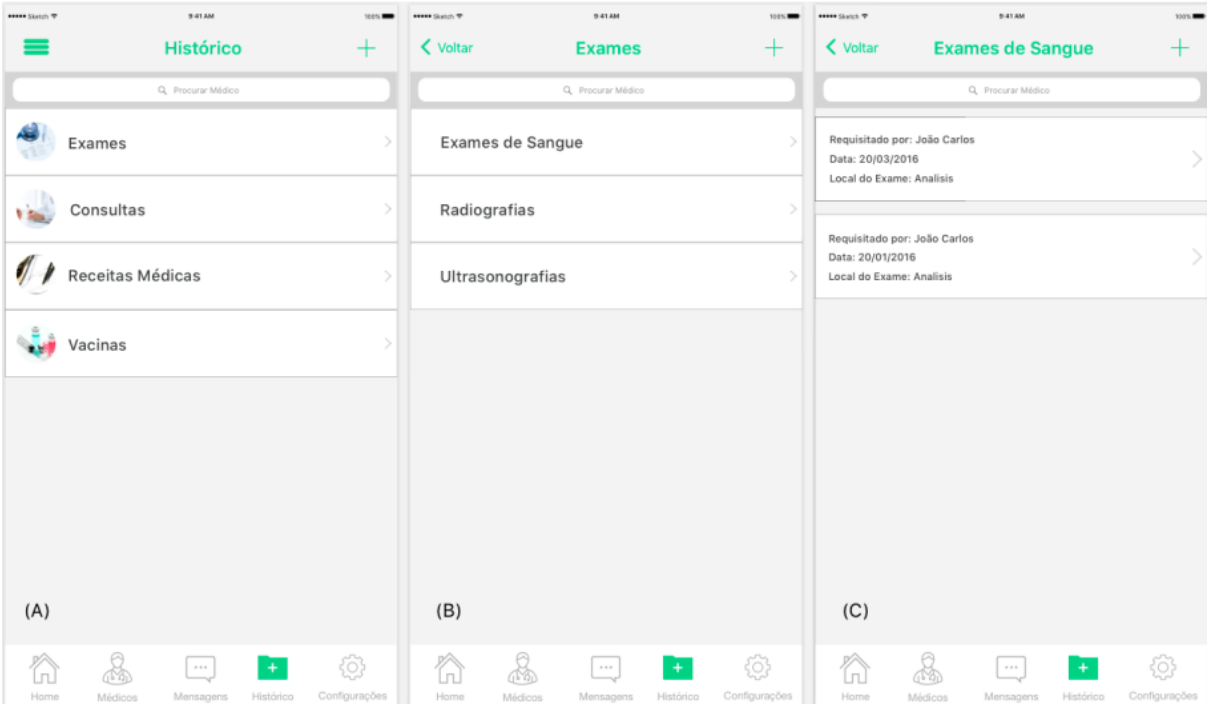
Figura 4 - Tela do Perfil Profissional

Fonte: Andrade (2017, p.41)

A Figura 4 permite acessar o perfil do profissional selecionado, onde deve estar contida informações como locais de atendimento, telefones e *e-mail* para

contato, além da opção de se iniciar um *chat* no próprio aplicativo (ANDRADE, 2017, p. 40).

Figura 5 - Tela de lista do Histórico Médico. (A) Tela de Histórico, (B) Tela de Exames e (C) Tela de Documentos Salvos



Fonte: Andrade (2017, p.43)

A Figura 5 contém as telas (A), (B) e (C), que mostram o histórico médico do paciente. Quando o botão “Histórico” da *tab bar* é acionado, o usuário é direcionado para a tela (A), a qual possui o seu acervo de documentos médicos, que estão subdivididos em categorias: “Exames”, “Consultas”, “Receitas Médicas” e “Vacinas”. Ao selecionar a categoria “Exames”, o paciente e então redirecionado para a tela (B), onde é possível escolher o tipo de exame que se deseja acessar, como, por exemplo, “Exames de Sangue”, “Radiografias”, “Ultrassonografias”, entre outros. Após a escolha do tipo de exame, o usuário terá acesso à tela (C), na qual estão contidos todos os exames do tipo selecionado que foram previamente anexados ao sistema, ordenados cronologicamente. É possível anexar novos documentos médicos ao histórico por meio do botão de adição, localizado no canto superior direito (ANDRADE, 2017, p. 42).

O trabalho é apenas uma aplicação para controle de médico e paciente, e não há forma de agendamento, conforme proposto neste trabalho.

2.7 Considerações do Capítulo

Neste capítulo foram apresentados conceitos do desenvolvimento de um aplicativo híbrido com base no *framework* Ionic e a linguagem Angular e também com uso de uma *API* de banco de dados e recursos necessários para seguir requisitos de acessibilidade *web*, foi de importante contribuição também os trabalhos correlatos de onde pode-se retirar ideias para a criação e evolução do aplicativo.

3 IMPLEMENTAÇÃO

Este capítulo discorre sobre a implementação do aplicativo proposto por este trabalho, então haverá imagens das interfaces criadas e de seus respectivos códigos, estando dividido em 7 seções. A seção 3.1 apresenta a criação da tela de cadastro de serviços, em seguida vem a 3.2 que será efetuado o desenvolvimento da tela de cadastro de cliente, após esta vem a 3.3 que será a tela responsável pela consulta de clientes (pacientes), a seção 3.4 apresentará o desenvolvimento da tela de histórico do paciente, já a 3.5 traz a tela de consulta de serviços e como foi sua criação, a 3.6 mostra uma das telas de anamnese e como elas foram geradas, por último vem a seção 3.7 que mostra a criação da tela de *home*, que também é a tela de agendamento de consultas.

3.1 Tela de Cadastro de Serviços

A interface de cadastro de serviços foi desenvolvida a partir do editor de código fonte *Visual Code* em conjunto com o *Framework* Ionic de onde foi utilizado alguns componentes conforme ilustrados nas figuras 6 e 7, as quais apresentam os códigos desenvolvidos na cadastro-serviços.page.html e que é base para toda a aplicação.

Figura 6 – Código da Tela de Cadastro de Serviços(cabeçalho)

```
<ion-header>
  <ion-toolbar  color="dark">
    <ion-buttons slot="start">
      <ion-back-button color="light" title="">
      </ion-back-button>
    </ion-buttons>
    <ion-title><h2><ion-icon name='hammer-outline'></ion-icon>Cadastro Serviços</h2></ion-title>
  </ion-toolbar>
</ion-header>
```

Fonte: Autoria própria (2021)

O bloco de código apresentado se refere apenas a parte do cabeçalho, utilizando o componente "*ion-header*", faz-se uso do componente "*ion-toolbar*" para a criação de uma barra com a "*color dark*" (cor escura), tem-se também um componente que serve para que a página retorne para a última página navegada antes de se chegar a ela, que seria o "*ion-back-button*" que foi definida a "cor *light*" (clara) para este botão e assim como o "*title*" (título) é vazio, pois deseja-se que apareça o nome padrão, no caso "*back*" (voltar). Em seguida é utilizado o componente "*ion-title*" para criar um título em destaque para a tela, no caso Cadastro Serviços.

Figura 7 – Código da Tela de Cadastro de Serviços (Corpo)

```
<ion-content>
  <ion-card  class="cardconteudo">
    <ion-card-header>
      <ion-card-title> <H1><ion-icon name="information-circle-outline"></ion-icon>Informações do Serviço</H1></ion-card-title>
    </ion-card-header>

    <ion-item color="clear">
      <ion-label  position="floating"><h2>Nome do Serviço</h2> </ion-label>
        <ion-input type="text"[(ngModel)]="servico.nome"></ion-input>
    </ion-item>

    <ion-item color="clear">
      <ion-label position="floating"><h2>Preço Unitario</h2></ion-label>
      <ion-input type="text"[(ngModel)]="servico.preco"></ion-input>
    </ion-item>

  <ion-item color="clear">
      <ion-label><h2>Duração de Sessão</h2></ion-label>
      <ion-select  [(ngModel)]="servico.duracao" value="brown" okText="confirmar" cancelText="cancelar">
        <ion-select-option value="15">15 minutos</ion-select-option>
        <ion-select-option value="30">30 minutos</ion-select-option>
        <ion-select-option value="60">1 hora</ion-select-option>
        <ion-select-option value="90">1 hora e 30 minutos</ion-select-option>
      </ion-select>
    </ion-item>

    <ion-item color="clear">
        <ion-label position="floating"><h2>Descrição</h2></ion-label>
        <ion-textarea type="text"[(ngModel)]="servico.descricao"></ion-textarea>
    </ion-item>

  </ion-card>
    <ion-button slot="" size="large" expand="block" color="danger" (click)="saveServico()"><h1>Salvar</h1></ion-button>
</ion-content>
```

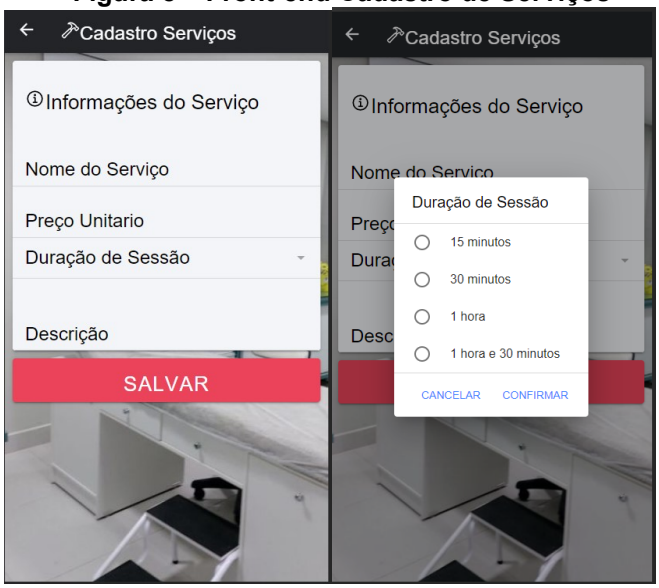
Fonte: Autoria própria (2021)

Este código se refere a parte do corpo, utilizando o componente "*ion-content*" e "*ion-card*", faz-se uso do componente "*ion-header*" novamente para criar um "*ion-card-title*" que apresenta o seguinte título "Informações do serviço", ainda dentro do "*ion- card*" tem a criação de "*ion-itens*" destinado a criação de cada item que existirá no formulário, junto deles há um "*ion-label*", destinado a colocar os nomes dos campos para que o usuário possa identificar, adicionando a esse componente a variável *position* = "*floating*" (posição= "flutuar"), que nesse caso gera uma animação. Seguindo tem um "*ion-input*", que será responsável por receber o tipo das informações preenchidas e no caso do item que terá um "*combo-box*" é utilizado um "*ion-select*" e dentro dele é criado vários "*ion-select-options*", de acordo com a necessidade do usuário. No caso da descrição ao invés de um "*ion-input*" é utilizado um "*ion-textarea*", que é um componente mais apropriado para quando se tem um campo que receberá uma grande quantidade de informação, tem-se também um componente para criação de botões o "*ion-button*" que no caso é utilizado para a criação do botão salvar.

Além do código *.html* é utilizado também o *.css* para a utilização de um plano de fundo disponibilizado pelo próprio cliente. Por fim se tem o resultado que é a

interface do projeto conforme ilustrado a Figura 8, que também foi moldado buscando atender padrões de web acessibilidade, como por exemplo tamanho de letra para pessoas com problemas de visão, cores fáceis de se distinguir, junto de ícones, para no caso o usuário possua algum tipo de daltonismo.

Figura 8 – Front-end Cadastro de Serviços



Fonte: Autoria própria (2021)

Após a criação desta classe deve-se criar uma classe Interface para os serviços como mostra a Figura 9:

Figura 9 – Interface Serviços

```
export interface Servicos {
    id?:string;
    nome?:string;
    descricao?: string;
    picture?:string;
    preco?:string;
    duracao?:string;
    createdAt?:number;
    userId?:string; }
```

Fonte: Autoria própria (2021)

Esta *interface* serve para criar as variáveis que serão usadas na classe de cadastro-de-servicos.ts (Figura11) e serviram como variáveis que serão armazenados no *FirebaseStorage* para receber as informações digitadas pelo o usuário.

Figura 10 – Servico.service

```
import { Injectable } from '@angular/core';
import { AngularFirestore, AngularFirestoreCollection, AngularFirestoreCollectionGroup } from '@angular/fire/firestore';
import { Servicos } from '../interfaces/servicos';
import { map } from 'rxjs/operators'
@Injectable({
  providedIn: 'root'
})
export class ServicoService {
  private servicosColletction: AngularFirestoreCollection<Servicos>;

  constructor(private afs: AngularFirestore) {
    this.servicosColletction = this.afs.collection<Servicos>('Servicos');

  }
  getServicos(){
    return this.servicosColletction.snapshotChanges().pipe(
      map(actions =>actions.map(a=>{
          const data = a.payload.doc.data();
          const id = a.payload.doc.id;

          return {id, ...data};
        }))
    );
  }

  addServico(servico:Servicos){
    return this.servicosColletction.add(servico);
  }

  getServico(id:string){
    return this.servicosColletction.doc<Servicos>(id).valueChanges();
  }

  updateServico(id:string, servico:Servicos){
    return this.servicosColletction.doc<Servicos>(id).update(servico);
  }

  deleteServico(id:string){
    return this.servicosColletction.doc<Servicos>(id).delete();
  }
}
```

Fonte: Autoria própria (2021)

Na classe servico.services (Figura 10) são apresentadas as funções responsáveis pela consulta de serviços disponíveis pela clínica. Esta classe é inicializada com uma variável *servicosCollecttion* que recebe uma função pronta do Angular *FireStore*, no caso o *AngularFirestoreCollection* e é passado junto a ela o parâmetro <Servicos> que se refere a interface servicos.ts. Logo abaixo tem-se uma situação semelhante, mas dessa vez dentro do método construtor e é passado *this.afs.Collection*<Servicos>('Servicos'), com isto uma *Collection* no banco *FireStore* é criada com o nome de Servicos.

Dentro da classe ServicoService tem-se as funções que são usadas na consulta de um ou todos os itens, neste caso as funções *get*Servicos() e *ge*tServico(*id:string*), o primeiro caso não recebe parâmetros de entrada pois é feita uma busca completa trazendo todos os ids que existem dentro desta *Collection*, no

segundo caso, como é uma busca específica é passado o id do item para que seja feita a consulta no banco.

Na função addServico(servico:Servico) é passado uma variável que recebe a interface Servicos assim adicionando um item, porem na função *update*Servico(*id:string*, servico:Servicos) para fazer a alteração é necessário que seja informado o id e o tipo que é enviado para efetuar a alteração das informações de um item e pôr fim a exclusão de um item por meio da função *delete*Servico(*id:string*) onde é passado o id do item para que seja efetuada a exclusão dele no banco. Com estas classes finalizadas pode-se partir para a classe cadastro-de-serviços.ts que é a última classe para o funcionamento da tela de cadastro de serviços.

Figura 11 – Cadastro.servicos.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { LoadingController, NavController, ToastController } from '@ionic/angular';
import { Subscription } from 'rxjs';
import { Servicos } from 'src/app/interfaces/servicos';
import { AuthService } from 'src/app/services/auth.service';
import { ServicoService } from 'src/app/services/servico.service';
@Component({
  selector: 'app-cadastro-servicos',
  templateUrl: './cadastro-servicos.page.html',
  styleUrls: ['./cadastro-servicos.page.scss'],
})
export class CadastroServicosPage implements OnInit {
  public servico: Servicos = {};
  private loading: any;
  private servicoId: string = null;
  private servicoSubscription: Subscription;


  constructor(
    private loadingCtrl: LoadingController,
    private toastCtrl: ToastController,
    private authService: AuthService,
    private activeRoute: ActivatedRoute,
    private servicoService: ServicoService,
    private navCtrl: NavController
    ) {

      this.servicoId = this.activeRoute.snapshot.params.id;
      if (this.servicoId) {this.loadServico();}
    }

  ngOnInit() {
  }
  loadServico(){

    this.servicoSubscription = this.servicoService.getServico(this.servicoId).subscribe(data =>{
      this.servico = data;
    });
  }
```

```
async saveServico(){
  await this.presentLoading();

  this.servico.userId = this.authService.getAuth().currentUser.uid;

  if(this.servicoId){
    try{
      await this.servicoService.updateServico(this.servicoId,this.servico);
      await this.loading.dismiss();

      this.navCtrl.navigateBack('/consultar-servicos');
    }catch(error){
      this.presentToast('Erro ao tentar salvar');
      this.loading.dismiss();
    }
  }else{
    this.servico.createdAt = new Date().getTime();

    try{
      await this.servicoService.addServico(this.servico);
      await this.loading.dismiss();

      this.navCtrl.navigateBack('/home');
    }catch(error){
      this.presentToast('Erro ao tentar salvar');
      this.loading.dismiss();
    }
  }

}

async presentLoading() {
  this.loading = await this.loadingCtrl.create({message: 'Por favor aguarde'});
  return this.loading.present();
}

async presentToast(message: string) {
  const toast = await this.toastCtrl.create({message,duration: 2000});
  toast.present();
}

}
```

Fonte: Autoria própria (2021)

A classe de cadastro-de-servico.ts é a classe responsável pelas operações na página de salvar e ler, também é responsável pelo controle de navegação e por passar o id do produto que será alterado em outra página. Esta classe precisa de alguns *imports*(importar), que é uma declaração onde indica que é feito o uso de funções, variáveis ou classes externas a este arquivo, dessa forma elas foram declaradas em outro arquivo e são disponibilizadas a partir da importação desse conteúdo no código. Logo após é apresentada a declaração de algumas variáveis que são utilizadas no código e outras que são declaradas dentro do método construtor, responsável por instanciar o objeto em memória a partir da classe que foi definida, por fim são apresentadas as funções que tem como propósito de promover algumas ações do código, algumas delas vem acompanhada da palavra *async* (assíncrona), funções que

são declaradas com o *async* antes tem o significado que podem executar assincronamente, ou seja, quem chamou não precisa esperar por sua execução e ela pode continuar normalmente sem bloquear a aplicação, assim quando o método chamado termina ele pode voltar para o ponto que foi chamado e dar continuidade ao que estava fazendo.

3.2 Tela de Cadastro de Clientes

A *interface* de cadastro de clientes foi desenvolvida a partir do editor de código fonte *Visual Code* em conjunto com o *Framework* Ionic de onde foi utilizado alguns componentes conforme ilustrados nas Figuras 12, 13 e 14, as quais apresentam os códigos desenvolvidos na cadastro-cliente.page.html.

Figura 12 – Cadastro.cliente.page.html(cabeçalho)

```
<ion-header>
  <ion-toolbar color="dark">
    <ion-buttons slot="start">
      <ion-back-button color="light" title="">
      </ion-back-button>
    </ion-buttons>

    <ion-title><ion-icon name='person-add'></ion-icon> Cadastro Cliente</ion-title>
  </ion-toolbar>
</ion-header>
```

Fonte: Autoria própria (2021)

Este cabeçalho do cadastro de cliente segue a mesma ideia do cabeçalho do cadastro de serviços. Já o corpo do cadastro de cliente, apresenta alguns componentes novos como, por exemplo, o componente *ion-datetime*, este serve para a criação de um componente que receberá as variáveis de dia, mês e ano. Outro componente é o *ion-select*, que gera uma tela de escolhas, nesse caso serve para escolher o gênero da pessoa.

Figura 13 – Cadastro.cliente.page.html(corpo)

```
<ion-content>
  <ion-card class="cardconteudo">

    <ion-card-header>
      <ion-card-title> <H1><ion-icon name="person-circle-outline"></ion-icon> Dados do Usuário</H1></ion-card-title>
    </ion-card-header>

      <ion-item color="clear">
        <ion-label position="floating"><h2>Nome</h2></ion-label>
        <ion-input type="text"[(ngModel)]="cliente.nome" ></ion-input>
      </ion-item>

      <ion-item color="clear">
        <ion-label position="floating"><h2>Email</h2></ion-label>
        <ion-input type="email"[(ngModel)]="cliente.email"></ion-input>
      </ion-item>

      <ion-item color="clear">
        <ion-label position="floating"><h2>CPF</h2></ion-label>
        <ion-input type="number"[(ngModel)]="cliente.cpf"></ion-input>
      </ion-item>

      <ion-item color="clear">
        <ion-label >Data de Nascimento</ion-label>
        <ion-datetime displayFormat="DD/MM/YYYY"  value="2002-09-23T15:03:46.789" [(ngModel)]="cliente.dataNascimento"></ion-datetime>
      </ion-item>

      <ion-item color="clear">
        <ion-label><h2>Sexo</h2></ion-label>
      <ion-select  [(ngModel)]="cliente.sexo" value="brown" okText="confirmar" cancelText="cancelar">
        <ion-select-option value="M">Masculino</ion-select-option>
        <ion-select-option value="F">Feminino</ion-select-option>
      </ion-select>
      </ion-item>

      <ion-item color="clear">
        <ion-label position="floating"><h2>Telefone</h2></ion-label>
        <ion-input type="tel"[(ngModel)]="cliente.telefone"></ion-input>
      </ion-item>

  </ion-card>
```

Fonte: Autoria própria (2021)

Figura 14 – Cadastro.cliente.page.html(corpo)

```
    <ion-item color="clear">
      <ion-label position="floating"><h2>Numero</h2></ion-label>
      <ion-input type="number"[(ngModel)]="cliente.numero"></ion-input>
    </ion-item>
  </ion-card-content>

  <ion-card-content>
    <ion-item color="clear">
      <ion-label position="floating"><h2>Complemento</h2></ion-label>
      <ion-input type="text"[(ngModel)]="cliente.complemento"></ion-input>
    </ion-item>
  </ion-card-content>

  <ion-card-content>
    <ion-item color="clear">
      <ion-label position="floating"><h2>Bairro</h2></ion-label>
      <ion-input type="text"[(ngModel)]="cliente.bairro"></ion-input>
    </ion-item>
  </ion-card-content>

  <ion-card-content>
    <ion-item color="clear">
      <ion-label position="floating"><h2>CEP</h2></ion-label>
      <ion-input type="number"[(ngModel)]="cliente.cep"></ion-input>
    </ion-item>
  </ion-card-content>

  <ion-card-content>
    <ion-item color="clear">
      <ion-label position="floating"><h2>Estado</h2></ion-label>
      <ion-input type="text"[(ngModel)]="cliente.estado"></ion-input>
    </ion-item>
  </ion-card-content>

  <ion-card-content>
    <ion-item color="clear">
      <ion-label position="floating"><h2>Cidade</h2></ion-label>
      <ion-input type="text"[(ngModel)]="cliente.cidade"></ion-input>
    </ion-item>
  </ion-card-content>

</ion-card>
<ion-button slot="" size="large" expand="block" color="danger" (click)="saveCliente()" ><h1>Salvar</h1></ion-button>
</ion-content>
```

Fonte: Autoria própria (2021)

As demais classes referentes ao cadastro de clientes, seguem a mesma ideia das classes do cadastro de serviço, modificando apenas sua *interface* como ilustra a Figura 15:

Figura 15 – Interface Cliente

```
export interface Cliente {
   id?:string;
   nome?:string;
   email?: string;
   cpf?:number;
   dataNascimento?:number;
   sexo?:string;
   telefone?:number;

   endereco?:string;
   numero?:number;
   complemento?:string;
   bairro?:string;
   cep?:number;
   estado?:string;
   cidade?:string;
   idade?:number;
   racaetnia?:string;
   profissao?:string;
   procedencia?:string;
   habitos?:string;
   descricao?:string;
   remedios?:string;
   descricaoTratamento?:string;

   createdAt?:number;
   userId?:string;
}
```

Fonte: Autoria própria (2021)

Esta *interface* é responsável por criar os atributos referentes as informações do cliente. Como pode-se observar há mais atributos do que a classe cadastro de clientes utiliza, pois além do cadastro simples do cliente ela possui também o cadastro de anamnese e tratamento, porém, só são mostradas quando for apresentada a tela de consulta de clientes. A Figura 17 ilustra como ficou a tela do cadastro de clientes. Em conjunto com esta classe Cliente, se tem a *interface extends* “Consultas”, responsável por armazenar os dados da consulta do cliente. A *interface* Cliente é composta por alguns atributos como mostra a Figura 16.

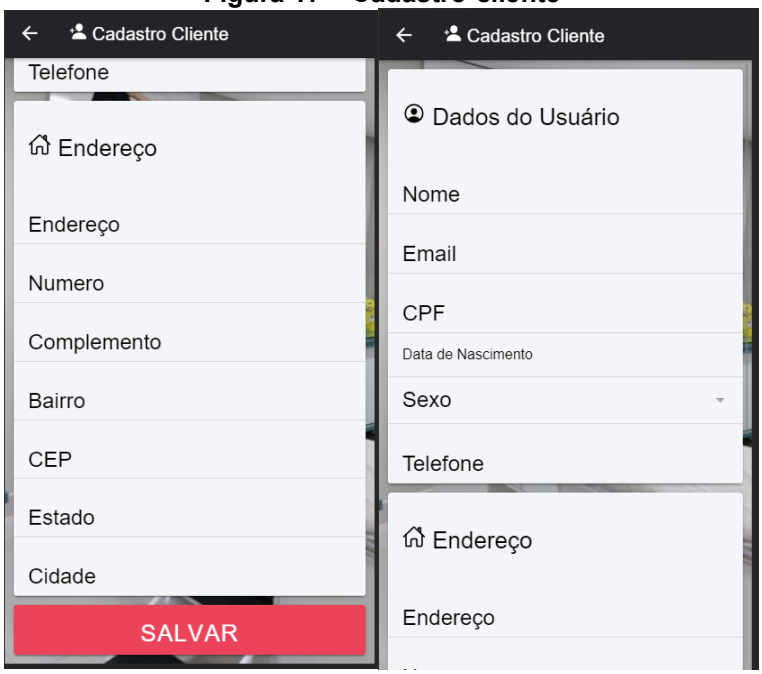
Figura 16 – Interface Consultas

```
export interface Consultas {
   id?: string;
   receituario?: string;
   prontuario?: string;
   createdAt?: number;
   userId?: string;
}
```

Fonte: Autoria própria (2021)

Esta *interface* contém os atributos como id, receituário, prontuário, *createdAt* e *userId*, que serão incluídas na classe cliente.

Figura 17 – Cadastro-cliente



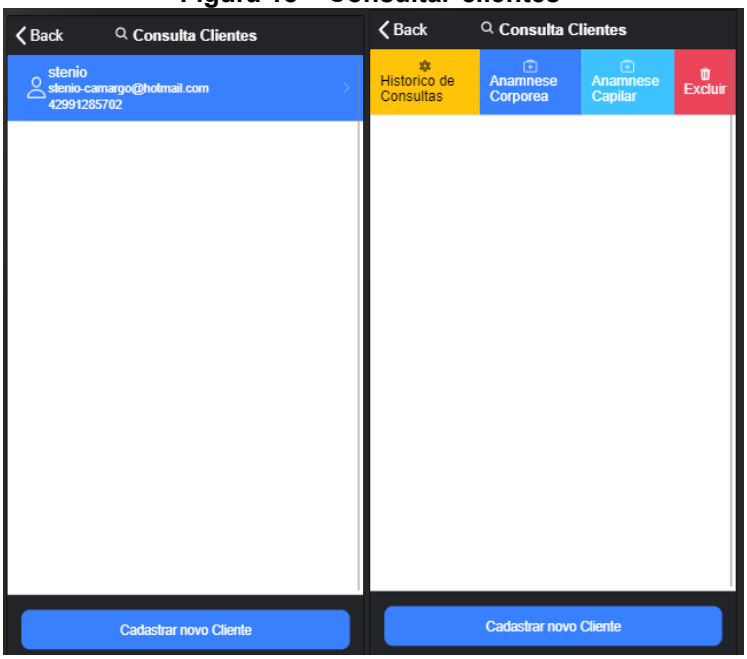
Fonte: Autoria própria (2021)

Esta tela busca a maior simplicidade e praticidade para que o usuário possa efetuar o cadastro do cliente, além de evitar o uso de uma grande paleta de cores para facilitar a compreensão de pessoas com dificuldades visuais (no caso daltonismo). O tamanho das letras são outro ponto interessante a se ressaltar, pois facilita sua visualização para a maioria das pessoas, o botão de salvar encontra-se no final da tela em uma cor vermelha, para dar destaque.

3.3 Tela de Consulta de Clientes (Pacientes)

Esta tela de consulta tem como objetivo apresentar todos os pacientes cadastrados pelo usuário, para que possa se fazer alterações ou acessar outras propriedades do paciente, como a anamnese corpórea, capilar e histórico de consultas, além de poder fazer a inclusão de um novo paciente ou a exclusão de um paciente já existente, conforme ilustrado na Figura 18.

Figura 18 – Consultar-clientes



Fonte: Autoria própria (2021)

Para a criação dessa tela Figura 19 e 20 foram utilizados alguns elementos já apresentados em outras *interfaces*, como por exemplo, “*ion-Button*”, “*ion-content*”, “*ion-footer*”, “*ion-title*”, “*ion-icon*”. As novidades nessa tela são os componentes “i*on-sliding*", utilizados para que cada item que existir na lista tenha uma função de deslizar da direita para a esquerda, podendo assim apresentar duas novas funções do aplicativo, o botão para tela de anamnese, histórico de consultas e o botão para a exclusão do item, possui também o componente “*ion-list*” que serve para que seja criada a lista com todos os pacientes, junto de um “*ion-animantion*” para dar um pouco de vida a esta lista de pacientes.

Dentro dos “*ion-label*” é incluído {Clientes.nome}, que serve para que apareça no item o nome do cliente encontrado em cada *id*. Também há a opção de clicar em um paciente e ser redirecionado por meio de um “*routerLink*” para a tela de cadastro de clientes, no qual será carregada as informações desses pacientes por meio do cliente.id passado pela tela de consulta através da navegação das páginas, com isto pode ser feito um *update* das informações deste paciente caso necessário.

Figura 19 – Consultar-clientes.page.html(cabeçalho)

```
<ion-header>
  <ion-toolbar color="dark" >
    <ion-buttons color="light" slot="start">
      <ion-back-button title="">
      </ion-back-button>
    </ion-buttons>
    <ion-title> <ion-icon name='search'></ion-icon> Consulta Clientes</ion-title>
  </ion-toolbar>
</ion-header>
```

Fonte: Autoria própria (2021)

Figura 20 – Consultar-clientes.page.html (corpo)

```
<ion-content>
  <ion-list *ngIf="!cliente.length">
    <ion-item *ngFor="let item of [0, 1, 2]">
      <ion-avatar slot="start">
        <ion-skeleton-text animated></ion-skeleton-text>
      </ion-avatar>
      <ion-label>
        <h3>
          <ion-skeleton-text animated style="width: 50%"></ion-skeleton-text>
        </h3>
        <p>
          <ion-skeleton-text animated style="width: 80%"></ion-skeleton-text>
        </p>
        <p>
          <ion-skeleton-text animated style="width: 30%"></ion-skeleton-text>
        </p>
      </ion-label>
    </ion-item>
  </ion-list>
  <ion-list >
    <ion-item-sliding  *ngFor="let clientes of cliente" >
      <ion-item color="primary" button [routerLink]="['/cadastro-cliente', clientes.id]">
        <ion-icon name="person-outline">  </ion-icon>
        <ion-label>
            {{ clientes.nome }}
          <p>{{ clientes.email }}</p>
          <p>{{ clientes.telefone}} </p>
        </ion-label>
      </ion-item>
      <ion-item-options side="end">
        <ion-item-option color="warning" [routerLink]="['/historico-paciente', clientes.id]">
          <ion-icon slot="top" name="medical-outline"></ion-icon>
          Historico de Consultas
        </ion-item-option>
        <ion-item-option color="primary" [routerLink]="['/paciente', clientes.id]">
          <ion-icon slot="top" name="medkit-outline"></ion-icon>
          Anamnese Corporea
        </ion-item-option>
        <ion-item-option color="secondary" [routerLink]="['/anamnese-capilar', clientes.id]">
          <ion-icon slot="top" name="medkit-outline"></ion-icon>
          Anamnese Capilar
        </ion-item-option>
        <ion-item-option color="danger" (click)="deleteCliente(clientes.id)">
          <ion-icon slot="top" name="trash"></ion-icon>
```

```
            Excluir
          </ion-item-option>
        </ion-item-options>
      </ion-item-sliding>
    </ion-list>
</ion-content>
<ion-footer>
  <ion-toolbar color="dark">
    <ion-grid>
      <ion-row >
        <ion-col size="12">
          <ion-button routerLink="/cadastro-cliente" expand="block">
            Cadastrar novo Cliente
          </ion-button>
        </ion-col>
      </ion-row>
    </ion-grid>
  </ion-toolbar>
</ion-footer>
```

Fonte: Autoria própria (2021)

Esta tela é muito semelhante as próximas telas que serão apresentadas de histórico do paciente e consulta de serviço, a principal diferença entre elas é que cada uma tem uma classe *service* para que seja feita a busca dos dados de acordo com o que for necessário Figura 21.

Figura 21 – Clientes.service

```
import { Consultas } from './../interfaces/consultas';
import { Cliente } from 'src/app/interfaces/cliente';
import { Injectable } from '@angular/core';
import { AngularFirestore, AngularFirestoreCollection, AngularFirestoreCollectionGroup } from '@angular/fire/firestore';
import { map } from 'rxjs/operators';


@Injectable({
  providedIn: 'root'
})
export class ClienteService {

  private clienteColletction: AngularFirestoreCollection<Cliente>;
  private consultasColletction: AngularFirestoreCollection<Consultas>;

  constructor(private afs: AngularFirestore) {
    this.clienteColletction = this.afs.collection<Cliente>('Clientes');
    this.consultasColletction = this.afs.collection<Cliente>('Consultas');

  }
  getClientes(){
    return this.clienteColletction.snapshotChanges().pipe(
      map(actions =>actions.map(a=>{
          const data = a.payload.doc.data();
          const id = a.payload.doc.id;

          return {id, ...data};
        }))
    );
  }
  getConsultas(ids: string){
    return this.clienteColletction.doc('Clientes/'+ids+'/Consultas').snapshotChanges().pipe(
      map(actions =>map(a=>{
          const data = actions.payload.data;
          const id = actions.payload.id;

          return {id, ...data};
        }))
    );
  }

  addCliente(cliente: Cliente){
    return this.clienteColletction.add(cliente);
  }

  getConsulta(id: string){
    return this.afs.collection('Clientes/'+id+'/Consultas').doc<Cliente>(id).valueChanges();
  }

  getCliente(id: string){
    return this.clienteColletction.doc<Cliente>(id).valueChanges();
  }

  updateCliente(id: string, cliente: Cliente){
    return this.clienteColletction.doc<Cliente>(id).update(cliente);
  }
  addConsulta(id: string, consulta: Consultas){
    return this.afs.collection('Clientes/'+id+'/Consultas').add(consulta);
  }

  deleteCliente(id: string){
    return this.clienteColletction.doc<Cliente>(id).delete();
  }
}
```

Fonte: Autoria própria (2021)

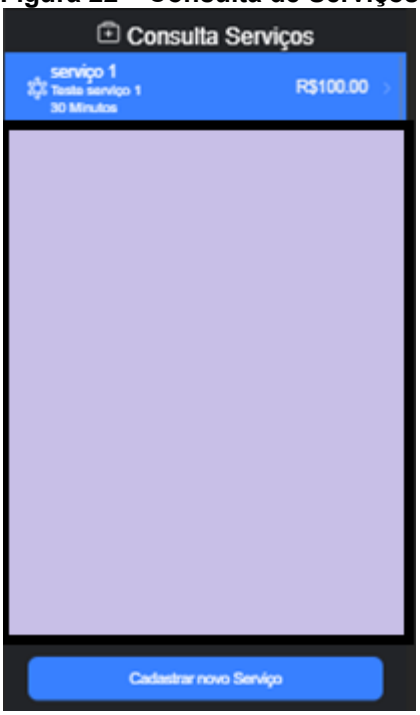
3.4 Tela de Histórico de Pacientes

A tela de histórico do paciente é uma tela onde pode ser encontrado todas as consultas já feitas pelo paciente, ela é uma extensão da tela de consulta de clientes. Nela pode ser consultada e alterada as consultas já feitas pelo paciente, como também disponibiliza um botão de criar uma nova consulta.

3.5 Tela de Consulta de Serviços

A tela de consulta de serviços, apresentada na Figura 22 serve para que o usuário possa alterar ou remover algum serviço depois de cadastrado. Esta tela segue as mesmas configurações da tela de consulta de cliente e histórico de cliente, levando em consideração que a única e principal mudança é que ela tem o seu próprio ".service "para executar as operações *CRUD* (*Create*, *Read, Update, Delete*). Sua finalidade é apresentar os serviços disponibilizados pela clínica.

Figura 22 – Consulta de Serviços



Fonte: Autoria própria (2021)

3.6 Tela de Anamnese

A tela ilustrada na Figura 23 tem como função cadastrar as informações de anamnese do paciente. Ela é dividida em anamnese capilar e anamnese corpórea, que são *extends* da *interface* cliente. Esta tela está associada à tela de consulta de clientes, e quando acessada recebe o cliente.*id* do paciente selecionado, com isto é feito um *update* dos dados que faltavam do mesmo, todos relacionados com a ficha médica. A imagem apresentada na Figura 23 é relacionada a anamnese capilar.

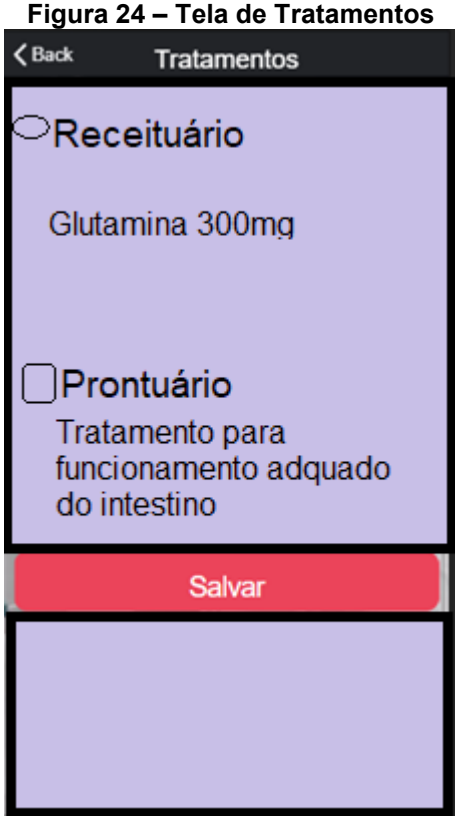
Figura 23 – Tela de Anamnese

Fonte: Autoria própria (2021)

3.7 Tela de Tratamentos (Receituário e Prontuário)

Nesta tela o usuário incluirá junto aos dados armazenados do seu usuário, informações da consulta, com o intuito de ter um histórico salvo de todos os tratamentos de seu paciente, a tela usará elementos já vistos anteriormente como nas telas de cadastro, a grande diferença desta vez é que em seu desenvolvimento através do tela-de-tratamentos.ts terá de guardar o valor do *id* de seu pai e repassa-lo junto ao nome da *subcollection* e o novo id de seu filho para que a gravação dos dados ocorra de forma correta.

Figura 24 – Tela de Tratamentos

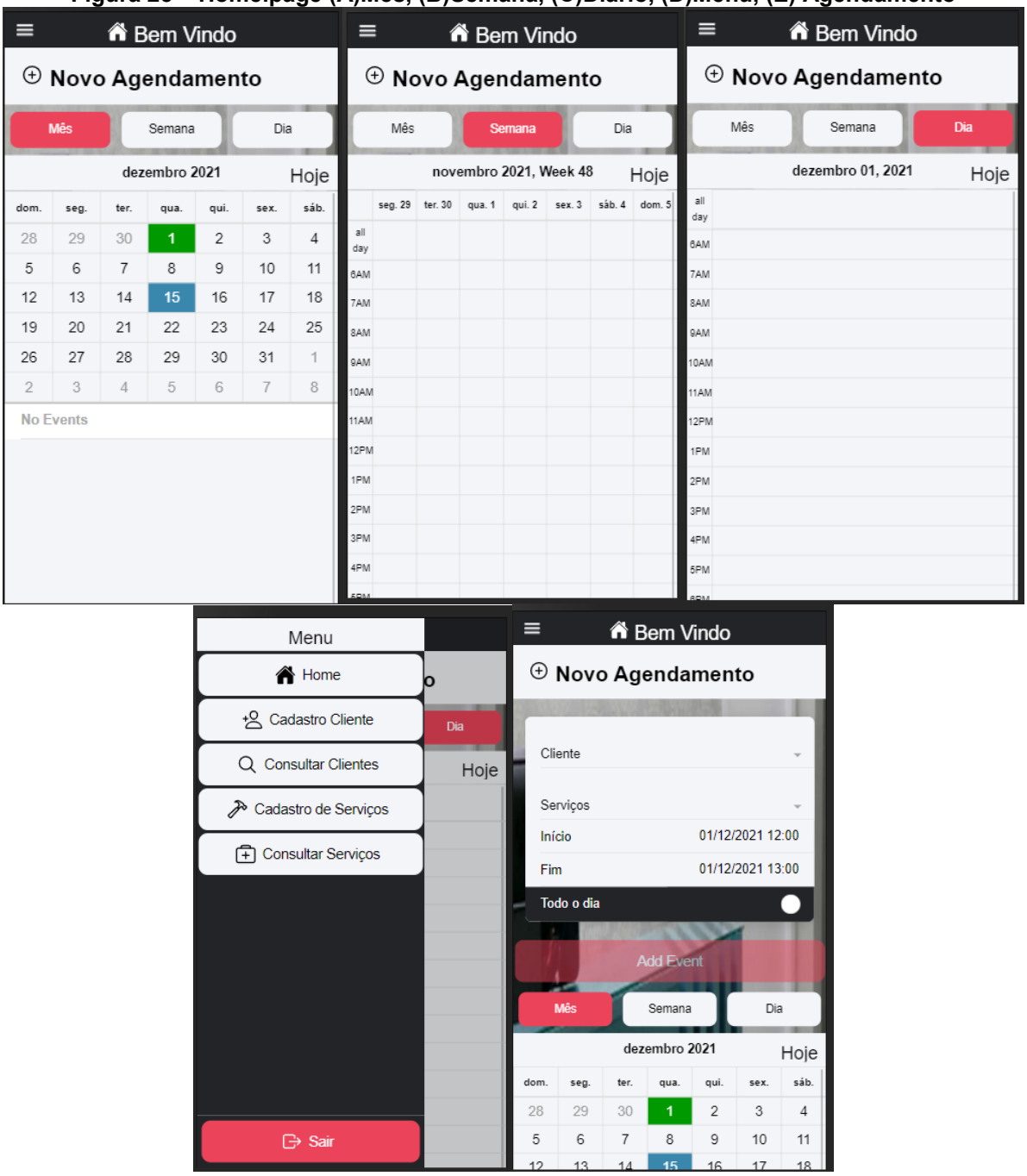


Fonte: Autoria própria (2021)

3.8 Tela Home

Nesta tela é apresentado ao usuário um calendário e uma área onde é possível ver suas consultas agendadas em dia, mês e ano, nesta tela também o usuário pode agendar os tratamentos.

Figura 25 – Home.page (A)Mês, (B)Semana, (C)Diário, (D)Menu, (E) Agendamento



Fonte: Autoria própria (2021)

A tela de home possui alguns componentes já apresentados e outros novos como é mostrado na Figura 25. Nela pode-se observar a presença de um "*Ion-menu-toggle*", que é usado junto de um "i*on-button*" e um "*ion-icon*", para a criação de um *Side Menu* (Figura25), permitindo navegar por todas as opções geradas nesse menu. Outro componente novo implementado nessa tela é o componente de calendário, nele é inserido algumas variáveis, como tempo de início e fim de expediente, início do dia

da semana, a função *onTimeSelected* que tem como propósito pegar a hora selecionada na agenda e atualizar a duração do agendamento.

Figura 26 – Home.page.html (Calendário)

```
  <calendar
[eventSource]="eventSource"
[calendarMode]="calendar.mode"
[currentDate]="calendar.currentDate"
(onEventSelected)="onEventSelected($event)"
(onTitleChanged)="onViewTitleChanged($event)"
(onTimeSelected)="onTimeSelected($event)"
startHour="6"
endHour="20"
step="30"
startingDayWeek="1">
>
</calendar>
```

Fonte: Autoria própria (2021)

Complementando o calendário há três botões, que quando um deles é selecionado *calendarMode* recebe um novo valor e altera o estilo do calendário e o botão muda de cor para representar que seu estado está ativo, no caso são três opções, diário, semana e mês (Figura 27).

Figura 27 – Home.page.html (Botões)

```
<ion-row color="light">

  <ion-col size="4">
    <ion-button expand="block" [color]="calendar.mode === 'month' ? 'danger' : 'light'" (click)="changeMode('month')">Mês</ion-button>
  </ion-col>
  <ion-col size="4">
    <ion-button expand="block" [color]="calendar.mode === 'week' ? 'danger' : 'light'" (click)="changeMode('week')">Semana</ion-button>
  </ion-col>
  <ion-col size="4">
    <ion-button expand="block" [color]="calendar.mode === 'day' ? 'danger' : 'light'" (click)="changeMode('day')">Dia</ion-button>
  </ion-col>

</ion-row>
```

Fonte: Autoria própria (2021)

Na área para agendamento (Figura 28) dos tratamentos disponíveis na clínica tem-se um <*ion-selected*>, porém este componente fará uso de itens que no caso não serão estáticos, os itens a serem selecionados serão trazidos do banco de dados *FireBase*, dentro do componente <*ion-select*> haverá um *ngModel*, é uma diretiva embutida que cria uma instância *FormControl* do modelo de domínio e o liga a um elemento de controle de formulário, com isto o usuário terá acesso a todos os clientes

e serviços para que possa fazer a escolha e adicionar ao seu agendamento. Uma vez os campos preenchidos, o <*ion-button*> responsável por salvar as informações mudará de uma cor opaca para uma cor mais viva, significando que está disponível, para que possa ser selecionado e sua ação ser realizada.

Figura 28 – Home.page.html (Agendamento)

```
<ion-content scrollY="true" >
  <ion-card-header color="light" tappable (click)="collapseCard = !collapseCard">
    <ion-card-title> <ion-icon name="add-circle-outline"></ion-icon> Novo Agendamento</ion-card-title>
  </ion-card-header>
  <ion-card color="light"  *ngIf="collapseCard">
        <ion-list>
          <ion-item class="input-item" color="light">
            <ion-label position="floating">Cliente</ion-label>
            <ion-select interface="popover" [(ngModel)]="event.title" (ionChange)="onClienteChange(clientSelectorForm.value)">
              <ion-select-option *ngFor="let cliente of clientes | async" [value]="cliente.id">
                {{cliente.nome}}
              </ion-select-option>
            </ion-select>
          </ion-item>
        </ion-list>

        <ion-list>
          <ion-item class="input-item" color="light">
            <ion-label position="floating">Serviços</ion-label>
            <ion-select interface="popover" [(ngModel)]="event.desc" (ionChange)="onServicoChange(servicoSelectorForm.value)">
              <ion-select-option *ngFor="let servico of servicos | async" [value]="servico.id">
                {{servico.nome}}
              </ion-select-option>
            </ion-select>
          </ion-item>
        </ion-list>

  <ion-item color="light">
    <ion-label>Início</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY HH:mm" pickerFormat="D MMM:HH:mm" [(ngModel)]="event.startTime" [min]="minDate"></ion-datetime>
  </ion-item>

  <ion-item color="light">
    <ion-label>Fim</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY HH:mm" pickerFormat="D MMM:HH:mm" [(ngModel)]="event.endTime" [min]="minDate"></ion-datetime>
  </ion-item>

  <ion-item color="dark">
    <ion-label>Todo o dia</ion-label>
    <ion-checkbox [(ngModel)]="event.diaTodo"></ion-checkbox>
  </ion-item>

  </ion-card>
  <ion-button  *ngIf="collapseCard" color="danger" size="large" expand="block" (click)="addEvent()" [disabled]="event.title === ''">Add Event</ion-button>
```

Fonte: Autoria própria (2021)

Após a apresentação deste código HTML pode-se compreender o .ts que será apresentado na Figura 29, para finalizar como funciona esta tela principal.

Figura 29 – Home.page.ts (Cabeçalho)

```
import { formatDate } from '@angular/common';
import { Component, Inject, LOCALE_ID, OnInit, ViewChild } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';
import { AlertController, ModalController } from '@ionic/angular';
import { CalendarComponent } from 'ionic2-calendar';
import { Observable } from 'rxjs';
import { Cliente } from 'src/app/interfaces/cliente';
import { AngularFirestore, AngularFirestoreCollection} from '@angular/fire/firestore';
import { Servicos } from 'src/app/interfaces/servicos';
import { Storage } from '@ionic/storage';
//import { storage } from 'firebase';


@Component({
  selector: 'app-home',
  templateUrl: './home.page.html',
  styleUrls: ['./home.page.scss'],
})
export class HomePage implements OnInit {
  @ViewChild(CalendarComponent) myCal: CalendarComponent;
  clientSelectorForm: FormGroup;
  selectedCliente: stri
  clientes: Observable<
  servicos: Observable<Servicos[]>;
  servicoSelectorForm: FormGroup;
  selectedServico: string;
  public clienteCollection: AngularFirestoreCollection<Cliente>;
  public servicosCollection: AngularFirestoreCollection<Servicos>;
```

```
(alias) interface Servicos
import Servicos
```

Fonte: Autoria própria (2021)

Assim como já visto na tela de consulta, o código da home.page.ts inicia com os *imports* necessários para realizar as funções nesta tela e a declaração de algumas variáveis, nesse caso é necessário fazer com que o *AngularFiresStoreCollection* traga os dados das *Collections* cliente e serviços, para serem consultadas e utilizadas no agendamento.

Figura 30 – Home.page.ts(event)

```
event = {
  title: '',
  desc: '',
  startTime: '',
  endTime: '',
  allDay: false
};
minDate = new Date().toISOString();
eventSource = [];
calendar = {
  mode: 'month',
  currentDate: new Date(),
};
viewTitle: '';
```

Fonte: Autoria própria (2021)

Na Figura 30 pode-se observar a criação do *event*(evento), composto de algumas variáveis como (titulo, desc, início, fim, dia todo), logo abaixo tem-se uma variável (*minDate*) usada para criar uma data mínima para esse evento, e logo abaixo tem uma lista gerada para estes eventos e o calendário, onde ele recebe o *mode* já abordado no html, responsável por escolher, se será diário, semana ou mês o modo do calendário e também *currenteDate*(dia de hoje), e por fim o *viewTitle*, que recebe um título, nesse caso o mês e ano.

Figura 31 – Home.page.ts (Construtor)

```
constructor( public storage: Storage,
  private alertCtrl: AlertController,
  @Inject(LOCALE_ID) private locale: string,
  private modalCtrl: ModalController,
  private afs: AngularFirestore,
  ) {



  this.clienteCollection = afs.collection<Cliente>(
    'Clientes');
  this.clientes = this.clienteCollection.valueChanges();
  this.clientSelectorForm = new FormGroup({ selectedCliente: new FormControl(), });
  this.clientes.subscribe(clientes => {
    this.selectedCliente = clientes[0].id;
    this.clientSelectorForm.controls.selectedCliente.patchValue(this.selectedCliente);
  });


  this.servicosCollection = afs.collection<Servicos>(
    'Servicos');
  this.servicos = this.servicosCollection.valueChanges();
  this.servicoSelectorForm = new FormGroup({ selectedServico: new FormControl(), });
  this.servicos.subscribe(servicos => {
    this.selectedServico = servicos[0].id;
    this.servicoSelectorForm.controls.selectedServico.patchValue(this.selectedServico);
  });


  }
```

Fonte: Autoria própria (2021)

Na sequência tem-se o método construtor e nele se encontra os métodos que são utilizados para selecionar na *Collections* o grupo de itens contidos na mesma, tanto para cliente quanto para serviços.

Figura 32 – Home.page.ts (funções)

```
async ngOnInit() {
  this.restEvent();
  await this.storage.create();
  this.storage.get('agendamento').then((val)=>{
    this.eventSource = val;
    this.myCal.loadEvents();
  });
}
onClienteChange(value) {
  this.selectedCliente = value.selectedCliente;
}
onServicoChange(value) {
  this.selectedServico = value.selectedServico;
}

restEvent(){
  this.event ={
    title: '',
    desc: '',
    startTime: '',
    endTime: '',
    allDay: false
  };
}
```

Fonte: Autoria própria (2021)

As duas primeiras funções são *onClienteChange* e *onServicoChange*, que são responsáveis por receber os valores selecionados no *Select-item* e alterar os valores das variáveis *selectedCliente* e *selectedServico*, já a *restEvent* serve para que após a confirmação ou gravação do agendamento as variáveis do evento sejam limpas para esperar um próximo evento.

Figura 33 – Home.page.ts (Adicionar-evento)

```
addEvent(){
  const eventCopy ={
    title: this.event.title,
    startTime: new Date(this.event.startTime),
    endTime: new Date(this.event.endTime),
    allDay: this.event.allDay,
    id: this.event.startTime + this.event.endTime,
    desc: this.event.desc
  };
  if(eventCopy.allDay){
    const start = eventCopy.startTime;
    const end = eventCopy.endTime;
    eventCopy.startTime = new Date(Date.UTC(start.getUTCFullYear(), start.getUTCMonth(), start.getUTCDate()));
    eventCopy.endTime = new Date(Date.UTC(end.getUTCFullYear(), end.getUTCMonth(), end.getUTCDate() + 1));
  }
  this.eventSource.push(eventCopy);
  this.storage.set('agendamento',this.eventSource);
  this.myCal.loadEvents();
  this.restEvent();
}
```

Fonte: Autoria própria (2021)

A função add*Event* é a principal função desta tela, ela é encarregada de fazer o armazenamento das informações inseridas nas variáveis do formulário em uma

estrutura, após esses dados são inseridos no *array eventSource*, logo em seguida é chamada à função própria do componente calendário *loadEvents*, ela é a responsável por atualizar o calendário trazendo nele o agendamento que acabou de ser efetuado.

Figura 34 – Home.page.ts (Outras funções)

```
changeMode(mode){
  this.calendar.mode = mode;
}

next() {
  // eslint-disable-next-line @typescript-eslint/dot-notation
  const swiper = document.querySelector('.swiper-container')['swiper'];
  swiper.slideNext();
}

back() {
  // eslint-disable-next-line @typescript-eslint/dot-notation
  const swiper = document.querySelector('.swiper-container')['swiper'];
  swiper.slidePrev();
}

onViewTitleChanged(title) {
  this.viewTitle = title;
}
today(){
  this.calendar.currentDate = new Date();
}
```

Fonte: Autoria própria (2021)

As funções de *changeMode* já foram abordadas, a função *back*(voltar) e *next*(próximo), servem para navegar pelas páginas do calendário, *today*(hoje) serve para retornar a data atual.

Figura 35 – Home.page.ts (onEventSelected e onTimeSelected)

```
async onEventSelected(event) {

  const start = formatDate(event.startTime, 'medium', this.locale);
  const end = formatDate(event.endTime, 'medium', this.locale);

  const alert = await this.alertCtrl.create({
    header: event.title,
    subHeader: event.desc,
    message: 'From: ' + start + '<br><br>To: ' + end,
    buttons: [{
      text: 'Ok',
      role: 'ok',
      handler: () => {
        console.log('Cancel clicked');
      }
    },
    {
      text: 'DELETE',
      handler: () => {
        console.log(this.eventSource);
        this.eventSource = this.eventSource.filter(eventSource => eventSource.id !== event.id);
        this.storage.set('agendamento',this.eventSource);
      }
    }
  ],
  });
  alert.present();
}

onTimeSelected(ev) {
  const selected = new Date(ev.selectedTime);
  this.event.startTime = selected.toISOString();
  selected.setHours(selected.getHours() + 1);
  this.event.endTime = (selected.toISOString());
}

}
```

Fonte: Autoria própria (2021)

A função de *onEventSelected*, traz a tela uma pequena janela, com detalhes do agendamento, e junto dela é apresentado um botão de delete, no qual o usuário pode excluir o agendamento caso necessite, para que não haja problema na deleção foi criado um campo id que no caso seria a soma do início e fim do agendamento, e por último a função *onTimeSelected* que quando se seleciona um dia ou data e hora de um dia específico no calendário, ele atualiza o formulário de agendamento para o usuário, colocando o horário de fim uma hora depois do horário de início, assim o usuário só terá de escolher o cliente e o serviço para o agendamento.

4 CONSIDERAÇÕES FINAIS DO TRABALHO

Este capítulo está dividido em duas seções. A primeira discorre sobre as conclusões obtidas no desenvolvimento do trabalho e a segundo apresenta alguns trabalhos que podem ser implementados para complementar este.

4.1 Conclusão

Este trabalho propôs o desenvolvimento de uma solução para uma clínica de estética localizada na cidade de Ponta Grossa, Paraná, com o objetivo de oferecer um controle mais adequado para a gestão de agendamento de tratamentos disponibilizados pela empresa, receituário e prontuário, serviços estes não disponibilizados pelos aplicativos analisados neste trabalho.

O aplicativo permite o cadastro de clientes, anamnese corporal e capilar, tratamentos (serviços) disponibilizados pela clínica, agendamento destes tratamentos, permitindo gerenciar as receitas indicadas para cada cliente e seus prontuários.

O desenvolvimento dos aplicativos utilizou a abordagem híbrida de construção de aplicativos, a qual permite construir um aplicativo utilizando *HTML*, *CSS* e acessar funcionalidades nativas dos dispositivos como câmera e conectividade que não podem ser acessadas apenas por *HTML*. Esta abordagem permitiu construir um aplicativo de qualidade com uma apresentação visual adequada, fazendo uso de recursos nativos dos dispositivos móveis e que consequentemente colaboram para uma melhor experiência de uso aos usuários.

Todas as ferramentas utilizadas no projeto são *open-source*, ou seja, não exigiram licença de desenvolvedor ou pagamento para utilização, portanto não houve gasto durante o desenvolvimento. Este fator é importante para as empresas de desenvolvimento de software, já que a redução de custos é almejada em várias empresas.

Para iniciar o projeto, foi efetuada a configuração da *API Cloud FireStore*, sem demais problemas, com ela foi possível o armazenamento das informações no banco de dados *FireBase*, efetuando o *login*(autenticação) corretamente, sem apresentar erros. A criação do projeto Ionic com base na linguagem Angular também foi efetuada com sucesso, o trabalho foi feito com base na linguagem mencionada.

Os requisitos necessários da aplicação foram levantados junto da clínica por meio da professora orientadora deste trabalho, onde foram abordadas as telas que

seriam necessárias, os conteúdos e informações, tanto ligados com a parte de desenvolvimento de aplicativo quanto com a área que seria aplicado o experimento, no caso a clínica de estética, então foram abordados fatos e informações necessárias para criação de formulários que estão contidos na aplicação.

Por último o caso de identificar critérios relacionados a web acessibilidade, neste caso alguns critérios foram utilizados, como por exemplo paleta de cores, o qual buscam ajudar que pessoas com distúrbio da visão(daltonismo), trazendo cores que podem ser mais fáceis para que o mesmo possa distinguir, o tamanho de texto tem como objetivo ajudar pessoas com hipermetropia, vista cansada e astigmatismo, botões com ícones e títulos para que se tornarem mais intuitivos, para pessoas com déficit de atenção, idosos ou pessoas que simplesmente não tem intimidade com a tecnologia.

Todo o projeto apresentou-se organizado na mesma estrutura, portanto a manutenção e as alterações no código-fonte tornaram-se muito mais fáceis. Grande parte do sucesso deste projeto deve-se, principalmente, às ferramentas utilizadas que facilitaram o desenvolvimento e a organização do mesmo.

4.2 Trabalhos Futuros

Como possíveis trabalhos futuros, pode-se apontar a implementação de uma função nativa da câmera para que possa ser usada na tela de tratamentos com o intuito de acrescentar ao usuário uma opção de poder tirar fotos das partes que serão tratadas. Com esta implementação feita será necessário também fazer mudanças no banco de dados.

Além da implementação citada anteriormente será interessante acrescentar as buscas, no caso nos <i*on-select*> a opção de pesquisar o nome do serviço ou do cliente que será buscado.

Outra ideia de trabalho futuro seria a implementação de notificação nativa para que o aparelho do usuário possa notificar o mesmo de um evento mesmo com o aplicativo fechado.

Permitir que o próprio cliente da clínica, possa realizar o agendamento dos tratamentos desejados, no momento, essa funcionalidade é realizada somente pela clínica de estética.

REFERÊNCIAS

ANDRADE, Leandro Paiva. **FaleMed**: um sistema para armazenar documentos médicos e melhorar a interação entre o profissional da saúde e seu paciente. TCC (Ciências da Computação) - Universidade Federal da Paraíba, João Pessoa, p. 61. 2017.

AMATYA, Suyesh; KURTI, Arianit. Cross-platform mobile development: challenges and opportunities. In: ICT Innovations 2013. **Springer International Publishing**, 2014. p. 219-229.

APPCELERATOR. Native vs. **HTML5 Mobile App Development**: Which option is best? 2012. Disponível em: http://www.appcelerator.com/enterprise/resource-center/white-papers/native-vs-html5-mobile-app-development/. Acesso em: 26 jun. 2021.

DA SILVA, Marcelo Moro; SANTOS, Marilde Terezinha Prado. Os Paradigmas de Desenvolvimento de Aplicativos para Aparelhos Celulares. In: **Revista TIS**, v. 3, n. 2, 2014. Disponível em: http://revistatis.dc.ufscar.br/index.php/revista/article/view/86. Acesso em: 26 jun. 2021.

DE ANDRADE, Ana Paula. **O que é IONIC?** Disponível em: https://www.treinaweb.com.br/blog/o-que-e-ionic. Acesso em 26 jun. 2021.

DE MENDONÇA, Vinıcius Rafael Lobo; BITTAR, Thiago Jabur; DE SOUZA DIAS, Márcio. **Um estudo dos Sistemas Operacionais Android e iOS para o desenvolvimento de aplicativos**. 2011. Disponível em: http://www.enacomp.com.br/2011/anais/trabalhos-aprovados/pdf/enacomp2011_submission_54.pdf. Acesso em: 26 jun. 2021.

EMAG. **Modelo de Acessibilidade em Governo Eletrônico/ Ministério do Planejamento, Orçamento e Gestão, Secretaria de Logística e Tecnologia da Informação** - Brasília: MP, SLTI, 2014.

FINCOTTO, Marcos Apolinário. Automação Comercial utilizando Aplicativos Móveis-Um Foco na Plataforma Android. In: **Revista TIS**, v. 3, n. 2, 2014. Disponível em: http://revistatis.dc.ufscar.br/index.php/revista/article/view/85. Acesso em: 26 jun. 2021.

FIREBASE Documentation . Disponível em: https://firebase.google.com/docs/firestore?hl=pt-br. Acesso em: 26 jun. 2021.

FUNDAÇÃO GETÚLIO VARGAS (FGV). **Brasil tem dois dispositivos digitais por habitante, revela pesquisa da FGV**. Disponível em: https://portal.fgv.br/noticias/brasil-tem-dois-dispositivos-digitais-habitante-revela-pesquisa-fgv. Acesso em: 26 jun. 2021.

FUNDAÇÃO GETÚLIO VARGAS (FGV). **Brasil tem 424 milhões de dispositivos digitais em uso, revela a 31ª Pesquisa Anual do FGVcia**. Disponível em: https://portal.fgv.br/noticias/brasil-tem-424-milhoes-dispositivos-digitais-uso-revela-31a-pesquisa-anual-fgvcia. Acesso em: 26 jun. 2021.

GOOGLE. **Nosso Planeta Mobile**: Brasil. 2013. Disponível em: https://services.google.com/fh/files/misc/omp-2013-br-local.pdf. Acesso em: 19 jul. 2021.

GUEDES, Marylene. **O que é angular e para que serve?** Disponível em: https://www.treinaweb.com.br/blog/o-que-e-o-angular-e-para-que-serve. Acesso em 26 jun. 2021.

IONIC. Ionic Documentation Overview. Disponível em: http://ionicframework.com/docs/overview/. Acesso em: 26 jun. 2021.

NETO, Georgea. **Aplicativo móvel multiplataforma para consulta e agendamento de serviços estéticos com geolocalização papum**. TCC (Bacharel em Ciências Da computação) - Ciências da Computação, Centro Universitário FACVEST, Lages, p. 107. 2020.

SANTOS, Silvanei Soares**. Aplicativo para agendamento de horário em salões de beleza**. 2016, 138p. TCC (Especialização em Engenharia de Software) - Especialização em Engenharia de Software, Universidade Tecnológica Federal do Paraná, Curitiba, 2016.

TYSKA, Vanessa**. O uso do Smartphone como Ferramenta de Pesquisa pelos Estudantes do Ensino Médio**. 2018, 67p. TCC (Bacharel em Biblioteconomia) - Faculdade de Biblioteconomia e Comunicação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2018.