

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Rodrigues Borges

**Desenvolvimento de Aplicação Mobile
Utilizando Metodologia Ágil SCRUM**

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Rodrigues Borges

**Desenvolvimento de Aplicação Mobile Utilizando
Metodologia Ágil SCRUM**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Orientador: Maria Adriana Vidigal de Lima

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2017

Bruno Rodrigues Borges

Desenvolvimento de Aplicação Mobile Utilizando Metodologia Ágil SCRUM

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 13 de dezembro de 2017:

Maria Adriana Vidigal de Lima
Orientador

Professor

Professor

Uberlândia, Brasil
2017

Dedico esse trabalho a meu falecido pai, a todos que contribuíram de alguma forma para sua conclusão e principalmente a minha mãe, que sempre esteve ao meu lado me auxiliando em tudo que foi possível para que pudesse chegar até aqui.

Agradecimentos

Agradeço aos professores, mestres e doutores que agregaram conhecimento a minha formação até aqui, agradeço a minha orientadora por todo auxílio e dedicação, assim como colegas de turma, amigos e a todos familiares que de alguma forma contribuíram para a conclusão de mais essa etapa na minha vida.

Resumo

A digitalização de processos do nosso cotidiano é uma realidade há alguns anos, e tudo que traz algum benefício para o trabalho das pessoas é válido. Pensando nisso, esse trabalho propõe o desenvolvimento de uma aplicação móvel que busca a otimização de um processo manual utilizado atualmente por profissionais esteticistas da área massoterápica. Neste processo, os profissionais costumam utilizar um formulário em papel, conhecido como *Ficha de Anamnese*, para o gerenciamento de seus clientes e atendimentos. Esse formulário possibilita que o profissional mantenha dados pessoais, contatos de emergência, medidas e histórico médico de seus clientes, além de informações sobre os atendimentos feitos. Uma proposta de solução digital para esse problema é uma aplicação móvel que possa estar a todo momento ao alcance desses profissionais e possa substituir o processo atual, buscando otimizar o uso do tempo de trabalho e evitar problemas decorrentes de perda de informações necessárias para o gerenciamento da atividade desses profissionais.

Palavras-chave: Aplicativo, Android, Mobile, SCRUM, Metodologia Ágil.

Lista de ilustrações

Figura 1 – Práticas do Scrum. Fonte: (PRATICAS..., 2017)	14
Figura 2 – Principais Atividades e Artefatos Fonte: (PRATICAS..., 2017)	15
Figura 3 – Recursos do Firebase	19
Figura 4 – Diagrama Caso de Uso Geral	23
Figura 5 – Estrutura do Projeto	23
Figura 6 – Diagrama Entidade Relacionamento do Banco	24
Figura 7 – Ficha Anamnese Corporal	25
Figura 8 – Tela Formulário Cliente	29
Figura 9 – Tela de Informação do Cliente	30
Figura 10 – Telas de Listagem de Clientes	30
Figura 11 – Tela do Formulário de Pacote	34
Figura 12 – Telas de Listagem dos Pacotes	37
Figura 13 – Tela de Listagem de Sessão	37
Figura 14 – Tela de Sessões Agendadas	38
Figura 15 – Telas de Finalização de Sessões	41
Figura 16 – Tela Formulário de Medida	42
Figura 17 – Telas de Listagem de Medidas	42
Figura 18 – Telas de Formulário de Histórico	43
Figura 19 – Tela de Visualização de Histórico	43
Figura 20 – Tela Formulário dos contatos de Emergência	47
Figura 21 – Tela Listagem de Contatos de Emergência	47
Figura 22 – Tela de Login	49
Figura 23 – Estrutura do armazenamento dos dados na nuvem (formato de documento)	53
Figura 24 – Pergunta 1	58
Figura 25 – Pergunta 2	58
Figura 26 – Pergunta 3	59
Figura 27 – Pergunta 4	59
Figura 28 – Pergunta 5	59
Figura 29 – Pergunta 6	60
Figura 30 – Pergunta 7	60
Figura 31 – Pergunta 8	60

Lista de abreviaturas e siglas

API	Application Programming Interface
CEP	Código de Endereçamento Postal
CRUD	Create, Read, Update e Delete
CSS	Cascading Style Sheets
DAO	Data Access Object
DDD	Discagem Direta a Distância
DER	Diagrama Entidade-Relacionamento
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Modelo-Visão-Controlador
NoSQL	Not Only SQL
SCRUM	Metodologia ágil para gestão e planejamento de projetos de software
SDK	Software Development Kit
SMS	Short Message Service
SQLite	Biblioteca em linguagem C que implementa um banco de dados SQL embutido
SSL	Secure Socket Layer
QA	Quality Assurance

Sumário

1	INTRODUÇÃO	10
1.1	Problema	10
1.2	Objetivos	11
1.2.1	Objetivos Específicos	11
1.3	Método	12
1.4	Resultado Esperado	12
1.5	Organização do Trabalho	12
2	REVISÃO BIBLIOGRÁFICA	13
2.1	Scrum	13
2.1.1	A Base do Scrum	13
2.1.2	Papéis fundamentais	14
2.1.3	Atividades Básicas	15
2.1.4	Artefatos	16
2.2	Sistema Operacional Android e Armazenamento	16
2.2.1	SQLite	17
2.3	Firestore	18
2.3.1	Realtime Document Database	19
2.3.2	Authentication	20
2.3.3	Crash Reporting	20
3	DESENVOLVIMENTO DA APLICAÇÃO MÓVEL UTILIZANDO METODOLOGIA SCRUM	21
3.1	Definição do Escopo do Produto (<i>Product Backlog</i>)	21
3.2	Configuração do projeto	22
3.3	Primeiro Sprint	24
3.3.1	Primeira Reunião de Planejamento do Sprint	25
3.3.2	Armazenamento de dados dos Clientes	25
3.3.3	Formulário de Cliente	27
3.3.4	Listagem de Clientes Salvos	28
3.3.5	Primeira Reunião de Revisão de Sprint	31
3.4	Segundo Sprint	31
3.4.1	Segunda Reunião de Planejamento de <i>Sprint</i>	31
3.4.2	Armazenamento de Pacotes e Sessões	32
3.4.3	Formulário de Pacotes	34
3.4.4	Listagem e Gerenciamento de Pacotes e Sessões	36

3.4.5	Agendamento de Sessões	37
3.4.6	Segunda Reunião de Revisão de Sprint	38
3.5	Terceiro Sprint	39
3.5.1	Terceira Reunião de Planejamento de Sprint	39
3.5.2	Gerenciamento de Sessões	39
3.5.3	Gerenciamento das Medidas	41
3.5.4	Gerenciamento do Histórico Clínico	42
3.5.5	Gerenciamento de Contatos de Emergência	45
3.5.6	Reunião de Revisão do Terceiro Sprint	46
3.6	Quarto Sprint	48
3.6.1	Quarta Reunião de Planejamento de Sprint	48
3.6.2	Autenticação de Acesso	48
3.6.3	Recuperação de Acesso	52
3.6.4	Recuperação de Dados	52
3.6.5	Quarta Reunião de Revisão de Sprint	55
3.7	Qualidade do Software	55
3.8	Pesquisa de Usabilidade	56
4	CONCLUSÃO	61
	Conclusão	61
	REFERÊNCIAS	62

1 Introdução

Desde o ano de 2007 com o lançamento do iPhone, o uso de *smartphones* e seus aplicativos não parou de crescer. No início, ainda com aplicativos nativos sem a possibilidade de baixar novos, até a criação de lojas virtuais para disponibilização e comercialização, o uso de aplicativos teve um crescimento estrondoso e estes começaram cada vez mais a fazer parte do dia-a-dia das pessoas.

São vários os tipos de aplicativos disponíveis atualmente, e através desses softwares a função primária de um telefone, fazer uma ligação, ficou completamente em segundo plano. Com os *smartphones* e seus aplicativos é possível fazer coisas que antes demandavam deslocamentos, como pagar uma conta, comprar o jornal do dia e fazer compras. Além disso, ferramentas muito úteis como agenda, câmera de vídeo, sistemas de comunicação por redes sociais, conversas por mensagem e até chamadas com vídeo em tempo real podem ser facilmente incorporadas.

Porém, apesar da imensa popularização tecnológica ainda é possível encontrar profissionais que utilizam meios manuscritos para realizar anotações, gerenciar processos, guardar informações entre outras coisas. E talvez o fazem dessa forma por não encontrar soluções digitais que satisfaçam suas necessidades. Com a disseminação dos *smartphones*, para onde se olha existe alguém com um deles em mãos, e nesse mercado tão grande o que não falta são empresas e pessoas buscando desenvolver soluções para situações e problemas específicos.

Esse trabalho tem como foco o desenvolvimento de uma aplicação móvel para atender um ramo da estética chamado massoterapia. Nessa área, os profissionais costumam utilizar um formulário em papel para gerenciar seus clientes e atendimentos. Esse formulário é conhecido como *Ficha de Anamnese*, e possibilita que o profissional tenha maior conhecimento sobre seu paciente. Nessa ficha são coletados dados pessoais, contatos de emergência, medidas e histórico médico de seus clientes, além de informações sobre atendimentos e sessões. Uma proposta de solução digital para esse problema é uma aplicação que possa estar a todo tempo ao alcance desses profissionais. Neste sentido, esta solução otimiza muito o uso do tempo de trabalho e evita problemas decorrentes de perda dessas fichas e, conseqüentemente, das informações necessárias para o exercício do trabalho.

1.1 Problema

Mesmo com toda a tecnologia disponível, a existência de processos manuais ainda em uso revela mercados ignorados ou mal atendidos por soluções digitais. Neste cenário,

este trabalho propõe a construção de um aplicativo móvel para atender à profissionais de estética, da área de massoterapia, para auxiliar no controle dos vários processos que devem ser realizados, como gerenciamento dos dados dos clientes e das sessões de terapia indicadas para os mesmos. Estes controles são feitos normalmente em fichas de papel. O armazenamento em papel torna o trabalho inviável, possibilitando o acontecimento de vários incidentes indesejáveis.

Dos pontos negativos encontrados no sistema atual de controle de clientes e pacotes de sessões de terapias, pode-se citar: a perda de fichas de papel o que ocasiona na perda de informações relevantes sobre os cliente; a logística destas fichas quando as mesmas devem ser levadas para os atendimentos; gasto com papel e impressão; vulnerabilidade no acesso a informações pessoais de clientes; além de ser necessária uma agenda também de papel para gerenciar horários de atendimento.

A digitalização ajuda na otimização desses processos como um todo, possibilita soluções em caso de contratempos, como perda de dados, alterações, consultas de informações e melhora a comunicação entre os envolvidos.

1.2 Objetivos

O presente trabalho tem como objetivo desenvolver uma aplicação móvel que consiga substituir e melhorar vários aspectos o processo já existente, automatizando-o e buscando a otimização do mesmo, além de diminuir gastos com papel, extinguir a logística com fichas atuais existentes e evitar a perda de informações de clientes.

1.2.1 Objetivos Específicos

1. Estudo do processo manual para definir os requisitos do desenvolvimento de um aplicativo que substitua o processo atual.
2. Estudo e definição de um método de desenvolvimento ágil para auxiliar no gerenciamento do projeto.
3. Estudo da API do Sistema *Android* assim como de todas as tecnologias envolvidas para o desenvolvimento de uma aplicação móvel em linguagem Java.
4. Definição da estrutura de armazenamento local dos dados e da forma de transmissão dos mesmos à uma base de dados em nuvem, garantindo a segurança dos dados.

1.3 Método

Para o entendimento do problema e definição do escopo do projeto foi feita uma reunião com os profissionais envolvidos na utilização da Ficha de Anamnese. O processo de desenvolvimento baseou-se nas técnicas do arcabouço SCRUM, uma metodologia usada para o gerenciamento e desenvolvimento de projetos ágeis. A linguagem Java juntamente com as ferramentas do ambiente de programação Java para Android (*SDK tools*) foram escolhidas para a construção do aplicativo. Considerando a etapa de construção do *backend* da aplicação, utilizou-se uma API do Google chamada Firebase.

1.4 Resultado Esperado

O resultado final pretendido desse trabalho é a entrega de uma aplicação móvel executável em sistema Android que consiga substituir e melhorar o processo convencional existente hoje através de um sistema de atendimento automatizado especial para massoterapeutas, sendo essa aplicação desenvolvida obedecendo todas as boas práticas do desenvolvimento de software ágil, com foco na qualidade do software final.

1.5 Organização do Trabalho

O presente documento está organizado da seguinte forma: o capítulo 2 apresenta os principais conceitos sobre as tecnologias e ferramentas utilizadas no desenvolvimento desse projeto. No capítulo 3 está definida a configuração do projeto, e a descrição de como foi realizado o desenvolvimento, com a apresentação dos resultados obtidos numa pesquisa de usabilidade sobre o produto desenvolvido. Por último, uma conclusão é apresentada diante do que foi proposto, com base nos resultados obtidos e nos testes realizados.

2 Revisão Bibliográfica

Nesse capítulo estão descritas as tecnologias e ferramentas utilizadas no planejamento e desenvolvimento do aplicativo. Para sua implementação foi aplicada a metodologia de desenvolvimento ágil Scrum (MORRIS, 2017) no gerenciamento do projeto e as tecnologias Android, SQLite e Firebase.

2.1 Scrum

A palavra Scrum tem sua origem em um esporte coletivo chamado Rugby, e dá nome a uma jogada que representa na prática a ideia do trabalho em equipe com sincronia, força e inteligência, utilizadas rumo à um objetivo. Isso é o que determina o *framework* proposto para o desenvolvimento ágil Scrum. Como no Rugby, cada membro do time Scrum deve possuir habilidades para juntos se apoiarem e chegarem ao objetivo final.

Os princípios do Scrum são coerentes com o manifesto ágil (BECK et al., 2001) e são usados para orientar o desenvolvimento de software em um processo que incorpora: requisitos, análise, projeto, evolução e entrega. Neste processo ocorrem tarefas realizadas dentro de um ciclo de processo chamado *sprint*. A quantidade de *sprints* prevista varia dependendo do tamanho e da complexidade do software a ser desenvolvido (PRESSMAN; MAXIM, 2016).

O Scrum é uma metodologia ágil focada no planejamento e na gerência de projetos de software, e que com pequenos ciclos e uma equipe bem reduzida de desenvolvimento busca *feedbacks* rápidos e conseqüentemente a garantia de qualidade do projeto e satisfação do cliente.

2.1.1 A Base do Scrum

O Scrum tem como base três pilares fundamentais:

- **Transparência:** Todos devem ter conhecimento dos requisitos de entrega, dos processos e do andamento do projeto;
- **Inspecção:** A todo momento o desenvolvimento é monitorado, seja nas reuniões diárias e no final de cada ciclo.
- **Adaptação:** O Scrum pode ser visto como adaptável em sua estrutura, pois, desde que respeitados os valores e práticas, seus processos podem ser modificados de acordo com o ambiente do problema.

Esta base é composta pelas práticas ilustradas na Figura 1. Nelas pode-se notar a definição clara de papéis para os envolvidos no projeto, as atividades básicas que determinam os ciclos de trabalho no Scrum, e o conjunto de artefatos gerados com o objetivo de organizar e inspecionar os processos, além de propiciar transparência sobre o andamento do projeto como um todo.



Figura 1 – Práticas do Scrum. Fonte: (PRATICAS..., 2017)

2.1.2 Papéis fundamentais

A equipe envolvida no desenvolvimento de software segundo a metodologia Scrum é separada em três papéis principais (PRATICAS..., 2017):

1. **Dono do Produto (*Product Owner*):** É a representação do cliente dentro do projeto. Ele conhece o problema como um todo e é responsável por ajudar a definir o Escopo do Produto (*Product Backlog*), priorizar os requisitos, auxiliar o time de desenvolvimento e aprovar ou não o que for entregue.
2. **Mestre Scrum:** É o líder do projeto e responsável pela fiel aplicação dos processos do Scrum no projeto. É dele também a função de facilitador, evitando tudo que possa comprometer a evolução do projeto.
3. **Time Scrum:** São os desenvolvedores do projeto (programadores, equipe de teste, DBA's). É um grupo auto-gerenciável que define como tudo será feito para que a entrega aconteça como planejada. Normalmente possui entre 6 e 10 integrantes.

2.1.3 Atividades Básicas

Os eventos do Scrum são cíclicos, e esta característica força a entrega dentro do prazo determinado, além de auxiliar a inspeção do código produzido e alguma adaptação necessária, conforme Figura 2. A cada ciclo existem atividades básicas que devem ser executadas, e segundo Morris (2017) essas atividades são:

- *Sprint* - Ciclo de trabalho num determinado período de tempo, ou seja, o momento em que o trabalho do time efetivamente é realizado: entre 2 e 4 semanas;
- Planejamento do *Sprint* - Reunião para que a equipe defina com o *Dono do Produto* as funcionalidades que serão desenvolvidas durante o *sprint* e como devem ser executadas para que a entrega ocorra dentro do prazo;
- *Scrum* Diário - Reunião diária para análise do progresso da equipe, do trabalho restante e de quaisquer impedimentos que surjam, sendo possível replanejar o que for necessário;
- Revisão do *Sprint* - Etapa de entrega do que foi feito e de compartilhamento do progresso com os interessados, buscando comentários que ajudem a promover possíveis adaptações do projeto e que cooperem com a motivação da equipe;
- Retrospectiva do *Sprint* - Reunião em que a equipe avalia o último *Sprint* e identifica possíveis ações de melhoria.

Como descrito em Praticas... (2017), ainda pode existir outra atividade, a Organização do Escopo do Produto (*Product Backlog Grooming*). Pode-se valer desta atividade para o refinamento dos itens do Escopo do Produto e a inclusão de algum novo item.



Figura 2 – Principais Atividades e Artefatos Fonte: (PRATICAS..., 2017)

2.1.4 Artefatos

Para proporcionar transparência e oportunidades de inspeção e adaptação dos processos, o Scrum define alguns artefatos para possibilitar a sustentação dos seus pilares fundamentais:

- Escopo do Produto (*Product Backlog*) - É uma lista definida pelo Dono do Produto contendo todas as funcionalidades desejadas para um software. Essa lista não precisa estar completa no início de um projeto. Ela pode ser iniciada com o que for mais óbvio em um primeiro momento, e com o passar do tempo, crescer e mudar à medida que se aprende mais sobre o produto e seus usuários.
- Escopo do *Sprint* - É uma sublista das funcionalidades do Escopo do Produto, contendo as tarefas que o Time Scrum se compromete a realizar durante o *Sprint*. Tais tarefas são definidas com base nas prioridades indicadas pelo Dono do Produto e na percepção da equipe sobre o tempo que será necessário para completar as várias funcionalidades.
- Incremento do Produto - É o conjunto de todas as tarefas encerradas no *Sprint*. Cada entrega deve ser utilizável para mostrar o progresso até aquele momento com objetivo de que o cliente consiga perceber o valor no que foi entregue.
- Definição de Pronto (*Definition of Done*) - É um documento feito em conjunto pela equipe que define quando cada tarefa do *sprint* pode ser considerada como pronta. Somente com essa definição clara de tarefa pronta é que se torna possível estimar o tempo de desenvolvimento e entrega de cada uma.

2.2 Sistema Operacional Android e Armazenamento

O sistema operacional Android baseia-se no núcleo do sistema Linux para funcionar em dispositivos móveis. É uma plataforma aberta (*open source*) e as aplicações desenvolvidas para esta plataforma são criadas utilizando-se a linguagem de programação Java (DEITEL; DEITEL; DEITEL, 2015). O Kit de Desenvolvimento de Software para Android (*Android SDK*) inclui uma lista de ferramentas de desenvolvimento em linguagem Java, entre elas *debugger*, bibliotecas, emulador, documentação, códigos de exemplo e tutoriais (SMYTH, 2017a; ABLESON; KING; SEN, 2012). Os aplicativos gerados para o sistema Android consistem em arquivos em formato *.apk* e são armazenados no diretório */data/app* do sistema operacional. Esta pasta é acessível somente ao usuário *root* por razões de segurança.

O sistema Android oferece várias opções para salvar dados de aplicativos de forma persistente, podendo-se considerar se os dados devem ser privados para a aplicação ou se

devem ser acessíveis para outras aplicações (e para o usuário) e quanto espaço os dados devem consumir. As opções de armazenamento de dados são ([ANDROID...](#), 2017):

- Preferências compartilhadas: dados primitivos privados são armazenados em pares chave-valor. Cada par representa uma preferência de usuário em relação à aplicação, contendo uma chave e seu valor associado.
- Armazenamento interno: dados privados são mantidos na memória do dispositivo. Os arquivos salvos no armazenamento interno pertencem ao aplicativo e não podem ser acessados por outros aplicativos ou diretamente pelo usuário.
- Armazenamento externo: dados públicos são mantidos na memória externa e podem ser vistos e compartilhados por vários usuários.
- Bancos de dados SQLite: possibilita o armazenamento de dados estruturados em um banco de dados relacional privado.
- Conexão de rede: provê o armazenamento de dados na *web* a partir de um servidor próprio de rede.

O *Android SDK* viabiliza a utilização das opções de armazenamento citadas, sendo que a escolha destas opções deve considerar as especificidades de tipo e uso das informações a serem mantidas. Para cada opção existem classes específicas dentro do *Android SDK* que viabilizam a implementação do tipo de armazenamento.

2.2.1 SQLite

O SQLite ([SQLITE](#), 2017) é composto de uma biblioteca que implementa um banco de dados SQL auto-suficiente, sem servidor ou configuração e transacional. É um mecanismo de banco de dados SQL incorporado, visto que não possui um processo de servidor em separado. Ele é completo podendo ter várias tabelas, índices, gatilhos (*triggers*) e visões, e isso tudo estando contido em um único arquivo de disco, o que otimiza o gasto com espaço em memória, podendo facilmente ser embarcado em dispositivos como telefones celulares, PDAs e MP3.

Quando comparado com outros "formatos de arquivo de aplicativo" tem vantagens convincentes sobre os mesmos, como transações ACID, linguagem de consulta de alto nível, atualizações incrementais e contínuas (quando somente o que foi alterado é atualizado na memória), capacidade de manipulação de grande base de dados (podendo chegar a terabytes), interface com varias linguagens como Python, Ruby, Delphi, Java, Javascript, Lua, .NET, PHP, entre outras.

2.3 Firebase

O Firebase é uma plataforma do Sistema Google que contém várias ferramentas e uma infraestrutura para auxiliar desenvolvedores de aplicações *web* e de dispositivos móveis na construção de software de alta qualidade e performance (HOW... , 2017; SMYTH, 2017b). Esta plataforma contém quatro segmentos de serviços: *Analytics*, *Develop*, *Grow* e *Earn*, como mostra a Figura 3.

O Firebase *Analytics* é uma solução construída para que o desenvolvedor possa gerar as métricas da aplicação e mensurar o comportamento do usuário. Do lado *Develop*, os diversos serviços têm como objetivo principal fornecer recursos de desenvolvimento de aplicativos com alta qualidade e tempo reduzido. São eles:

- *Cloud Messaging*: permite a entrega e o recebimento de mensagens e notificações entre as plataformas iOS, Android e Web.
- *Authentication*: possibilita conhecer a identidade do usuário e manter o controle do acesso ao aplicativo. Permite ainda utilizar provedores de identidades externos para autenticação como as contas do Google, Facebook, Twitter e GitHub.
- *Realtime Document Database*: disponibiliza um banco de dados NoSQL hospedado em nuvem em que os dados são armazenados em formato JSON e sincronizados em tempo real com todos os clientes conectados.
- *Storage*: facilita o armazenamento de arquivos como imagens, vídeos e áudio, além de outros conteúdos gerados por usuários.
- *Hosting*: oferece hospedagem HTML, CSS e JavaScript para uma página web específica, além de outros ativos fornecidos pelo desenvolvedor, como gráficos, fontes e ícones. Possui certificado SSL fornecido automaticamente, sendo ideal para aplicativos web e para dispositivos móveis.
- *Remote Config*: armazena pares de chave-valor especificados pelo desenvolvedor que permite alterar o comportamento e a aparência do aplicativo sem exigir que os usuários baixem uma atualização do aplicativo.
- *Test Lab*: fornece infraestrutura em nuvem para a realização de testes em aplicativos Android.
- *Crash Reporting*: permite a criação de relatórios de erro detalhados para os aplicativos Android e iOS. Os erros são agrupados em conjuntos e organizados de acordo com o possível impacto para os usuários.

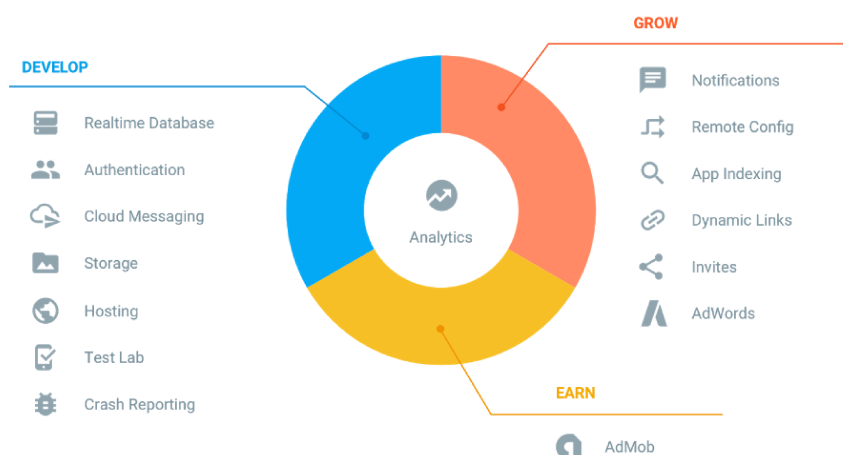


Figura 3 – Recursos do Firebase, Fonte: (HOW..., 2017)

O segmento *Grow* disponibiliza recursos voltados ao envolvimento e conquista dos usuários para a aplicação, e o *Earn* proporciona mecanismos para monetizar aplicativos com publicidade segmentada e gerar receita sem prejudicar a experiência do usuário.

O Firebase possibilita o desenvolvimento em várias plataformas, é escalável, tem suporte gratuito, fácil implementação, além de disponibilizar relatórios que ajudam nas tomadas de decisões sobre os usuários. Essas características fizeram com que o Firebase fosse escolhido para ser usado no desenvolvimento desse trabalho.

2.3.1 Realtime Document Database

O Realtime Document Database é um banco de dados NoSQL, orientado a documento, com hospedagem de dados em nuvem (REALTIME..., 2017). Ele possibilita o armazenamento e a sincronização dos dados em tempo real, seja pela Web ou a partir de dispositivos móveis. O Firebase disponibiliza SDKs para desenvolvimento em várias plataformas e utiliza memória local do dispositivo para aplicar e armazenar atualizações de dados. Em consequência, quando o dispositivo volta a ficar conectado, os dados locais podem ser sincronizados automaticamente através do Realtime Database.

Para conseguir o acesso em tempo real, o Firebase não se utiliza de requisições HTTPs típicas, e sim sincronização de dados. Com isso, sempre que uma informação é modificada, todos os ouvintes têm acesso a essa modificação em milissegundos. Todos os dados do Realtime Database são armazenados como objetos JSON. Desta forma, a base de dados é representada por uma árvore JSON hospedada na nuvem.

Juntamente com outro produto, o *Authentication*, o Realtime Document Database disponibiliza uma linguagem de regras flexíveis baseadas em expressão, denominadas regras de segurança. Esta linguagem permite definir como os dados são estruturados, quem tem acesso, a quais dados tem acesso e como esses dados podem ser alcançados.

2.3.2 Authentication

O serviço de autenticação é utilizado em diversas aplicações que necessitam conhecer a identidade de um usuário. Este conhecimento permite que a aplicação armazene de forma segura dados do usuário na nuvem e ainda ofereça a mesma experiência personalizada para as funcionalidade e dispositivos daquele usuário.

Para realizar a autenticação, o Firebase disponibiliza a ferramenta Firebase Auth (FIREBASE..., 2017), desenvolvida por equipes que auxiliaram na construção do login do Google, *Smart Lock* e *Chrome Password Manager*, por exemplo. Sua configuração no projeto pode ser feita com poucas linhas de código (aproximadamente 10(dez) linhas), e ainda evita empenho com tempo de desenvolvimento e manutenção de um sistema de autenticação próprio, além da redução de custos.

Com o Firebase Auth é possível gerenciar usuários de maneira simples e segura, com métodos de autenticação frequentemente utilizados, como o de e-mail/senha, autenticação por dispositivo e até mesmo por provedores de terceiros, como Facebook e Twitter.

2.3.3 Crash Reporting

O Crash Reporting é um serviço de notificação de erros do Firebase. Com ele é possível diagnosticar problemas nas aplicações com o auxílio de relatórios detalhados de falhas, com filtragem por frequência e gravidade de impacto. Com este serviço é possível ainda monitorar a integridade geral do sistema e os fluxos de usuários, além de disponibilizar o uso de notificações por e-mail.

Para usar o Crash Reporting, basta uma linha de código nas configurações do projeto, e as falhas nos dispositivos que utilizam seu sistema são capturadas automaticamente. Com isso é possível monitorar e tratar rapidamente os erros, evitando impactar muitos usuários e consequentemente uma má avaliação do sistema.

Os relatórios são gerenciados por um *dashboard* completo que possui gráficos e análises detalhadas possibilitando o agrupamento, segmentação e filtragem dos erros, auxiliando na avaliação minuciosa de tudo que ocorre enquanto o sistema é executado.

3 Desenvolvimento da Aplicação Móvel Utilizando Metodologia SCRUM

A aplicação foi desenvolvida utilizando-se a metodologia de desenvolvimento ágil SCRUM, por ser um projeto relativamente pequeno e pelo curto período de tempo definido para a entrega. A equipe de desenvolvimento foi composta por apenas duas pessoas, uma responsável pelo papel de Dono do Produto, no caso o cliente, e eu pelos papéis de Mestre Scrum e Time Scrum, responsável respectivamente pela aplicação da metodologia SCRUM e desenvolvimento do produto.

O contato próximo com o cliente facilitou a compreensão do problema e diminuiu o número de reuniões necessárias para a criação do Escopo do Produto (*Product Backlog*) e a definição dos Escopos dos Sprints (*Sprint Backlogs*).

3.1 Definição do Escopo do Produto (*Product Backlog*)

O Escopo do Produto foi definido e organizado priorizando-se as funcionalidades mais relevantes para o usuário final, levando-se em conta também o fluxo de execução e o custo de desenvolvimento de tais funcionalidades. As funcionalidades do sistema são ilustradas no Caso de Uso apresentado na Figura 4. Assim, a listagem dos requisitos do sistema pode ser definida por:

1. Salvar e alterar informações de clientes, como: nome completo, data de nascimento, telefones, endereço, foto, assim como todas as outras informações pessoais contidas no formulário “Dados pessoais” mostrado na Figura 7.
2. Visualizar as informações dos clientes salvos.
3. Remover determinado cliente, e consequentemente todos os dados referentes a esse cliente.
4. Criar e excluir pacotes de atendimento para os clientes cadastrados. No caso de exclusão, as sessões referentes a esse pacote devem também ser apagadas.
5. Acessar informações dos pacotes criados.
6. Criação e agendamento das sessões. Na criação do pacote deve ser disponibilizada a possibilidade de se agendar ou não, as sessões desse pacote. As sessões devem obrigatoriamente estar dentro de um pacote.

7. Visualização da agenda, mostrando os atendimentos diários.
8. Finalizar/cancelar sessões com a assinatura do cliente.
9. Salvar e permitir a visualização das medidas do cliente.
10. Salvar e permitir a visualização de contatos de emergência do cliente.
11. Salvar e possibilitar acesso à informações referentes ao histórico clínico do cliente, hábitos diários, etc., assim como às outras informações presentes no formulário “Histórico” conforme Figura 7.
12. Prover autenticação de usuários para acesso ao aplicativo.
13. Possibilitar a recuperação de acesso.
14. Possibilitar a recuperação de dados na nuvem para um determinado usuário: seus clientes, pacotes, sessões, etc.

A Figura 4 ilustra o Diagrama de Caso de Uso Geral que organiza e esquematiza os processos elicitados no Escopo do Produto, mostrando as interações dos mesmos com os atores *Cliente* e *Usuário* (que representam respectivamente o cliente/paciente e os profissionais de massoterapia).

3.2 Configuração do projeto

Antes de iniciar o desenvolvimento foi necessário organizar e configurar o projeto como um todo. O projeto foi criado com a estrutura padrão de um projeto Android através da IDE de desenvolvimento Android Studio (DEITEL; DEITEL; DEITEL, 2015) e adaptado para utilização do padrão de projeto DAO (CORE..., 2009) de acesso aos dados, e do padrão MVC (JAVA..., 2009) para a estruturação arquitetural do projeto, como mostrado na Figura 5.

O Android permite usar um conjunto de pacotes, contendo classes pré-definidas, que possibilitam o acesso aos recursos do sistema operacional e a incorporação dos mesmos nos aplicativos em desenvolvimento. Estes pacotes auxiliam no tratamento e uso das convenções de aparência, comportamento e diretrizes de estilo específicas de aplicativos do Sistema Android (DEITEL; DEITEL; DEITEL, 2015). Neste contexto, foram definidos padrões de cores utilizadas no layout, fonte dos textos, elementos de *Material Design* do Android e definidos os componentes visuais da aplicação como imagens de fundo de tela, padrão de fontes de texto e botões. Em seguida definiu-se quais bibliotecas seriam usadas no projeto.

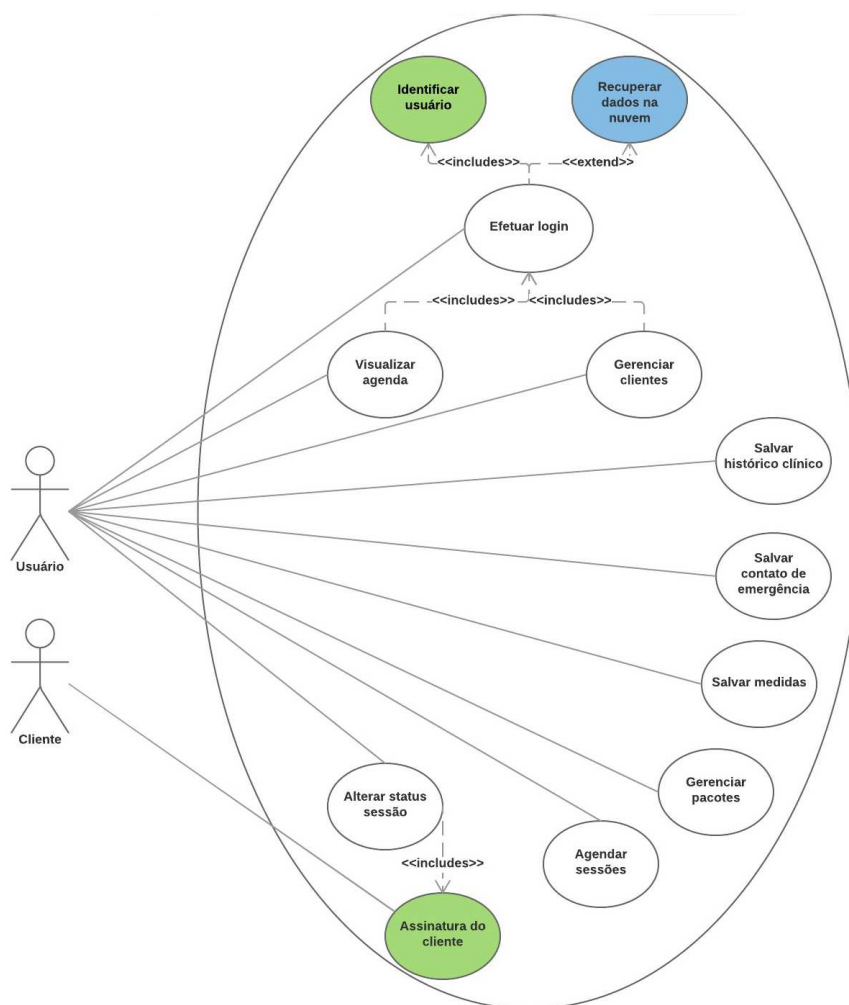


Figura 4 – Diagrama Caso de Uso Geral

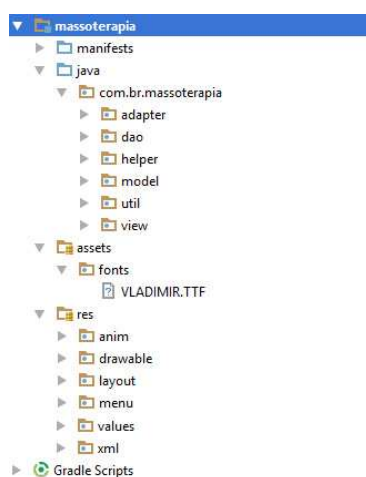


Figura 5 – Estrutura do Projeto

Para o armazenamento de dados, estabeleceu-se que todas as informações da aplicação seriam armazenadas utilizando-se o banco de dados SQLite, que seria responsável

por manter o registro das informações localmente nos dispositivos móveis, e o Firebase Realtime Database seria necessário para o armazenamento das informações em nuvem. Desta forma, o SQLite é responsável por registrar os dados que serão processados durante a execução da aplicação. A utilização do Realtime Database serve para manter um *backup* caso haja necessidade de recuperação dos dados salvos no SQLite. A estrutura do banco SQLite pode ser vista na Figura 6. Nela podem-se notar as tabelas definidas para o armazenamento dos dados pessoais de clientes, suas medidas físicas, seus contatos de emergência, os pacotes oferecidos e sessões associadas, e um histórico.

Para acesso a um serviço *Rest* via *webService*, como por exemplo, fazer busca de endereço por CEP no formulário de clientes, foi definida a utilização da biblioteca *Volley* (KHAN, 2015). No processo de coleta da assinatura dos clientes através da escrita na tela do dispositivo durante o fechamento de uma sessão, utilizou-se no projeto a biblioteca de código aberto *SignatureView* (SIGNATUREVIEW, 2017). As funções de carregamento e exibição de imagens foram confeccionadas com o auxílio das bibliotecas *Picasso* e *RoundedImageview* respectivamente.

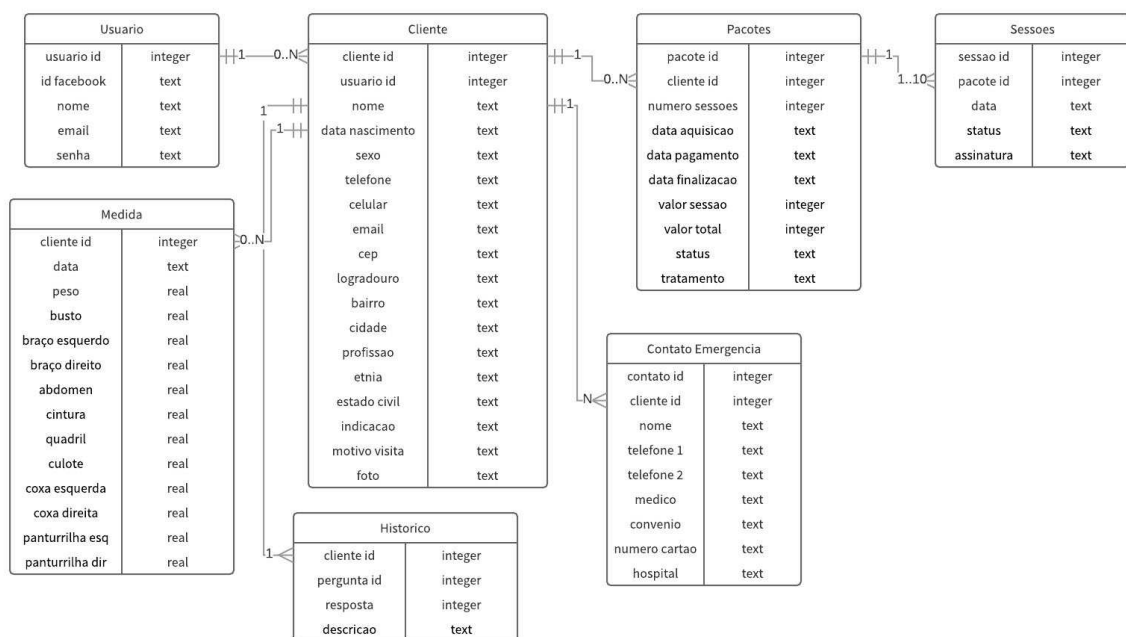


Figura 6 – Diagrama Entidade Relacionamento do Banco

3.3 Primeiro Sprint

Após a definição do Escopo do Produto (ou *Product Backlog*), a próxima etapa foi reunir o Time Scrum com o cliente do produto para a realização de uma reunião.

3.3.1 Primeira Reunião de Planejamento do Sprint

A primeira Reunião de Planejamento do Sprint com o cliente da aplicação foi feita para que as funcionalidades de maior prioridade fossem identificadas e para que se pudesse, através de questionamentos ao cliente, dividir tais funcionalidades em tarefas técnicas e definir o *Escopo do Primeiro Sprint*.

Na conversa ressaltou-se a importância para os usuários da aplicação de se ter as informações de seus pacientes, pelo fato de serem de extrema necessidade para a resolução do problema inicial e também para o desenvolvimento do restante do projeto. Assim definiu-se que o primeiro *sprint* contemplaria o desenvolvimento dos itens 1, 2 e 3 do Escopo do Produto. Na reunião ainda foi definido o prazo de 15 dias para a entrega do produto final do *sprint* e a realização da Reunião de Revisão do *sprint*.

[illegible]

Figura 7 – Ficha Anamnese Corporal

3.3.2 Armazenamento de dados dos Clientes

Seguindo como modelo a Ficha de Anamnese Corporal (Figura 7), as informações dos clientes que devem ser salvas no aplicativo são as mesmas da seção “Dados Pessoais” do formulário presente na ficha. A estrutura da tabela Cliente está representada no Diagrama de Entidade-Relacionamento da Figura 6.

Foi utilizada a classe abstrata *SQLiteOpenHelper* da API do Android para auxiliar na criação e no gerenciamento do banco SQLite, como especificado na documentação em

([SQLITEOPENHELPER, 2017](#)). Parte do código pode ser visto a seguir:

```
1 public class GerenciadorSqlite extends SQLiteOpenHelper {
2
3     // Versao do Banco
4     public static final int DATABASE_VERSION = 2;
5
6     // Nome do banco
7     public static final String DATABASE_NAME = "Massoterapia.db";
8     ...
9     public GerenciadorSqlite(Context context) {
10         super(context, DATABASE_NAME, null, DATABASE_VERSION);
11     }
12
13     @Override
14     public void onCreate(SQLiteDatabase db) {
15         db.execSQL(CREATE_TABLE_CLIENTE);
16     ...
17     }
18     private void dropTables(SQLiteDatabase db){
19         db.execSQL("DROP TABLE IF EXISTS " + TABELA_CLIENTE + ";");
20     ...
21     }
22     @Override
23     public void onUpgrade(SQLiteDatabase db, int oldVersion, int
24         newVersion) {
25         dropTables(db);
26         onCreate(db);
27     }
28 }
```

Para a realização das operações *insert*, *delete* e *update* no banco local SQLite foi utilizada a classe `SQLiteDatabase`. Com relação ao armazenamento em nuvem, a referência ao banco é dada pela classe `DatabaseReference` da API do Firebase, conforme mostra o trecho de código da classe `ClienteDAO`.

```
1 public class ClienteDAO {
2
3     private SQLiteDatabase database;
4     private GerenciadorSqlite gerenciadorSqlite;
5     private Context context;
```

```
6
7     private final String LOG = "ClienteDAO";
8     private FirebaseUser firebaseUser;
9     private FirebaseAuth mAuth;
10    private DatabaseReference databaseReference;
11
12    // Continua...
```

3.3.3 Formulário de Cliente

O formulário do aplicativo, além das informações da Ficha Anamnese Corporal (Figura 7), foi implementado para permitir o registro de uma foto do cliente caso o profissional deseje e o cliente permita. A imagem fica armazenada num diretório do aplicativo na memória do telefone, o que dificulta o acesso e sua visualização fora do aplicativo pois é gravado num formato `.tif` em uma pasta desconhecida para os usuários comuns do sistema Android. Ao abrir o formulário, o profissional pode ativar a câmera clicando em um ícone sugestivo localizado no topo da tela. Após tirar a foto e solicitar que seja armazenada, o caminho de pastas completo para o registro da imagem é salvo em memória para visualizações futuras.

A coleta e o armazenamento dos dados no formulário foram tratados utilizando-se componentes do Sistema Android, da linguagem Java e do Google (SMYTH, 2017a). As informações textuais foram tratadas utilizando-se o componente `EditText` do Android, com exceção das informações pré-definidas como sexo, etnia e estado civil, que são coletadas utilizando o componente `Spinners` com opções que podem ser selecionadas pelo usuário. Dessas informações, ficou definido que apenas os campos nome, data de nascimento, sexo, telefone celular, logradouro e bairro teriam preenchimento obrigatório, ficando os demais como campos opcionais.

Foram previstas validações em alguns dos campos. Por exemplo, para o nome do cliente não é possível registrar nome com menos de três caracteres e sem sobrenome. Para o telefone será verificado se o número digitado possui 11 ou mais dígitos, padronizando com a estrutura mínima (operadora e telefone com 8 dígitos). No caso do e-mail deve-se verificar se contém as partes de usuário e provedor separadas pelo símbolo `@` e o símbolo `.` sendo utilizado para detalhar nome/entidade do provedor.

Outra funcionalidade desenvolvida para facilitar o preenchimento do formulário é a busca de endereço pelo número de CEP. Ao preencher esse campo uma busca automática deve ser iniciada, utilizando o serviço `Via CEP`¹, para trazer o nome de logradouro, bairro e país de acordo com o CEP informado. Se houver sucesso na busca, os campos são

¹ Serviço Web gratuito para consultar Códigos de Endereçamento Postal (CEP) do Brasil - <https://viacep.com.br>

preenchidos automaticamente, caso contrário uma mensagem é mostrada informando a necessidade de preenchimento manual.

Ao final do preenchimento dos dados, ao clicar no botão “Salvar”, um objeto **Cliente** é criado e enviado através da classe **ClienteDAO** e as informações são armazenadas no banco SQLite. Posteriormente ao registro do cliente, é chamada uma instância da API do banco do Firebase para que o objeto também seja registrado na nuvem.

A tela do formulário de cliente foi basicamente toda criada dentro de um componente **ScrollView** para que fosse possível a visualização de todo o formulário em qualquer dispositivo, já que o mesmo é extenso e ultrapassa o limite da tela dos dispositivos existentes atualmente.

Um componente **ImageView**, configurado com escuta na ação de um clique, foi utilizado para ativar a câmera caso desejado e salvar uma foto que identifique o cliente. Um **FloatingActionButton** foi o tipo de botão utilizado para receber o clique do usuário para validar uma ação, e caso tudo esteja correto, submeter as informações digitadas. O resultado visual final da tela de formulário está representado na Tela de Formulário de Cliente da Figura 8.

3.3.4 Listagem de Clientes Salvos

Após a gravação dos dados de um cliente, suas informações ficam disponíveis para consulta no banco local do dispositivo. A visualização reduzida das informações do cliente é feita através de uma tela composta por um componente Android *Listview*, que mostra uma listagem dos clientes salvos no banco. Para a visualização completa dos dados do cliente, foi criada outra tela, acessível através da seleção de um cliente na lista, que abre a Tela de Informação do Cliente, ilustrada na Figura 9 que mostra todos os dados pessoais do cliente selecionado.

Com o dispositivo na vertical (*Portrait Orientation*), apenas os nomes dos clientes são exibidos na lista, juntamente com um botão que possibilita a exclusão de cada um. Estando o dispositivo na posição horizontal (*Landscape Orientation*), além do nome, são exibidos para cada cliente da lista o número do celular e seu e-mail.

Caso um toque longo seja feito sobre algum cliente da lista, uma janela estilo *pop-up* com algumas opções de ações é mostrada para o usuário, como:

- Ligar para o usuário usando o número de celular salvo no cadastro;
- Enviar um SMS para o mesmo número;
- Abrir um mapa com a localização do endereço do cliente;
- Enviar um e-mail;

← Cadastro Cliente



Nome completo

Data de Nascimento

Sexo ▾

Telefone Residencial
(0xx)xxxx-xxxx

Telefone Celular
(0xx)xxxxx-xxxx

E-mail
email@provedor.com

CEP
00.000-000

Logradouro
Rua Aplicação, 100

Bairro

Cidade

Profissão

Etnia ▾

Estado Civil ▾

Indicação

Motivo da visita



Figura 8 – Tela Formulário Cliente

- Apagar o cliente selecionado da base de dados;
- Editar as informações pessoais do cliente selecionado;



← Info Cliente

Nome completo
Bruno Rodrigues Borges.

Data de Nascimento
08/05/1991.

Sexo
Masculino.

Telefone Residencial
.

Telefone Celular
(034)99991-9010.

E-mail
brunorb0805@gmail.com.

CEP
38408354.

Logradouro
Rua João Caetano de Rezende.

Bairro
Segismundo Pereira.

Cidade
Uberlândia.

Profissão

Figura 9 – Tela de Informação do Cliente

Ao selecionar a opção de apagar um cliente, seja no botão da lista ou na opção de exibida após um clique longo na tela sobre o item da lista, um alerta é mostrado para o usuário pedindo confirmação para apagar o cliente. Em caso positivo, uma remoção em cadeia deve ocorrer em todas as tabelas do banco, apagando todas as linhas pertencentes ao cliente removido. Da mesma forma, deve haver a remoção no banco do Firebase, onde as referências ao objeto do cliente também devem ser excluídas.

Na tela de listagem dos clientes, ainda existe um botão do tipo *FloatingActionButton* que, ao ser selecionado, abre o formulário e possibilita o cadastro de outros clientes. O protótipo da tela de listagem, tanto na vertical como na horizontal pode ser visto na imagem da Figura 10.

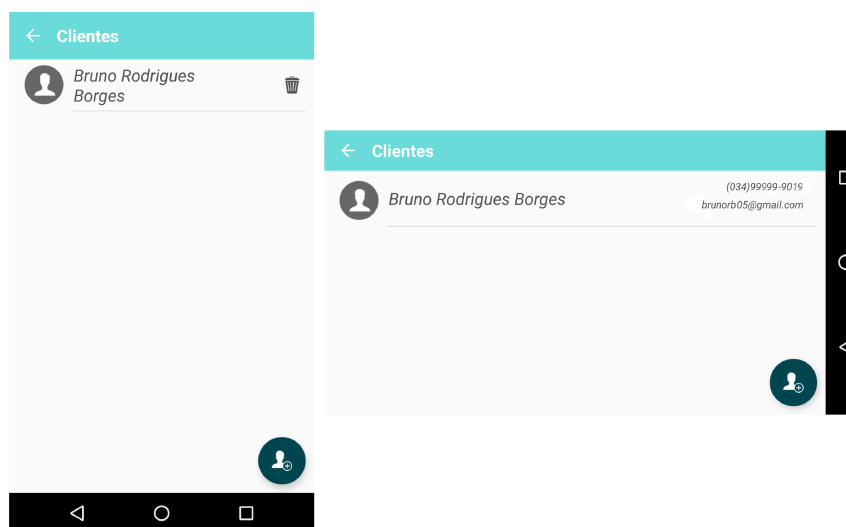


Figura 10 – Telas de Listagem de Clientes

3.3.5 Primeira Reunião de Revisão de Sprint

Ao final do primeiro *sprint* foi realizada a primeira Reunião de Revisão de Sprint para discutir e mostrar os objetivos alcançados naquele ciclo. Inicialmente o objetivo foi o desenvolvimento de toda a parte de gerenciamento de clientes do aplicativo.

A primeira atividade desenvolvida incluiu as funções para salvar e alterar informações de clientes de acordo com o formulário contido na Ficha Anamnese Corporal (Figura 7). A base de dados foi criada contendo uma tabela para armazenar os dados pessoais de clientes e foi implementada a tela contendo o formulário. Um pequeno problema na validação do formulário foi identificado e tratado prontamente.

A segunda atividade foi a exibição das informações salvas na base de dados. As camadas *Controller* e *View* foram desenvolvidas para que os dados pudessem ser visualizados em forma de lista, com a opção inicial de disponibilizar a visualização reduzida das informações de todos os clientes, e exibir as informações de um cliente de forma completa quando o mesmo for selecionado.

Na terceira atividade foi desenvolvida a funcionalidade de remoção de um cliente, considerando apenas a remoção de seus dados pessoais, devendo esta função ser revista para a remoção dos dados associados ao cliente (pacotes, sessões, histórico e medidas), cujas funções serão implementadas nos próximos ciclos.

3.4 Segundo Sprint

Com a conclusão do primeiro *sprint*, deu-se início à definição das funcionalidades que estariam presentes no segundo ciclo para a delimitação do Escopo do Segundo Sprint (*Sprint Backlog*). Esta atividade foi realizada durante a Segunda Reunião de Planejamento de *Sprint* (*Sprint Planning Meeting*).

3.4.1 Segunda Reunião de Planejamento de *Sprint*

Dentro do Escopo do Segundo Sprint ficou o desenvolvimento das funcionalidades de gerenciamento dos pacotes de atendimento, que são os itens 4, 5, 6 e 7 do Escopo do Produto (*Product Backlog*), apresentado na Seção 3.1. Esses itens foram escolhidos pois são de importância ímpar para a resolução do problema inicial e para o funcionamento do aplicativo, não sendo maior que o gerenciamento de clientes pois depende dele para ser implementado.

Atualmente as informações sobre pacotes de atendimento são escritas em uma seção da Ficha Anamnese Corporal (Figura 7), abaixo do formulário de medidas, onde são coletadas informações sobre número da sessão, data da sessão, tratamento previsto, entre outras. Da forma como foi definida a ficha em papel, só é possível guardar informações

de um único pacote por ficha, o que torna inviável esta tarefa, por ser necessário iniciar uma nova ficha, replicando dados do cliente e consumindo mais papel, à medida em que novos pacotes são adquiridos.

Dessa forma, ficou definido que para um único cliente poderão ser criados vários pacotes e para cada pacote várias sessões, diferentemente do mecanismo utilizado em papel, com várias fichas para um mesmo cliente. Neste contexto, definiu-se que durante a criação de um pacote já devem ser criadas as suas respectivas sessões de acordo com a quantidade escolhida.

3.4.2 Armazenamento de Pacotes e Sessões

Para o armazenamento das informações de pacotes e sessões, foi utilizada a mesma estrutura de gerenciamento vista na Seção 3.3.2. Para operações no banco SQLite local usou-se a classe `SQLiteDatabase`, e para armazenamento na nuvem a classe `DatabaseReference` da API do Firebase. O código abaixo mostra a utilização destas classes pré-definidas dentro da classe `PacoteDAO`, responsável pela criação de pacotes de terapia para os clientes, e da classe `SessaoDAO`.

```
1 public class PacoteDAO {
2
3     private SQLiteDatabase database;
4     private GerenciadorSqlite gerenciadorSqlite;
5     private Context context;
6
7     private FirebaseUser firebaseUser;
8     private FirebaseAuth mAuth;
9     private DatabaseReference databaseReference;
10
11     // Continua...
12
13 public class SessaoDAO {
14
15     private SQLiteDatabase database;
16     private GerenciadorSqlite gerenciadorSqlite;
17     private Context context;
18
19     private FirebaseUser firebaseUser;
20     private FirebaseAuth mAuth;
21     private DatabaseReference databaseReference;
22
23     // Continua...
```

A estrutura de armazenamento de sessões e pacotes foi implementada de formas diferente no banco SQLite local e na nuvem do Firebase. Localmente foram criados duas tabelas no banco, conforme o DER da Figura 6, uma para Pacotes e outra para Sessões, com cardinalidade de relacionamento (1:N), ou seja, um pacote com várias sessões. De forma remota, a estrutura NoSQL do banco do Firebase define, em forma de documento JSON, um Pacote contendo uma estrutura de lista para organizar todas as Sessões desse Pacote. Assim, não há uma estrutura individual para Sessões.

Para cada Pacote são registradas as informações:

- Id do pacote
- Id do cliente
- Quantidade de sessões do pacote
- Data de aquisição
- Data de pagamento
- Data de finalização
- Valor de cada sessão
- Valor total
- Status do pacote (Aberto, Fechado, Cancelado)
- Tratamento a ser executado

No caso de cada Sessão, deverão ser armazenados os seguintes itens:

- Id da sessão
- Id do pacote
- Id do cliente
- Nome do cliente
- Data de agendamento
- Status da sessão (Aberta, Fechada, Cancelada)
- Assinatura de finalização

3.4.3 Formulário de Pacotes

O formulário de Pacotes foi implementado para coletar informações tanto do Pacote quanto das Sessões a serem criadas para esse Pacote. Para capturar estas informações foram usados componentes do Android como **Spinners** para dados pré definidos, **EditText** para informações textuais e **CheckBoxes** para dados binários e de múltipla escolha.

Figura 11 – Tela do Formulário de Pacote

Neste formulário o usuário deverá informar qual tratamento será executado nas sessões, o valor de cada sessão e a quantidade delas no pacote. Deve também informar se o pacote já esta sendo pago no ato da contratação e fornecer detalhes sobre o agendamento de cada uma das sessões. Para agendar, o formulário possibilita selecionar os dias da semana em que as sessões serão executadas e ainda os horários, que podem ser fixos, com todas as sessões atendidas no mesmo horário em todos os dias escolhidos, ou variados para cada dia.

Especificamente neste formulário, todas as informações devem obrigatoriamente ser preenchidas. Assim, ao ser finalizado, um novo objeto **Pacote** é construído para que sejam mantidos: o valor da sessão, o valor do pacote (calculado de acordo com o valor e a quantidade de sessões definida), o tratamento, a data de pagamento (que deverá ser o dia corrente caso o **CheckBox** de “Pacote pago” esteja selecionado) e as informações de agendamento.

Na realização do agendamento é possível definir se os horários das sessões serão fixos ou variáveis dependendo de cada dia da semana. Caso seja fixo, o usuário seleciona um horário e os dias da semana em que as sessões ocorrerão. No caso da opção para dias/horas variáveis, a opção de selecionar o horário para cada dia deve ser disponibilizada com a

desmarcação de um `CheckBox` com o título “Horario fixo?”.

Quando o botão “Salvar” é acionado, uma agenda é gerada de acordo com os dados confirmados, e vários objetos **Sessões** são criados para o pacote de acordo com essa agenda. Caso nenhum horário ou dia da semana seja selecionado, as sessões são geradas sem agendamento. Por fim, tanto o objeto **Pacote** quanto o **Sessão** são inseridos no banco e disponibilizados para consulta.

A listagem abaixo mostra a função criada para gerar uma agenda, de acordo com as informações passadas pelo usuário, e retornar as sessões do pacote.

```

1  /*
2   * Cria agendamento de acordo com as entradas do cliente
3   */
4  private List<String> createAgenda( List<Horario> listDays, int
      numSessions ) {
5
6      List<String> listSchedule = new ArrayList<>();
7      List<Horario> auxList;
8      SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd");
9      Calendar c = Calendar.getInstance();
10
11     auxList = sortSchedulers( listDays, c.get(Calendar.
        DAY_OF_WEEK) - 1);
12
13     boolean bool = true;
14     int count = 0;
15     while( bool ) {
16         for (Horario h : auxList) {
17             while( c.get(Calendar.DAY_OF_WEEK) != weekdays[h.
                getDiaSemana()] ) {
18                 c.add(Calendar.DATE, 1);
19             }
20
21             listSchedule.add( sdf.format(c.getTime()) + h.
                getHora().replace(":", "") + "00" );
22             count++;
23             c.add(Calendar.DATE, 1);
24             if( count == numSessions ) {
25                 bool = false;
26                 break;
27             }
28         }
29     }
30 }

```

```
29         }  
30         return listSchedule;  
31     }  
32  
33     // Continua...
```

A função `createAgenda` recebe uma lista de objetos `Horario`, que possuem dia da semana e seus respectivos encaixes de horários, e a quantidade de sessões do pacote. Com essas informações faz uma verificação e ordena os dias a partir do dia da semana atual, de modo que o primeiro dia da lista seja o próximo em relação ao dia da criação do pacote. Desta forma, as sessões podem ser agendadas já no próximo dia da semana, após o dia atual.

3.4.4 Listagem e Gerenciamento de Pacotes e Sessões

Cada Pacote comprado por cliente deve ser armazenado na base de dados a partir do preenchimento dos dados na tela de formulário de Pacotes. Após armazenado, um pacote pode ser recuperado para exibição na tela. Esta funcionalidade foi definida utilizando-se o mesmo processo realizado para a exibição de dados pessoais de clientes, na forma de lista.

Dois *layouts* foram criados para visualização da lista na vertical e horizontal. Com a tela na vertical, a lista informa somente o tratamento definido em cada pacote e seu status na forma de texto e de imagem, além do botão para apagar o pacote. Se o dispositivo estiver posicionado na horizontal, ao invés do botão para apagar, são exibidas outras duas informações: a data da última atualização de status e o valor do pacote. O *layout* definido está apresentado na Figura 12.

Na tela de listagem de Pacotes, o usuário pode realizar três ações: apagar o pacote, acessar as sessões de um pacote e abrir o formulário para incluir um novo pacote. Ao clicar no botão de apagar para algum pacote da lista, uma mensagem deve ser mostrada solicitando a confirmação da remoção do pacote. Caso a remoção seja confirmada, tanto o pacote quanto suas sessões são removidos do banco de dados, além do elemento ser retirado da lista. Para acessar as sessões de um pacote basta clicar no item da lista que o representa, assim uma nova tela será aberta contendo uma relação de todas as sessões desse pacote. Ainda na tela de listagem de Pacotes, é possível abrir o formulário e salvar novos pacotes clicando em um botão no canto inferior direito da tela.

A Tela de Listagem de Sessão, mostrada na Figura 13, é aberta quando um pacote é selecionado. Neste momento são mostradas todas as sessões desse pacote ordenadas pela data de agendamento das mesmas, caso estejam agendadas. Nesta tela o usuário tem acesso a duas ações sobre as sessões. Quando uma sessão da lista é clicada essas as opções

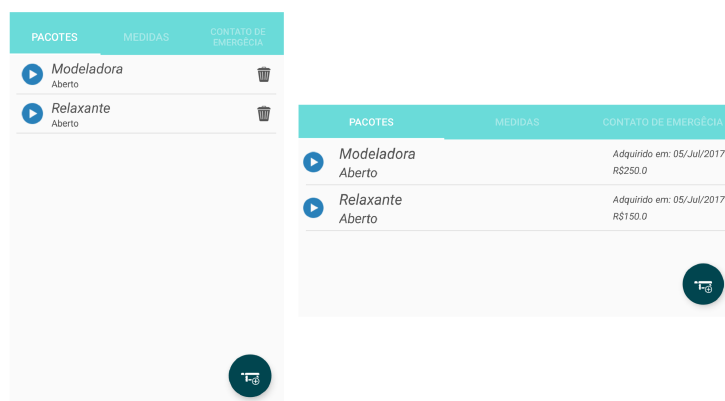


Figura 12 – Telas de Listagem dos Pacotes

aparecem em um componente `Dialog` (componente do Android). Este componente agrupa as ações para: “Finalizar”, caso o atendimento tenha sido concluído e “Cancelar”, caso seja necessário cancelar a sessão. Estas opções, já previstas neste *sprint*, foi implementada no ciclos seguintes.

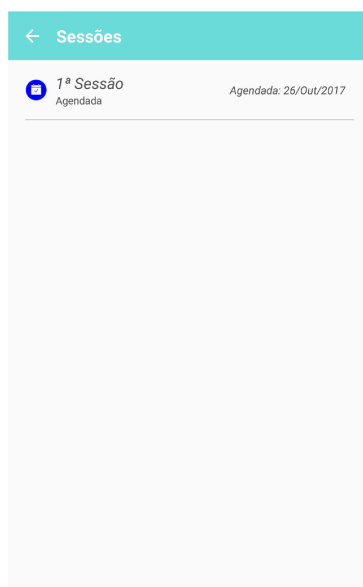


Figura 13 – Tela de Listagem de Sessão

3.4.5 Agendamento de Sessões

O agendamento das sessões é feito no próprio formulário de pacotes, onde são definidos os dias da semana de cada sessão e seus respectivos horários de execução. Assim, uma agenda é gerada e os pacotes/sessões são registrados de acordo com esse agendamento. Existe ainda a possibilidade da criação de sessões sem agendamento.

Após a criação dessas sessões, além da listagem das mesmas através da seleção do pacote que elas pertencem, uma nova tela foi confeccionada para que fosse possível a

visualização de todas as sessões agendadas para o usuário, sendo definida como a agenda, conforme Figura 14. Através dessa agenda é possível verificar quais os próximos atendimentos a serem efetuados no dia selecionado. Nessa tela também é possível finalizar e cancelar a sessão clicando sobre a mesma.

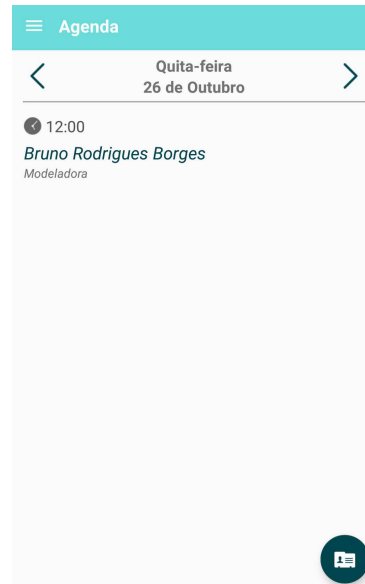


Figura 14 – Tela de Sessões Agendadas

3.4.6 Segunda Reunião de Revisão de Sprint

Para o encerramento do segundo *sprint* foi feita a segunda reunião para apresentação dos objetivos alcançados. Nesse *sprint* o objetivo foi a implementação da gestão de pacotes do aplicativo além da criação e visualização de sessões desses pacotes.

O primeiro objetivo foi desenvolver as funcionalidades de criação e remoção de pacotes e suas sessões. Para isso foi criado um formulário para que as informações iniciais sobre o pacote fossem inseridas e o pacote salvo na memória. No caso de remoção do pacote, foi implementada a remoção de suas sessões. O item foi concluído integralmente e sem observações por parte do cliente.

O segundo item a ser implementado foi o acesso às informações de pacotes. Para isso foi criada uma tela com componente de lista (**ListView**) em que são mostrados todos os pacotes de um determinado cliente e informações de tratamento e status desses pacotes. Esse requisito foi entregue 100% completo, mas com uma ressalva do cliente sobre a forma de listagem dos pacotes: as informações poderiam ser mais detalhadas. Tal ressalva indicou a definição deste detalhamento em uma futura atualização do aplicativo.

Para atender ao terceiro e quarto requisitos, implementou-se uma agenda. A criação do agendamento foi desenvolvida primeiro juntamente com o formulário de pacotes. Posteriormente, a tela de visualização dos agendamentos foi implementada. Além da lis-

tagem de todos os atendimentos agendados para o usuário, foi disponibilizada a possibilidade de gerenciamento desses agendamentos também através da tela de agenda. Com isso foi possível fazer todas as ações disponíveis na listagem de sessões (Finalizar, Cancelar) também através desta tela.

3.5 Terceiro Sprint

Terminado e entregue o segundo *sprint*, iniciou-se o processo de definição do próximo. E para dar início ao desenvolvimento foi marcada a Reunião de Planejamento do terceiro *sprint*.

3.5.1 Terceira Reunião de Planejamento de Sprint

Para o início do terceiro *sprint*, uma nova reunião foi feita para a definição dos requisitos que seriam desenvolvidos nessa nova etapa. Definiu-se que para a finalização do gerenciamento de sessões, o item 8 do Escopo do Produto (Seção 3.1) faria parte desse *sprint*, e junto com ele, os item 9, 10 e 11 que tratam dos gerenciamentos de medidas, contatos de emergência e histórico clínico dos clientes respectivamente.

No formulário da Ficha Anamnese Corporal, detalhado na Figura 7, após a finalização de cada sessão é feita uma validação do atendimento com a assinatura do cliente, comprovando a conclusão do mesmo. Assim, definiu-se que seria necessário e de grande importância e valor para a aplicação a implementação de uma forma de recolhimento dessa assinatura através da tela do dispositivo.

Para finalizar, fazendo com que o aplicativo contemple todos os formulários da ficha, a implementação dos requisitos sobre medidas, contato de emergência e histórico do cliente foi proposta nesse *sprint*.

3.5.2 Gerenciamento de Sessões

Para finalizar o gerenciamento das sessões, já iniciado no *sprint* anterior, o desenvolvimento do fechamento das sessões através da coleta de assinatura do cliente foi desenvolvido nesse terceiro *sprint*.

Para este fim foi utilizada uma biblioteca chamada **SignatureView** que, através do toque na tela do dispositivo, desenha o caminho desse toque e consegue capturar a escrita na tela. Para o uso desta biblioteca foi necessária apenas a importação da mesma nas dependências definidas no arquivo `build.gradle` do projeto:

```
1 dependencies {  
2     ...  
3     compile 'com.kyanogen.signatureview:signature-view:1.0'
```

```
4      ...  
5  }
```

Essa biblioteca disponibiliza um componente de interface, como uma lousa branca, capaz de ler os toques na tela e gerar uma resposta visual para o usuário. Essa interface ainda possui vários parâmetros de customização dos quais foram usados:

- `sign:penSize` - Define a espessura do risco gerado pelo toque na tela;
- `sign:backgroundColor` - Define a cor de fundo da “lousa”;
- `sign:penColor` - Define a cor do risco na “lousa”.

Na camada Controller foram usadas duas funções de biblioteca, uma para captura do conteúdo do *background* do componente de interface e outra para limpá-lo. Para a captura foi utilizada a função `getSignatureBitmap()` que retorna o objeto *Bitmap* que possibilitou a validação da assinatura, verificando se houve alguma alteração no componente de escrita e também a captura do que foi escrito. Para limpar o conteúdo escrito na tela, foi utilizada a função `clearCanvas()`.

Com as ferramentas acima citadas foram desenvolvidas as Telas de Finalização da Sessão, conforme mostra a Figura 15. De acordo com a opção selecionada pelo cliente (Finalizar ou Cancelar a sessão) ou status da sessão, essa tela se mostra de uma forma diferente. Em ambos os casos, são discriminadas na tela as informações de:

- ID da sessão
- Nome do cliente
- Data (Dia atual se o status estiver aberto ou dia da mudança do último status)
- Tratamento do pacote

As mudanças ocorrem no título, no botão de confirmação e no componente de escrita da tela. O título é definido como “Finalizar sessão”, caso essa opção seja selecionada pelo usuário, “Cancelar sessão” na escolha de cancelar e “Info sessão” no caso do status ser diferente de “Aberto”. Com o status “Aberto”, a opção de escrita é ativada, o botão de confirmação é mostrado, assim como uma linha de referência para a assinatura e um outro botão para limpar a tela. No caso de outro status, apenas a representação da assinatura é mostrada.

Com a assinatura recolhida e o botão de confirmação clicado, várias alterações são feitas na sessão, notadamente no status, na data de mudança desse status e na assinatura em si, que é então registrada. O usuário é redirecionado para a tela de listagem de sessões.



Figura 15 – Telas de Finalização de Sessões

3.5.3 Gerenciamento das Medidas

Outro formulário presente na Ficha Anamnese Corporal (Figura 7) é o de medidas. Seguindo a mesma estrutura de gerenciamento de todas as informações salvas no aplicativo, as operações do banco SQLite local também são feitas usando-se a classe `SQLiteDatabase` e as operações do banco em nuvem utilizam a API do Firebase. A seguir podem ser vistos trechos de código da classe DAO de medidas.

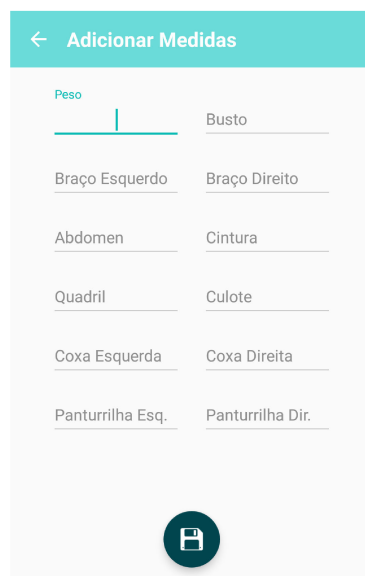
```

1 public class ListaMedidaDAO {
2
3     private SQLiteDatabase mDatabase;
4     private GerenciadorSqlite mGerenciadorSqlite;
5     private Context mContext;
6
7     private FirebaseUser mFirebaseUser;
8     private FirebaseAuth mAuth;
9     private DatabaseReference mDatabaseReference;

```

Nesse formulário são informadas as medidas do corpo do cliente para que possam ser comparadas ao fim de cada tratamento, com o intuito de demonstrar resultados numéricos ao cliente. Estas medidas podem ou não ser retiradas, dependendo da região do tratamento. Por isso o preenchimento de todos os campos nesse caso não é obrigatório no formulário.

Na aplicação, a representação desse formulário pode ser vista na Tela Formulário de Medidas, ilustrada na Figura 16. Foram usados componentes `EditText` para que o usuário entre com as medidas retiradas. Estes campos recebem apenas números e possui uma máscara de valores flutuantes com uma casa decimal apenas(###,##). Após o usuário entrar com as medidas que deseja, com o clique no botão salvar, na parte inferior central da tela, as informações passadas são salvas no banco através da classe DAO e disponibilizadas para as consultas futuras.



← Adicionar Medidas

Peso

Busto

Braço Esquerdo

Braço Direito

Abdomen

Cintura

Quadril

Culote

Coxa Esquerda

Coxa Direita

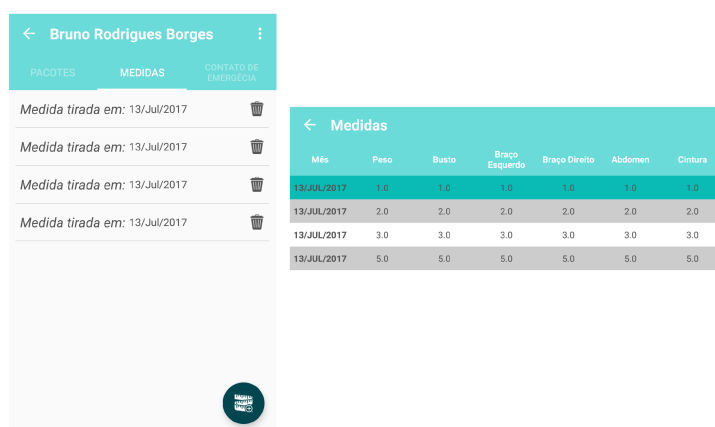
Panturrilha Esq.

Panturrilha Dir.

Salvar

Figura 16 – Tela Formulário de Medida

Quando armazenadas, as medidas são exibidas na forma de lista em uma **ListView** dentro de um **Fragment** na mesma tela dos pacotes, de forma decrescente a partir da data de retirada da última medida. Para cada componente dessa lista é mostrada a data em que a medida foi retirada, e ainda existe um botão disponível para que a medida seja removida. Com o clique sobre um item da lista, são mostradas todas as informações dessa medição e das demais já retiradas, em uma tela de comparação. Dessa forma pode ser feita uma análise das medidas e verificar se houve algum resultado em relação as medidas subsequentes à selecionada na lista. Essa duas telas podem ser vistas na Figura 17.



← Bruno Rodrigues Borges

PACOTES MEDIDAS CONTATO DE EMERGÊNCIA

Medida tirada em: 13/Jul/2017

Medida tirada em: 13/Jul/2017

Medida tirada em: 13/Jul/2017

Medida tirada em: 13/Jul/2017

← Medidas

Mês	Peso	Busto	Braço Esquerdo	Braço Direito	Abdomen	Cintura
13/JUL/2017	1.0	1.0	1.0	1.0	1.0	1.0
13/JUL/2017	2.0	2.0	2.0	2.0	2.0	2.0
13/JUL/2017	3.0	3.0	3.0	3.0	3.0	3.0
13/JUL/2017	5.0	5.0	5.0	5.0	5.0	5.0

Salvar

Figura 17 – Telas de Listagem de Medidas

3.5.4 Gerenciamento do Histórico Clínico

O Histórico Clínico do cliente é um formulário bem extenso da Ficha Anamnese Corporal, conforme apresentado na Figura 7, que contempla perguntas a respeito de

comportamento diário até cirurgias feitas pelo cliente. Com essas informações o usuário profissional consegue programar as terapias que serão aplicadas e o que será usado durante o tratamento.

Este formulário foi dividido em três telas, como mostra a Figura 18, e após seu preenchimento o histórico é salvo e pode ser consultado sempre que necessário. Esse formulário fica acessível na aplicação na tela de cliente, no menu do topo da tela, e caso o mesmo já tenha sido preenchido, suas informações são mostradas completamente em uma tela.

Cadastro Histórico

Costuma permanecer muito tempo sentado(a)?
☐ Sim ☒ Não

Antecedentes cirúrgicos?
☐ Sim ☒ Não Qualis?

Tratamento estético anterior?
☐ Sim ☒ Não Qual?

Antecedentes alérgicos?
☐ Sim ☒ Não Qualis?

Funcionamento intestinal regular?
☐ Sim ☒ Não Obs.:

Pratica atividade física?
☐ Sim ☒ Não Qualis?

É fumante?
☐ Sim ☒ Não

Alimentação balanceada?
☐ Sim ☒ Não Tipo?

Cadastro Histórico

Antecedentes oncológicos?
☐ Sim ☒ Não Qual?

Ciclo menstrual regular?
☐ Sim ☒ Não Obs.:

Usa método anticoncepcional?
☐ Sim ☒ Não Qual?

Varizes?
☐ Sim ☒ Não Grau?

Lesões?
☐ Sim ☒ Não Qualis?

Hipertensão?
☐ Sim ☒ Não

Hipotensão?
☐ Sim ☒ Não

Epilepsia?
☐ Sim ☒ Não

Cadastro Histórico

É gestante?
☐ Sim ☒ Não

Filhos?
☐ Sim ☒ Não Quantos?

Tem algum problema ortopédico?
☐ Sim ☒ Não Qual?

Faz algum tratamento médico?
☐ Sim ☒ Não Qual?

Usa ou já usou ácido na pele?
☐ Sim ☒ Não Qualis?

Já fez algum tratamento ortomolecular?
☐ Sim ☒ Não Qual?

Cuidados diários e produtos em uso?
☐ Sim ☒ Não Qual?

Portador de Marcapasso?
☐ Sim ☒ Não Qual?

Figura 18 – Telas de Formulário de Histórico

Como o formulário é basicamente composto por perguntas de respostas binárias, foram utilizados componentes **RadioGroup** com as opções “Sim” e “Não” para que o usuário registre as respostas das perguntas. Para aquelas que possuem um campo adicional para a entrada de resposta textual foram usados componentes **EditText** para a captura da resposta. Nenhuma das telas possui validação nos campos por não ser necessário responder a todas as perguntas e a navegação delas é feita sequencialmente sem a possibilidade de voltar. Chegando ao fim, as informações preenchidas podem ser salvas no banco e visualizadas na tela mostrada na Figura 19.

Histórico

Costuma permanecer muito tempo sentado(a)?
 Não.

Antecedentes cirúrgicos?
 Não.

Tratamento estético anterior?
 Não.

Antecedentes alérgicos?
 Não.

Funcionamento intestinal regular?
 Não.

Pratica atividade física?
 Não.

É fumante?
 Não.

Alimentação balanceada?
 Não.

Ingere líquidos com frequência?
 Não.

É gestante?
 Não.

Filhos?
 Não.

Antecedentes oncológicos?
 Não.

Ciclo menstrual regular?
 Não.

Usa método anticoncepcional?
 Não.

Varizes?
 Não.

Lesões?
 Não.

Hipertensão?
 Não.

Hipotensão?
 Não.

Epilepsia?
 Não.

Filhos?
 Não.

Tem algum problema ortopédico?
 Não.

Faz algum tratamento médico?
 Não.

Usa ou já usou ácido na pele?
 Não.

Já fez algum tratamento ortomolecular?
 Não.

Cuidados diários e produtos em uso?
 Não.

Portador de Marcapasso?
 Não.

Presença de metais?
 Não.

Figura 19 – Tela de Visualização de Histórico

A estrutura da tabela de histórico, que pode ser vista no diagrama da Figura 6, foi

pensada para salvar todas as respostas para as perguntas referentes a todos os clientes em uma mesma tabela. Durante o preenchimento do formulário, um objeto `HistoricoCliente` vai persistindo todos os objetos `ItemHistoricoCliente` (que representam cada resposta do questionário para um cliente) em uma lista. Ao final do preenchimento esta lista é percorrida e cada item é salvo em uma linha da tabela contendo o ID do referido cliente, o ID da pergunta, e a resposta, com a escolha selecionada e sua descrição.

A seguir são mostradas partes do código fonte do projeto referentes às classes `HistoricoCliente`, `ItemHistoricoCliente` e `HistoricoDAO` respectivamente.

- Classe `HistoricoCliente`:

```
1 public class HistoricoCliente implements Serializable{
2
3     private static final long serialVersionUID = 1L;
4
5     private Long mIdCliente;
6     private List<ItemHistoricoCliente> mListHiscCliente = new
        ArrayList<ItemHistoricoCliente>();
7
8
9     public ItemHistoricoCliente getItemHistCliente(int pos){
10         return listHiscCliente.get(pos);
11     }
12
13     ... Continua
```

- Classe `ItemHistoricoCliente`:

```
1 public class ItemHistoricoCliente implements Serializable {
2
3     private static final long serialVersionUID = 1L;
4
5     private int mResposta;
6     private String mDescricao;
7
8
9     public int getResposta() {
10         return resposta;
11     }
12
13     public void setResposta(int resposta) {
14         this.resposta = resposta;
15     }
```

```
16
17     public String getDescricao() {
18         \\tratativa para resposta que nao possuem descricao
19         if( descricao == null ) {
20             return "null";
21         }
22         return mDescricao;
23     }
24
25     public void setDescricao(String descricao) {
26         this.mDescricao = descricao;
27     }
28 }
```

- Classe HistoricoDAO:

```
1 public class HistoricoDAO {
2
3     private SQLiteDatabase mDatabase;
4     private GerenciadorSqlite mGerenciadorSqlite;
5     @SuppressWarnings("unused")
6     private Context mContext;
7
8     private FirebaseUser mFirebaseUser;
9     private FirebaseAuth mAuth;
10    private DatabaseReference mDatabaseReference;
11
12    ... Continua
```

Como pode-se notar na parte de código contida na classe `HistoricoDAO`, a estrutura de gerenciamento das informações também aplicada às classes de armazenamento anteriores foi adotada nesse caso.

3.5.5 Gerenciamento de Contatos de Emergência

Finalizando-se os formulários de entrada de informações, foi desenvolvido o gerenciamento de contatos de emergência. Esse contato é usado pelo usuário profissional como um auxílio no caso de uma emergência durante o atendimento.

Na Ficha Anamnese Corporal, Figura 7, é possível o cadastramento de apenas um contato para casos de emergência. Na aplicação o usuário poderá salvar quantos contatos forem necessários. As informações salvas no aplicativo são as mesmas presentes na ficha de papel:

- Nome do contato
- Telefone de contato 1
- Telefone de contato 2
- Nome de um médico
- Convênio
- Número de convênio
- Nome de algum hospital

Dessas informações, apenas o nome do contato e um número de telefone têm preenchimento obrigatório. Com apenas um contato de emergência pode-se finalizar o cadastro.

A estrutura da tabela de contatos de emergência também pode ser vista no DER da Figura 6, e da mesma forma dos outros formulários, as informações são salvas utilizando-se a estrutura presente em todos os formulários até aqui, com `SQLiteDatabase` para o armazenamento interno e para armazenamento na nuvem a classe `DatabaseReference` da API do Firebase.

A tela de formulário dos contatos de emergência pode ser vista na Figura 20 e possui apenas entradas textuais. Com isso foram usados apenas componentes `EditText` para o preenchimento das informações. Existe uma validação tanto no nome como nos telefones inseridos para checar se o nome tem um padrão “Nome” e “Sobrenome” e o nome não possua menos de 3 caracteres. Já para o telefone verifica-se a existência de no mínimo 11 dígitos (8 do número e 2 do DDD). O botão padrão da aplicação foi usado para que o usuário salve as informações no banco.

Após salvar as informações o usuário poderá acessar os contatos de emergência em uma lista na tela principal do cliente. Para este fim foi criada uma tela, conforme Figura 21, estruturada com um objeto `Fragment` contendo uma `ListView` para a exibição dos dados. Cada item dessa lista exibe apenas o nome do contato e o seu número principal para contato. Ao clicar em algum item da lista, uma ligação é imediatamente feita para esse número principal do contato. Cada item da lista também possui o botão para apagar, localizado no canto direito da tela.

3.5.6 Reunião de Revisão do Terceiro Sprint

Para finalizar o terceiro *sprint* foi realizada uma Reunião de Revisão para a avaliação e discussão dos objetivos alcançados e requisitos concluídos. Nessa fase do projeto

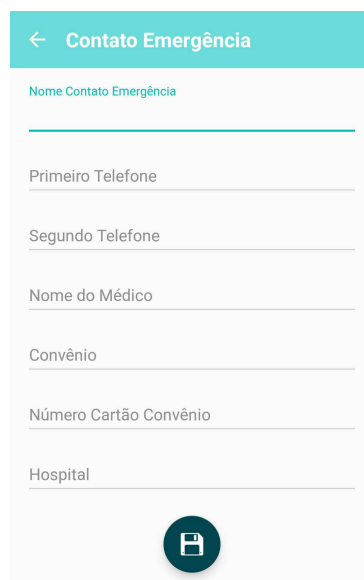


Figura 20 – Tela Formulário dos contatos de Emergência

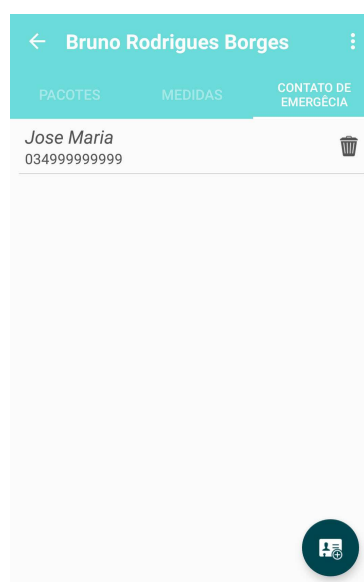


Figura 21 – Tela Listagem de Contatos de Emergência

ficaram definidos o desenvolvimento do gerenciamento de sessão, medidas, contato de emergencia e histórico do cliente.

O primeiro requisito desenvolvido foi a criação de pacotes e sessões. Este requisito foi concluído e, ao clicar em alguma sessão na lista de pacotes, o cliente pode finalizar e cancelar a sessão com o registro da sua assinatura digital, além de poder visualizar posteriormente as informações das sessões finalizadas ou canceladas.

O segundo requisito do *sprint* era a implementação do gerenciamento de medidas, banco de dados, controller e views, e o objetivo foi concluído completamente.

O terceiro ponto a ser tratado foi o gerenciamento de histórico clínico dos clientes.

Para isso foi desenvolvida também toda a parte de acesso ao banco de dados, controllers e programação da interface gráfica (views) do formulário e visualização de informações.

Já o quarto e último item desse *sprint* foi o gerenciamento de contatos de emergência. Neste ponto foi implementada toda a estrutura necessária para a persistência das informações dos contatos e também as telas de formulário e de listagem. O acesso aos contatos foi otimizado, podendo ser feita rapidamente uma ligação telefônica caso necessário.

3.6 Quarto Sprint

Para o último *sprint* desse projeto restaram três requisitos definidos no Escopo do Produto: a autenticação de usuários para acessar o aplicativo, a recuperação de senha e o mecanismo de recuperação de informações do *backup* em nuvem. A última Reunião de Planejamento de *sprint* foi realizada para a definição dos objetivos.

3.6.1 Quarta Reunião de Planejamento de Sprint

Nesta reunião houve a discussão da forma como seriam implementadas as funcionalidades do *sprint*, qual tecnologia a ser usada para autenticar usuários, tipo de autenticação, informações relevantes para o funcionamento do *backup* e forma de recuperação do acesso.

Para o desenvolvimento dessas funcionalidades ficou definido que seria usada a API do Firebase que disponibiliza uma infraestrutura completa e gratuita para diversos casos de uso, além de ser expansível, já que a aplicação mesmo sendo de pequeno porte, possui grande expectativa futura de evolução.

3.6.2 Autenticação de Acesso

Para a solução desse requisito foi utilizado o serviço **Authentication** da API do Firebase, mais especificamente o método de autenticação de e-mail e senha. Essa autenticação usa um e-mail, de qualquer provedor, que será o usuário de login e caso necessário, uma mensagem de recuperação de senha será enviada para este e-mail. A senha será usada para completar a autenticação do acesso. O cadastro é feito somente pelo aplicativo, sendo que as contas podem ser gerenciadas pelo administrador da aplicação pela interface do Firebase.

Foi criada uma interface gráfica para a tela de login, como mostra a Figura 22, com dois campos para que e-mail e senha sejam inseridos. A depender da situação do usuário, o aplicativo atua de formas diferentes, seja tentando fazer a autenticação e em caso de falha, verificando se o usuário deseja criar uma nova conta de acesso.

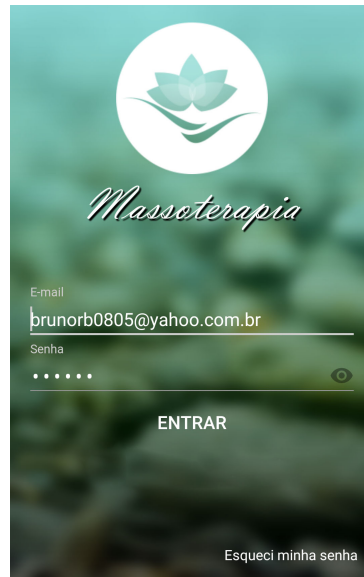


Figura 22 – Tela de Login

Caso seja a primeira vez do usuário no aplicativo, ele ainda não possuirá cadastro. Assim quando entrar com um e-mail e senha não conseguirá prosseguir. Em consequência, uma mensagem perguntando se o mesmo deseja criar uma conta será mostrada, e em caso de sucesso uma nova conta será criada usando o e-mail e senha passados. Após isso já será possível fazer o login normalmente no aplicativo e uma chave é gerada localmente para que ele consiga acessar a conta mesmo estando *offline*. Essa autenticação é feita localmente apenas nesse caso, sempre que houver conexão com a internet ela será feita pelo servidor.

A autenticação é feita usando a função `signInWithEmailAndPassword`, que recebe como parâmetros o login e a senha. Em caso de sucesso do login, o usuário é direcionado para a tela principal, e no caso de falha verifica-se se o erro foi falta de acesso ou falha na autenticação por senha inválida. O código abaixo mostra como isso é feito.

```

1 mAuth.signInWithEmailAndPassword(emailLogin, passLogin)
2   .addOnCompleteListener(LoginActivity.this, new
3     OnCompleteListener<AuthResult>() {
4       @Override
5       public void onComplete(@NonNull Task<AuthResult> task) {
6
7         if (!task.isSuccessful()) {
8           /*****
9            * FALHOU O LOGIN *
10          *****/
11           if( progressDialog != null || progressDialog.isShowing
12             () )
13             progressDialog.dismiss();

```

```

13
14         try {
15             throw task.getException();
16         } catch (FirebaseAuthInvalidUserException e) { //
17             Usuario nao cadastrado no firebase
18
19             *****
20             * PERGUNTA SE DESEJA CRIAR *
21             *****/
22             alerta.alertConfirmChoice(LoginActivity.this,
23                 getString(R.string.invalid_login_new_user),
24                 getString(R.string.btn_yes), getString(R.string
25                     .btn_no), new ClickAlertaInterface() {
26
27                 @Override
28                 public void metodoPositivo() {
29                     createNewUser(emailLogin, passLogin);
30                 }
31
32                 @Override
33                 public void metodoNegativo() {}
34
35             } );
36             return;
37         } catch (FirebaseAuthInvalidCredentialsException e) {
38             //Senha referente ao e-mail esta errado
39             edPassword.setError(getString(R.string.
40                 user_password_invalid));
41             edPassword.requestFocus();
42             return;
43         } catch (Exception e) {
44             Toast.makeText(LoginActivity.this, R.string.
45                 error_login, Toast.LENGTH_LONG).show();
46             return;
47         }
48     }
49
50     *****
51     * AUTENTICACAO COM SUCESSO, ACESSO PERMITIDO NO APP *
52     *****
53     else {

```

```

48         if( progressDialog != null && progressDialog.
           isShowing() )
49             progressDialog.dismiss();
50
51         ctrl.salvarUsuario( user );
52         user = ctrl.buscaUsuario(emailLogin);
53         enterApp();
54     }
55 }
56 });

```

No caso de falta de acesso, o cadastro pode ser feito. Para este fim, a API disponibiliza uma função (`createUserWithEmailAndPassword`) que recebe como parâmetro duas strings, e-mail e senha, respectivamente. O código abaixo exemplifica como é feito este cadastro.

```

1 mAuth.createUserWithEmailAndPassword(emailLogin, passLogin)
2     .addOnCompleteListener(LoginActivity.this, new
      OnCompleteListener<AuthResult>() {
3         @Override
4         public void onComplete(@NonNull Task<AuthResult> task) {
5             /* *****
6             * SE FALHAR a CRIACAO DE CONTA UMA MENSAGEM EH MOSTRADA*
7             ***** */
8             if (!task.isSuccessful()) {
9
10                if( progressDialog != null && progressDialog.
                  isShowing() )
11                    progressDialog.dismiss();
12                    Toast.makeText(LoginActivity.this, R.string.
                        error_login, Toast.LENGTH_LONG).show();
13
14            }
15            /* *****
16            * CONTA CRIADA COM SUCESSO O USUARIO *
17            * EH AUTENTICADO E ENTRA NO APP *
18            ***** */
19            else {
20                if( progressDialog != null && progressDialog.
                      isShowing() )
21                    progressDialog.dismiss();
22
23                if (firebaseUser.getUid() != null) {

```

```
24         ctrl.salvarUsuario( user );
25         user = ctrl.buscaUsuario( emailLogin );
26         enterApp();
27
28     } else {
29         Toast.makeText(LoginActivity.this, R.string.
30             error_login, Toast.LENGTH_SHORT).show();
31     }
32 }
33 });
```

Ainda na autenticação existe a tratativa de senha inválida. Neste caso, uma mensagem de aviso é mostrada na tela, permitindo ao usuário tentar novamente o acesso ou trocar a senha. Esta funcionalidade também está implementada nesse *sprint*.

3.6.3 Recuperação de Acesso

A recuperação de acesso é feita totalmente através do Firebase e usa o e-mail de cadastro para enviar o link de recuperação. Para a implementação desse requisito foi usada apenas uma função `FirebaseAuth.sendPasswordResetEmail("emailDoUsuario")`, que recebe como argumento o e-mail do usuário.

Para realizar a troca de senha no aplicativo, o usuário deve clicar no texto “Esqueci minha senha” no canto inferior direito na tela de login (Figura 22). Em seguida, um componente `Dialog` é mostrado para que o e-mail seja fornecido. Assim que for preenchido o e-mail, a função é chamada passando como parâmetro o e-mail informado, e caso ele esteja cadastrado na base, uma mensagem de recuperação é enviada para o usuário.

Ao acessar o e-mail recebido e clicar no link, o usuário é direcionado para uma tela de recuperação, e após efetuada a troca de senha, o acesso já pode ser concluído com a nova senha cadastrada. No caso de falha no envio da mensagem de recuperação, existe uma tratativa para “Erro de conexão” e outra para “E-mail inválido”. Nos demais casos uma mensagem de erro padrão é mostrada.

3.6.4 Recuperação de Dados

Outra funcionalidade feita com o uso da API do Firebase é a recuperação de dados. Como mostrado em todos os CRUDs (*Create*, *Read*, *Update* e *Delete*) de dados cadastrados no aplicativo, junto com a persistência local, também era persistidos os dados no banco externo na nuvem. Desta forma, os mesmos dados foram inseridos em memória local e remotamente. Isso foi feito para que fosse possível a recuperação posteriormente.

Como o próprio nome *Realtime Database* já diz, o banco do Firebase possibilita em tempo real à todas as alterações e inserções feitas. Porém, para a solução do problema atual, o uso dessa funcionalidade não foi necessária. O acesso aos dados é feito somente no caso do cliente logado no aplicativo possuir dados na nuvem e o banco de dados local esteja vazio. Isso ocorrerá quando o cliente por exemplo logar em um aparelho diferente do de costume, ou ainda por algum motivo ter apagado o aplicativo do celular. Em caso de remoção de dados pelo aplicativo a recuperação é comprometida, pois a operação de remoção exclui também os dados do banco na nuvem.

Para conseguir acessar o *backup* dos dados, após logar no aplicativo, é necessário selecionar a opção “Recuperar dados” no menu lateral da tela de agendamentos, que é responsável por acionar a recuperação dos dados existentes na nuvem. Desta forma, os dados são buscados na base do *Realtime Database* usando o id do usuário “logado” e são posteriormente incluídos no banco SQLite local.

O *Realtime Document Database* é um banco NoSQL orientado a documento, porém como ele foi usado apenas para backup e por recomendações da documentação do *Firebase*, o mesmo foi estruturado de uma forma não convencional, mas semelhante à estrutura relacional presente no banco SQLite local da aplicação. O banco remoto possui agrupamentos de documentos que representam as “tabelas”, e esses documentos são subdivididos por documentos que representam os usuários já cadastrados, ou seja, quando se salva um cliente na base, o mesmo é persistido no documento “Clientes” dentro do subdocumento do usuário que executou a ação de persistência. Já nos casos de Pacotes, Medidas, Histórico, por exemplo, além da subpasta do usuário, outro subdocumento é criado para referenciar o cliente a qual tal informação pertence. A figura 23 exemplifica como estão estruturadas as informações salvas no *Realtime Database*.

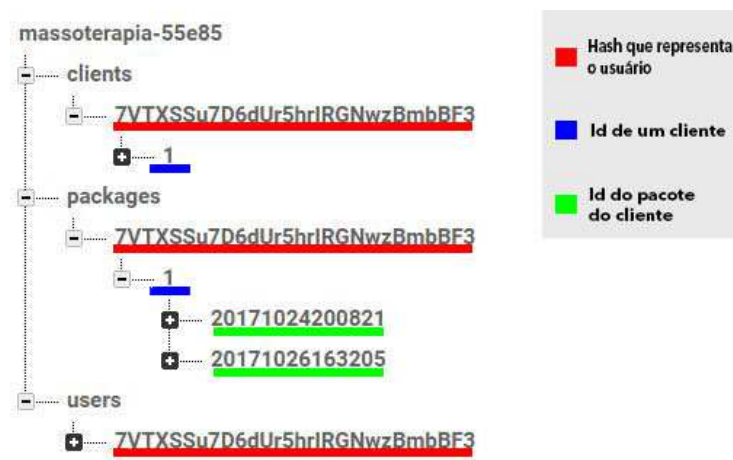


Figura 23 – Estrutura do armazenamento dos dados na nuvem (formato de documento)

No trecho de código abaixo é mostrada a recuperação de todos os clientes de um determinado usuário. Isso é feito com a sobrecarga do método `onDataChange` do padrão

observer do banco. O método `addListenerForSingleValueEvent` foi chamado, neste caso, para escutar todos os dados da árvore de clientes (`clients`) do usuário representado pelo valor em `userid`. Assim, caso existam dados, eles são retornados e inseridos, um a um no banco local.

```
1  /*
2  * RECUPERA CLIENTES DO USUARIO LOGADO
3  */
4  public void downloadBackup() throws InterruptedException {
5      final String userId = firebaseUser.getUid();
6
7      progressDialog = ProgressDialog.show(activity, null, "
          Recuperando dados...");
8      databaseReference.child("clients").child(userId).
          addListenerForSingleValueEvent(
9          new ValueEventListener() {
10
11              @Override
12              public void onDataChange(DataSnapshot dataSnapshot) {
13
14                  FormularioClienteController ctrlClient = new
                      FormularioClienteController(activity);
15                  Cliente c = null;
16
17                  for (DataSnapshot postSnapshot: dataSnapshot.
                      getChildren()) {
18                      c = postSnapshot.getValue(Cliente.class);
19                      ctrlClient.salvarCliente( c, true );
20                      downloadBackupPacotes( userId, c.getId() );
21                  }
22
23                  progressDialog.dismiss();
24                  activity.recreate();
25
26              }
27
28              @Override
29              public void onCancelled(DatabaseError databaseError) {
30                  Log.w(TAG, "getUser:onCancelled", databaseError.
                      toException());
31                  alert.exibirMensagem(activity.getString(R.string.
                      title_error),
```

```
32             activity.getString(R.string.  
33                 str_falha_backup_clientes));  
34         progressDialog.dismiss();  
35         activity.recreate();  
36     }  
37 }  
38 }
```

3.6.5 Quarta Reunião de Revisão de Sprint

Com o fim do período designado para o desenvolvimento do quarto *sprint*, a reunião para revisão e entrega dos requisitos foi feita. Para esse *sprint* estavam planejadas a implementação das funcionalidades de autenticação e a recuperação de acesso e de dados remotos. Esses três pontos foram desenvolvidos utilizando-se as ferramentas disponibilizadas pela API do Firebase e ambas foram totalmente concluídas dentro do prazo.

3.7 Qualidade do Software

Como o SCRUM é uma metodologia com foco no tempo de desenvolvimento, a qualidade do software pode ser afetada por essa busca pela agilidade na entrega. Porém o uso de uma metodologia ágil não implica em um software mal feito, pelo contrário, metodologias ágeis se bem implementadas garantem a criação e evolução do software buscando sempre a qualidade e geração de valor para o cliente no produto final.

Pensando nisso, a qualidade do aplicativo foi um ponto gerenciado desde o início do projeto. O código foi analisado usando a ferramenta *Analyze* do próprio Android Studio, que identifica várias possibilidades de falhas e indica boas práticas de programação. O uso da *Analyze* possibilita não deixar os testes de QA (*Quality Assurance*) somente para o final, visto que projetos ágeis são construídos de maneira iterativa e incremental. Os testes e a garantia da qualidade como um todo ocorreram em paralelo com o desenvolvimento do produto, sendo que desde a definição dos requisitos, os processos foram analisados e desenvolvidos com foco na qualidade do produto final.

No SCRUM, dois eventos centrais podem ser usados para gerenciar a qualidade: a Revisão de *Sprint* (*Sprint Review*) e a Retrospectiva de Revisões (*Retrospective Review*). No final de cada desenvolvimento de um requisito dentro dos *sprints*, casos de testes foram desenvolvidos e erros e inconformidades encontrados foram prontamente corrigidos. Em consequência, na Revisão de *sprint* aqueles requisitos desenvolvidos poderiam ser definidos com status concluído e consequentemente o *sprint* também. Já na Retrospectiva foram observados como esses objetivos foram alcançados, ou seja, como o desenvolvimento foi

feito. O sucesso de cada *sprint* foi usado para o desenvolvimento dos próximos ciclos e assim a qualidade foi garantida até a entrega.

3.8 Pesquisa de Usabilidade

Pensando em medir o valor agregado ao produto final, foi realizada uma pesquisa com alguns usuários da Ficha Anamnese Corporal (Figura 7) para uma simples avaliação de desempenho e usabilidade do aplicativo em comparação com a ficha, assim como a aceitação do desenho da interface gráfica.

Nesta avaliação foram usadas questões de múltipla escolha, e cada uma contendo de 3 a 5 opções de respostas. Abaixo estão relacionadas as questões apresentadas à um conjunto de usuários finais:

1. Qual sua opinião sobre usar o aplicativo para gerenciar os atendimentos em relação ao formulário de papel?
 - () Totalmente inútil
 - () Inútil
 - () Indiferente
 - () Útil
 - () Muito útil
2. O que você achou do visual, cores, ícones do aplicativo?
 - () Muito desagradáveis
 - () Desagradáveis
 - () Indiferente
 - () Agradáveis
 - () Muito agradáveis
3. Os menus do aplicativo são intuitivos?
 - () Nada intuitivos
 - () Um pouco intuitivos
 - () Bem Intuitivos.
4. Como você definiria as mensagens informativas, de erro e de aviso?
 - () Complexas e difíceis de entender
 - () Pouco intuitivas
 - () Indiferente
 - () Intuitivas

- () Muito intuitivas
5. Em relação a ficha de papel, os formulários podem ser preenchidos mais rapidamente?
- () Não, foram mais lentos
() Indiferente
() Sim, foi mais rápido
6. Considerando backup e acesso às informações por pessoas não autorizadas, você achou mais segura a utilização do aplicativo em relação à ficha de papel?
- () Não, achei mais inseguro
() Indiferente
() Sim, é mais seguro
7. Você achou o Massoterapia fácil de usar?
- () Muito difícil
() Difícil
() Normal
() Fácil
() Muito Fácil
8. Você utilizaria o Massoterapia no seu dia-a-dia?
- () Não utilizaria
() Utilizaria se houvesse melhorias
() Sim, Utilizaria

A amostragem da pesquisa foi pequena(dez pessoas), usuários do gerenciamento através da ficha de Anamnese em papel. Para responder essas questões os usuários antes fizeram procedimentos cotidianos com a ficha, e depois utilizando o aplicativo para isso. Os objetivos foram:

- Fazer login;
- Acessar a lista de clientes e adicionar um novo;
- Modificar os dados deste cliente;
- Adicionar um novo pacote e agendar sessões para este cliente;
- Salvar as medidas deste cliente;
- Salvar um contato de emergência;
- Acessar o formulário de histórico clínico;

- Acessar as informações deste cliente;
- Finalizar uma sessão do pacote criado;
- Finalizar/Cancelar uma sessão pela agenda;

As respostas dadas foram processadas e reunidas para que os resultados fossem representados nos gráficos abaixo.



Figura 24 – Pergunta 1

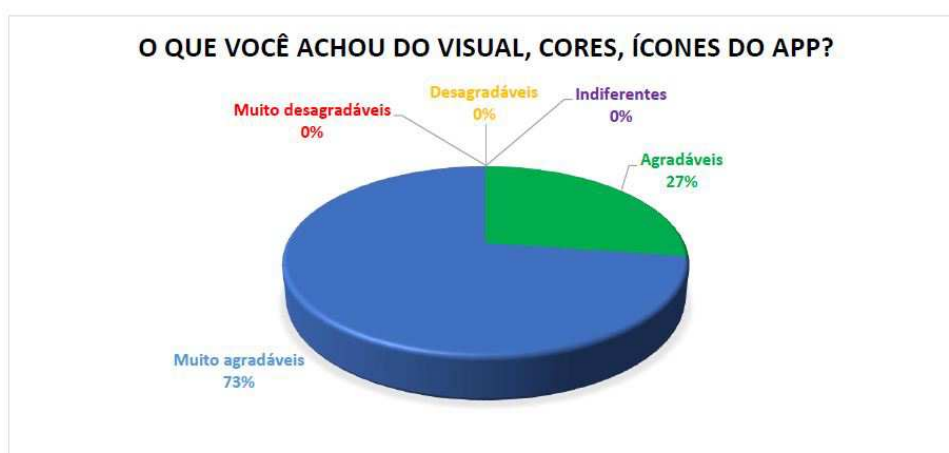


Figura 25 – Pergunta 2

Analisando os resultados da pesquisa pode-se ver que o aplicativo tem valor agregado e auxilia muito o trabalho do profissional no gerenciamento de seus clientes e atendimentos.



Figura 26 – Pergunta 3



Figura 27 – Pergunta 4



Figura 28 – Pergunta 5



Figura 29 – Pergunta 6

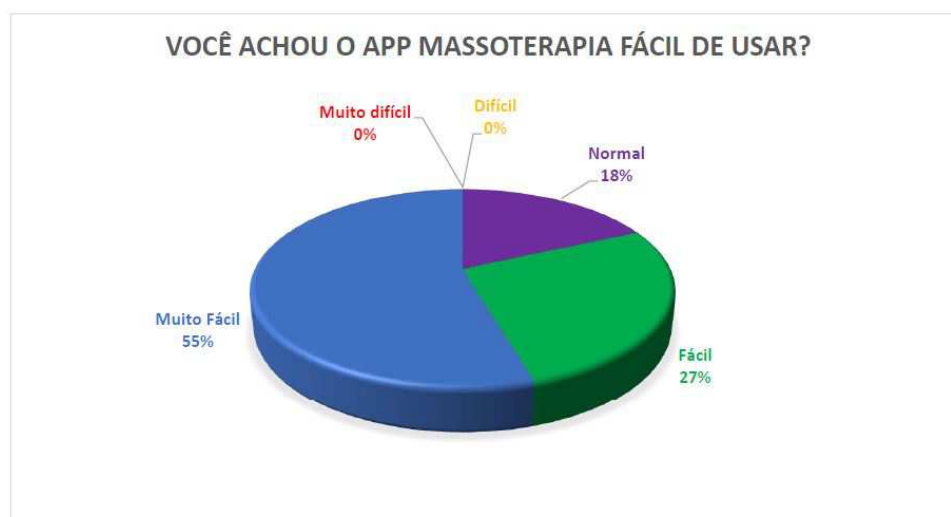


Figura 30 – Pergunta 7



Figura 31 – Pergunta 8

4 Conclusão

Após analisar e entender o problema inicial, ficaram perceptíveis os benefícios que a digitalização do processo através da aplicação móvel trouxe para os usuários da ficha de Anamnese em papel. Os problemas causados pela perda de fichas e consequentemente de informações dos clientes foram extintos com o backup em nuvem. O emprego de um aplicativo contendo todas as informações na palma das mãos dos terapeutas, otimizou muito o trabalho de gerenciamento, propiciando a união de ferramentas como agenda, ficha de Anamnese e recolhimento de assinaturas num só lugar.

O entendimento e uso de um método de desenvolvimento ágil ajudou muito no gerenciamento das entregas dentro do prazo e ainda contribuiu para assegurar a qualidade do produto final. Além disso, o uso de metodologia ágil colaborou para que a interação entre cliente e o desenvolvedor fosse clara e objetiva.

O estudo das tecnologias desenvolvidas para sistemas Android usadas na implementação do aplicativo possibilitou agregar novos conhecimentos na área. Do início do desenvolvimento até a etapa final ocorreram diversas otimizações de acordo com o que era estudado a respeito da plataforma e suas bibliotecas.

Após a finalização do desenvolvimento da primeira *release* do aplicativo e dos testes experimentais com profissionais da área, os resultados obtidos nos testes de usabilidade tornam evidente a contribuição desse trabalho para os massoterapeutas, podendo-se dizer que o problema inicial foi solucionado e os objetivos propostos para esse projeto de Trabalho de Conclusão de Curso foram alcançados.

Referências

ABLESON, F.; KING, C.; SEN, R. *Android em ação, 3a ed.* Elsevier Brasil, 2012. ISBN 9788535248418. Disponível em: <<https://books.google.com.br/books?id=zQLJY6KCFo8C>>. Citado na página 16.

ANDROID Data Storage. 2017. <https://developer.android.com/guide/topics/data/data-storage.html>. Acesso em Setembro de 2017. Citado na página 17.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <<http://www.agilemanifesto.org/>>. Citado na página 13.

CORE J2EE Patterns - Data Access Object. 2009. <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>. Acesso em Junho de 2017. Citado na página 22.

DEITEL, H.; DEITEL, P.; DEITEL, A. *Android: Como programar - 2ed.*. Bookman Editora, 2015. ISBN 9788582603482. Disponível em: <<https://books.google.com.br/books?id=aGMfCgAAQBAJ>>. Citado 2 vezes nas páginas 16 e 22.

FIREBASE Auth. 2017. <https://firebase.google.com/products/auth/>. Acesso em Setembro de 2017. Citado na página 20.

HOW Using Firebase Can Help You Earn More. 2017. <https://admob.googleblog.com/2016/11/how-using-firebase-can-help-you-earn-more.html>. Acesso em Setembro de 2017. Citado 2 vezes nas páginas 18 e 19.

JAVA SE Application Design With MVC. 2009. <http://www.oracle.com/technetwork/articles/javase/index-142890.html>. Acesso em Junho de 2017. Citado na página 22.

KHAN, B. *ANDROID Volley Tutorial Fetching JSON Data from URL*. 2015. <https://www.simplifiedcoding.net/android-volley-tutorial-fetch-json/>. Acesso em Setembro de 2017. Citado na página 24.

MORRIS, D. *Scrum in easy steps: An ideal framework for agile projects*. In Easy Steps, 2017. (In Easy Steps). ISBN 9781840787825. Disponível em: <<https://books.google.com.br/books?id=nkmkDgAAQBAJ>>. Citado 2 vezes nas páginas 13 e 15.

PRATICAS do SCRUM. 2017. <http://www.mindmaster.com.br/scrum/>. Acesso em Setembro de 2017. Citado 3 vezes nas páginas 6, 14 e 15.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software - 8ª Edição.*. [S.l.: s.n.], 2016. Citado na página 13.

REALTIME Database. 2017. <https://firebase.google.com/products/database/>. Acesso em Setembro de 2017. Citado na página 19.

SIGNATUREVIEW. 2017. <https://github.com/zahid-ali-shah/SignatureView>. Acesso em Junho de 2017. Citado na página 24.

- SMYTH, N. *Android Studio 2.3 Development Essentials - Android 7 Edition*:. CreateSpace Independent Publishing Platform, 2017. ISBN 9781544275437. Disponível em: <<https://books.google.com.br/books?id=SuxcDgAAQBAJ>>. Citado 2 vezes nas páginas 16 e 27.
- SMYTH, N. *Firebase Essentials - Android Edition*:. CreateSpace Independent Publishing Platform, 2017. ISBN 9781546660330. Disponível em: <<https://books.google.com.br/books?id=9i4tDwAAQBAJ>>. Citado na página 18.
- SQLITE. 2017. <https://www.sqlite.org/about.html>. Acesso em Setembro de 2017. Citado na página 17.
- SQLITEOPENHELPER. 2017. <https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>. Acesso em Julho de 2017. Citado na página 26.