Desenvolvimento de uma Aplicação Android para a Busca por Prestadores de Serviços

Maurício Rodríguez Roehrs[1], Mirkos Ortiz Martins[1]

[1]Curso de Sistemas de Informação – Universidade Franciscana
CEP 97010-032 – Santa Maria – RS – Brasil

mauricio.roehrs@ufn.edu.br, mirkos.martins@ufn.edu.br

Abstract. *This article presents the development of a mobile application, for Android, which concentrates the most diverse types of services, all arranged in a list, with the possibility to perform a search for the specialty and to classify them by a score scale, after the conclusion of service. The application aims to reduce the time to find and hire a service as the user needs also aims to replace classified ads in newspapers, for example, because there is nothing more practical than being able to find a suitable professional to solve the most diverse problems in the palm of your hand.*

Resumo. *O Presente artigo apresenta o desenvolvimento de um aplicativo móvel, para Android, que concentre os mais diversos tipos de serviços, todos dispostos em uma lista, com a possibilidade de realizar uma busca pela especialidade e classificá-los mediante uma escala de pontuação após o término do serviço. A aplicação visa se tornar outro meio para encontrar e contratar um serviço conforme a necessidade do usuário, tem o objetivo de substituir anúncios em classificados impressos em jornais, por exemplo, pois não há nada mais prático que ter a possibilidade de encontrar um profissional adequado, para solucionar os mais diversos problemas, na palma da mão.*

1. Introdução

A terciarização da economia é uma tendência observada mundialmente. Há uma mudança na estrutura produtiva (agropecuária, indústria e serviços) que torna o setor de serviços cada vez maior, tanto no que se refere a sua participação no Produto Interno Bruto (PIB) quanto no emprego das economias [Pereira 2014].

O IBGE (2018) afirma que dois grupos, somados, representaram 62,0% do número de empresas: serviços prestados principalmente às famílias, com 415,2 mil empresas ou 31,2%; e serviços profissionais, administrativos e complementares, com 409,9 mil empresas ou 30,8%.

O número de pessoas que trabalham por conta própria ou em vagas sem carteira assinada superou o daqueles que têm um emprego formal pela primeira vez em 2017, é o que apontam os dados divulgados pelo Instituto Brasileiro de Geografia e Estatística (IBGE) [Cury *et al*. 2018].

O ano de 2017 se encerrou com 34,31 milhões de pessoas trabalhando por conta própria ou sem carteira, contra 33,321 ocupados em vagas formais. Em 2016, cerca de 34 milhões trabalhavam sob o regime de CLT (Consolidação das Leis do Trabalho), contra

32,6 milhões ocupados em vagas sem carteira assinada ou como autônomos [Cury *et al*. 2018].

O avanço do trabalho sem carteira e por conta própria mostra o crescimento da informalidade na economia. O chamado "por conta própria" é uma categoria que inclui profissionais autônomos, como advogados e dentistas, mas também trabalhadores informais, como vendedores ambulantes [Cury *et al*. 2018].

1.1. Justificativa

De acordo com as pesquisas realizadas, foi analisado que o setor de serviços continua crescendo exponencialmente no Brasil, desse modo, levando em consideração a preocupação cada vez maior dos profissionais em prospectar novos clientes, este projeto irá apresentar o desenvolvimento de uma aplicação para dispositivo móvel para que estes profissionais sejam facilmente encontrados pela população em geral. Em função de se estar na era da conectividade, esta aplicação também visa substituir anúncios em classificados impressos em jornais, por exemplo, pois não há nada mais prático que ter a possibilidade de encontrar um profissional adequado, para solucionar os mais diversos problemas, na palma da mão.

1.2. Objetivo geral

O objetivo deste trabalho é desenvolver um aplicativo móvel, para Android, que concentre os mais diversos tipos de serviços, todos dispostos em uma lista, com a possibilidade de realizar uma busca pela especialidade e classificá-los mediante uma escala de pontuação após o término do serviço. A aplicação visa se tornar um meio alternativo para encontrar e contratar um serviço conforme a necessidade do usuário.

1.3. Objetivos específicos

- Fazer estudo das tecnologias necessárias para desenvolver o projeto;
- Fazer uso da linguagem Java com foco à dispositivos móveis;
- Fazer uso do banco de dados *Cloud Firestore*, Sistema Gerenciador de Banco de Dados (SGBD) para armazenamento das informações dos usuários e prestadores de serviços;
- Fazer uso do *Firebase Authentication* para controle de acesso ao aplicativo;
- Utilizar o *framework Viewmodel* e a classe *Livedata* a fim de controlar o ciclo de vida na interface de usuário e o componente *Navigation* para auxiliar na transação entre as telas do aplicativo;
- Utilizar a metodologia de desenvolvimento será FDD ("*Feature Driven Developement*").

2. Referencial Teórico

Esta seção tem por objetivo apresentar os conceitos para um melhor entendimento deste trabalho.

2.1. Serviços

Segundo o IBGE (2018), o setor de serviços é caracterizado por atividades bastante heterogêneas quanto ao porte das empresas, à remuneração média e à intensidade no uso de tecnologias. Nas últimas décadas, o desempenho das atividades que compõem o setor vem se destacando pelo dinamismo e pela crescente participação na produção econômica brasileira. Serviços são atividades onde o cliente não obtém a posse exclusiva do que foi adquirido, com exceção dos casos em que há contrato de exclusividade. Um serviço é o conjunto de atividades realizadas por uma empresa ou pessoa para responder às expectativas e necessidades do cliente.

Segundo Meirelles (2006), na análise conceitual dos serviços consiste em compreender que serviço é fundamentalmente diferente de um bem ou de um produto. Serviço é trabalho em processo, e não o resultado da ação do trabalho; por esta razão elementar, não se produz um serviço, e sim se presta um serviço.

2.2. Profissionais

A principal característica do profissional autônomo é que ele é um prestador de serviços e, portanto, não possui vínculo empregatício com nenhuma empresa [Santos 2018].

Os trabalhadores que se encaixam nessa categoria possuem autonomia econômica e profissional. Ou seja, desempenham suas atividades sem precisar, necessariamente, seguir regras específicas e modelos de trabalho das organizações. Além disso, eles assumem todos os riscos daquele serviço prestado, que pode variar conforme seus conhecimentos e habilidades [Santos 2018].

2.3. Tecnologias que foram utilizadas

Nesta subseção serão apresentadas todas as tecnologias que serão usadas no desenvolvimento deste trabalho.

2.3.1. Java

A linguagem Java tem sua própria estrutura, regras de sintaxe e paradigma de programação, assim como qualquer outra linguagem de programação, conforme Perry (2016). Ainda de acordo com o autor, a linguagem Java é baseada no conceito de Programação Orientada a Objetos.

Quanto à estrutura, a linguagem Java se divide em pacotes que contém classes, que podem conter métodos e/ou atributos. [Perry 2016].

2.3.2. Android

O Android é um sistema operacional que executa sobre o núcleo Linux. A plataforma Android permite que aplicativos sejam escritos em Java para controlar o dispositivo através de bibliotecas desenvolvidas pelo Google, com o foco de tornar esta plataforma flexível, aberta e facilitando a migração para os fabricantes de dispositivos [Petroni *et al*. 2014].

2.3.3. *Cloud Firestore*

O *Cloud Firestore* é um banco de dados NoSQL hospedado na nuvem que os aplicativos do iOS, do Android e da Web podem acessar diretamente por meio de *Software development kits* (SDKs) nativos [Cloud Firestore 2019].

Seguindo o modelo de dados NoSQL do *Cloud Firestore*, é possível armazenar dados em documentos que contêm mapeamentos de campos para valores. Esses documentos são armazenados em coleções, que são contêineres de documentos que podem ser utilizados para organizar dados e criar consultas. Os documentos são compatíveis com muitos tipos de dados diferentes, desde *strings* e números simples a objetos complexos e aninhados. Também é possível criar subcoleções dentro dos documentos e criar estruturas de dados hierárquicas que podem ser escalonadas à medida que o banco de dados cresce [Cloud Firestore 2019].

2.3.4. *ViewModel*

A Classe *ViewModel* foi projetada para armazenar e gerenciar dados relacionados à interface do usuário. A classe *ViewModel* permite que os dados sobrevivam a alterações na configuração, como rotações de tela [Android Developers 2019c].

Segundo o Android Developers (2019c), o *ViewModel* gerencia os ciclos de vida dos controladores da interface do usuário, como *activities* e *fragments*. O *framework* pode decidir destruir ou recriar um controlador de interface de usuário em resposta a determinadas ações do usuário ou eventos do dispositivo.

2.3.5. *Livedata*

De acordo com o Android Developers (2019a), a *LiveData* é uma classe de suporte de dados que pode ser observada em um determinado ciclo de vida. Isso significa que um observador pode ser adicionado a um par com um *LifecycleOwner* e esse observador será notificado sobre modificações dos dados agrupados apenas se o *LifecycleOwner* emparelhado estiver no estado ativo. O *LifecycleOwner* é considerado ativo se seu estado for *Lifecycle.State.STARTED* ou *Lifecycle.State.RESUMED*. Um observador adicionado via *observeForever(Observer)* é considerado como sempre ativo e, portanto, sempre será notificado sobre modificações. Para esses observadores, é necessário chamar manualmente *removeObserver(Observer)*.

Esta classe foi projetada para armazenar campos de dados individuais do *ViewModel*, mas também pode ser usada para compartilhar dados entre diferentes módulos em aplicativos de maneira dissociada [Android Developers 2019a].

2.3.6. *Firebase Authentication*

A maioria dos aplicativos precisa reconhecer a identidade do usuário. Ter essa informação permite que um aplicativo salve os dados do usuário na nuvem com segurança e forneça a mesma experiência personalizada em todos os dispositivos do usuário.

O Firebase Authentication (2019) diz que o *Firebase Authentication* fornece serviços de *back-end*, SDKs e bibliotecas de interface de usuário prontas para autenticar usuários no aplicativo. Ele oferece suporte à autenticação por meio de senhas, números de telefone e redes sociais, como Google, Facebook, Twitter e muito mais.

Para conectar um usuário ao aplicativo, primeiro é necessário ter as credenciais de autenticação do usuário. Essas credenciais podem ser o endereço de e-mail e a senha do usuário ou um *token* disponibilizado por uma rede social. Em seguida, são transmitidas essas credenciais para o SDK do *Firebase Authentication*. Os serviços de *back-end*, do *Firebase Authentication*, as verificarão e enviarão uma resposta ao cliente [Firebase Authentication 2019].

Após fazer *login*, o aplicativo terá acesso às informações básicas do perfil do usuário e é possível controlar o acesso do usuário aos dados armazenados em outros produtos do *Firebase*.

2.3.7. *Navigation*

O *Navigation* se refere às interações que permitem aos usuários navegar, entrar e sair de diferentes partes do conteúdo no aplicativo. O componente do *navigation* do Android ajuda a implementar a navegação, desde simples cliques em botões até padrões mais complexos, como barras de aplicativos e a gaveta de navegação. Esse componente também garante uma experiência do usuário consistente e previsível por meio da adesão a um conjunto de princípios estabelecido [Android Developers 2019b].

Segundo o Android Developers (2019b), o componente do *navigation* consiste de três partes principais. O primeiro, o Gráfico do *navigation*, é um recurso XML (*Extensible Markup Language*) que contém todas as informações relacionadas à navegação em um local centralizado. Isso inclui todas as áreas de conteúdo individual no aplicativo, chamadas destinos e todos os caminhos que podem ser percorridos pelo usuário no aplicativo.

O segundo, NavHost, é um contêiner vazio que mostra destinos do gráfico de navegação. O componente do *navigation* contém uma implementação NavHost padrão, NavHostFragment, que mostra os destinos do fragmento. E, por último, o NavController, que é um objeto que gerencia a navegação do aplicativo em um NavHost. O NavController organiza a troca do conteúdo de destino no NavHost conforme os usuários navegam pelo aplicativo [Android Developers 2019b].

2.4. Trabalhos correlatos

Nesta subseção serão apresentados os trabalhos relacionados ao sistema proposto, os quais possuem algumas características semelhantes ao presente trabalho, com o intuito de contribuir como base de conhecimento para a elaboração e organização desse projeto.

2.4.1. Aplicativo para contratação de diaristas: Donamaid

A plataforma, criada para disponibilizar o contrato de diaristas, nasceu na Universidade Federal de Pelotas (UFPel) e promete agilidade e segurança para quem contrata os profissionais. Depois de fazer um cadastro, o cliente pode selecionar - por meio do site - quantas horas de trabalho deseja contratar, que pode ser de uma a oito horas. O preço é calculado conforme a hora e varia entre R$29,90 e R$134,90. É possível contratar os serviços para limpezas em residências, pequenos escritórios e consultórios. Depois de selecionar a quantidade de horas, o cliente escolhe a data e a hora para contratar a faxina, que pode ser feita, inclusive, em domingos e feriados [Diário de Santa Maria 2017].

2.4.2. Aplicativo para agendamento de serviços: Triider

O Triider é uma plataforma que conecta prestadores de serviços especializados e clientes. São mais de 40 tipos de serviços para casa, escritório e até eventos reunidos no mesmo local. Nada mais é que uma lista de profissionais que vai desde o pedreiro para reformar a casa até um fotógrafo para a festa de formatura. Para garantir serviços de qualidade para os usuários, o Triider só seleciona os prestadores e os indica na plataforma após um controle de antecedentes e documentação. Além disso, pelo menos três referências de trabalhos anteriores são consultadas. Após realizar um rápido cadastro, que pode ser feito via redes sociais, é preciso escolher e detalhar o tipo de serviço desejado. Então, a plataforma compartilha a solicitação com os prestadores, que respondem com um orçamento. As respostas ficam disponíveis na página do cliente, que pode contatar o profissional via chat ou contratar o serviço imediatamente [Gaúcha ZH 2018].

2.4.3. Aplicativo para agendamento de serviços: SmartGest

O SmartGest possui dois aplicativos distintos, um direcionado ao cliente e outro ao profissional prestador do serviço.

O Aplicativo para o cliente serve basicamente para agendar horário com o profissional de sua preferência, mas antes é necessário realizar um cadastramento que pode ser feito por meio de redes sociais. Após o cadastro, basta selecionar a cidade, buscar, em uma lista corrida, os mais diversos prestadores disponíveis e agendar um horário.

Na versão profissional do aplicativo, o prestador realiza um cadastro informando nome da empresa, área de atuação, cidade, telefone de contato e e-mail. Após cadastrado, sua empresa já aparece para os clientes na versão do aplicativo voltado para os clientes que estiverem na mesma cidade. O cliente pode agendar um serviço ou, caso o cliente não possua o aplicativo, porém realizou um agendamento feito por telefone, o SmartGest serve como uma agenda, que, quando o profissional cadastrar o evento, selecionando uma data e horário, facilitará para o mesmo se organizar e também este horário agendado não estará disponível para os utilizadores do aplicativo versão cliente.

2.4.4. Análise sobre os trabalhos correlatos

Conclui-se que os aplicativos apresentados possuem o mesmo foco que o proposto pelo presente trabalho, que é a divulgação de serviços por parte dos profissionais e o agendamento destes serviços por parte dos clientes. A proposta é trazer esta facilidade de encontrar um profissional de forma rápida para a cidade de Santa Maria, onde somente o Donamaid disponibiliza o serviço, sendo este direcionado exclusivamente à contratação de diaristas. Mas não somente isso, propõe-se alguns diferenciais, como uma interface intuitiva, para que o cliente em poucos cliques possa realizar o agendamento com o profissional desejado, apresentar os profissionais mais, próximos dentro de um raio selecionado pelo usuário, possibilidade para o cliente marcar seus profissionais favoritos, agilizando uma contratação futura, bem como classificá-los para que outros clientes sintam segurança ao contratar um serviço.

3. Metodologia

Este projeto faz uso da metodologia *Feature Driven Development* (FDD). Segundo Goyal (2007), essa é uma metodologia considerada ágil, sendo assim, tem suporte para mudanças que venham a ocorrer no projeto.

O FDD é um processo de desenvolvimento de *software* ágil e altamente adaptável que enfatiza a qualidade em todos os passos, oferece resultados de trabalho frequentes e tangíveis em todos os passos. O fornecimento de informações precisas e significativas de progresso e status, com o mínimo de sobrecarga e interrupção para os desenvolvedores, é apreciado pelos clientes, gerentes e desenvolvedores [Goyal 2007].

Goyal (2007) diz que o FDD é compreendido em cinco processos. O primeiro é a construção do modelo geral, o segundo é construir uma lista de funcionalidades a serem desenvolvidas, o terceiro é o planejamento de cada funcionalidade individualmente, o quarto é o detalhamento das tarefas e, por fim, o quinto é o desenvolvimento das tarefas (o quarto e o quinto processo são executados de uma forma iterativa para cada funcionalidade).

3.1. Modelo abrangente

Nesta etapa é realizado um estudo detalhado sobre o domínio do negócio, além da definição do escopo do projeto [Silva e Miranda 2009]. A visão geral do aplicativo pode ser observada por meio do Diagrama de Domínio (Apêndice A).

3.2. Lista de funcionalidades

Com base no modelo abrangente, é realizada a coleta dos requisitos funcionais e não funcionais (Apêndice B) necessários para a construção e cumprimento das necessidades do cliente.

3.3. Detalhamento das funcionalidades

Nesta seção são detalhadas as principais funcionalidades do sistema. Segundo Heptagon (2017), tem como finalidade possibilitar o entendimento do fluxo de execução do sistema, quais as atividades que podem ser realizadas e como funciona seu tratamento dentro do mesmo.

Foram desenvolvidos os descritivos de caso de uso (Apêndice C), os diagramas de atividades (Apêndice C) e o Diagrama de Classes. Nesse detalhamento, não foi desenvolvido o Diagrama de Entidade e Relacionamento (DER).

Na *Unified Modeling Language* (UML), em diagrama de classe, uma classe é representada por um retângulo com três divisões, são elas: o nome da classe, seus atributos e, por fim, os métodos. Como pode ser visto, a seguir, na Figura 1.

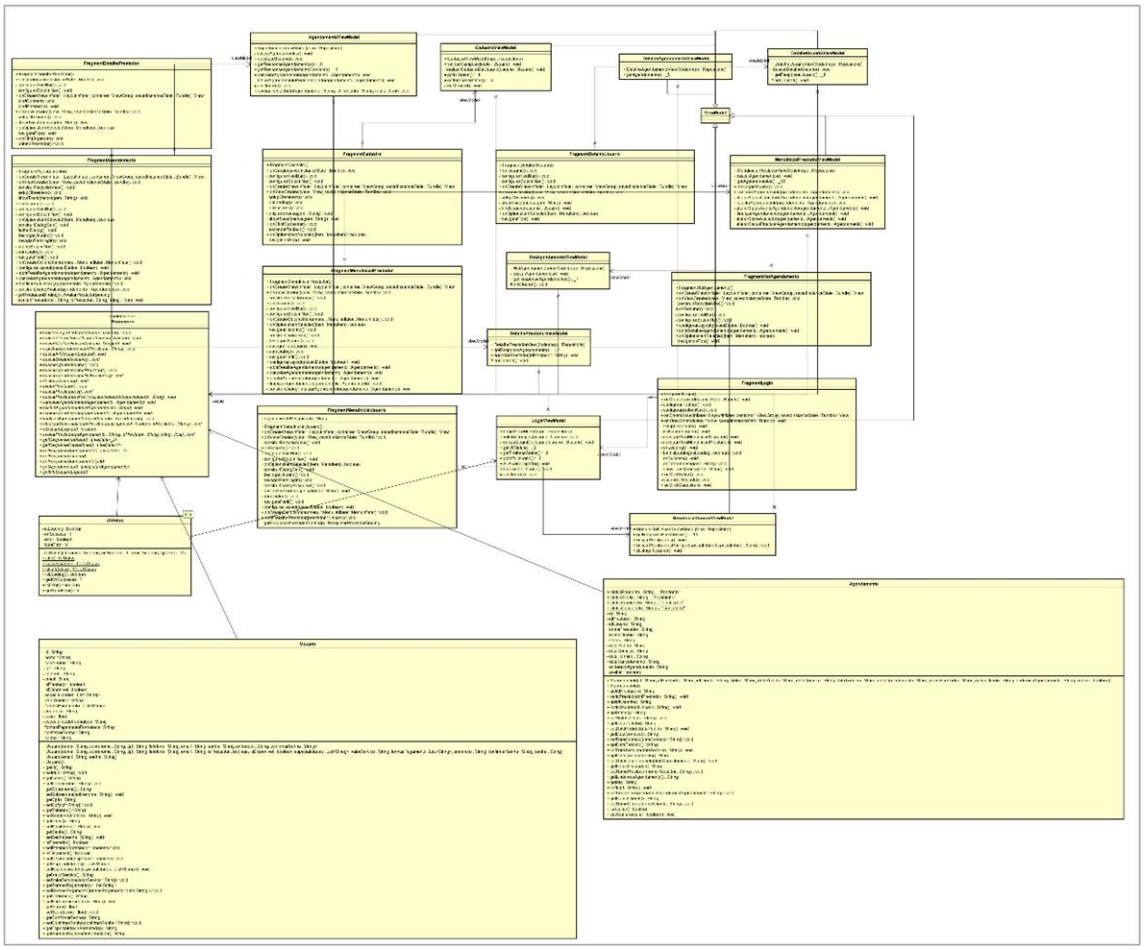

Figura 1. Diagrama de classes

3.4. Construção por funcionalidade

Nesta etapa ocorreu então a implementação dos requisitos utilizando a linguagem de programação Java. Devido à utilização do banco de dados NoSQL, serão apresentados alguns trechos de códigos relacionados a essa tecnologia.

Utilizando o *Firebase Authentication* para criar as contas de usuário, foi possível realizar o *login* de usuário no aplicativo, utilizando métodos nativos para o desenvolvedor disponibilizados por essa tecnologia, a Figura 2 representa o trecho de código utilizado para cadastrar um novo usuário no sistema; o método *realizarCadastroDeUsuario* recebe um objeto do tipo *Usuario* e, através de métodos *get*, recebe os dados necessários para que seja possível realizar a gravação dos dados no banco de dados apresentado no método *enviarInfoParaNuvem*.

```java
@Override
public void realizarCadastroDeUsuario(Usuario usuario) {
    auth.createUserWithEmailAndPassword(usuario.getEmail(), usuario.getSenha()).addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            usuario.setId(auth.getCurrentUser().getUid());
            enviarInfoParaNuvem(usuario);
        } else {
            try {
                FirebaseAuthException firebaseAuthException = (FirebaseAuthException) task.getException();
                statusFirebase.postValue(UiStatus.failure(firebaseAuthException.getMessage()));
            } catch (Exception ignored) {
                statusFirebase.postValue(UiStatus.failure("Erro ao realizar esta ação. Tente novamente em instantes"));
            }
        }
    });
}

@Override
public void enviarInfoParaNuvem(Usuario cliente) {
    FirestoreHelper.usuarioCollectionReference.add(cliente).addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            auth.signOut();
            statusFirebase.postValue(UiStatus.success(true));
        } else {
            try {
                auth.signOut();
                auth.getCurrentUser().delete();
                FirebaseFirestoreException firebaseFirestoreException = (FirebaseFirestoreException) task.getException();
                statusFirebase.postValue(UiStatus.failure(firebaseFirestoreException.getMessage()));
            } catch (Exception ignored) {
                statusFirebase.postValue(UiStatus.failure("Erro ao realizar esta ação. Tente novamente em instantes"));
            }
        }
    });
}
```

Figura 2. Método para criação de usuários

Após o usuário ter realizado o seu cadastro no sistema, o mesmo poderá realizar o *login*. A Figura 3 apresenta o método *realizarLoginFirebase*, que é responsável por realizar a autenticação do usuário, recebendo como parâmetro um objeto do tipo *Usuario* e, se o e-mail e senha informada forem válidos, o mesmo consegue ter acesso ao aplicativo, caso contrário, será mostrada uma mensagem de erro ao tentar logar no sistema.

```java
@Override
public void realizarLoginFirebase(Usuario usuario) {
    auth.signInWithEmailAndPassword(usuario.getEmail(), usuario.getSenha()).addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            statusFirebase.postValue(UiStatus.success(true));
        } else {
            try {
                FirebaseAuthException firebaseAuthException = (FirebaseAuthException) task.getException();
                statusFirebase.postValue(UiStatus.failure(firebaseAuthException.getMessage()));
            } catch (Exception ignored) {
                statusFirebase.postValue(UiStatus.failure("Erro ao realizar esta ação. Tente novamente em instantes"));
            }
        }
    });
}
```

Figura 3. Método para autenticação de usuários

Após a autenticação do usuário ser validada, o mesmo será direcionado para o menu inicial onde haverá duas guias: “Prestadores” e “Agendamentos”, apresentados na Figura 4. O usuário poderá, na guia “Prestadores” (Figura 4A), realizar ações como buscar um prestador informando a especialidade desejada; ao selecionar o prestador, o usuário terá acesso às informações do profissional escolhido, como avalição, endereço, formas de pagamento e o valor cobrado tendo a opção de solicitar atendimento a este prestador

através do botão “Chamar Prestador”, essas informações estão representadas na Figura 4B.

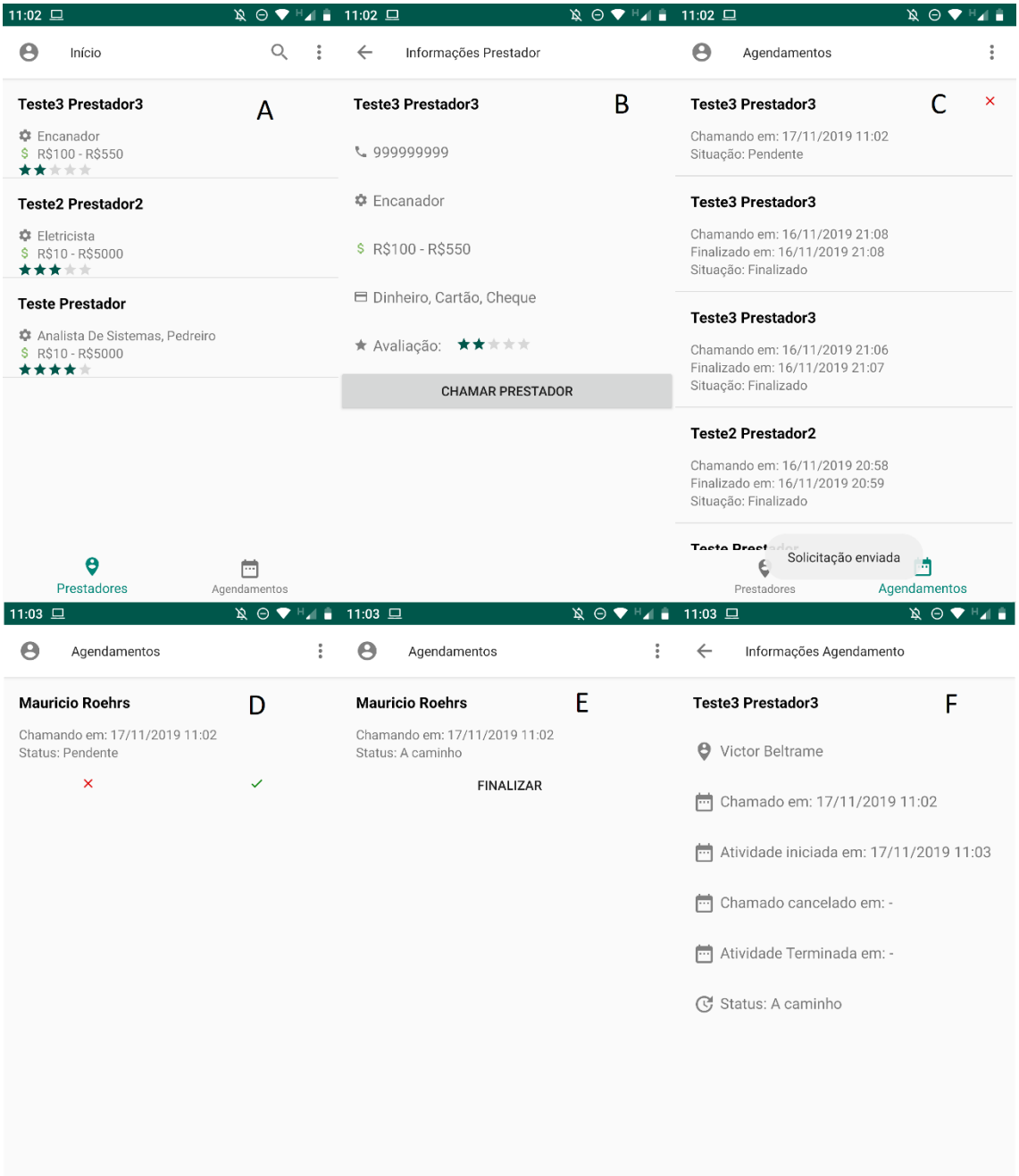

Figura 4. Processo de solicitação e aceite de um serviço

Acessando a guia “Agendamentos”, Figura 4C, o usuário terá acesso aos status do agendamento solicitado bem como ao histórico de todas as solicitações passadas. Para o usuário com perfil de Prestador, só haverá a guia “Agendamentos”, apresentado na Figura 4D, onde o mesmo poderá aceitar, recusar ou finalizar um atendimento (Figura 4E) e terá acesso ao histórico dos atendimentos realizados.

Na Figura 4F, o cliente pode acompanhar todos os status do seu atendimento, como: endereço no qual foi solicitado; data e hora que foi realizado o chamado; data e hora em que o chamado foi iniciado; caso o chamado seja cancelado pelo prestador,

também será exibida data e hora quando a atividade foi terminada; e, por fim, o status do atendimento que pode ser diferenciado entre pendente, a caminho, cancelado e finalizado.

Na Figura 5 é apresentado o código responsável pelo agendamento de um serviço no momento em que o botão “Chamar Prestador” é pressionado (Figura 4B).

```
public void registrarAgendamento(String idPrestador) {
    String idUsuario = auth.getCurrentUser().getUid();

    FirestoreHelper.usuarioCollectionReference.whereEqualTo( field: "id", idPrestador).get().addOnCompleteListener(prestadorTask -> {
        try {
            if (prestadorTask.isSuccessful()) {
                if (prestadorTask.getResult() != null) {
                    for (QueryDocumentSnapshot document : prestadorTask.getResult()) {
                        Usuario prestador = document.toObject(Usuario.class);

                        FirestoreHelper.usuarioCollectionReference.whereEqualTo( field: "id", idUsuario).get().addOnCompleteListener(usuarioTask -> {
                            if (usuarioTask.isSuccessful()) {
                                if (usuarioTask.getResult() != null) {
                                    for (QueryDocumentSnapshot documentUsuario : usuarioTask.getResult()) {
                                        Usuario usuario = documentUsuario.toObject(Usuario.class);

                                        String idDocument = Formatter.removerCaracteresData(Formatter.formatarData(new Date()));

                                        Agendamento agendamento = new Agendamento(idDocument, idPrestador, idUsuario,
                                                Agendamento.statusPendente, Formatter.formatarData(new Date()),
                                                dataComeco: "", dataTermino: "", dataCancelamento: "",
                                                nomePrestador: prestador.getNome() + " " + prestador.getSobrenome(),
                                                nomeCliente: usuario.getNome() + " " + usuario.getSobrenome(),
                                                usuario.getEndereco(), avaliar: false);

                                        FirestoreHelper.agendamentoCollectionReference.document(idDocument).set(agendamento).addOnCompleteListener(agendamentoTask -> {
                                            if (agendamentoTask.isSuccessful()) {
                                                responseAgendamento.postValue(UiStatus.success(true));
                                            } else {
                                                responseAgendamento.postValue(UiStatus.failure("Não foi possível registrar o seu agendamento"));
                                            }
                                        });
                                    }
                                } else {
                                    responseAgendamento.postValue(UiStatus.failure("Não foi possível registrar o seu agendamento"));
                                }
                            } else {
                                responseAgendamento.postValue(UiStatus.failure("Não foi possível registrar o seu agendamento"));
                            }
                        });
                    }
                } else {
                    responseAgendamento.postValue(UiStatus.failure("Não foi possível registrar o seu agendamento"));
                }
            } else {
                responseAgendamento.postValue(UiStatus.failure("Não foi possível registrar o seu agendamento"));
            }
        } catch (Exception ignored) {
            responseAgendamento.postValue(UiStatus.failure("Não foi possível registrar o seu agendamento"));
        }
    });
}
```

Figura 5. Processo de solicitação e aceite de um serviço

Na Figura 5, é possível identificar algumas decisões de agendamento. Em um primeiro momento, é feita uma consulta ao banco de dados para buscar informações do prestador. Após algumas validações, como verificar o resultado da consulta, o método realiza uma nova consulta ao banco de dados buscando informações do usuário e realiza as mesmas validações feitas na consulta dos dados do prestador. Estando tudo conforme, é criado um objeto do tipo Agendamento com todos os dados necessários e, por fim, efetivar a gravação no banco de dados.

4. Resultados e Discussões

O maior desafio foi utilizar integrações com a nuvem como o *Cloud Firestore*, onde a conexão é feita através de uma API, disponibilizada pelo *Cloud Firestore,* que cria um arquivo *JSON* e este arquivo deve ser colocado dentro da pasta *APP* do projeto. Nesse arquivo, estão todas as informações necessárias para que seja possível realizar a comunicação com o *Cloud Firestore*. O *Cloud Firestore* permite realizar transações *off-line* através de algumas configurações realizadas dentro do aplicativo; neste trabalho, não há transações *off-line*, visto que os dados precisam estar sempre atualizados em tempo

real. A Sincronia dos dados é feita pela própria API do *Cloud Firestore*, o mesmo guarda todas as informações em cache no aparelho e, quando o dispositivo conecta a Internet, o *Cloud Firestore* detecta esta conexão e envia os dados para o banco de dados, se estiver *off-line* na hora em que a ação foi executada. Caso esteja *on-line*, o *Cloud Firestore* envia os dados para o banco de dados via API. O *login* ocorre através de uma API *do Firebase*, chamada *Firebase Authentication*; quando o usuário é criado no sistema, ele é registrado na lista de autenticação do *Firebase*, e, quando o *login* é realizado, o *Firebase* consulta esta lista e retorna; se o usuário existir, *True*, caso contrário, *False*.

Durante o desenvolvimento da aplicação, alguns requisitos não puderam ser concluídos por desconhecimento da tecnologia, como a busca por proximidade, RF06 (Apêndice B); já que todas as tentativas de utilizar as API's não deram certo e como o tempo começou a ficar escasso, essa funcionalidade teve que ser desconsiderada. Tempo hábil como o cadastro dos prestadores, este cadastro será feito pelo administrador do sistema diretamente via banco de dados e os dados de Login e Senha serão passados para o prestador, bem como o RF07 (Apêndice B) que se refere à possibilidade do prestador de adicionar ou remover especialidades, essas mudanças devem ser solicitadas ao administrador para realizar os ajustes diretamente na base de dados.

Todos os testes foram realizados de forma fictícia, ou seja, um prestador fictício simulando as solicitações realizadas pelos usuários também fictícios apenas para validar as funcionalidades da aplicação e correção de erros.

4. Conclusão

Este trabalho apresentou a implementação de um aplicativo móvel, para Android, que concentre os mais diversos tipos de serviços, todos dispostos em uma lista, com a possibilidade de realizar uma busca pela especialidade e classificá-los mediante uma escala de pontuação após o termino do serviço. A aplicação visa auxiliar o usuário em encontrar e contratar um serviço conforme sua necessidade. Houve sucesso em atender os objetivos específicos que foram propostos na respectiva sessão.

Com a realização das pesquisas, foi constatado que o setor de prestação de serviços é um dos que mais cresce no Brasil, tornando essa aplicação um meio alternativo para a prospecção de novos clientes por estes prestadores.

Foi identificado, através da pesquisa, que todas as aplicações semelhantes acabam por ser burocráticas em excesso e o propósito deste aplicativo é agilizar a contratação do prestador de serviço apenas com alguns toques, por ambas as partes, cliente e prestador.

Como desenvolvimentos futuros, sugere-se a adição da busca por proximidade para que o cliente possa solicitar atendimento pelo prestador mais próximo da sua atual localização. Criar uma separação dos prestadores por categoria, pedreiros, encanadores, pintores e demais áreas, a fim de ser possível criar uma classificação dos prestadores com a maior classificação e facilitar a busca dos mesmos pelos usuários dentro dessas categorias. Criar a possibilidade de adicionar comentários nas avaliações. Criar uma área para os prestadores se cadastrarem pelo aplicativo. Fazer com que o aplicativo aceite dois perfis logados simultaneamente, cliente e prestador, tornando uma utilização prática pelos usuários.

Referências

Android Developers. (2019a) "*LiveData*", https://developer.android.com/reference/androidx/lifecycle/LiveData.html, Outubro.

Android Developers. (2019b) "*Navigation*", https://developer.android.com/guide/navigation, Agosto.

Android Developers. (2019c) "*ViewModel*", https://developer.android.com/reference/androidx/lifecycle/ViewModel, Setembro.

Cloud Firestore. (2019) "Documentação do Cloud Firestore", https://firebase.google.com/docs/firestore?hl=pt-br, Agosto.

Cury, A. *et al*. (2018) "Trabalho sem carteira assinada e 'por conta própria' supera pela 1ª vez emprego formal em 2017, aponta IBGE", https://g1.globo.com/economia/noticia/trabalho-sem-carteira-assinada-e-por-conta-propria-supera-pela-1-vez-emprego-formal-em-2017-aponta-ibge.ghtml, Setembro.

Diário de Santa Maria. (2017) "Aplicativo facilita contratação de diaristas em Santa Maria", https://diariosm.com.br/not%C3%ADcias/economia/aplicativo-facilita-contrata%C3%A7%C3%A3o-de-diaristas-em-santa-maria-1.2013289, Setembro.

Duarte, D. (2012) "UML-Diagrama de Atividades", https://www.purainfo.com.br/artigos/uml-diagrama-de-atividades/, Agosto.

Firebase Authentication. (2019) "Documentação do Firebase Authentication", https://firebase.google.com/docs/auth, Agosto.

Gaúcha ZH. (2018) "Novo aplicativo facilita contratação de profissionais em mais de 40 áreas", https://gauchazh.clicrbs.com.br/tecnologia/conteudo-publicitario/2018/03/novo-aplicativo-facilita-contratacao-de-profissionais-em-mais-de-40-areas-cjeg58uer00ra01p4q4n9lngh.html, Setembro.

Goyal, S. (2007) "Agile Techniques for Project Management and Software Engineering", https://pdfs.semanticscholar.org/35c8/a718b8c9483d5a3b6dc08dc61036fe0f54e0.pdf , Setembro.

Heptagon. (2017) "FDD – Feature Driven Development", http://heptagon.com.br/tag/fdd/, Agosto.

IBGE. (2018) "Serviços", https://brasilemsintese.ibge.gov.br/servicos.html, Setembro.

Pereira, V. R. (2014) "O setor de serviços do Brasil", https://repositorio.unesp.br/bitstream/handle/11449/126327/000840571.pdf?sequence=1, Outubro.

Perry, J. S. (2016) "Fundamentos da linguagem Java: programação orientada a objetos", IBM – developerWorks.

Petroni, B. C., Schuster, C. E., e Oliveira, C. L. V. (2014) "Avaliação da usabilidade da IDE Android Studio", *Revista Eletrônica de Tecnologia e Cultura,* v. 14, p. 132-140.

Santos, B. (2018) "O que é um profissional autônomo e como formalizar?", https://blog.hotmart.com/pt-br/profissional-autonomo/, Setembro.

Silva, I. J. O., e Miranda, K. O. S. (2009) "Impactos do bem-estar na produção de ovos", http://heptagon.com.br/tag/fdd/, Agosto.

Smartgest. (2018) “Smartgest”, https://www.smartgest.com.br/, Agosto.

Meirelles, D. S. (2006) “O Conceito de Serviço”, http://www.scielo.br/pdf/rep/v26n1/a07v26n1.pdf, Dezembro.

Apêndice A. Modelo Abrangente

O modelo geral nada mais é do que o diagrama de domínio que mostra a visão simplificada.

A Figura 6 representa a visão macro do sistema, contendo tópicos significativos para o funcionamento do sistema. O usuário solicitará atendimento ao prestador desejado utilizando o sistema e acompanhará quando foi realizado o aceite e término do serviço.

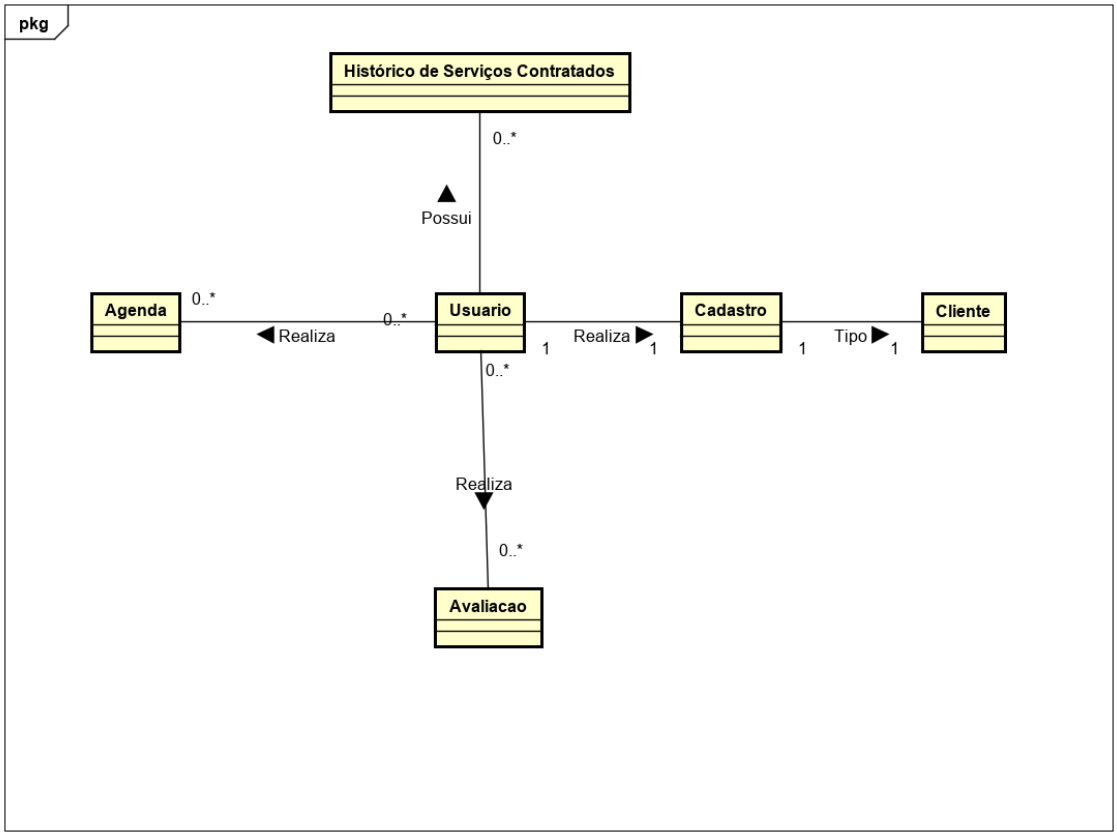

Figura 6. Diagrama de domínio

Apêndice B. Lista de Funcionalidades

Nesta seção serão apresentados os requisitos funcionais e não funcionais do sistema. Entende-se como funcionalidade tudo aquilo que é possível fazer utilizando o sistema.

- RF01 – Busca por serviços: o sistema deverá buscar os serviços conforme necessidade do usuário;
- RF02 – Cadastro de Usuários: o sistema deverá possibilitar uma área para cadastro de novos usuários com perfil de cliente.
- RF03 – Controle de Agendamento: o sistema deverá possibilitar o agendamento de um serviço, pelo cliente, bem como a exclusão do mesmo antes ou depois do prestador ter aceito o serviço. O só pode rejeitar antes do aceite, caso contrário a única forma é finalizando o atendimento;
- RF04 – Avaliação do Serviço: o sistema deverá possibilitar que o cliente avalie o serviço prestado pelo profissional;
- RF05 – Histórico de Serviços: o sistema deve possibilitar que, tanto o cliente quanto o profissional prestador do serviço, acessem uma área com o histórico de todos os serviços contratados e/ou realizados;
- RF06 – Busca por proximidade: o sistema deverá possibilitar a visualização dos prestadores em um raio pré-determinado pelo usuário;
- RF07 – Selecionar Especialidades: o sistema deve possibilitar que o profissional prestador de serviços selecione as suas especialidades de acordo com as opções já pré-definidas pelo administrador do sistema, também possibilitando alteração e exclusão das mesmas;
- RNF01 – Utilizar linguagem de programação Java;
- RNF02 – Utilizar o banco de dados Cloud Firestore;
- RNF03 – Utilizar o Sistema Operacional para Dispositivos Móveis Android.

As funcionalidades citadas anteriormente são representadas pelo Digrama de Caso de Uso, na Figura 7.

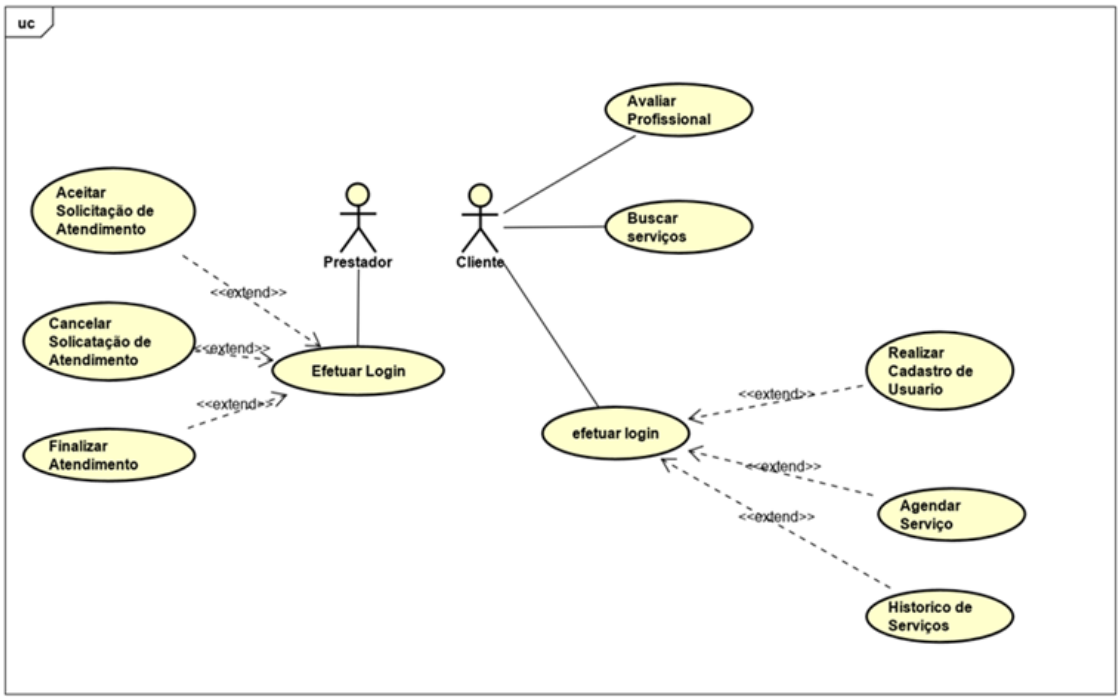

Figura 7. Diagrama de caso de uso

Apêndice C. Detalhamento das Funcionalidades

Nesta seção são detalhadas as principais funcionalidades do sistema.

O descritivo de caso de uso, a seguir, descreve a funcionalidade Cadastro de Usuário.

Tabela 1. Caso de uso 01 - cadastro de usuário

Identificação	**UC01**
Caso de uso	Cadastro de usuário.
Descrição	O sistema deverá gerenciar as funções de inclusão de um novo usuário com perfil de cliente.
Fluxo Principal [FP]	1. Usuário acessa o sistema. 2. Usuário seleciona a opção desejada: Cadastrar [FA1]. 3. Fim do caso de uso.
Fluxo de Exceção [FE]	1. Usuário cadastrado. a. O sistema verifica se o usuário já está cadastrado em caso positivo retorna ao [FP1].
Fluxo Alternativo [FA]	1. Cadastrar a. Usuário seleciona a opção de cadastro; b. Usuário preenche o formulário de Cadastro [FE1]; c. O Sistema insere os dados no banco de dados; d. O Sistema apresenta mensagem na tela; e. O Sistema retorna ao [FP2].

O descritivo de caso de uso, a seguir, descreve a funcionalidade Busca por Serviços.

Tabela 2. Caso de uso 02 - buscar por serviços

Identificação	**UC02**
Caso de uso	Busca por serviços.
Descrição	O sistema deverá buscar os serviços conforme necessidade do usuário.
Fluxo Principal [FP]	1. Usuário acessa o sistema. 2. Usuário acessa opção de busca por serviços [FA1]. 3. Fim do caso de uso.
Fluxo de Exceção [FE]	1. Busca de especialidade de profissional. a. O sistema verifica se a especialidade já está cadastrada, em caso positivo retorna ao [FA1b]. Caso não exista retorna ao [FA1d].
Fluxo Alternativo [FA]	1. Buscar a. Usuário digita a especialidade dos profissionais desejados; b. Sistema exibe os profissionais na tela [FE1]; O Sistema retorna ao [FP2].

O descritivo de caso de uso abaixo descreve a funcionalidade Controle de Agendamento.

Tabela 3. Caso de uso 03 - controle de agendamento

Identificação	**UC03**
Caso de uso	Controle de agendamento.
Descrição	O sistema deverá possibilitar o agendamento de um serviço, pelo cliente, bem como a exclusão do mesmo antes ou depois do prestador ter aceito o serviço. O só pode rejeitar antes do aceite, caso contrário a única forma é finalizando o atendimento.
Fluxo Principal [FP]	1. Usuário acessa o sistema. 2. Usuário seleciona o prestador desejado e clica na opção Chamar Prestador [FA1]. 3. Usuário acompanha andamento da solicitação na tela Agendamentos [FA2]. 4. Usuário cancela solicitação na tela Agendamentos [FA3]. 5. Fim do caso de uso.
Fluxo de Exceção [FE]	-
Fluxo Alternativo [FA]	1. Chamar Prestador a. Usuário seleciona o profissional desejado; b. Usuário clica em chamar prestador; c. O sistema insere os dados no banco de dados; d. O sistema apresenta mensagem na tela; e. O sistema retorna ao [FP1]. 2. Agendamentos a. Usuário seleciona o agendamento; b. Sistema exibe informações sobre o agendamento na tela; c. O sistema retorna ao [FP1]. 3. Cancelar a. Usuário seleciona a solicitação desejada e clica em cancelar; b. O sistema insere os dados no banco de dados; c. O sistema apresenta uma mensagem na tela; d. O sistema retorna ao [FP1].

A seguir, são explicitados os Diagramas de Atividades. Duarte (2012) diz que um diagrama de atividade é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra e que serão empregados para fazer a modelagem de aspectos dinâmicos do sistema.

Na Figura 8 é apresentado o Diagrama de Atividades sob a visão do cliente contratante do serviço.

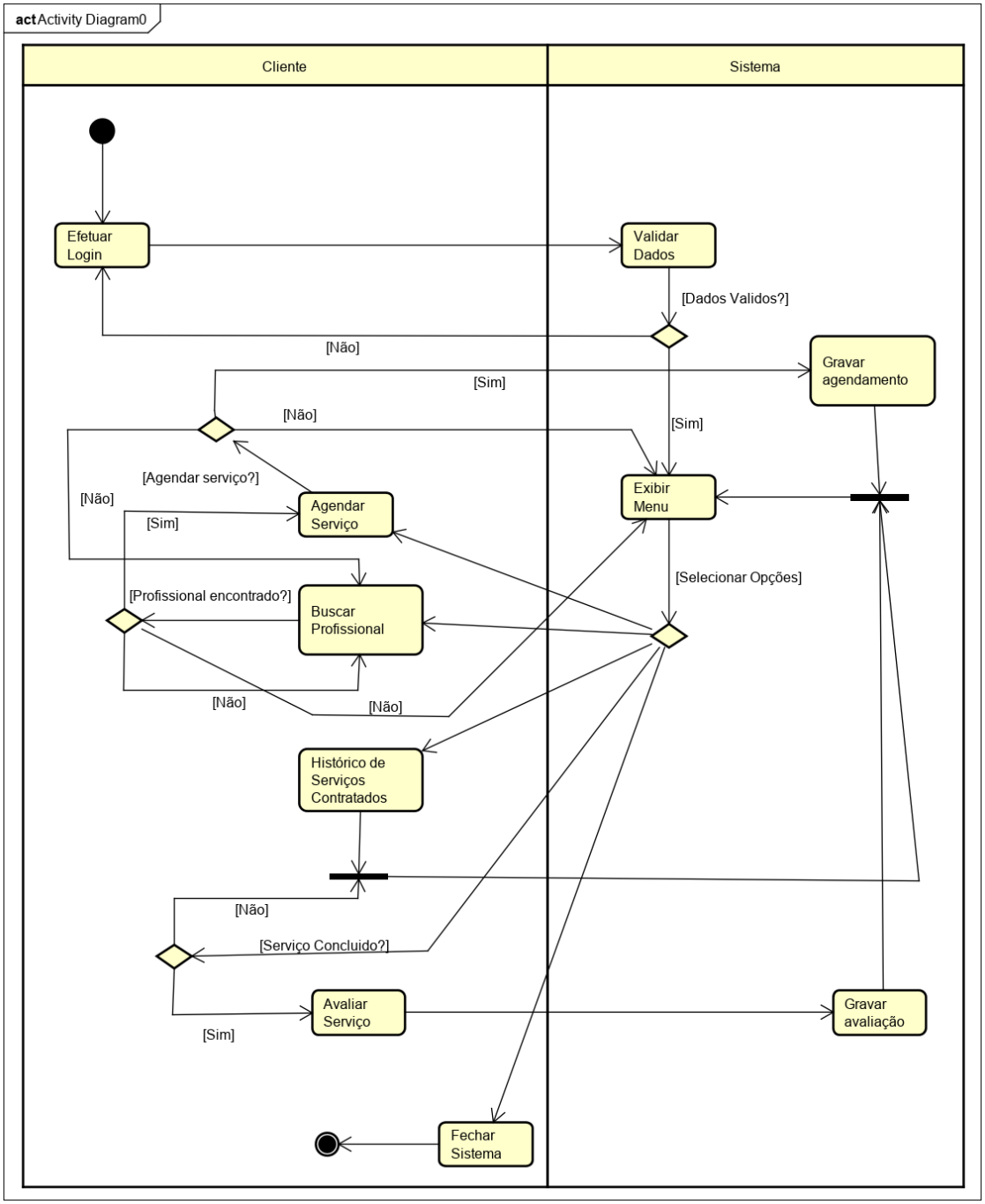

Figura 8. Diagrama de atividades sob a visão do cliente contratante do serviço

A Figura 9, a seguir, apresenta o Diagrama de Atividades na visão do profissional prestador do serviço.

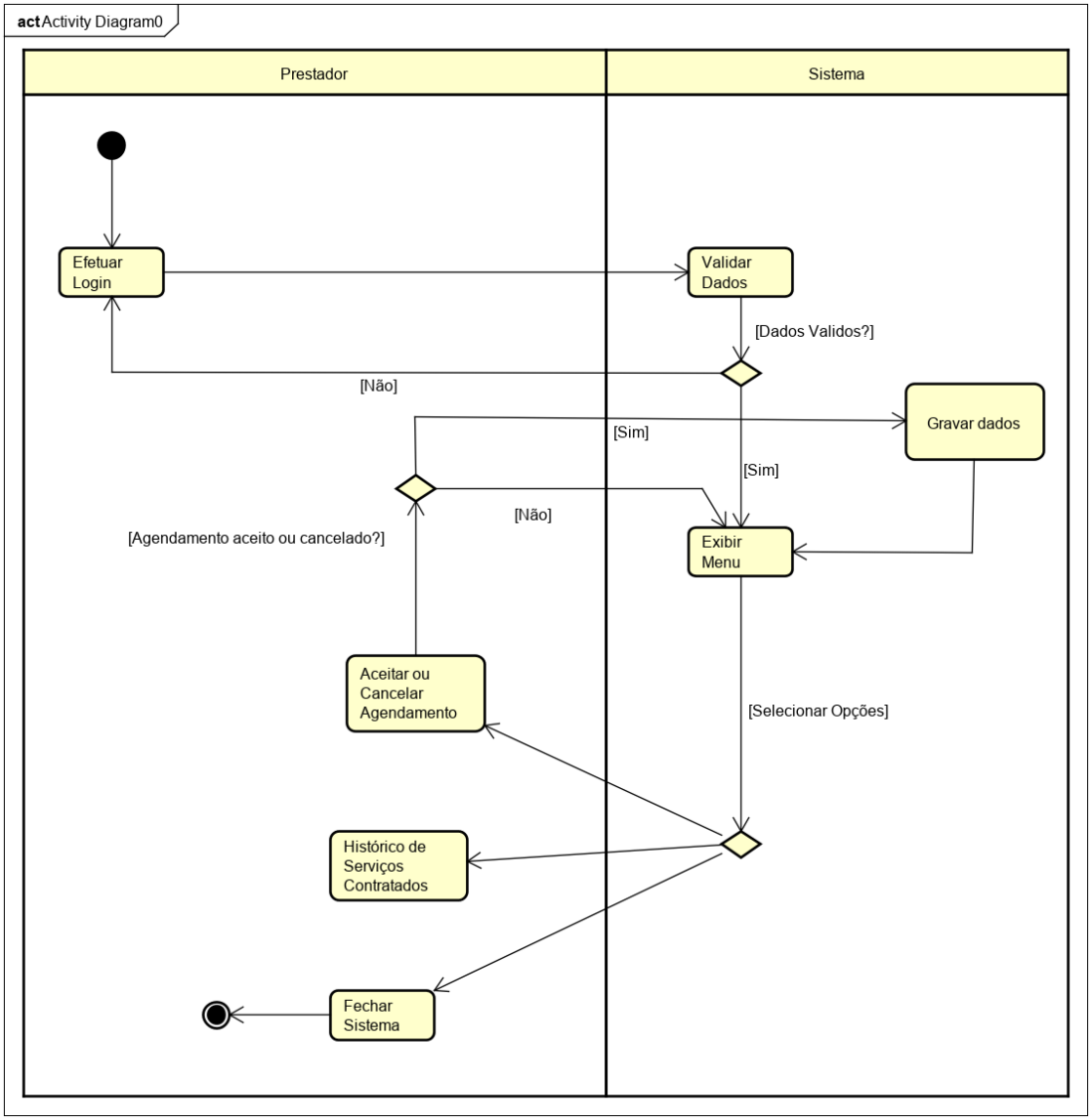

Figura 9. Diagrama de atividades na visão do profissional prestador do serviço