UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADRIANO GUIMARÃES SOARES

PoupaGrana: Aplicativo Gerenciador de Finanças Pessoais com Interface Conversacional

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Marcelo Soares Pimenta

Porto Alegre
2019

“Every book you pick up has its own lesson or lessons,
and quite often the bad books have more to teach than the good ones.”
— Stephen King

AGRADECIMENTOS

Agradeço a minha família por ter me ajudado nos momentos difíceis do caminho durante todos esses anos e aos meus amigos por terem me feito companhia e não terem desistido de mim durante os momentos corridos.

Por fim, agradeço a todos os professores da UFRGS, que contribuíram na minha formação profissional, e especialmente ao meu orientador Dr. Marcelo Soares Pimenta por ter me orientado e ter possibilitado a criação deste trabalho.

RESUMO

Este trabalho tem como objetivo apresentar a concepção e desenvolvimento de um aplicativo iOS para realizar o gerenciamento de finanças pessoais do usuário. A aplicação tem como diferencial o uso de uma interface conversacional, similar a um chat, tornando, assim, a entrada de dados mais intuitiva para o usuário. O texto também irá descrever as tecnologias e técnicas utilizadas no desenvolvimento, além de apresentar uma avaliação da aplicação desenvolvida.

Palavras-chave: Gerenciador de Finanças Pessoais. Aplicativo. iOS.

PoupaGrana: Personal Finance Manager App with Conversational Interface

ABSTRACT

This work aims to present the design and development of an iOS application to perform the management of personal finances of the user. The application has as a differential the use of a conversational interface, similar to a chat, making data entry more intuitive for the user. The text will also describe the technologies and techniques used in development, and will present an evaluation of the application developed.

Keywords: Personal Finance Manager. App. iOS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BaaS	Backend as a Service
FaaS	Function as a Service
iOS	iPhone Operating System
JSON	JavaScript Object Notation
MVC	Model–View–Controller
MVVM	Model–View–ViewModel
SDK	Software Development Kit
UML	Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

O número de brasileiros que fazem controle financeiro tem crescido no Brasil. Segundo a pesquisa do (SPCBRASIL, 2019), em 2018, passou de 55% para 63% o número de brasileiros que acompanham e analisam os seus ganhos e gastos por meio de um orçamento. O método mais utilizado é o caderno de anotações, com 33%, seguido dos 20% que utilizam planilhas e 10% que utilizam aplicativos para Smartphone. Considerando métodos informais, 19% das pessoas dizem que fazem o controle de cabeça, e 13% não adotam qualquer método.

O controle financeiro é uma ferramenta importante para o conhecimento da nossa realidade financeira. É realizado através do registro dos ganhos e gastos, possibilitando saber o quanto e onde o dinheiro é realmente gasto. Com essas informações é possível analisar e identificar os principais gastos, tornando possível a realização de cortes de gasto e hábitos que devem ser mudados.

1.1 Motivação

Apesar do número de brasileiros que fazem o controle de sua vida financeira estar aumentando anualmente, 62,3% dessas pessoas diz ter dificuldades para realizar o controle efetivo. Parte delas afirma não ter encontrado mecanismo simples para fazer esse controle.

Diante dessas informações, somos levados ao seguinte questionamento: Como fornecer uma experiência simples e intuitiva para a registro de ganhos e despesas?

1.2 Objetivo

Este trabalho tem como objetivo o desenvolvimento de um aplicativo com interface conversacional, que seja simples e intuitivo para realizar o controle financeiro pessoal, detalhando todo o processo de desenvolvimento necessário para a sua construção.

1.3 Organização do texto

O trabalho está dividido em 6 capítulos. O capítulo 2 apresenta conceitos e tecnologias importantes para a compreensão do projeto. O capítulo 3 apresenta aplicativos com funções semelhantes à desenvolvida neste trabalho. Além disso, mostra uma análise comparativa de funcionalidades. O capítulo 4 apresenta detalhadamente o desenvolvimento da aplicação. O Capítulo 5 apresenta uma análise da facilidade de uso da aplicação desenvolvida neste trabalho. O capítulo 6 apresenta a conclusão, sobretudo sobre possíveis melhorias.

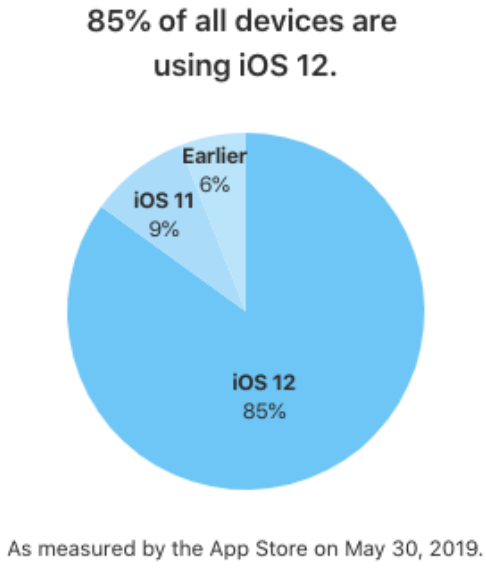
2 CONCEITOS E TECNOLOGIAS

Neste capítulo, serão apresentadas as tecnologias, ferramentas e conceitos utilizados para a elaboração da solução proposta. Também serão abordados os motivos para a escolha destas tecnologias e ferramentas. Os conceitos abordados serão *iOS*, *Swift*, *RxSwift*, *Firebase* e Interfaces conversacionais.

2.1 IOS

O *iOS* é o sistema operacional para dispositivos moveis da *Apple*, e foi a plataforma escolhida para o desenvolvimento da solução apresentada no trabalho. O principal fator motivador da escolha da plataforma se deu pela grande taxa de adoção das versões mais recentes do sistema operacional, como é possível ver na figura 2.1, 85% de todos os dispositivos moveis da *Apple* utilizam a versão 12 do *iOS*, e isso faz com que seja possível utilizar os recursos mais recentes do dispositivo. O principal motivo para a escolha do *iOS* como plataforma de desenvolvimento foi o conhecimento prévio do autor.

Figura 2.1: Taxa de adoção da versão mais recente do iOS



Fonte: <https://developer.apple.com/support/app-store>

2.2 Swift

Swift é uma das linguagens de programação disponíveis para a criação de aplicativos *iOS*. Foi criado pela *Apple* em 2014 para ser o substituto do *Objective-C* e, em 2015, se tornou *Open Source* (SWIFT, 2015). Uma das principais vantagens do *Swift* é a facilidade com que possibilita escrever um código seguro, devido aos seus recursos para tratamento de erro e incorporação de fundamentos de linguagens funcionais.

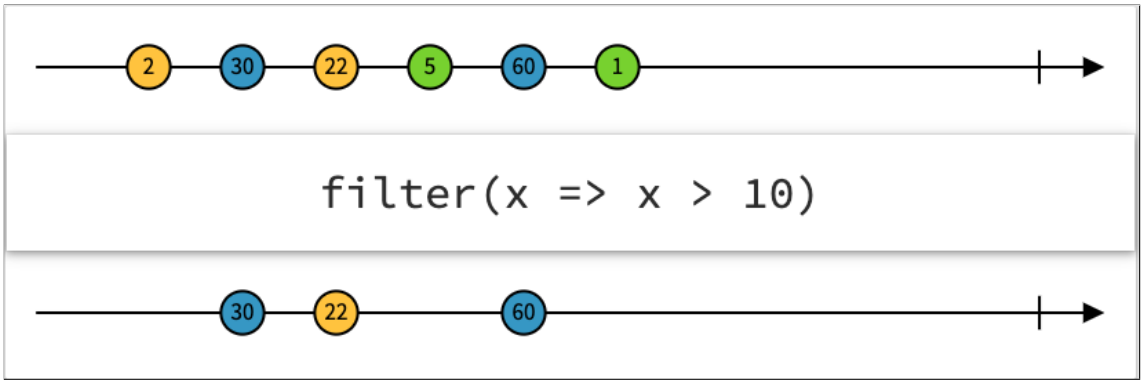
Em pesquisa feita com desenvolvedores (STACKOVERFLOW, 2018) ficou entre as 10 primeiras no ranking de linguagens mais amadas e também figura entre as 10 primeiras no ranking de linguagens que os desenvolvedores mais querem aprender.

2.3 RxSwift

É a versão em *Swift* de um *framework* de programação reativa, que tem como objetivo facilitar o desenvolvimento de interfaces baseadas em eventos. Utiliza o conceito de fluxos de dados para propagar mudanças de forma assíncrona pela aplicação. Existem versões dela para diversas linguagens de programação.

Ela implementa operadores que atuam sobre esses fluxos de dados. Esses operadores podem ser encadeados para criar um pipeline para transformar os dados. Na figura 2.2 podemos ver o operador filter e um exemplo de seu funcionamento. Outros operadores podem ser vistos de forma interativa em RxMarbles (2019).

Figura 2.2: Operador filter na ferramenta RxMarbles



Fonte: <https://rxmarbles.com/#filter>

2.4 Firebase

É uma plataforma de ferramentas voltada para o desenvolvimento de aplicativos. Tem como objetivo ajudar as empresas com as tarefas de desenvolvimento, qualidade e crescimento.

Na parte de desenvolvimento, ela possui serviços de autenticação, banco de dados *NoSQL* em tempo real, serviço de hospedagem de arquivos e sites estáticos. Para a parte de qualidade, ela entrega ferramentas de relatório de falhas, relatório de performance e laboratório de testes, onde é possível testar o aplicativo em dispositivos variados. Para o crescimento, eles disponibilizam ferramentas para *analytics*, envio de notificações e testes A/B.

O uso de um serviço de *BaaS* diminui muito o tempo de desenvolvimento da aplicação, não sendo necessário se preocupar com o desenvolvimento de um *backend* próprio e sua hospedagem.

2.5 Interfaces conversacionais

São interfaces que simulam uma conversa com humanos. Existem 2 tipos básicos de interfaces conversacionais: as baseadas em inteligência artificial e as baseadas em regras (LANDBOT, 2019).

As interfaces baseadas em inteligência artificial são aquelas que utilizam processamento natural de linguagem e aprendizado de máquina para se comunicar com os usuários, e é o tipo que é encontrado nos assistentes de voz, como a *Siri*. Esse tipo de interface permite mais liberdade ao usuário, contudo, são muito mais complexas e custosas para se desenvolver.

Interfaces baseadas em regras são aquelas que possuem um roteiro de conversa bem definido, geralmente limitando o escopo da conversa, e deixando o fluxo da conversa no controle da interface. Este tipo de interface é bem mais simples de ser desenvolvida e tem um custo bem menor que as baseadas em inteligência artificial.

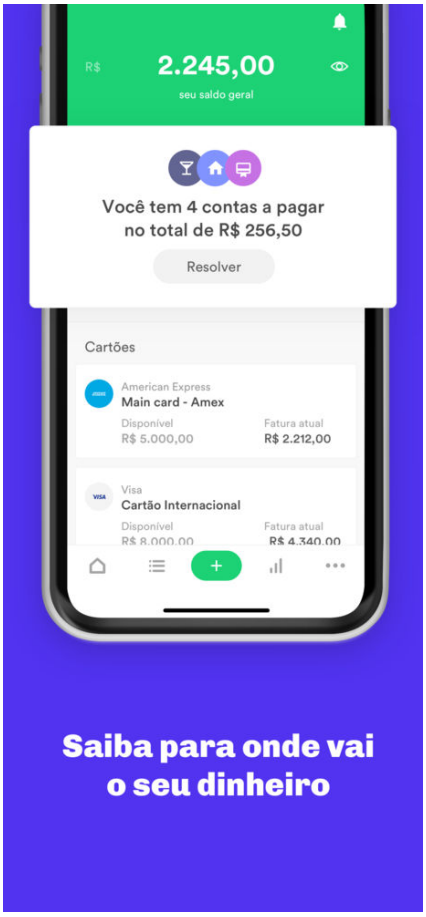
3 TRABALHOS RELACIONADOS

Neste capitulo, serão apresentados e comparados alguns aplicativos que possuem finalidade semelhante à solução proposta neste trabalho. A escolha dos aplicativos analisados foi feita entre os mais bem avaliados da AppStore na categoria finanças.

3.1 Organizze

O Organizze (ORGANIZZE, 2019) é um aplicativo brasileiro para controle financeiro pessoal. Ele está disponível para as plataformas Android, iOS e Web. Não é possível cadastrar novas contas ou cartões de crédito na versão gratuita, apenas na versão premium. Possui categorização de compras e importação de arquivo de conciliação bancária na versão Web. Atualmente, não possui nenhum método de sincronização automática, mas a funcionalidade está em desenvolvimento. Na figura 3.1 podemos ver um exemplo de sua interface.

Figura 3.1: Organizze

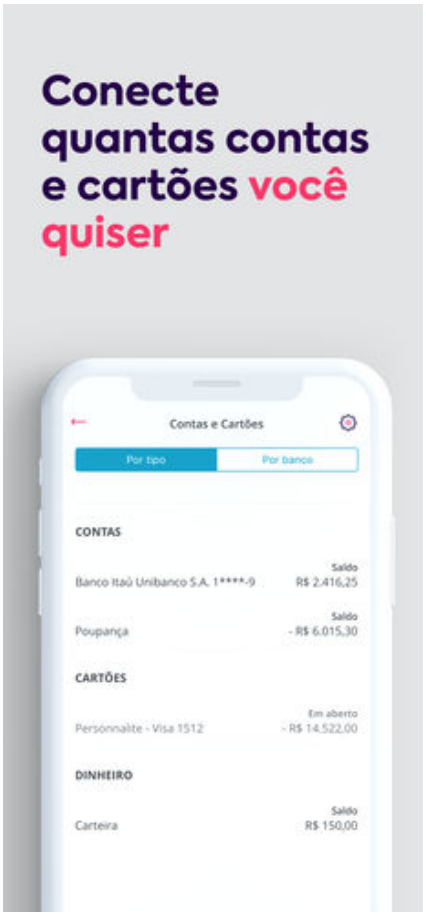


Fonte: AppStore

3.2 Guia Bolso

O Guia Bolso (GUIABOLSO, 2019) é um aplicativo brasileiro, e possui versões Android, iOS e Web. Tem como grande diferencial a sincronização automática de algumas contas bancárias e cartões de crédito como, por exemplo o *Nubank* e o *Inter*. Para a sincronização ser possível, é necessário informar o usuário e a senha da conta. Possui categorização de compras e oferece dicas de onde é possível fazer economia. O serviço é totalmente gratuito, sendo monetizado através de outros serviços, como o oferecimento de empréstimos e a monitoria de CPF. Na figura 3.2 podemos ver um exemplo de sua interface.

Figura 3.2: Guia Bolso



Fonte: AppStore

3.3 Mobills

O Mobills (MOBILLS, 2019) é um aplicativo brasileiro, e possui versões Android, iOS e Web. Assim como o Organizze, não é possível adicionar novas contas ou cartões de crédito na versão gratuita, apenas na versão premium. Além disso, possui a opção de adicionar uma categoria ao cadastrar uma nova despesa ou ganho. Não possui a funcionalidade de sincronia automática com as contas bancárias, apenas a possibilidade de importação na Web de arquivos de conciliação bancária na versão paga. Na figura 3.3 podemos ver um exemplo de sua interface.

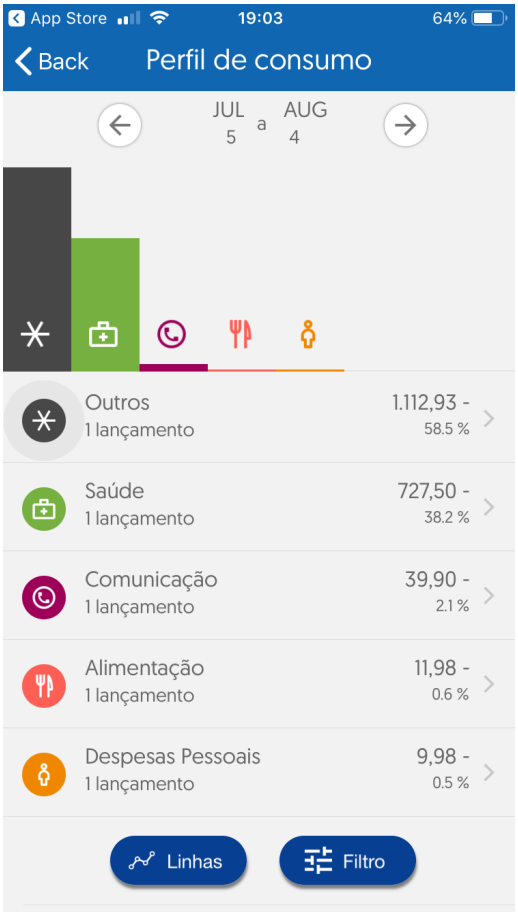
Figura 3.3: Mobills

Fonte: AppStore

3.4 Banco do Brasil

Diferente dos aplicativos citados anteriormente, o aplicativo do Banco do Brasi (BANCODOBRASIL, 2019) não é um gerenciador financeiro propriamente dito, e sim um aplicativo bancário com funções de gerenciador financeiro. O aplicativo possui sincronização automática dos produtos da própria instituição, não sendo possível adicionar contas ou cartões de crédito de terceiros. Porém, é possível a inclusão manual de transações, só que elas ficam misturadas no meio das importadas automaticamente. Também existe a possibilidade de recategorizar as transações. Na figura 3.4 podemos ver um exemplo de sua interface.

Figura 3.4: Banco do Brasil



Fonte: AppStore

3.5 Comparativo de Funcionalidades

Nesta seção, é apresentada uma comparação dos aplicativos analisados anteriormente. A tabela 3.1 apresenta as funcionalidades de cada sistema com relação a gestão financeira pessoal.

Tabela 3.1: Comparativo de funcionalidades

	Organizze	Guia Bolso	Mobills	Banco do Brasil
Adicionar conta	Sim[1]	Sim	Sim[1]	Não
Adicionar cartão de crédito	Sim[1]	Sim	Sim[1]	Não
Categorização	Sim	Sim	Sim	Sim
Sincronização	Não	Sim	Não	Sim
Pago	Sim	Não	Sim	Não
Tipo de interface	Convencional	Convencional	Convencional	Convencional

Fonte: Autor

Analisando a tabela comparativa, podemos ver que, dos aplicativos analisados, apenas o Guia Bolso possui a funcionalidade de adicionar novas contas e cartões de crédito na versão gratuita. O Organizze e Mobills possuem uma mensalidade para liberar a função enquanto no aplicativo do Banco do Brasil não é possível adicionar outras contas.

Além disso, todos os aplicativos aqui analisados possuem interfaces convencionais, e nenhum deles implementa interfaces do tipo conversacionais.

[1] Apenas na versão paga

4 PROJETO E DESENVOLVIMENTO DO POUPAGRANA

Neste capítulo, será apresentado o projeto e desenvolvimento do aplicativo iOS *PoupaGrana*, cujo objetivo é facilitar o controle e planejamento financeiro pessoal, utilizando uma interface conversacional. Nas seções seguintes, serão apresentados os casos de uso, a arquitetura do aplicativo, o servidor e seu banco de dados. Além disso, serão apresentadas as telas do aplicativo e suas devidas funcionalidades.

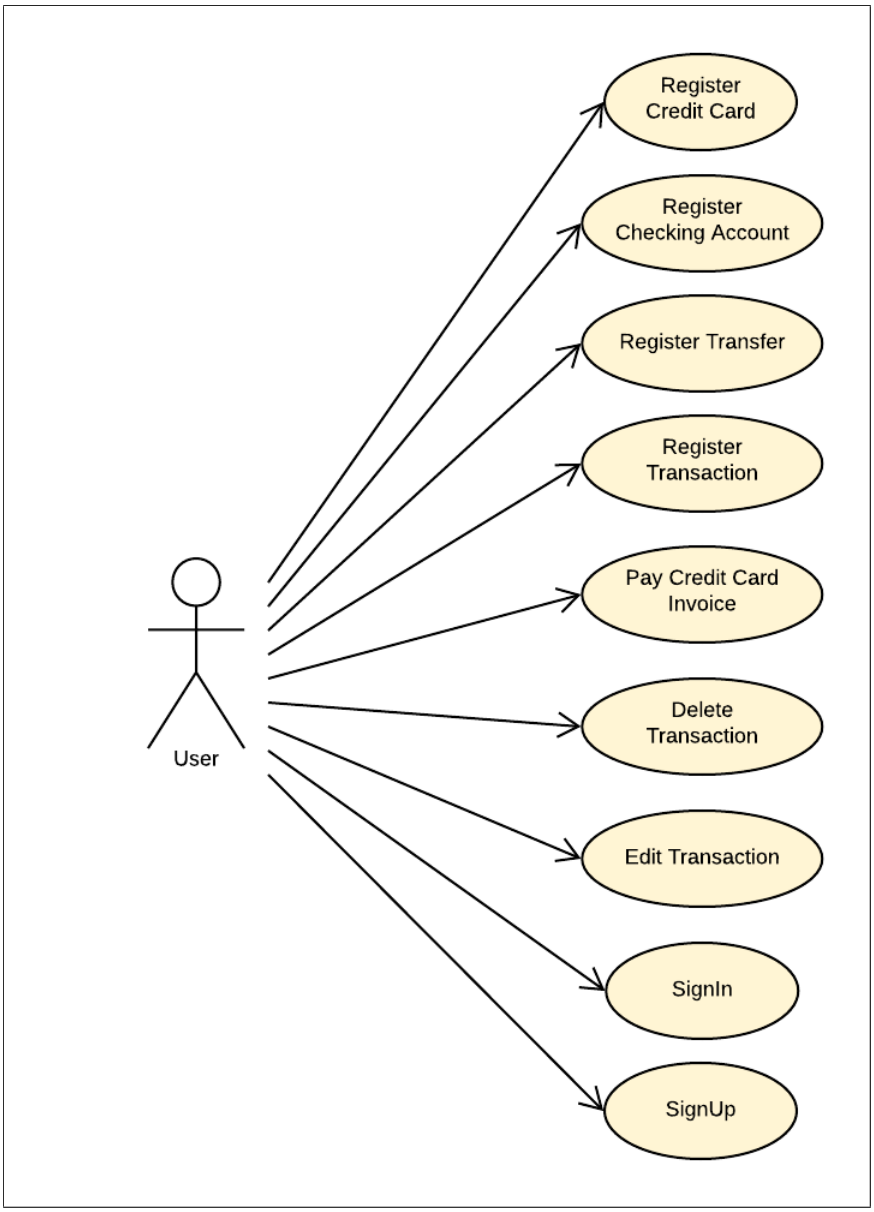
4.1 Modelagem do Sistema

Nessa seção, será apresentada a modelagem do sistema através do UML. Serão utilizados diagramas de caso de uso para listar todas as funcionalidades do sistema e diagramas de atividade para as principais funcionalidades.

4.1.1 Casos de Uso

O diagrama de caso de uso é responsável por representar as funcionalidades na perspectiva dos seus atores, sem se aprofundar em detalhes técnicos, apenas nas suas interações com o sistema. Na figura 4.1 podemos ver os principais casos de uso do aplicativo.

Figura 4.1: Casos de Uso



Fonte: Autor

4.1.1.1 Cadastrar conta corrente

O caso de uso apresentado na tabela 4.1 ocorre quando o usuário vai cadastrar uma nova conta, selecionando o nome, tipo de conta e o saldo inicial da conta, que será utilizado como base para todos os registros de transações. O servidor atualiza o saldo consolidado da conta usuário.

Tabela 4.1: Caso de Uso - Cadastrar conta corrente

Cadastrar conta corrente	
Pré-Condições	• Usuário deve estar logado
Atores	Usuário, Servidor
Pós-Condições	• O servidor deve adicionar a conta corrente à lista de contas do usuário • O servidor deve atualizar o saldo consolidado do usuário
Fluxo Principal	• Usuário informa o nome da conta • Usuário escolhe o tipo de conta • Usuário informa o saldo em conta • Usuário confirma as informações

Fonte: Autor

4.1.1.2 Cadastrar cartão de crédito

O caso de uso apresentado na tabela 4.2 é semelhante ao caso de uso de cadastrar conta corrente, mudando apenas as informações solicitadas e também que não ocorre a atualização do saldo consolidado. O usuário deve informar um nome de identificação para o cartão, o dia de corte, o dia de vencimento para o aplicativo lembrar o usuário de pagar a conta e o limite do cartão.

Tabela 4.2: Caso de Uso - Cadastrar um cartão de crédito

Cadastrar Cartão de Crédito	
Pré-Condições	• Usuário deve estar logado
Atores	Usuário, Servidor
Pós-Condições	• O servidor deve adicionar o cartão à lista de cartões do usuário
Fluxo Principal	• Usuário informa o nome do cartão de crédito • Usuário informa o dia de corte do cartão de crédito • Usuário informa o dia de vencimento do cartão de crédito • Usuário informa o limite do cartão de crédito • Usuário confirma as informações

Fonte: Autor

4.1.1.3 Registrar uma transação

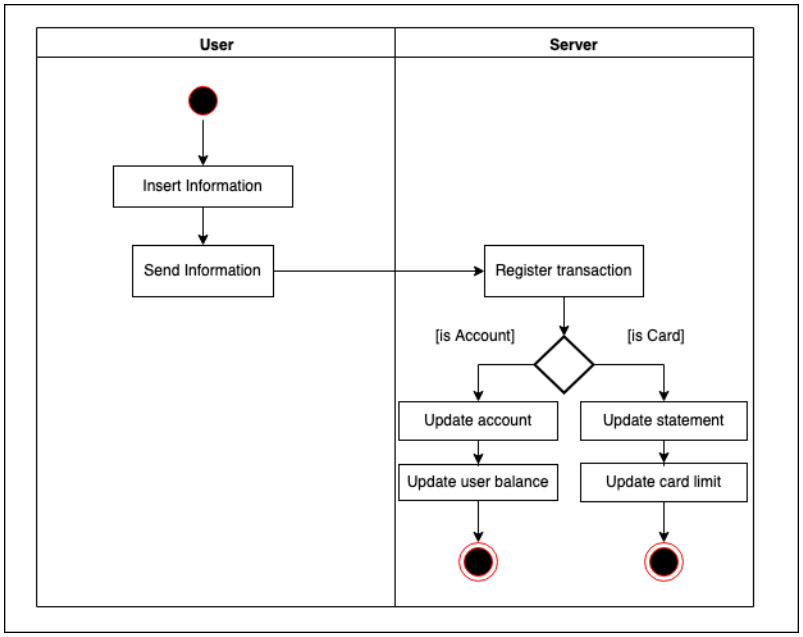
O caso de uso apresentado na tabela 4.3 ocorre quando o usuário vai cadastrar uma nova transação, seja ela para uma conta ou para um cartão de crédito. Se ela for para o cartão de crédito, o servidor a adiciona na fatura correspondente da data. Caso seja em uma conta, o servidor atualiza o saldo da conta e o saldo consolidado. O usuário pode escolher entre uma das categorias predefinidas. Na figura 4.2 podemos ver o diagrama de atividades que representa esse caso de uso.

Tabela 4.3: Caso de Uso - Registrar uma transação

Registrar uma transação	
Pré-Condições	• O usuário deve estar logado • O usuário deve possuir conta corrente ou cartão de credito cadastrado
Atores	Usuário, Servidor
Pós-Condições	• O servidor deve atualizar o saldo caso necessário
Fluxo Principal	• Usuário informa o tipo de transação • Usuário informa valor da transação • Usuário informa uma descrição para a transação • Usuário escolhe o cartão ou a conta da transação • Usuário informa a data que ocorreu a transação • Usuário seleciona a categoria • Usuário confirma as informações

Fonte: Autor

Figura 4.2: Diagrama de atividades - Registrar uma transação



Fonte: Autor

4.1.1.4 Transferência entre contas

No caso de uso apresentado na tabela 4.4, o usuário deve escolher uma conta de origem e uma conta de destino. O servidor deve atualizar o saldo das duas contas, e o saldo consolidado permanece inalterado.

Tabela 4.4: Caso de Uso - Transferência entre contas

Transferência entre contas	
Pré-Condições	• O usuário deve estar logado • O usuário deve possuir, no mínimo, duas contas correntes cadastradas
Atores	Usuário, Servidor
Pós-Condições	• O servidor deve atualizar o saldo das contas envolvidas
Fluxo Principal	• Usuário informa o valor da transferência • Usuário seleciona a conta de origem • Usuário seleciona a conta de destino • Usuário informa a data que ocorreu a transferência • Usuário confirma as informações

Fonte: Autor

4.1.1.5 Pagar uma fatura do cartão de crédito

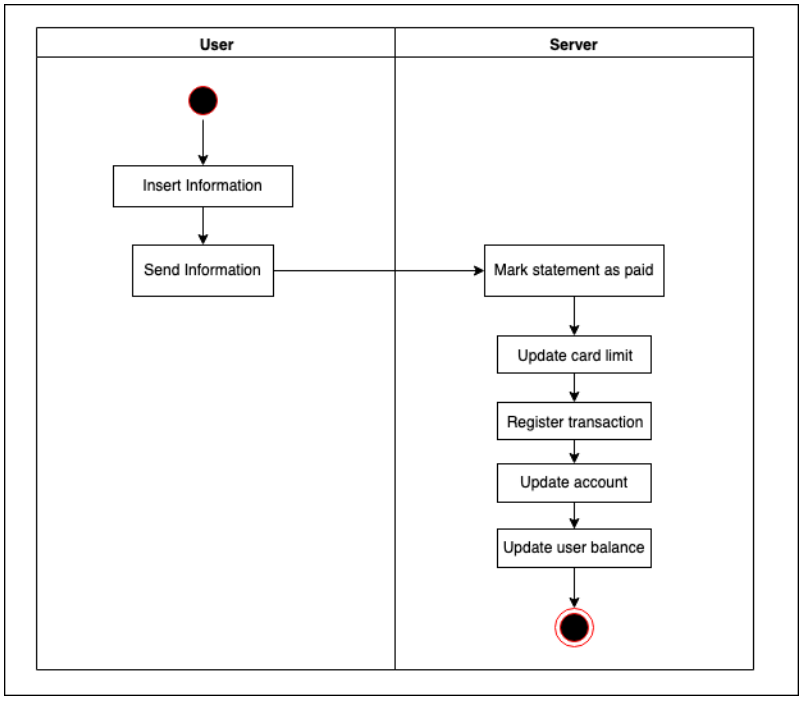
O caso de uso apresentado na tabela 4.5 ocorre quando o usuário quer registrar que pagou uma fatura do cartão. Após informar o valor pago e a conta corrente de onde vai sair o saldo, o servidor deve marcar a fatura como paga. Caso não for pago o valor integral, deve adicionar como saldo devedor na próxima fatura. Além disso, deve atualizar o saldo da conta usada e o saldo consolidado. Na figura 4.3 podemos ver o diagrama de atividades que representa esse caso de uso.

Tabela 4.5: Caso de Uso - Pagar uma fatura do cartão de crédito

Pagar uma fatura do cartão de crédito	
Pré-Condições	• O usuário deve estar logado • O usuário possuir cartão de credito e conta corrente cadastrada
Atores	Usuário, Servidor
Pós-Condições	• O servidor deve marcar a fatura como paga • O servidor deve atualizar o saldo das contas envolvidas
Fluxo Principal	• Usuário seleciona a fatura que quer pagar • Usuário informa quanto quer pagar • Usuário seleciona a conta corrente que vai usar para pagar • Usuário informa a data que ocorreu o pagamento • Usuário confirma as informações

Fonte: Autor

Figura 4.3: Diagrama de atividades - Pagar uma fatura do cartão de crédito



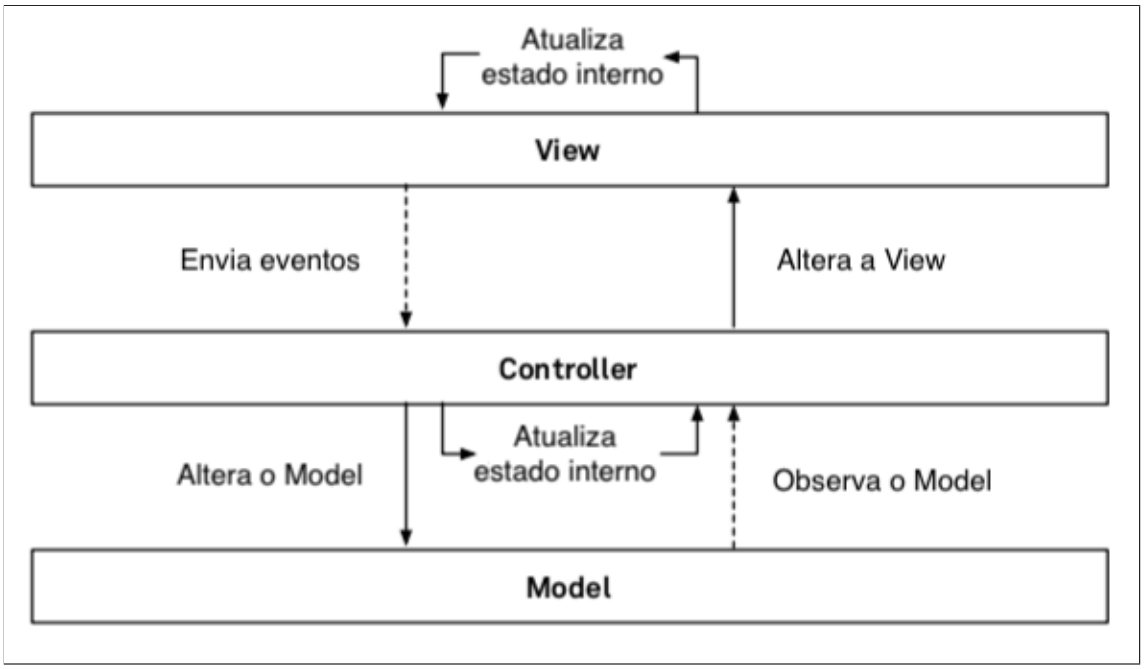
Fonte: Autor

4.2 Arquitetura

A arquitetura em que é desenvolvida a maior parte das aplicações para iOS é a *MVC*. Esse padrão arquitetural possui três camadas com funções distintas (Model, View, Controller), como pode ser visto na figura 4.4.

Model: é responsável pelos dados da aplicação e lógica de negocio.
View: é a camada que exibida na tela, sendo responsável pela interação com o usuário e pela representação gráfica dos dados.
Controller: é o que une a *View* com a *Model*, transformando as interações com a *View* em mudanças no *Model* e as mudanças no *Model* em alterações visuais na *View*.

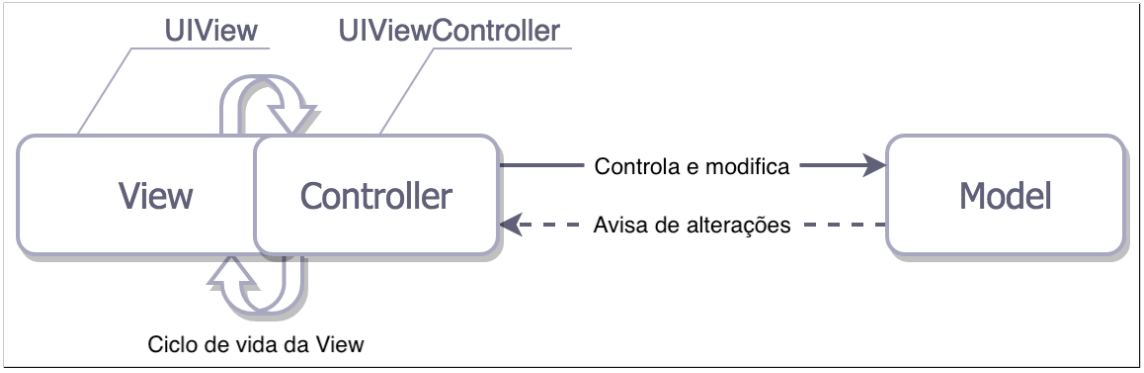
Figura 4.4: Arquitetura MVC



Fonte: (EIDHOF; GALLAGHER; KUGLER, 2018)

Apesar do *MVC* ser um padrão arquitetural bem conhecido, a *Apple* acaba utilizando de uma maneira diferente do convencional, como mostrado na figura 4.5. É utilizada uma estrutura chamada de *ViewController*, onde as responsabilidades da *View* são delegadas para a *Controller*, criando, desta forma, um forte acoplamento entre essas classes, o que acaba dificultando o reuso e a testabilidade. Desenvolvedores *Apple* acabam utilizando a *ViewController* para praticamente tudo, como por exemplo: manipular *Models*, fazer requisições *HTTP* e tratar as interações do usuário, criando o que é chamado pela comunidade de *Massive View Controller* (HUDSON, 2018).

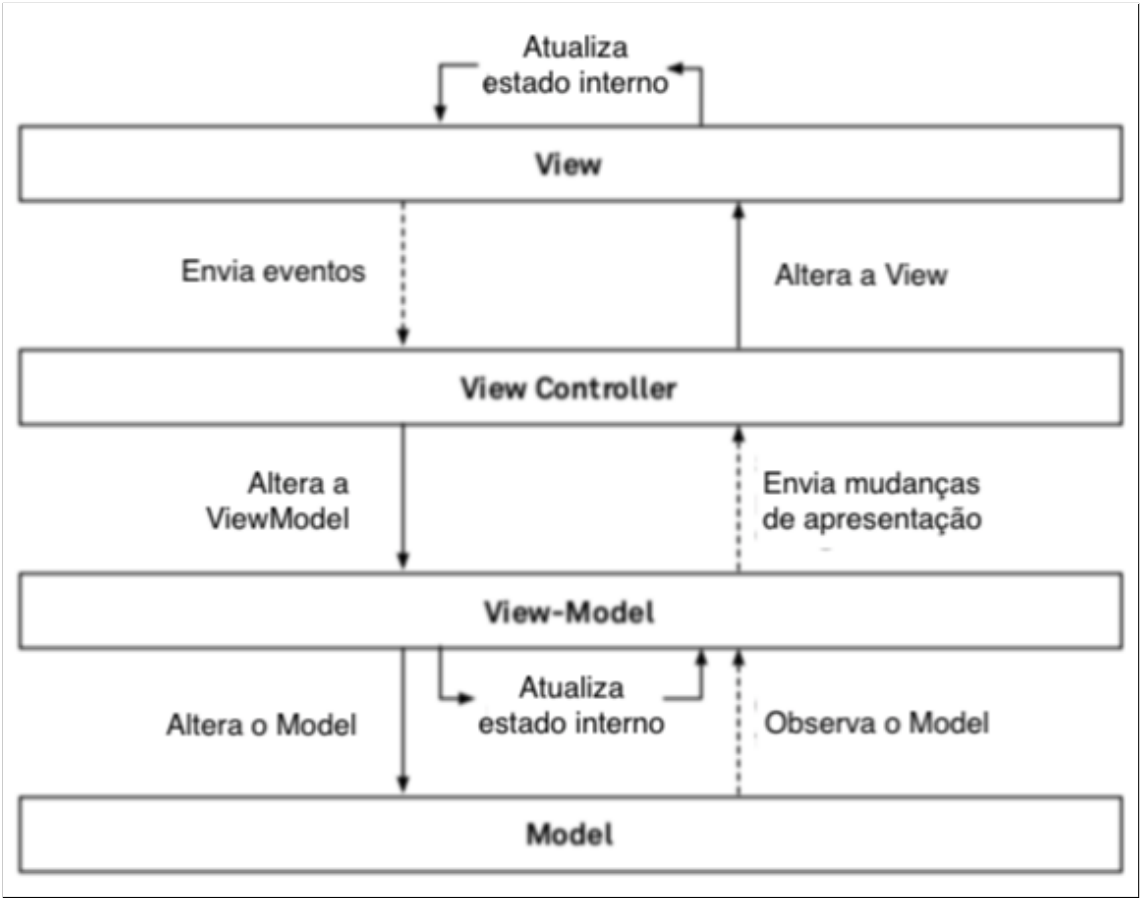
Figura 4.5: Arquitetura MVC usado pela Apple



Fonte: (ORLOV, 2015)

Para contornar esses problemas, foi decidido usar outra arquitetura, a *Model-View-ViewModel* (*MVVM*), que é uma variação do *MVC* criada pela *Microsoft* em 2005. Essa arquitetura possui uma etapa adicional chamada *ViewModel*, que tem função conter as lógicas de negócio e guardar os estados da *View*, desta forma, diminuindo as responsabilidades da *ViewController*, como pode ser visto na figura 4.6. Essa camada também pode ser a responsável por realizar requisições *HTTP*, tornando a *ViewController* testável e reutilizável (EIDHOF; GALLAGHER; KUGLER, 2018). Para facilitar a comunicação entre a *ViewModel* e a *ViewController*, é possível usar Programação Reativa para criar um *binding* que atualizará a UI quando algum dado for alterado na *ViewModel*.

Figura 4.6: Arquitetura MVVM



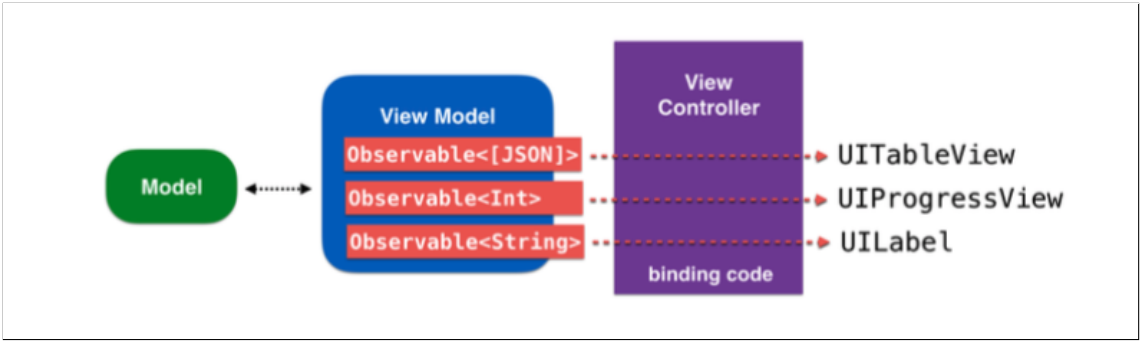
Fonte: (EIDHOF; GALLAGHER; KUGLER, 2018)

4.2.1 Programação reativa

Anteriormente, vimos que atualização da interface ocorre através do *binding* entre a *ViewModel* e a *View*. O *binding* é feito usando a biblioteca *RxSwift* (RXSWIFT, 2019),

e seus tipos, *Observable* e *Subjects*. Assim sempre que as variáveis do tipo *Observable* da *ViewModel* sofrem alguma mudança é disparado um evento para que as *Views* mudem para refletir essa mudança, como pode ser visto na figura 4.7.

Figura 4.7: Arquitetura MVVM com RxSwift



Fonte: (PILLET et al., 2017)

Como exemplo apresentado na figura 4.8, vemos a *ViewModel* responsável pelo login e pelo cadastro. Podemos ver que o *binding* foi feito como um fluxo de dados unidirecional, recebendo como entrada os *Observables* que representam os eventos da interface, sendo eles o e-mail, senha e os eventos de toque nos botões de login e cadastro, retornando os *Observables* que representam os eventos da *ViewModel*.

Figura 4.8: Exemplo de ViewModel reativa

```
class AuthViewModel {

    func bind(
        email: Observable<String?>,
        password: Observable<String?>,
        signInButton: Observable<Void>,
        signUpButton: Observable<Void>
    ) → (
        isValid: Observable<Bool>,
        userLoggedIn: Observable<Event<User>>
    ) {
        let credentials = Observable.combineLatest(email, password)

        let isValid = credentials
            .map({ (email, password) in
                  return email?.isValidEmail && password?.isValidPassword
            })
            .startWith(false)

        let signIn = signInButton
            .withLatestFrom(credentials, resultSelector: { $1 })
            .flatMapLatest({ (email, password) in
                AuthService()
                    .signIn(withEmail: email, andPassword: password)
                    .materialize()
            })
            .share(replay: 1)

        let signUp = signUpButton
            .withLatestFrom(credentials, resultSelector: { $1 })
            .flatMapLatest({ (email, password) in
                AuthService()
                    .signUp(withEmail: email, andPassword: password)
                    .materialize()
            })
            .share(replay: 1)

        let userLoggedIn = Observable.merge(signIn, signUp)

        return (
            isValid: isValid,
            userLoggedIn: userLoggedIn
        )
    }
}
```

Fonte: Autor

Na *ViewControler* apresentada na figura 4.9 podemos ver o *binding* da interface sendo feito para habilitar os botões caso os campos estejam preenchidos corretamente, e

para receber o evento que corresponde ao Login do usuário.

Figura 4.9: Exemplo de ViewController reativa

```
class AuthViewController: UIViewController {

    private let viewModel: AuthViewModel
    private var disposeBag = DisposeBag()

    init(viewModel: AuthViewModel) {
        self.viewModel = viewModel
        super.init(nibName: nil, bundle: nil)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        bindViewModel()
    }

    private func bindViewModel() {
        let (
            isValid,
            userLoggedIn
        ) = viewModel.bind(
            email: emailTextField.rx.text.asObservable(),
            password: passwordTextField.rx.text.asObservable(),
            signInButton: signInButton.rx.tapWithThrottle.asObservable(),
            signUpButton: signUpButton.rx.tapWithThrottle.asObservable()
        )

        isValid
            .not()
            .bind(to: signInButton.rx.isEnabled, signUpButton.rx.isEnabled)
            .disposed(by: self.disposeBag)

        userLoggedIn
            .subscribe(onNext: { [weak self] event in
                switch event {

                case .next(let user):
                    let home = AppRouter.home(user: user)
                    self?.present(home, animated: true, completion: nil)

                case .error(let error):
                    self?.showError(message: error.localizedDescription)

                case .completed: break
                }
            })
            .disposed(by: self.disposeBag)
    }
}
```

Fonte: Autor

4.3 Servidor

O *Firebase* é uma plataforma de desenvolvimento Web e Mobile que oferece diversos serviços, entre eles o *Firestore*, que é um *BaaS* com um banco de dados *NoSQL Realtime*. Esse banco de dados pode ser acessado usando uma *API RESTful* ou através do uso do *SDK* provido pelo serviço. (FIREBASE, 2019)

O *Firebase* possui também o *Cloud Functions*, um *FaaS* que que permite a criação de funções que podem ser ativadas através de um *endpoint REST*, ou até mesmo em resposta a uma mudança no banco de dados. Outro serviço que o *Firebase* fornece é o *Authentication*. Com ele é possível fazer o controle de sessão, além de autenticar um usuário através da senha, número de telefone ou de integrações com provedores de identidade como *Google* e *Facebook*.

Figura 4.10: Cloud Function - Atualização do saldo do usuário

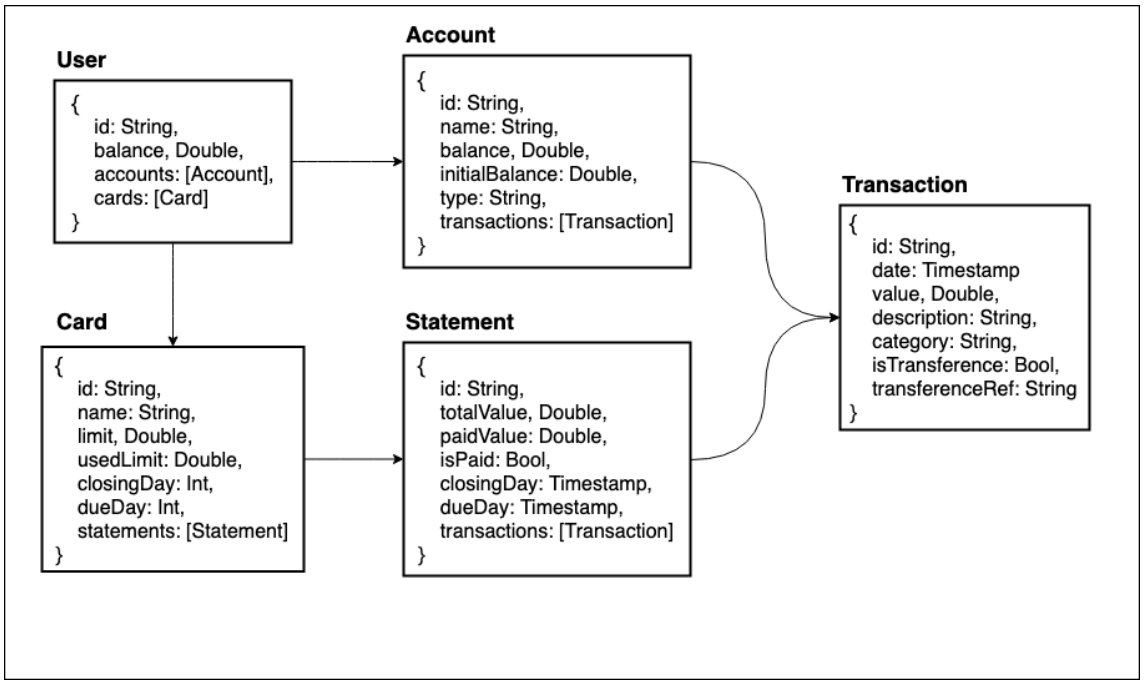
```
const functions = require('firebase-functions');
const admin     = require('firebase-admin');

admin.initializeApp();
var db = admin.firestore();

exports.updateUserBalance = functions
    .firestore
    .document('users/{userId}/accounts/{accountId}')
    .onWrite((change, context) => {
        var accountsRef = db.collection('users')
            .doc(context.params.userId)
            .collection('accounts')

        var userRef = db.collection('users')
            .doc(context.params.userId)

        return db.runTransaction(transaction => {
            return transaction.get(accountsRef).then(accounts => {
                var accountBalance = accounts.docs.reduce((a, c) => a + c.get('balance'), 0)

                return transaction.update(userRef, {
                    balance: accountBalance
                })
            })
        });
    })
```

Fonte: Autor

4.3.1 Banco de dados

O Firestore, como visto anteriormente, é uma base de dados NoSQL orientada a documentos do tipo JSON, trazendo grande flexibilidade para armazenar dados de forma hierárquica. Esses documentos são armazenados em coleções. Os documentos, além de dados, também pode conter subcoleções de documentos. Na figura 4.11 podemos ver o esquema da base de dados utilizado nesse trabalho.

Figura 4.11: Esquema da base de dados



Fonte: Autor

4.3.1.1 User

Esta coleção é utilizada para armazenar parte dos dados do usuário. A parte de autenticação fica armazenada no Firebase Authentication. Nela, armazenamos o saldo consolidado do usuário e as subcoleções que contém as contas correntes e os cartões de crédito. O saldo consolidado é atualizado sempre que o saldo de alguma conta muda. Essa estrutura é criada por uma Cloud Function no momento em que o usuário cria a conta no aplicativo.

4.3.1.2 Account

Esta coleção é utilizada para armazenar os dados das contas do usuário. Nela são armazenados o nome para identificação da conta, o saldo consolidado, o saldo inicial e o tipo da conta (corrente ou poupança). No documento também é armazenada uma subcoleção contendo todas as transações realizadas referentes a essa conta. Sempre que o saldo inicial é alterado ou transações são adicionadas, removidas ou editadas, o saldo consolidado da conta é recalculado usando uma Cloud Function.

4.3.1.3 Card

Esta coleção é responsável por armazenar os dados do cartão de crédito. Na criação do cartão é armazenado o nome para identificação, o dia de corte da fatura, o dia de vencimento da fatura, o limite total e o limite usado. Além disso, o documento possui uma subcoleção de faturas e, sempre que houver uma alteração em alguma fatura, o limite usado é recalculado para o cartão correspondente.

4.3.1.4 Statement

Esta coleção é utilizada para armazenar os dados da fatura. Ela é criada automaticamente através de uma Cloud Function sempre que chega o dia de fechamento do cartão de crédito, ou quando é inserida uma transação para um mês futuro. Neste documento é armazenado o dia de fechamento, o dia de vencimento, se a fatura foi paga, o total pago e o valor total dela, que é calculado sempre que uma transação é adicionada, removida ou editada.

4.3.1.5 Transaction

Esta coleção é responsável por armazenar os dados da transação e transferência. Ela possui uma descrição curta, data, valor e uma categoria entre as disponíveis no App. Caso essa transação seja uma transferência, é gerada uma outra transação na outra conta com o valor positivo, e adicionada uma referência entre as duas. Assim, sempre que uma é alterada, a outra também sofrerá as mesmas alterações.

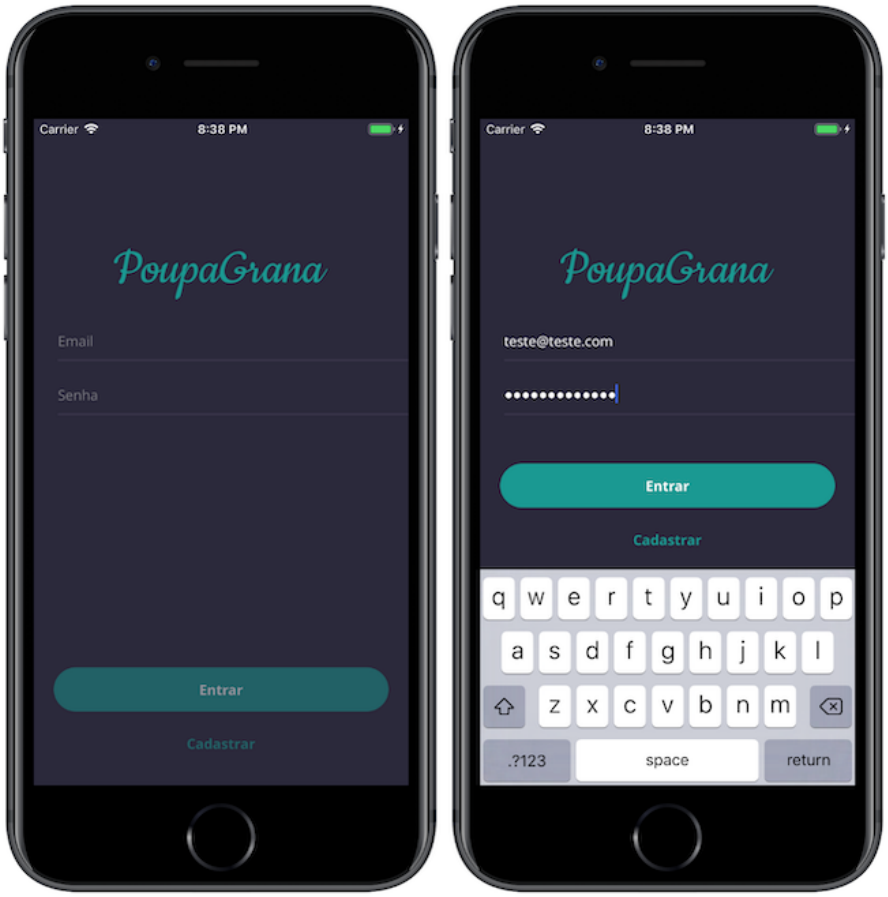
4.4 Telas

Nesta seção serão mostradas as principais telas do aplicativo e suas principais funcionalidades, as quais foram citadas anteriormente.

4.4.1 Tela de login e cadastro

A tela de login possui dois campos, sendo eles para e-mail e para senha, e dois botões, um para cadastro e o outro para login. Os botões só se tornam ativos quando for preenchido um e-mail válido e uma senha. Na figura 4.12 podemos ver a demonstração desta tela.

Figura 4.12: Tela de login e cadastro

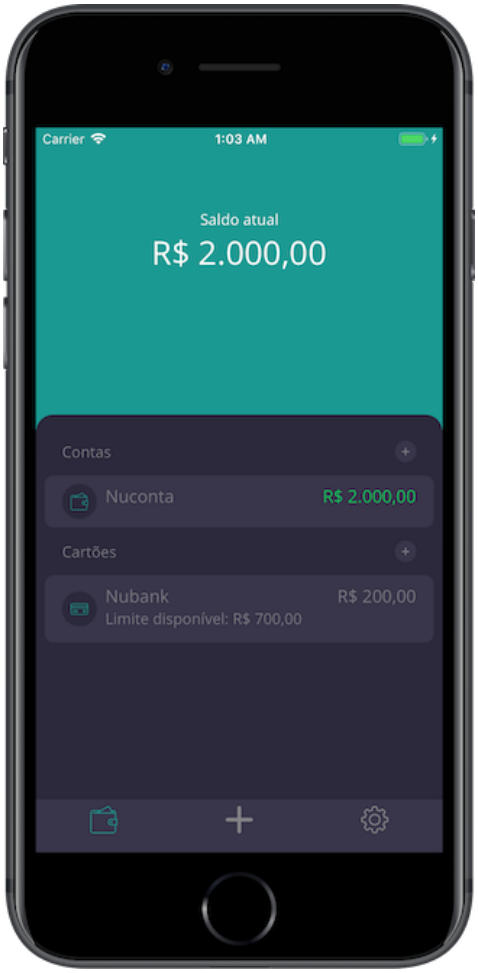


Fonte: Autor

4.4.2 Tela inicial

Na tela principal, é possível o usuário visualizar o seu saldo acumulado, o saldo de todas as suas contas e o valor atual das suas faturas do cartão de crédito. Nesta tela, ainda é possível ir para os fluxos de adicionar uma conta ou um cartão de crédito e de registrar uma movimentação financeira. Na figura 4.13 é possível vermos a tela principal do aplicativo.

Figura 4.13: Tela inicial



Fonte: Autor

4.4.3 Tela registro de movimentação

Nesta tela, o aplicativo faz diversas perguntas ao usuário a fim de obter as informações que serão necessárias para o registro da movimentação. Como método de entrada,

será usado o teclado para entradas livres, um conjunto de botões para selecionar entre um número pequeno de opções e um componente de seleção para um número maior de opções e para datas. Na figura 4.14 podemos ver um exemplo do funcionamento da tela.

Figura 4.14: Tela registro de movimentação

Fonte: Autor

5 AVALIAÇÃO DO APLICATIVO

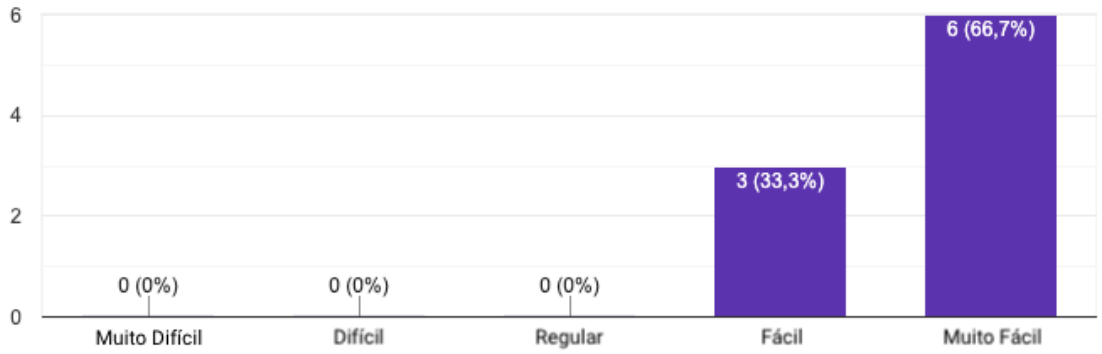
Este capitulo tem como objetivo apresentar a avaliação dos resultados obtidos com a solução proposta e o método utilizado para a avaliação.

Foi disponibilizado aos usuários um dispositivo com a aplicação já instalada e uma conta já cadastrada. Após isso, foi pedido para que os usuários realizassem as ações de adicionar um cartão de crédito, adicionar uma conta corrente, adicionar uma transação e pagar a fatura do cartão de crédito. Após as tarefas serem completadas, foi aplicado o questionário presente no apêndice A.

A pesquisa foi aplicada com 9 usuários dos quais 44,4% têm o ensino superior completo, 22,2% estão cursando o ensino superior e 33,3% têm o ensino médio completo.

Quando os usuários foram perguntados "*Como você avaliaria a tarefa de adicionar um cartão de crédito?*", 66,7% avaliaram como muito fácil, e os 33,3% restantes avaliaram como fácil.

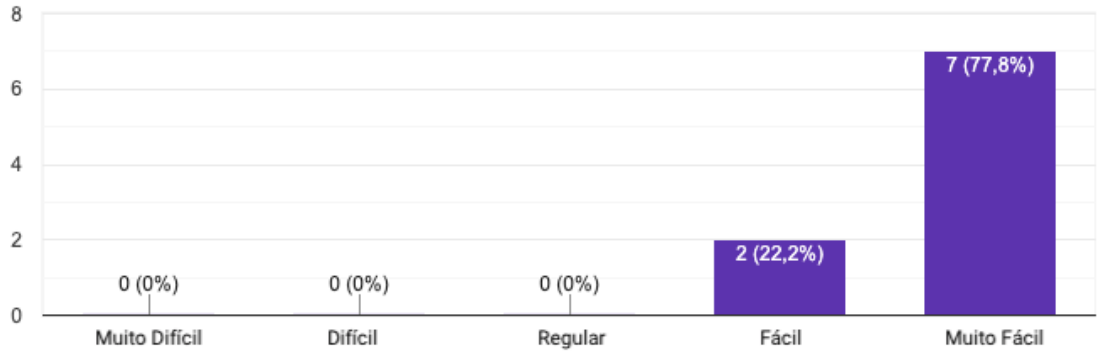
Figura 5.1: Pesquisa - Adicionar um cartão de crédito



Fonte: Autor

Quando os usuários foram perguntados "*Como você avaliaria a tarefa de adicionar uma nova conta corrente?*", 77,8% avaliaram como muito fácil, e os 22,2% restantes avaliaram como fácil.

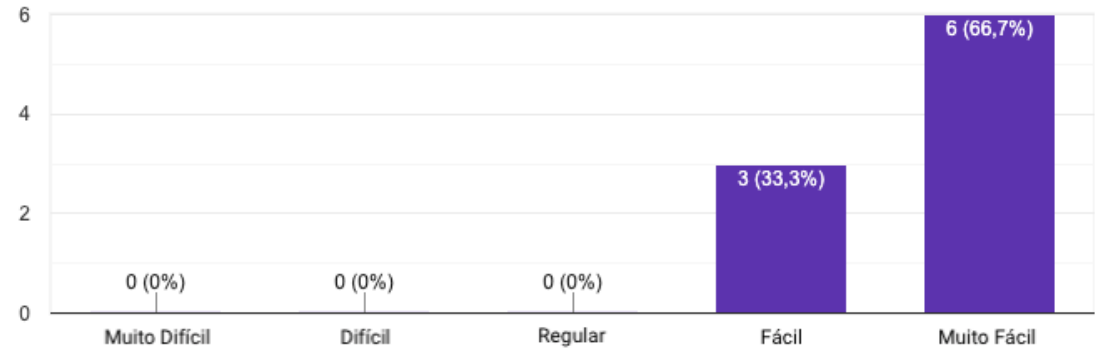
Figura 5.2: Pesquisa - Adicionar uma nova conta corrente



Fonte: Autor

Quando os usuários foram perguntados "*Como você avaliaria a tarefa de adicionar uma nova transação?*", 66,8% avaliaram como muito fácil, e os 33,3% restantes avaliaram como fácil.

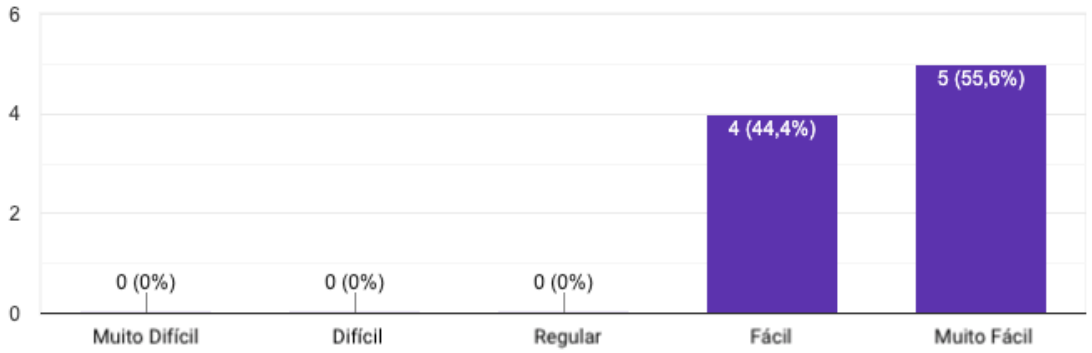
Figura 5.3: Pesquisa - Adicionar uma nova transação



Fonte: Autor

Quando os usuários foram perguntados "*Como você avaliaria a tarefa de pagar a fatura do cartão de crédito?*", 55,6% avaliaram como muito fácil, e os 44,4% restantes avaliaram como fácil.

Figura 5.4: Pesquisa - Pagar a fatura do cartão de crédito



Fonte: Autor

Também foi dado espaço para os usuários darem sugestões de melhorias e novas funcionalidades para o aplicativo.

Uma das sugestões dos usuários foi a de dar a possibilidade de, ao cadastrar uma despesa, informar que é uma despesa fixa e, assim, sendo cobrada mensalmente sem a necessidade de incluir manualmente todo os meses. Essa sugestão também serviria para compras parceladas que seriam cobradas por um número especifico de meses.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento da versão inicial de um aplicativo de gestão financeira pessoal que utiliza uma interface conversacional para facilitar o controle de gastos da vida financeira do usuário. Também foram apresentadas as principais tecnologias e técnicas utilizadas para o desenvolvimento das funcionalidades.

Como podemos ver nos resultados da pesquisa feita para avaliar as funcionalidades do aplicativo, apresentada no capítulo 5, a solução teve êxito em criar uma experiência intuitiva para as funcionalidades definidas no escopo inicial. A interface conversacional também se provou um método bastante simples para a entrada de dados do usuário, resultando em um aplicativo bem fácil de ser utilizado.

6.1 Limitações

Apesar dos resultados positivos obtidos com os usuários, o fato de não ter sido desenvolvido um sistema de sincronização automática com contas bancárias e cartões de crédito não elimina alguns dos principais motivos para as pessoas não realizarem o controle financeiro em suas vidas: falta de tempo e o não ter o hábito.

6.2 Trabalhos futuros

Considerando que a aplicação desenvolvida neste trabalho é apenas a versão inicial, foram pensadas as seguintes melhorias para serem implementadas em versões futuras:

Sincronização de contas: um sistema que possibilite ao usuário importar dados de suas contas e cartões de crédito de maneira automática, não sendo necessário o registro manual para toda e qualquer compra realizada.

Outras plataformas: não é nenhum segredo que, no Brasil, o número de dispositivos Android é bem maior que o número de dispositivos iOS, principalmente devido ao preço reduzido de seus dispositivos de entrada. Visto que o objetivo da aplicação é ajudar no controle financeiro dos brasileiros, é indispensável o desenvolvimento de uma versão Android e uma versão Web

Transações recorrentes: Implementar a possibilidade de ao registrar uma despesa marcar ela como despesa fixa ou parcelada, fazendo com que seja inseridas automaticamente inúmeras transações, sem que haja a necessidade de o usuário fazer isso manualmente.

REFERÊNCIAS

BANCODOBRASIL. **BancoDoBrasil**. 2019. <https://www.bb.com.br/>. Acessado em: 20/06/2019.

EIDHOF, C.; GALLAGHER, M.; KUGLER, F. **App Architecture: iOS Application Design Patterns in Swift**. CreateSpace Independent Publishing Platform, 2018. ISBN 1719030251. Available from Internet: <https://www.amazon.com/App-Architecture-Application-Design-Patterns/dp/1719030251>.

FIREBASE. **Firebase**. 2019. <https://firebase.google.com/>. Acessado em: 30/05/2019.

GUIABOLSO. **GuiaBolso**. 2019. <https://www.guiabolso.com.br/>. Acessado em: 20/06/2019.

HUDSON, P. **Swift Design Patterns**. HACKING WITH SWIFT, 2018. Available from Internet: <https://www.hackingwithswift.com/store/swift-design-patterns>.

LANDBOT. **The Ultimate Guide to Conversational Design**. 2019. <https://landbot.io/blog/guide-to-conversational-design>. Acessado em: 24/06/2019.

MOBILLS. **Mobills**. 2019. <https://www.mobills.com.br/>. Acessado em: 20/06/2019.

ORGANIZZE. **Organizze**. 2019. <https://www.organizze.com.br/>. Acessado em: 20/06/2019.

ORLOV, B. **iOS Architecture Patterns**. 2015. <https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52>. Acessado em: 15/05/2019.

PILLET, F. et al. **RxSwift: Reactive Programming with Swift, Second Edition**. Razeware LLC, 2017. ISBN 194287846X. Available from Internet: <https://store.raywenderlich.com/products/rxswift>.

RXMARBLES. **RxMarbles**. 2019. <https://rxmarbles.com/>. Acessado em: 15/06/2019.

RXSWIFT. **RxSwift**. 2019. <https://github.com/ReactiveX/RxSwift>. Acessado em: 30/05/2019.

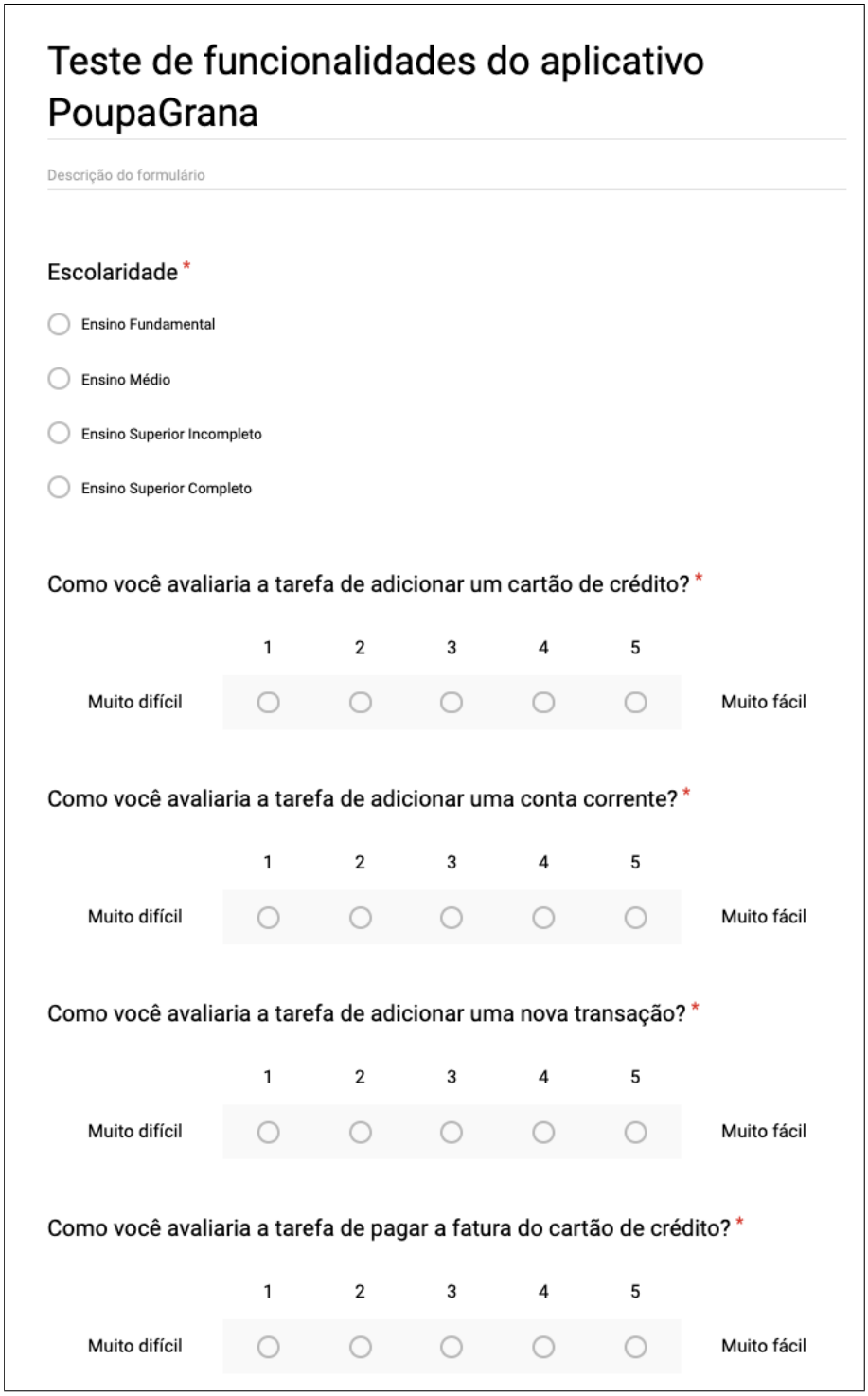
SPCBRASIL. **Cresce para 63% o número de consumidores que controlam suas finanças, revelam CNDL/SPC Brasil e Banco Central**. 2019. <https://www.spcbrasil.org.br/pesquisas/pesquisa/5873>. Acessado em: 14/06/2019.

STACKOVERFLOW. **Developer Survey Results 2018**. 2018. <https://insights.stackoverflow.com/survey/2018>. Acessado em: 15/06/2019.

SWIFT. **Swift**. 2015. <https://swift.org/about>. Acessado em: 15/06/2019.

APÊNDICE A — PESQUISA REALIZADA COM OS USUÁRIOS

Figura A.1: Pesquisa sobre as funcionalidades



Teste de funcionalidades do aplicativo PoupaGrana

Descrição do formulário

Escolaridade *

- Ensino Fundamental
- Ensino Médio
- Ensino Superior Incompleto
- Ensino Superior Completo

Como você avaliaria a tarefa de adicionar um cartão de crédito? *

	1	2	3	4	5	
Muito difícil	○	○	○	○	○	Muito fácil

Como você avaliaria a tarefa de adicionar uma conta corrente? *

	1	2	3	4	5	
Muito difícil	○	○	○	○	○	Muito fácil

Como você avaliaria a tarefa de adicionar uma nova transação? *

	1	2	3	4	5	
Muito difícil	○	○	○	○	○	Muito fácil

Como você avaliaria a tarefa de pagar a fatura do cartão de crédito? *

	1	2	3	4	5	
Muito difícil	○	○	○	○	○	Muito fácil

Fonte: Autor