Desenvolvimento de Funcionalidades na Aplicação Móvel iOS *M24* do Banco Montepio

Rui Pedro Garcia da Costa

Nº 17005 – Regime Diurno

Orientação

Luís Gonzaga Martins Ferreira

Thiago Gomes Garcia

Ano letivo 2020/2021

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

Identificação do Aluno

Rui Pedro Garcia da Costa

Aluno número 17005, regime diurno

Licenciatura em Engenharia de Sistemas Informáticos

Orientação

Luís Gonzaga Martins Ferreira

Professor Adjunto

Thiago Gomes Garcia

ITSector – Arquiteto iOS

Informação sobre o Estágio

ITSector, SA.

Av. Sidónio Pais, 153 Torre A, 6º Andar, 4100-467 Porto

RESUMO

Cada vez mais as entidades bancárias optam pelo desenvolvimento de aplicações móveis e *web* de forma a fornecer aos seus clientes um controlo mais fácil e rápido das suas contas.

Uma vez que existem inúmeras entidades bancárias, a diferenciação de funcionalidades disponibilizadas de imediato a partir de um dispositivo com o acesso à internet poderá ser um aspeto a ter em conta por parte do cliente. Assim, e cada vez mais, os bancos disponibilizam funcionalidades muito para além da consulta do saldo, ou da execução de pagamentos, existindo já atualmente entidades que permitem até a abertura de uma nova conta deste modo, sem a necessidade do deslocamento ao espaço físico da empresa.

Assim, e sendo a *ITSector* uma empresa de desenvolvimento de *software* especializada na transformação digital para instituições financeiras, com este trabalho foi desenvolvida uma nova funcionalidade, integrada na aplicação móvel para *iOS* "M24" do Banco Montepio, aplicação esta que foi desenvolvida de raiz pela *ITSector*.

ABSTRACT

More and more banking entities choose mobile and web applications development in order to offer their customers a easier and faster control of their accounts.

Once there are countless financial institutions the differentiation of features available immediately from a device with internet access may be an aspect to consider by the customer. So, and increasingly, banking entities provide features far beyond balance inquiry and making payments, existing already companies that even allow account opening without the need of displacement to the bank physical space.

This way, and *ITSector* being a software development company specialized in digital transformation for financial institutions with this job was developed a new feature integrated on the *iOS* mobile application “M24” of Banco Montepio, application that was developed from scratch by *ITSector*.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE EXCERTOS DE CÓDIGO

Glossário

JSON – Objeto estruturado sobre uma forma lógica em formato JSON.

MOCK-UP – Representação de um projeto, em escala ou tamanho real.

SCRUM – Método ágil de desenvolvimento de software.

SPRINT – Período de tempo no qual uma versão incremental e usável de um produto é desenvolvida.

USER STORY – Descrição concisa de uma necessidade do utilizador do produto.

Siglas e Acrónimos

API – Application Programming Interface

IDE - Integrated development environment

JSON - JavaScript Object Notation

MVC – Model-View-Controller

PDA – Personal Digital Assistant

UI – User Interface

URL – Uniform Resource Locator

UX – User Experience

XML - Extensible Markup Language

1. Introdução

Os *smartphones* cada vez mais estão a deixar de ser vistos como um "brinquedo" ou como um dispositivo de entretenimento, e estão a passar, progressivamente, a serem objetos indispensáveis na vida de qualquer pessoa comum, uma vez que nos permitem efetuar diversas tarefas de formas muito mais simples e rápidas.

Não haverá qualquer tipo de dúvida que nos dias de hoje qualquer pessoa gosta de ter acesso aos seus dados bancários através do seu *smartphone,* bem como de efetuar pagamentos de despesas ou até mesmo de compras realizadas no mesmo dispositivo, pelo que é sem dúvida fundamental, por parte das entidades bancárias, investirem em produtos que permitem aos seus clientes satisfazer essas suas necessidades, neste caso, através do desenvolvimento de aplicações móveis.

Disto isto, e de forma a tentar superar a concorrência, as aplicações móveis relativas a entidades bancárias atuais, possuem cada vez mais funcionalidades, de forma a cativar clientes, não querendo as mesmas ficar atrás dos seus concorrentes, sendo que atualmente algumas aplicações até já permitem a abertura de contas.

Assim sendo, este relatório serve de auxiliar ao trabalho realizado na implementação de uma nova funcionalidade, que descreverei mais tarde, para ser integrada na aplicação móvel "M24" do Banco Montepio, desenvolvida pela *ITSector.*

1.1. Objetivos

O objetivo deste projeto é a implementação de uma nova funcionalidade, chamada de "Arredondamentos".

Essa funcionalidade consiste, como o próprio nome indica, no arredondamento do valor das compras efetuadas pelo cliente, sendo que a diferença entre o valor real e o arredondado irá reverter a favor de uma conta poupança, de um mealheiro, ou até mesmo de uma instituição, através de um donativo, ficando a escolha do destino ao critério do utilizador.

1.2. Contexto

Este projeto foi realizado no âmbito da Licenciatura em Engenharia de Sistemas Informáticos, na empresa *ITSector*, em regime de teletrabalho. O estágio curricular teve duração de sensivelmente 4 meses, tendo iniciado a 15 de março e terminado a 25 de Junho de 2021, num regime de 4 dias semanais.

A *ITSector* foi fundada em 2005, no Porto, onde está ainda agora sediada. É uma empresa especialista no desenvolvimento de *software* para o setor financeiro.

Com mais de 15 anos de experiência, e uma equipa de mais de 500 especialistas distribuídos pelos 6 centros de desenvolvimento em Portugal (Porto, Lisboa, Braga, Aveiro, Bragança e Castelo Branco), tem vindo a afirmar-se no Mercado nacional e internacional com crescimentos continuados, tendo sido recentemente adquirida pela multinacional francesa *Alten*.

1.3. Estrutura do Documento

Este documento está dividido em vários capítulos, estando os mesmos divididos em subcapítulos, que explicam sucintamente os problemas propostos, bem como a sua abordagem e resolução.

Neste capítulo, Introdução, foram dadas a conhecer informações relativas ao estágio curricular, bem como da empresa onde o mesmo foi realizado, tendo sido também feita uma introdução ao produto que se pretende desenvolver.

No capítulo “Estado de Arte” é feita uma abordagem geral aquilo que são as tecnologias móveis, focando no *iOS,* bem como as suas linguagens de programação oficiais.

Posteriormente, no capítulo “Academia iOS” é relatado todo o trabalho desenvolvido durante a academia em que eu participei antes da minha integração no projeto.

Dado isto, no capítulo “Programa Arredondamentos” é escrita uma explicação do objetivo, bem como do conteúdo deste projeto.

Seguidamente, existe um capítulo respetivo a cada uma das *sprints* onde eu tive oportunidade de atuar, respeitando a mesma estrutura em todos os capítulos, começo por revelar as *User Stories*, passando depois a enunciar as tarefas que me

foram atribuídas, seguindo a minha abordagem para a sua resolução e assim terminando com a apresentação de alguns resultados obtidos.

Em forma de desfecho, é feita uma apreciação global descrita no capítulo “Conclusão”.

2. Estado de Arte

Neste capítulo será abordado o estado de arte, neste caso, o desenvolvimento *iOS*.

2.1. Desenvolvimento Móvel

O desenvolvimento de aplicações e sistemas para dispositivos móveis, vulgarmente designado por “desenvolvimento móvel” ou “desenvolvimento mobile”, consiste em todas as atividade e processos relativos ao desenvolvimento de *software* para dispositivos móveis como *smartphones, tablets,* consolas portáteis, PDAs, entre outros.

Uma vez que existem milhares de modelos de dispositivos móveis, seria impensável desenvolver um produto de software que fosse apenas compatível com um dispositivo em específico, visto que para conseguir alcançar todo o mercado seria necessário implementar milhares de aplicações. Tendo em conta que todos os dias são lançados novos produtos para o mercado, não seria possível implementar *software* personalizado para todos eles, para não falar de que os custos seriam impensáveis para qualquer instituição.

Assim sendo, um produto de *software* móvel deve ser desenvolvido com a consciência de que deverá ser possível utilizá-lo em milhares de dispositivos com diferentes características, desde a capacidade de processamento até às dimensões do ecrã.

2.1.1. iOS

O iOS é o sistema operativo para dispositivos móveis da Apple, pelo que a empresa não permite que o mesmo seja executado em *hardware* de terceiros.

O ambiente de desenvolvimento chama-se *Cocoa Touch*. Está escrito maioritariamente na linguagem de programação Objective-C e segue uma arquitetura de software MVC.

Atualmente, a Apple oferece suporte a duas linguagens de programação, sendo assim o Objective-C e o Swift, consideradas as linguagens oficiais da multinacional.

2.2. Objective-C

O *Objective-C*, criado no início dos anos oitenta, é uma linguagem de programação desenvolvida de forma a permitir uma abordagem orientada a objetos, é assim conhecido como um pequeno mas poderoso conjunto de extensões sobre a linguagem *C*, essas extensões são maioritariamente baseadas em *Smalltalk,* uma das primeiras linguagens de programação orientadas a objetos.

Assim, esta foi uma linguagem desenvolvida para adicionar ao C capacidades de programação orientada a objetos de uma forma simples e a pensar no futuro.

Ainda é muito utilizada atualmente, estando, no entanto, a perder o seu protagonismo para a nova tecnologia oficial da Apple, o *Swift*.

2.3. Swift

O *Swift,* anunciado em 2014*,* é uma linguagem de programação poderosa e intuitiva, desenvolvida pela Apple para desenvolvimento *iOS, macOS, watchOS* e *tvOS*. Esta tecnologia teve como inspiração inúmeras linguagens de programação, de entre elas o *Objective-C, Rust, Haskell, Ruby, Python e C#.*

Esta linguagem foi desenvolvida de forma a manter a compatibilidade com a *API Cocoa* e com código existente em *Objective-C*.

É atualmente uma das linguagens mais famosas e consequentemente mais utilizadas em todo o mundo, sendo considerada sucessora do *Objective-C* como principal linguagem de programação da Apple.

3. Academia iOS

A *ITSector* oferece a todos os seus colaboradores, sejam estes estagiários ou contratados, antes da sua integração em qualquer projeto, uma academia de formação relativa ao tema onde os mesmos vão atuar.

Para mim foi sem dúvida fundamental uma vez que eu nunca tinha tido contacto com tecnologias de desenvolvimento móvel, pelo que tive a oportunidade, durante a academia, que teve uma duração de 4 semanas, de me familiarizar com as tecnologias *Objective-C* e *Swift*.

Durante esta academia foram-me propostos três desafios para avaliação de conhecimentos. Será de salientar que em termos de complexidade lógica os exercícios eram acessíveis, no entanto o principal desafio era a sua implementação nas novas linguagens que estava a aprender.

3.1. Teste 1 iOS

Neste exercício era pedido um projeto de consola, com um nome específico, foi também pedida uma classe com o nome FizzBrain e dentro da mesma seria necessário um método chamado startFizzing onde estaria implementada toda a lógica.

Seria então necessário imprimir na consola os números de 1 a 100, mas para múltiplos de três, em vez de apresentado o número, seria impressa a *string* “Fizz” e para múltiplos de cinco, seria impressa a *string “*Buzz*”,* no caso de ser múltiplo de ambos os números, a *string* impressa seria “FizzBuzz”.

3.1.1. Abordagem

Como à data que esta proposta de exercício foi apresentada já tínhamos abordado na academia alguns componentes de UI, eu decidi, com a aprovação dos formadores, não só implementar o pedido, a ser impresso na consola, mas também começar a praticar um bocadinho a parte visual, pelo que implementei um algoritmo um bocadinho mais complexo que em vez de utilizar os números três e cinco, deixava o utilizador escolher os números, e o mesmo acontecia com as palavras, ao invés de escrever as palavras “Fizz” e “Buzz” dava também ao utilizador a oportunidade de definir as palavras utilizadas, por último, também o tamanho da amostra poderia ser

definido pelo utilizador. Caso não fosse inserido algum dos valores, ele utilizaria os valores predefinidos do exercício.

Este exercício foi implementado tanto em Objective-C, como em Swift, utilizando a mesma lógica em ambos.

3.1.2. Solução

A solução começou por criar um objeto com todos os valores que poderiam ser alterados pelo utilizador para posteriormente poder instanciá-lo e efetuar as leituras dos valores introduzidos para o objeto e posteriormente utilizá-lo para mostrar os resultados, através de um *for loop* semelhante em ambas as linguagens, apenas com as respetivas diferenças a nível de sintaxe, mostrando a seguinte figura a implementação em *Objective-C.*

```
for (int i = 1; i <= _info.size; i++)
{
    if (i % _info.num1 == 0 || i % _info.num2 == 0)
    {
        if (i % _info.num1 == 0 && i % _info.num2 == 0)
        {
            NSMutableString *auxiliar = [[NSMutableString alloc] init];
            [auxiliar appendString:[NSString stringWithFormat:@"%@",_info.word1]];
            [auxiliar appendString:[NSString stringWithFormat:@"%@\n",_info.word2]];
            [fulltext appendString:(auxiliar)];
        }

        else if (i % _info.num1 == 0)
            [fulltext appendString:[NSString stringWithFormat:@"%@\n",_info.word1]];

        else
            [fulltext appendString:[NSString stringWithFormat:@"%@\n",_info.word2]];

    }

    else
    {
        [fulltext appendString:[NSString stringWithFormat:@"%d\n",i]];
    }
```

Excerto de Código 1 - Loop Teste 1 (Objective-C)

Apesar de não ter sido pedido no enunciado do exercício, eu implementei também uma pequena interface gráfica, recorrendo ao *storyboard* do *XCode*.

O *storyboard* é uma interface visual, disponibilizada pelo *IDE XCode,* que permite a criação do *design* do ecrã sem a necessidade de escrever o *XML*, permitindo assim adicionar e arrastar elementos de *UI/UX* como *Views, Labels, TableViews*, etc.

A figura seguinte mostra o resultado obtido no storyboard do *XCode*.

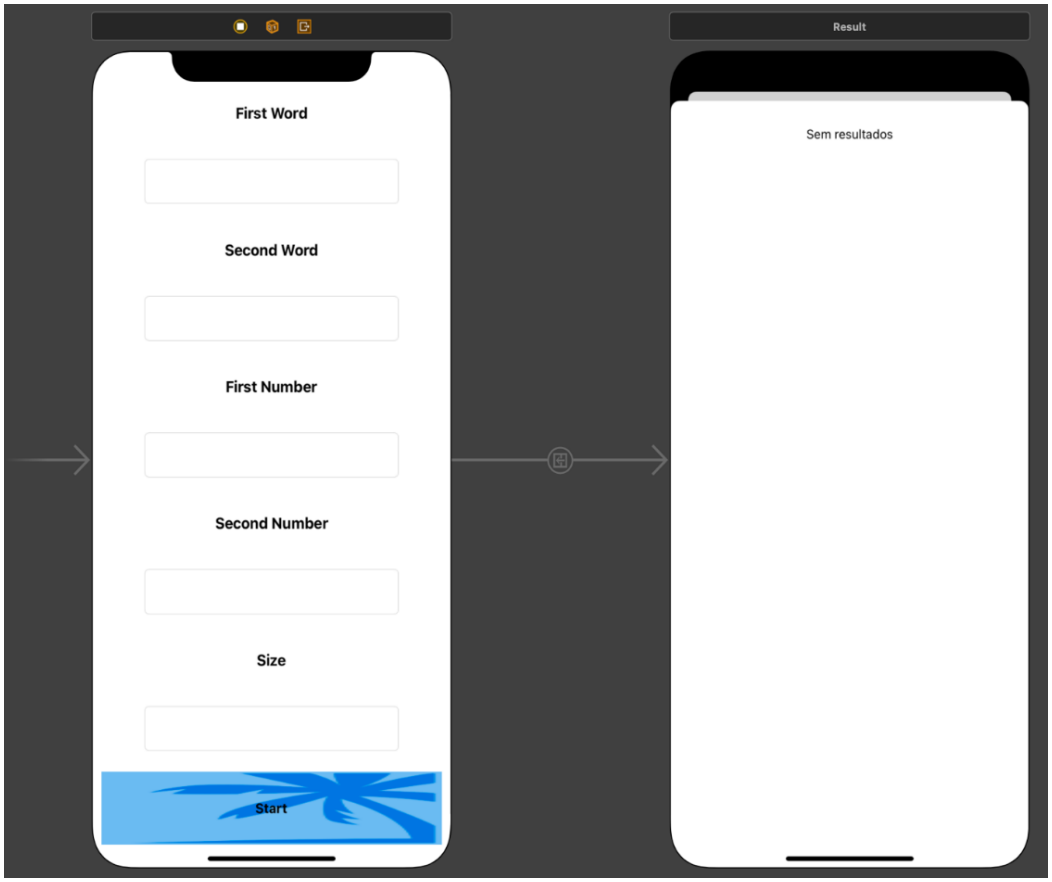


Figura 1 - Storyboard Teste 1

3.1.3. Resultado

O resultado obtido foi bom, uma vez que consegui fazer mais do que o que tinha sido inicialmente pedido, e consegui com que tudo ficasse a funcionar como eu tinha idealizado inicialmente.

Nesta fase, no entanto, ainda não tínhamos abordado a questão do *auto-layout* e como consequência este ecrã apenas fica devidamente formatado no iPhone 12 Pro Max, em qualquer outro irá ficar desformatado devido às diferentes dimensões do ecrã. Seguem-se, no entanto, as imagens do resultado final no simulador do iPhone 12 Pro Max.

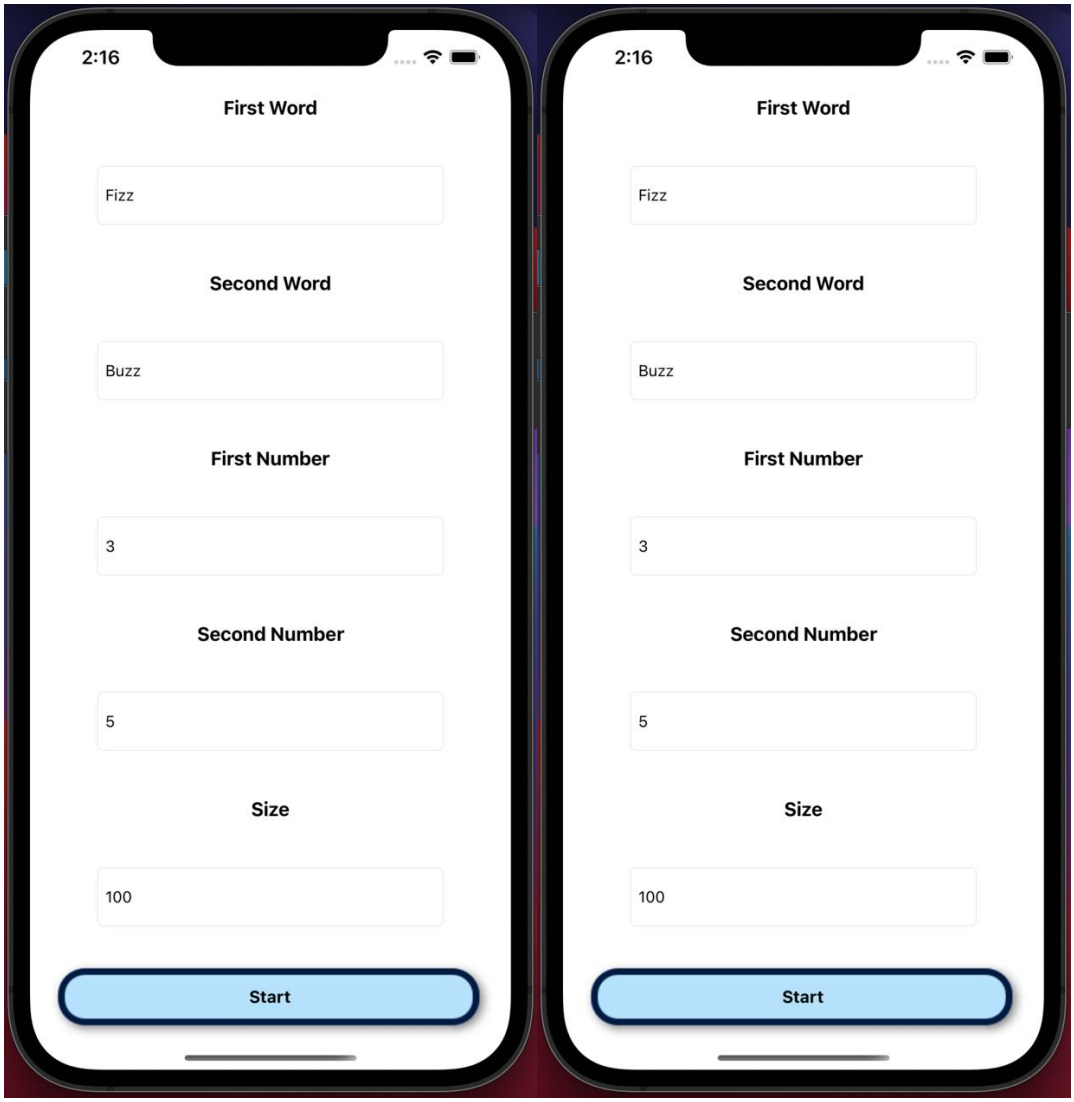


Figura 2 - Resultado Teste 1

3.2. Teste 2 iOS

Este exercício tinha como objetivo demonstrar conhecimentos relativamente à parte de *UI/UX*. Assim, foram nos dados 3 ecrãs que deveríamos replicar e que deveriam ficar bem formatados em qualquer um dos modelos de *iPhone*, entrando aqui a parte do *auto-layout.* Será também de salientar que um dos ecrãs se deveria adaptar para funcionar também em modo *landscape*. Em termos de código não relativo a *UI/UX* apenas foi necessário implementar dois contadores, um para cada uma das equipas que fosse incrementado quando um dos botões fosse clicado.

3.2.1. Abordagem

Tendo em conta que o objetivo deste exercício era testar o meu conhecimento na parte dos componentes de *UI/UX* foi precisamente por onde eu comecei, por montar os *layouts*, recorrendo ao *storyboard* e atribuindo-lhe as *constraints* para que os mesmos se conseguissem adaptar a qualquer dimensão de ecrã. Após implementado e testado passei sim à parte do código, onde nada mais fiz do que implementar dois contadores, e transportar diferentes informações para um outro ecrã dependendo da equipa que fosse selecionada.

3.2.2. Solução

Decidi recorrer a *StackViews* pois achei serem estruturas muito uteis quando se trata do *auto-layout* uma vez que elas próprias se conseguem adaptar ao tamanho do ecrã onde são apresentadas e redimensionam os componentes que a constituem. Decidi também implementar uma *Scroll View* no ecrã que deveria funcionar em *landscape* para possibilitar o *scroll* caso nesse modo algum componente ficasse "escondido". A figura seguinte mostra o resultado no *storyboard.*

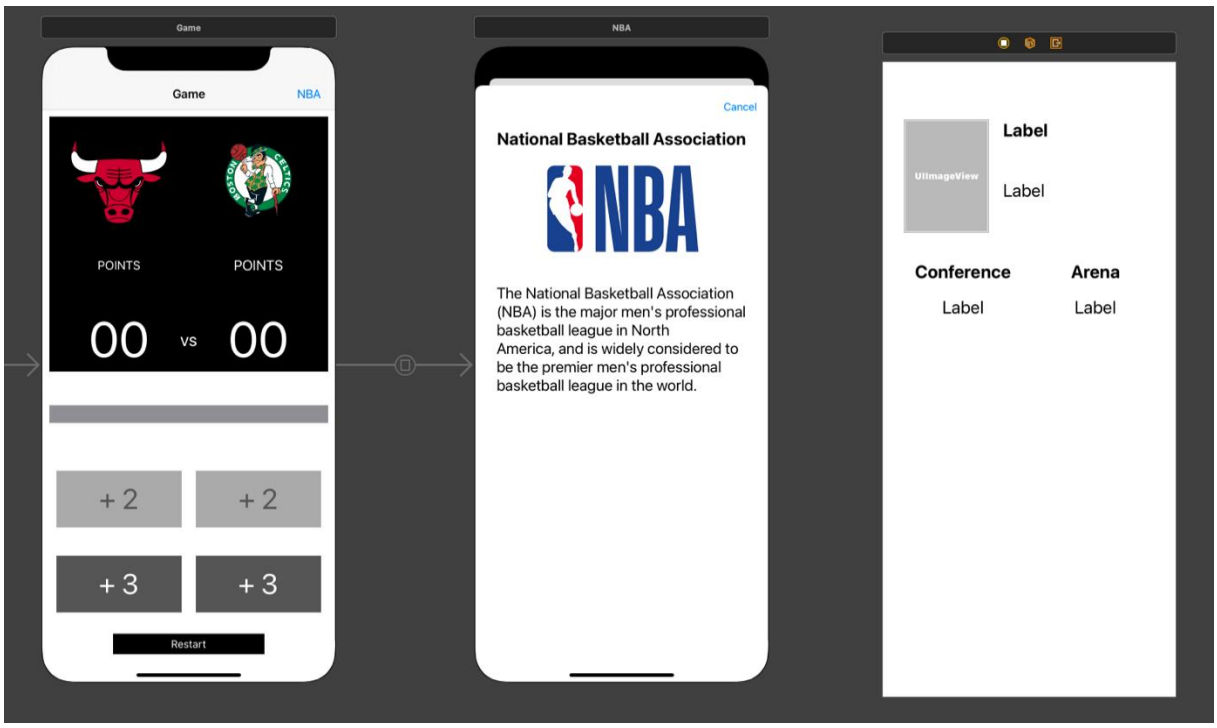


Figura 3 - Storyboard Teste 2 iOS

3.2.3. Resultado

Mais uma vez, consegui implementar o pedido, e tive também tempo para descobrir mais algumas possibilidades, como por exemplo a execução de áudio quando clicado no botão, sendo que quando se clica no logótipo de uma das equipas para além de ser apresentada a informação sobre a mesma é reproduzido o seu hino. O *landscape* ficou funcional graças à *Scroll View* e o ecrã dos pontos ficou bem formatado independentemente do dispositivo onde era apresentado. Para além disso também os contadores ficaram a funcionar conforme o esperado.

As seguintes figuras mostram o resultado em tempo de execução em dispositivos com diferentes dimensões.

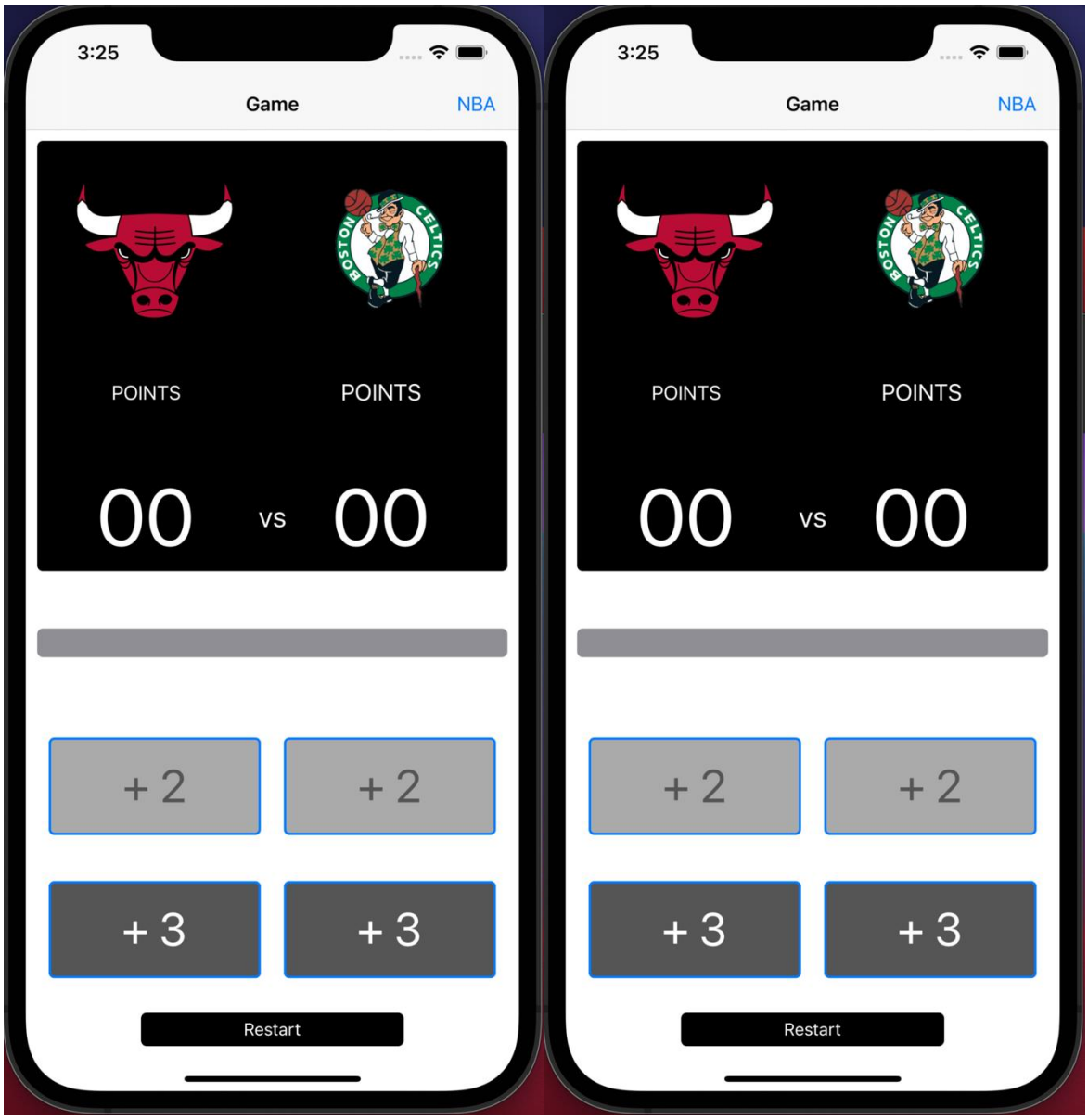


Figura 4 - Resultado do Ecrã dos Pontos (Teste 2)

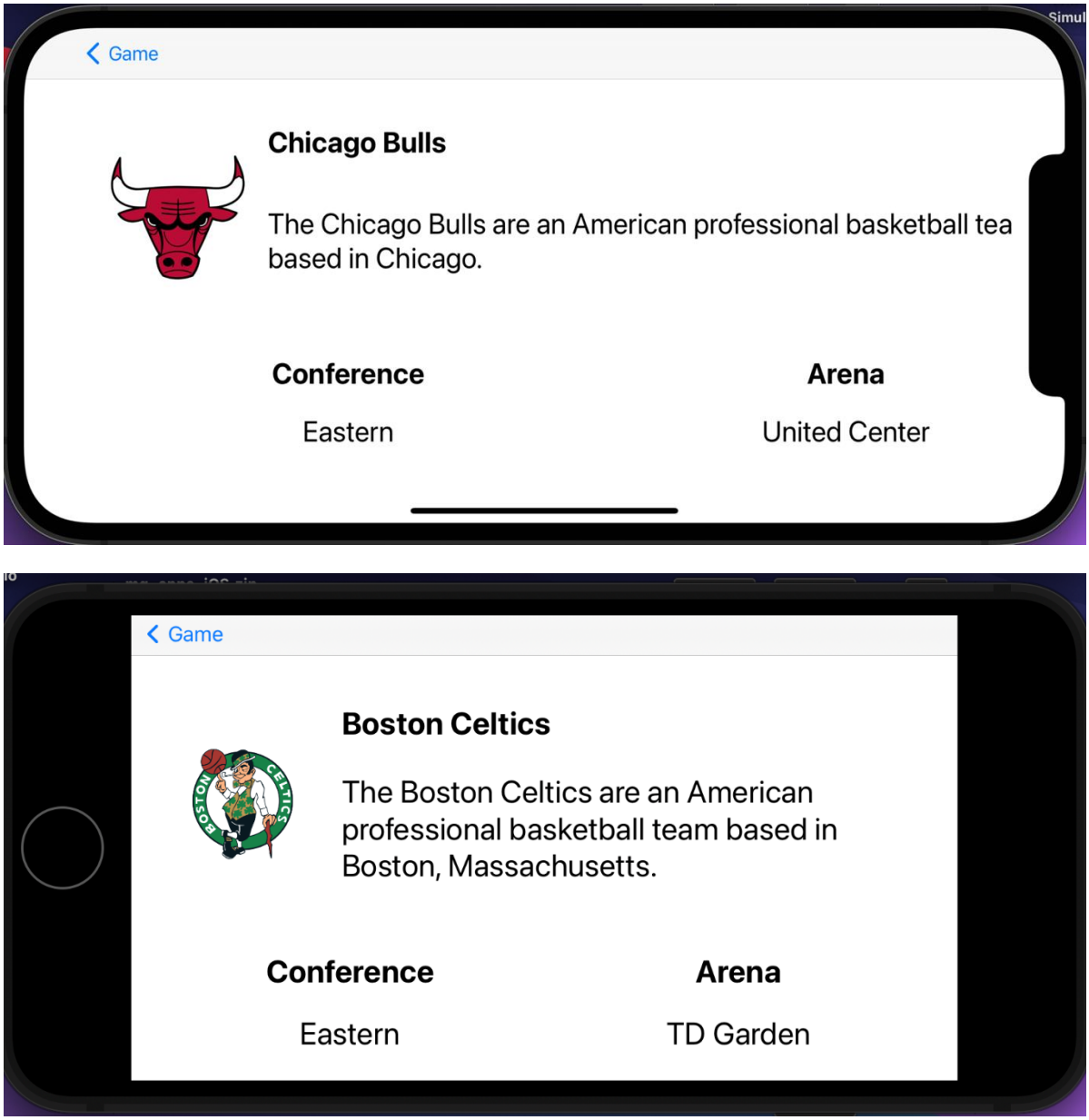


Figura 5 - Resultado modo landscape (Teste 2)

3.3. iOS Project – Currency Converter App

Como projeto de conclusão de academia, foi-me pedido que desenvolvesse um conversor de moeda, com *Swift* e *Objective-C* em paralelo, de forma a aplicar os conhecimentos obtidos sobre cada uma das linguagens, bem como de *UI/UX* pois era necessário seguir os layouts fornecidos no enunciado, sendo também necessário utilizar *REST APIs* que devolvessem os valores das moedas.

3.3.1. Abordagem

Decidi começar pela construção das *cells* para as *TableViews* e pelos layouts para posteriormente poder fazer a conexão aos serviços para poder receber os dados e tratá-los de forma a mostrar as informações necessárias. As condições impostas foram de que a parte de *Networking*, ou seja, a parte de ir buscar informação aos serviços, teria de ser implementada em *Objective-C* e os *controllers* teriam de ser implementados em *Swift*.

3.3.2. Solução

Tendo em conta que a ideia seria mostrar a mesma informação vezes sem conta, a *TableView* seria a melhor solução uma vez que a mesma mostra a informação, em modo de tabela, as vezes que forem necessárias, tratando também do *scroll* caso seja necessário.Assim, após a construção das *cells* e de as ter associado à respetiva *TableView* tratei da implementação do *layout* que permitia efetuar conversões.

Posto isto e após a conexão aos serviços, utilizando o *pod AFNetworking* utilizei também o *pod Kingfisher* para tratar do carregamento das imagens na *TableView.* Os *pods* nada mais são do que bibliotecas externas que nos podem fornecer algumas funcionalidades extra, sem a necessidade de as implementar uma vez que alguém já tratou disso por nós, sendo necessário o gestor de dependências CocoaPods para a utilização das mesmas, daí a origem do nome “pod”.

Tendo em conta que um dos serviços apenas devolvia os valores relativos a um código, foi necessário cruzar essa informação com a de um outro serviço que devolvia o nome da moeda bem como a bandeira do seu país para conseguir construir os layouts semelhantes aos que foram pedidos. As seguintes imagens são relativas ao esquema do *storyboard* bem como às *cells* da *TableView.*

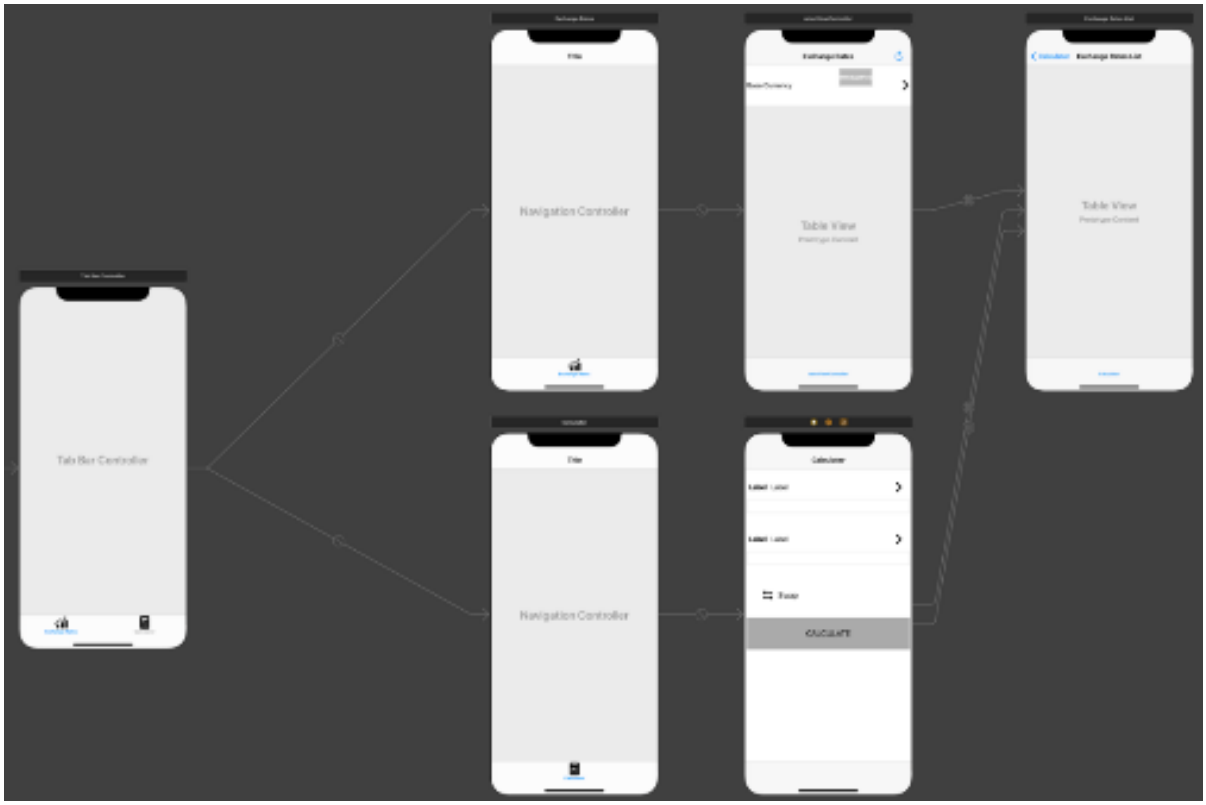


Figura 6 - Solução iOS Project Storyboard

Figura 7 - Solução iOS Project (Custom Cells)

O seguinte excerto de código mostra uma chamada aos serviços que devolvem os valores das moedas relativamente a um código de moeda em específico, passado por parâmetro, com o auxilio do *pod "AFNetworking"*, anteriormente referido, para tratar do pedido aos serviços.

```
+(void)getRates:(NSString *)code withCompletion:(void (^)(NSDictionary *))Block
{
    NSString *requestString = [[NSString alloc] init];
    requestString = [NSString stringWithFormat:
      @"https://api.exchangerate.host/latest?base=%@",code];
    NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration
      defaultSessionConfiguration];
    AFURLSessionManager *manager = [[AFURLSessionManager alloc]
      initWithSessionConfiguration:configuration];
    NSURL *url = [NSURL URLWithString:requestString];
    NSURLRequest *request = [NSURLRequest requestWithURL:url];

    NSURLSessionDataTask *dataTask = [manager dataTaskWithRequest
      :request uploadProgress:nil downloadProgress:nil completionHandler:
      ^(NSURLResponse *response, id responseObject, NSError *error)
    {
        if (error)
            NSLog(@"Error: %@", error);

        else
            NSDictionary *jsonResponse = [responseObject valueForKey:@"rates"];
     }

}
```

Excerto de Código 2 - Solução iOS Project (Chamada Serviços)

Já no próximo excerto de código, em *Swift*, podemos ver a chamada de dois métodos, o "*getRates"* e o "*getCountries" fazendo* um cruzamento da informação para associar os valores de moeda recebidos por parte de uma *API* e associar à bandeira e nome do país recebidos pela outra *API*.

```
Networking.getRates(userDefaults.string(forKey: "default"),
    withCompletion: { jsonResponse in
      if let jsonResponse = jsonResponse {

        for (key, value) in jsonResponse
        {
          let valor:NSNumber = value as! NSNumber

          Networking.getCountries(key.description, withCompletion: { country in
            if let country = country {

              //Variável Auxiliar
              let ratesModel = RatesModel()

              //Atribuir valores ao elemento ratesModel
              ratesModel.flagPng = country.flagPng
              ratesModel.currencyName = country.currencyName
              ratesModel.currencyCode = key.description
              ratesModel.currencyValue = valor.stringValue
              let date: Double = Date().timeIntervalSinceReferenceDate
              ratesModel.timeStamp = Date(timeIntervalSinceReferenceDate: date)

              //Atribuir os valores aos componentes da View
              if (key.description == (self.userDefaults.string(forKey: "default")!))
              {
                self.currencyNameLabel.text = country.currencyName
                self.currencyCodeLabel.text = key.description

                let url = URL(string: country.flagPng)
                self.imageView.kf.setImage(with: url)
              }

              //Adicionaar o elemento a um array
              self.rates.append(ratesModel)

              DispatchQueue.main.async {
                //Reload da tableView
                self.tableView.reloadData()
              }
            }
          })
```

Excerto de Código 3 - iOS Project Cruzamento dos Valores Recebidos

3.3.3. Resultado

Como todos os outros exercícios, também este tinha um *deadline* eu tive alguns problemas ao início, na criação do projeto que me demoraram algum tempo a resolver pelo que o meu tempo para implementação do código em si ficou muito reduzido. Apesar de ter conseguido entregar o projeto acabado e a funcionar, não fiquei muito satisfeito com o código produzido porque achei que ficou muito confuso e que caso tivesse mais tempo certamente teria optado por implementar de outra forma, no entanto e em termos de *UI/UX* estava de acordo com o pedido e mesmo o conversor em si estava funcional.

As seguintes imagens mostram os ecrãs do programa em tempo de execução.

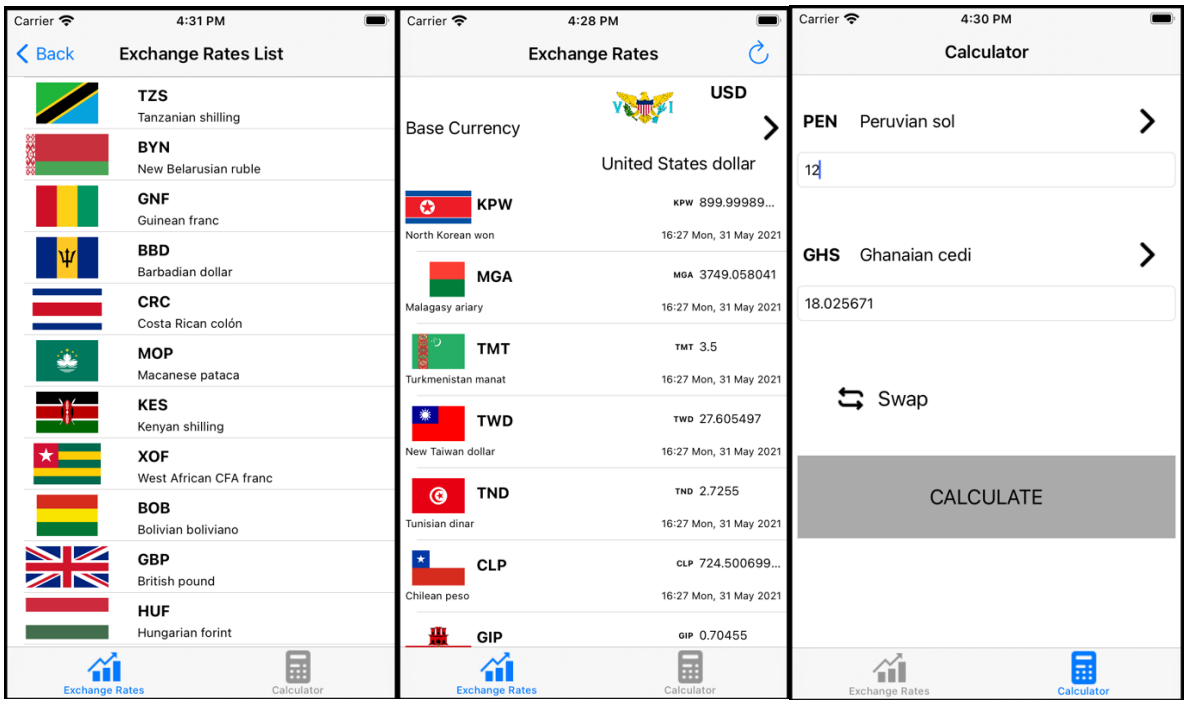


Figura 8 - Resultado iOS Project

3.4. Balanço Final da Academia

Com a conclusão da academia, sinto-me realmente preparado para integrar um projeto, seja este desenvolvido em *Swift* ou *Objective-C*. Claro está que a complexidade dos problemas será outra, no entanto em termos de sintaxe e de estruturas de dados sinto que estou completamente consciente do que cada uma das linguagens oferece. Tendo em conta que nunca tinha trabalhado com tecnologia móvel, esta academia forneceu-me bases e ferramentas fundamentais para que a minha integração no projeto seja bem-sucedida.

4. Programa Arredondamentos

O projeto em que fui inserido tem como nome “Produto Arredondamentos” e consiste na adição de uma nova funcionalidade, denominada “Arredondamentos”, à aplicação móvel do Banco Montepio, a M24, desenvolvida pela IT Sector.

A metodologia de desenvolvimento utilizada neste projeto é o *Scrum.* Cada *sprint* tem a duração de duas semanas e começa com uma reunião onde estão presentes todos os intervenientes do projeto, tendo a mesma a duração de sensivelmente uma manhã. Nesta reunião é feito todo o planeamento da *sprint,* todas as *User Stories* são analisadas em conjunto de forma a estimar os tempos de execução das tarefas.

Existem também reuniões diárias onde cada parte integrante deverá relatar o trabalho desenvolvido no dia anterior de forma a perceber se as tarefas estão a cumprir os tempos estimados ou se existe algum impedimento que irá influenciar a quantidade de trabalho que será desenvolvida na *sprint*.

A plataforma utilizada para gestão de tarefas é o *Azure DevOps*.

5. Primeira Sprint

Na minha primeira *sprint* já me foram atribuídas algumas tarefas no *DevOps*, e tendo em conta que ainda era tudo muito novo para mim, acredito que, apesar de não ter sido perfeito, o meu desempenho tenha sido no mínimo satisfatório.

5.1. User Stories

Tendo em conta que algumas das tarefas consistam na construção do layout, semelhante ao apresentado nas *User Stories* e nas *mock-ups* que me foram disponibilizadas no *Figma, p*ara que seja percetível neste relatório qual o trabalho que me foi pedido, seguem-se as imagens das *User Stories* onde eu tive que atuar.

- Eu, enquanto utilizador do Montepio24, quero ter a opção de selecionar de entre as minhas poupanças em qual ou em quais vou aplicar o meu arredondamento.

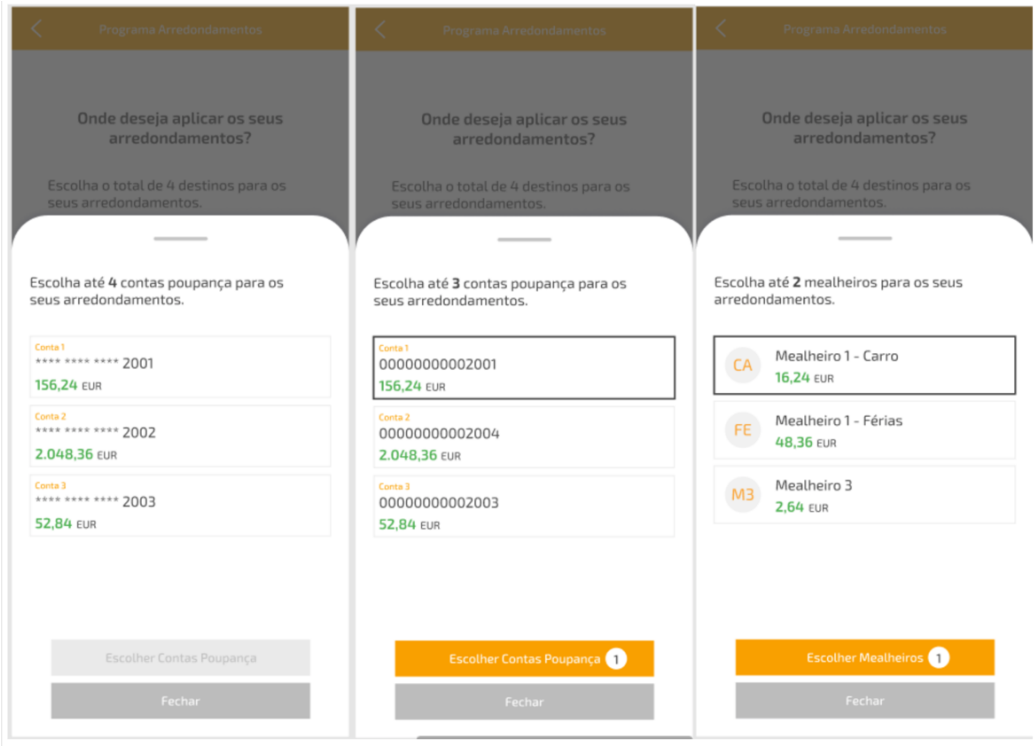


Figura 9 - User Story Selecionar as minhas poupanças – Modal

- Eu, enquanto utilizador do Montepio24, quero ter a opção de selecionar de entre as minhas poupanças em qual ou em quais vou aplicar o meu arredondamento.

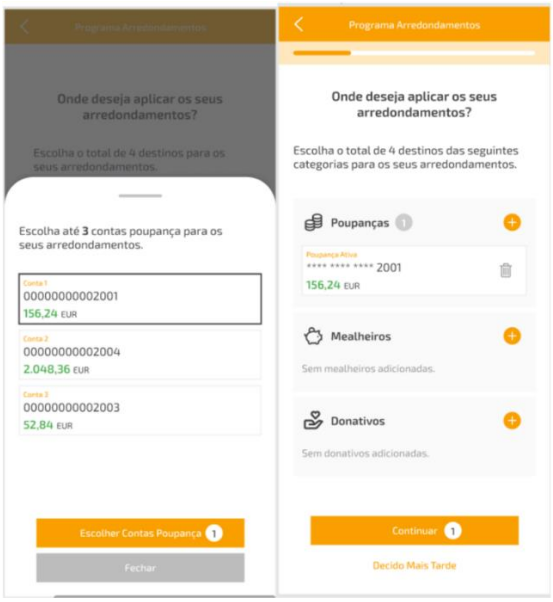
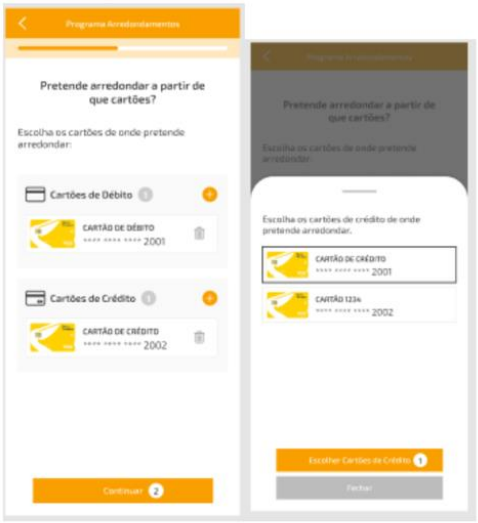


Figura 10 - User Story Selecionar as minhas poupanças (criação do layout)

- Eu, enquanto aderente (utilizador do Montepio 24), quero saber quantos cartões de débito, dos quais sou titular, para os associar ao Programa de Arredondamento, escolhi.

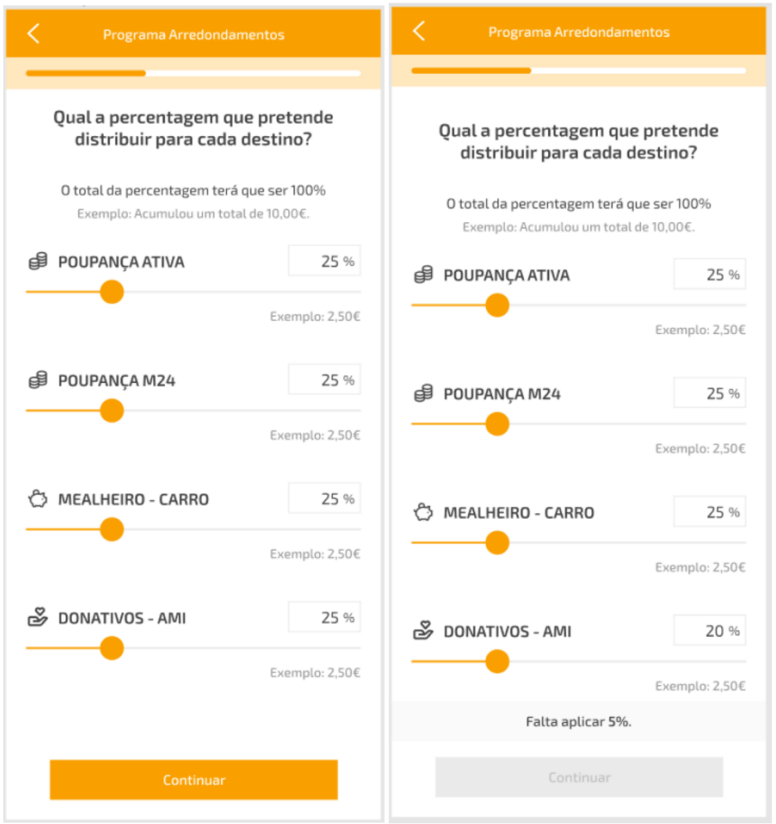


Texto dos botões que poderão considerar contadores:

1. Continuar
2. Escolher Cartões de Débito
3. Escolher Cartões de Crédito

Figura 11 - User Story Selecionar os meus cartões de débito – Contadores

- Eu, enquanto utilizador do Montepio24, quero ter a possibilidade de definir a percentagem de distribuição pelos meus destinos, através de barras de preenchimento.



Textos:

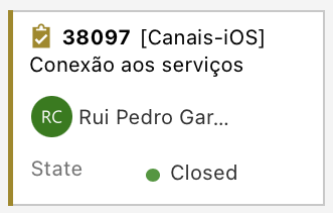
1. Qual a percentagem que pretende distribuir para cada destino?
2. O total da percentagem terá de ser 100%.
3. Exemplo: Acumulou um total de 10,00 €

Figura 12 - User Story Distribuição pelos meus destinos – Barras

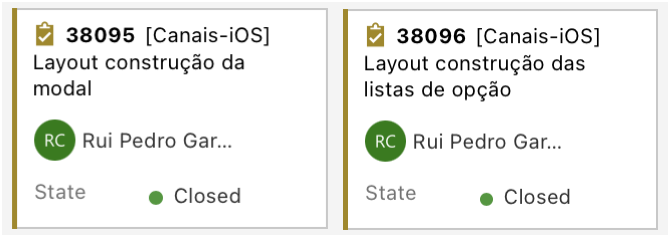
5.2. Tarefas Atribuídas

As imagens anteriores mostram as *User Stories* onde eu tive que atuar, passo agora a enumerar as tarefas que tive que desempenhar relativamente a cada uma delas.

- **Selecionar as minhas poupanças – Modal**

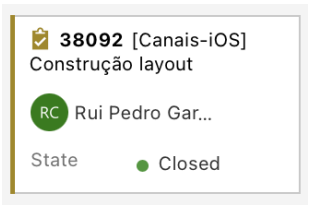


- **Selecionar as minhas poupanças (criação do layout)**



- **Selecionar os meus cartões de débito – Contadores**

- **Distribuição pelos meus destinos – Barras**



Como é percetível pelas imagens anteriores, na primeira User Story o meu papel foi o de estabelecer a conexão com os serviços, relativamente à segunda, tive de criar o *layout* da *modal* e das listas de opção, em relação à terceira apenas tive de implementar alguma lógica de contadores e introduzir o contador no botão e para terminar, na *Story* das barras o meu papel foi também o de criar o *layout*.

5.3. Soluções Implementadas

Neste subcapítulo irei explicar a minha abordagem para a resolução das tarefas que me foram propostas, por *User Story*.

5.3.1. Selecionar as minhas poupanças – Modal

Neste projeto existe uma classe chamada "*ITSTransactionsManager*" que contém um método responsável pela chamada aos serviços, devolvendo um *NSDictionary* da resposta em *json.* Sendo apenas necessário definir o tipo de transação, ao qual está associado o *URL* da mesma.

O código seguinte mostra um excerto do que está implementado dentro do *Transaction Manager,* tendo em conta que é este que retorna o dicionário "*result*". Esse dicionário é posteriormente utilizado para popular um *NSMutableArray* chamado "*resultObj*" de onde será retirado o conteúdo relativo às contas poupança, conteúdo esse que será depois enviado para o ecrã que irá ser apresentado de seguida, que é o ecrã da modal de seleção de poupanças, como se pode ver na *User Story*.

```
self.resultObj = [[AutomaticallySavingsOnboardStep0 alloc] initWithDictionary:result];
for (AutomaticallySavingsDestinos *savingType in self.resultObj.listaOpcoes)
        {
            if ([savingType.index isEqualToString:@"poup"])
            {
                self.initialCounter = [savingType.numMaxSel intValue];
                self.choiceCounter = self.initialCounter;
                vc.counter = self.choiceCounter;
                vc.cellType = 1;
                vc.delegate = self;
                [self.allSavings addObjectsFromArray:savingType.lista];
                vc.dataObject = self.allSavings;
                [self.navigationController
                presentViewController:vc animated:YES completion:nil];
                return;
            }

        }
```

Excerto de Código 4 - Popular as poupanças através dos serviços

5.3.2. Selecionar as minhas poupanças (criação do layout)

Esta User Story foi bastante trabalhosa, pois a mesma modal iria ser apresentada com conteúdos diferentes, pelo que decidi atribuir um inteiro a cada "tipo de chamada" que a mesma fosse receber. Assim, e para além de criar o *layout* da *modal* houve a necessidade de criar as *cells* para cada um dos tipos de dados que poderiam ser mostrados na mesma, contas poupanças, mealheiros, cartões de crédito e cartões de débitos.

As seguintes imagens mostram as quatro *TableViewCells* que foram criadas, sendo que a "*CardOriginTableViewCell*" funciona tanto com cartões de crédito como com cartões de débito.

Figura 13 – CardOriginTableViewCell

Figura 14 – DigitalMoneyTableViewCell

Figura 15 – DonationTableViewCell

Figura 16 – SavingsTableViewCell

Após a criação das *TableViewCells* foi necessário criar o *layout* da *modal,* representado pela figura abaixo, tendo em atenção que a mesma apenas deveria ocupar dois terços do ecrã, e permitir que a *TableView* nele contida conseguisse mostrar diferentes *cells* dependendo de onde era chamado.

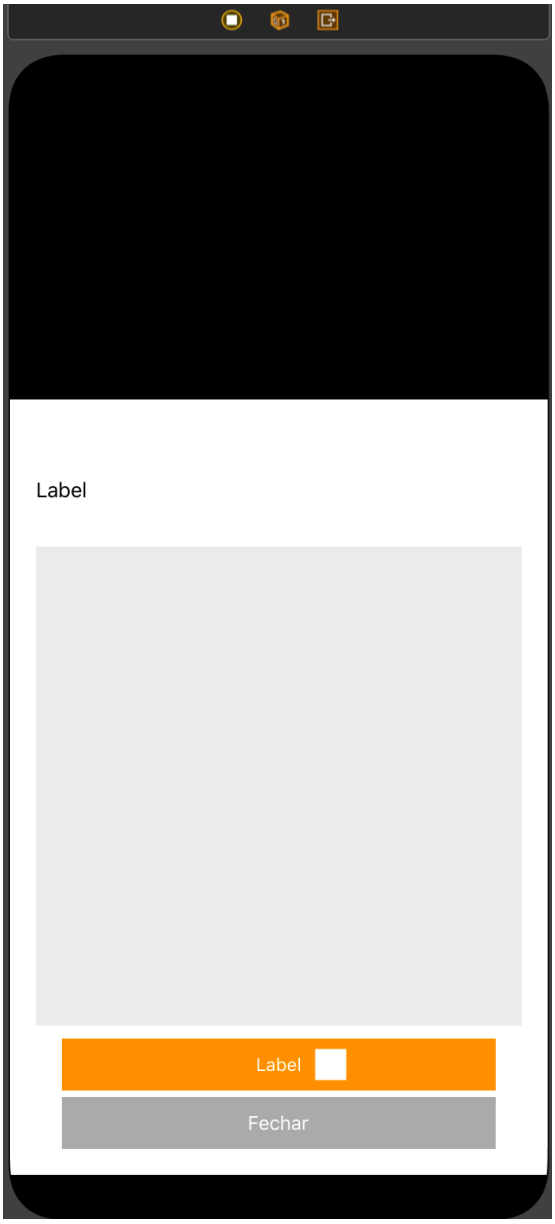


Figura 17 - Layout da Modal

Foi então necessário o seguinte código na *ViewController* da *modal* de forma a perceber qual a *cell* que deveria ser utilizada tendo em conta o inteiro que fosse recebido na variável "*cellType*"

```
-(void)selectCell
{
   switch (self.cellType)
   {
      case 1:
         [self.tableView registerNib:[UINib nibWithNibName:@"SavingsTableViewCell"
            bundle:nil] forCellReuseIdentifier:@"savingsCell"];
         self.cell = @"savingsCell";
         break;
      case 2:
         [self.tableView registerNib:[UINib nibWithNibName:@"DigitalMoneyTableViewCell"
            bundle:nil] forCellReuseIdentifier:@"digitalMoneyCell"];
         self.cell = @"digitalMoneyCell";
         break;
      case 3:
         [self.tableView registerNib:[UINib nibWithNibName:@"DonationTableViewCell"
            bundle:nil] forCellReuseIdentifier:@"donationCell"];
         self.cell = @"donationCell";
         break;
      case 4:
      case 5:
         [self.tableView registerNib:[UINib nibWithNibName:@"CardOriginTableViewCell"
            bundle:nil] forCellReuseIdentifier:@"cardOriginCell"];
         self.cell = @"cardOriginCell";
         break;
      default:
```

Excerto de Código 5 - Escolha da cell a ser mostrada

Relativamente à criação das listas de opção inicialmente pensei reaproveitar as *TableViewCells* tendo em conta que eram muito parecidas, no entanto, e depois de perder muito tempo a tentar fazer com que funcionasse, percebi que não era possível adicionar *UIViews* do tipo *TableViewCell*, pelo que foi necessário criar mais três *UIViews* personalizadas, para as contas poupança, mealheiros e donativos, no entanto a única funcionalidade disponível de momento é a das contas poupança.

As UIViews são exatamente iguais às *TableViewCells* anteriores, apenas com a particularidade de terem também um botão com um símbolo de caixote do lixo.

Assim, penso que a parte mais complexa desta tarefa terá sido a de adicionar as *UIView* à *View* mãe programaticamente, sendo necessário uma pequena "fórmula" para calcular a altura do seu "*container*" em função do número de poupanças selecionadas, sendo o seguinte excerto de código responsável por tudo isso.

```
if ([destiny.index isEqualToString:@"poup"])
{
    [item.label2 setHidden:NO];
    [item.ciurcularView setHidden:YES];

    item = [[SavingsSelectorCellCollectionViewCell alloc]
      initWithFrame:CGRectMake(0, 0, self.savingsContainer.frame.size.width,
      ITSDestinyHight + (8 + ITSSubDestinyHeight) * self.selectedSavings.count)];
    item.stackViewHeight.constant = (8 + ITSSubDestinyHeight) * self.selectedSavings.count;
    self.savingsContainerHeight.constant = ITSDestinyHight + item.stackViewHeight.constant;

    for (int i = 0; i < self.selectedSavings.count; i++)
    {
        [item.stackView addArrangedSubview:
          [self savingsConstructor:self.selectedSavings[i]
          andWidth:item.stackView.frame.size.width]];
    }

    if (self.selectedSavings.count)
    {
        [item.label2 setHidden:YES];
        [item.ciurcularView setHidden:NO];
        item.counterLabel.text = [NSString
          stringWithFormat:@"%ld", self.selectedSavings.count];
    }

    item.label1.text = destiny.descritivo;
    item.label2.text = destiny.nota;
    item.delegate    = self;
    item.row         = i;
    [item.ivType setImage: [UIImage imageNamed:destiny.index]];
    [self.savingsContainer addSubview:item];
}
```

Excerto de Código 6 - Inserir os itens da lista na View mãe

5.3.3. Selecionar os meus cartões de débito – Contadores

Nesta User Story a minha tarefa nada mais foi do que implementar um contador de contas poupança selecionadas, o que em termos de complexidade de lógica não é necessário nada de extraordinário, simplesmente tinha de mostrar o valor do contador do *array* onde estavam a ser registadas as contas selecionadas.

Para além disso foi necessário adicionar o contador ao botão, no entanto o componente do botão não permite tal funcionalidade, pelo que foi necessário colocar um botão transparente por cima de uma *UIView* onde estaria então uma *Label* com o contador. A *UIView* é um componente que permite ter outros componentes dentro, já a *Label* é um componente com a função de mostrar texto.

Esta foi, sem dúvida, a tarefa mais simples que eu tive de resolver nesta *sprint*.

5.3.4. Distribuição pelos meus destinos – Barras

Relativamente a esta User Story eu tinha mais tarefas associadas, no entanto a única que ficou concluída a tempo da entrega dentro do *deadline* da *sprint* foi a da construção do layout. Ao longo das duas semanas, houve então um problema que me demorou um bocadinho mais de tempo para perceber como se iria resolver e para além disso, houveram algumas falhas e manutenção da parte dos serviços, que não permitiam que eu conseguisse testar o que implementava, pelo que sempre que os serviços estavam em baixo eu não conseguia implementar nada. Também esse atraso existiu na parte da equipa de desenvolvimento *Android.*

Assim, esta tarefa é única e exclusivamente sobre a criação do *layout*, que foi criado utilizando a lógica do *layout* das listas de opção, pelo que criei uma *UIView* personalizada com um *slider* dentro e depois adicionava à View mãe tantas quanto fossem necessárias, sendo que havia um máximo de quatro destinos definido, sendo que as próximas imagens são relativas à sua implementação no *storyboard.*

O Slider é um componente que permite ser arrastado e nos devolve o valor da sua posição atual.

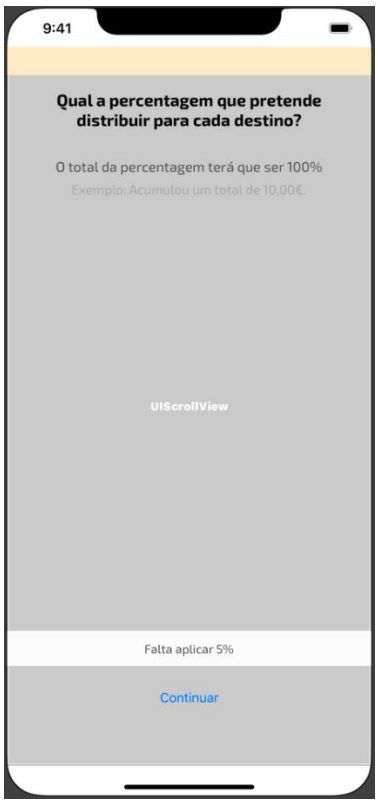


Figura 18 - Layout Distribuição pelos meus destinos

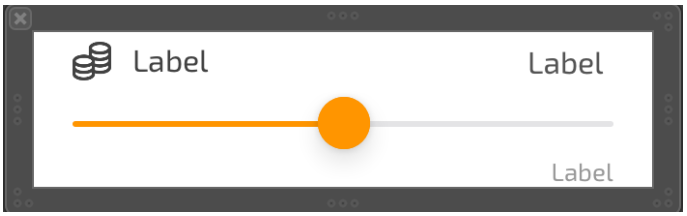


Figura 19 – PercentageSelectorView

5.4. Resultados

As seguintes imagens mostram testes das *TableViewCells* com dados "mockados" apenas para testar a lógica antes de implementar no projeto e fazer a ligação aos serviços, até porque nos serviços, à data, apenas existem informações relativas às contas poupança e aos cartões de débito.

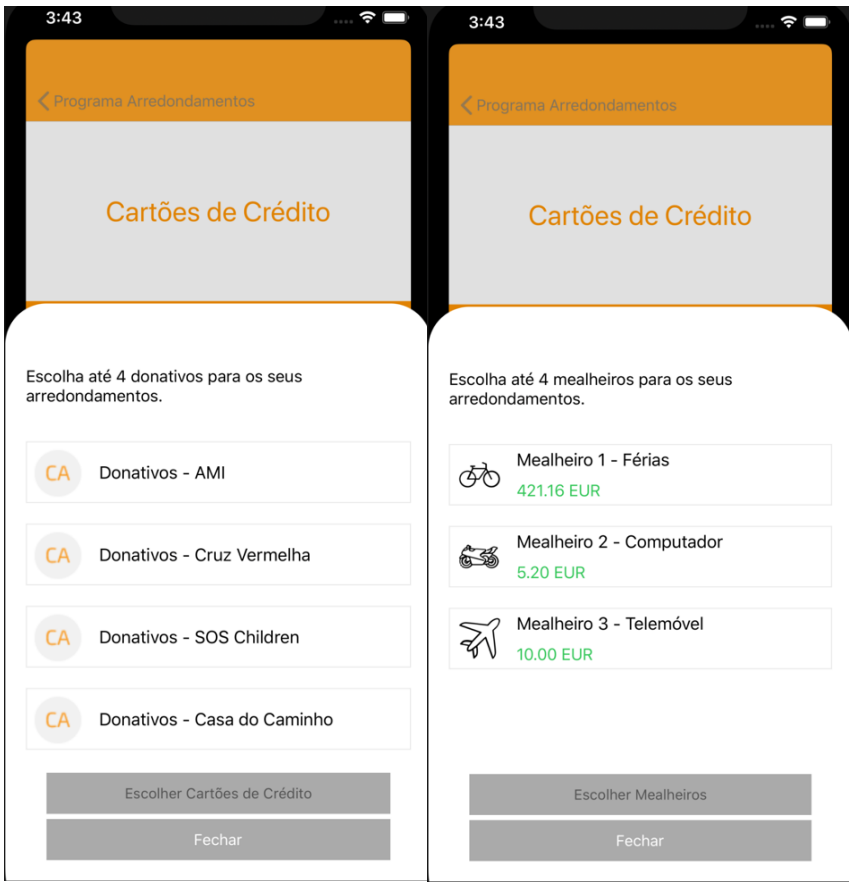


Figura 20 - Resultado donativos e mealheiros

Figura 21 - Resultado cartões de crédito e contas poupança

As imagens seguintes mostram os ecrãs após implementados na aplicação e após a conexão aos serviços.

As formatações do saldo das contas poupança por algum motivo não são visíveis no simulador que é onde eu tenho sempre testado tudo uma vez que não tenho iPhone. No entanto, os meus colegas testaram em dispositivos físicos e lá realmente funciona.

No entanto tudo o resto é percetível por estas imagens, e mesmo os dados apresentados são “reais” e estão associados a um utilizador Montepio criado para testes.

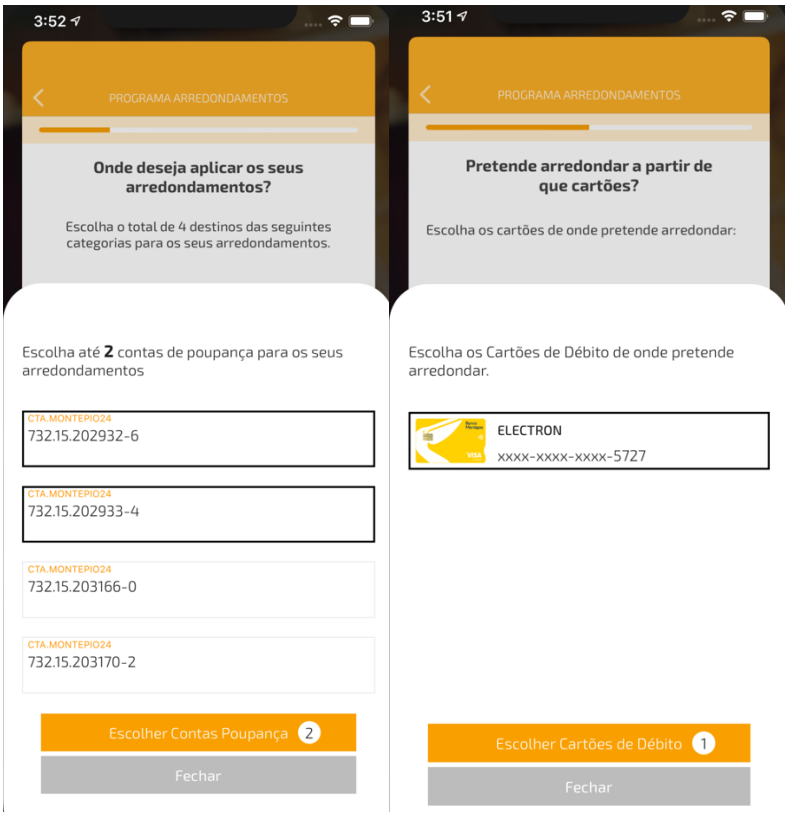


Figura 22 - Resultado cartões de débito e contas poupança

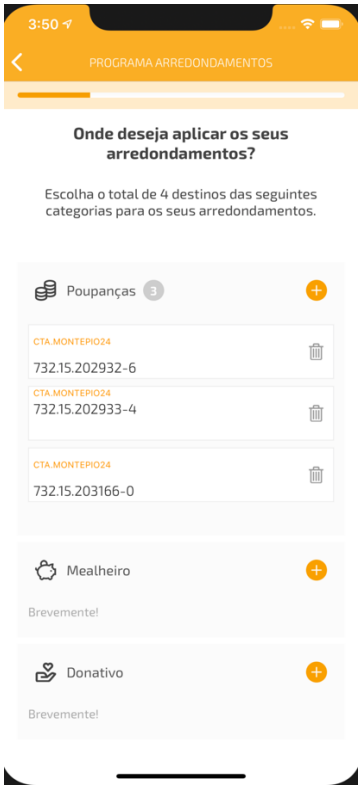


Figura 23 - Resultado lista de opção e contador

Na imagem anterior é possível verificar que as contas poupança selecionadas realmente são apresentadas, bem como o seu contador.

Relativamente ao layout das barras, nesta fase não há qualquer resultado a não ser a parte da implementação no storyboard pois não foi implementada qualquer lógica de forma a fazer o ecrã aparecer antes da conclusão desta sprint, pelo que essas tarefas irão passar para a próxima *sprint*.

6. Segunda Sprint

Neste segunda Sprint, em primeiro lugar foram concluídas algumas tarefas que não foi possível terminar a tempo na sprint anterior, e tendo em conta que um membro da equipa de desenvolvimento *Android* iria entrar de férias, não existiram muitas tarefas a nível de desenvolvimento para não ficar com o trabalho de *iOS* mais adiantado do que o de *Android*.

6.1. User Stories

Mais uma vez, para melhor perceção das tarefas realmente desenvolvidas por mim, passo a mostrar as *User Stories* onde essas tarefas estavam inseridas.

- Eu, enquanto utilizador do Montepio24, quero ter a possibilidade de definir a percentagem de distribuição pelos meus destinos, através de barras de preenchimento.

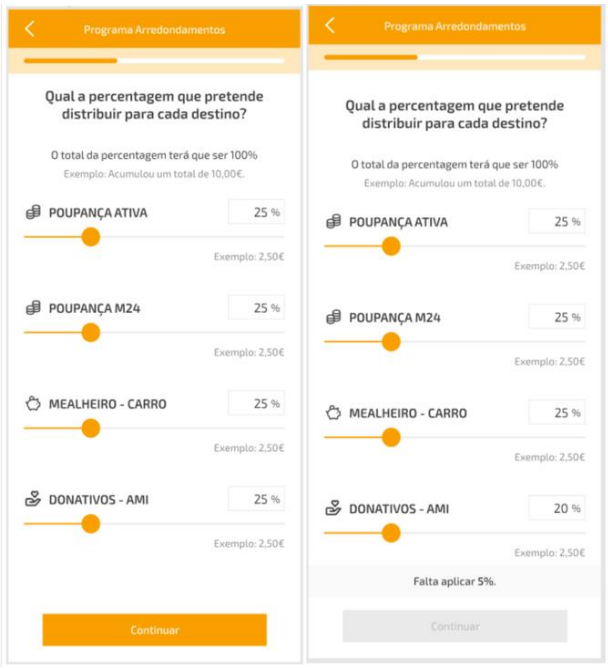


Figura 24 - User Story Distribuição pelos meus destinos – Barras

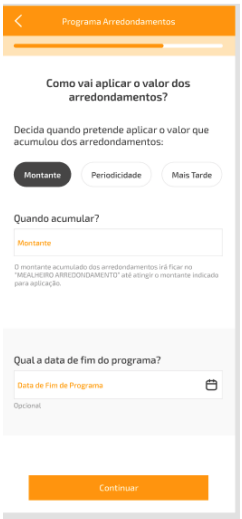
- Eu, enquanto utilizador do Montepio 24, quero ter opção de aplicar nos destinos os meus arredondamentos, apenas quando atingir um valor acumulado de x.

Textos (aplicar nos demais ecrãs Montante, Periodicidade e Mais tarde)

1. *Quando pretende aplicar os seus arredondamentos?*
2. *Indique quando pretende aplicar o valor acumulado dos seus arredondamentos:*
3. *Quando acumular?*
4. *O montante acumulado dos arredondamentos irá ficar no "MEALHEIRO DE ARREDONDAMENTO" até atingir o montante indicado para aplicação.*
5. *Quando pretende terminar o Programa?*
6. *Data fim do Programa* (em laranja)
7. ("Opcional" deverá estar mais visível)

Figura 25 - User Story Valor acumulado

- Eu, enquanto utilizador do Montepio24, quero ter a opção de aplicar nos destinos os meus arredondamentos, apenas quando atingir um valor acumulado de x.



extos (aplicar nos demais ecrãs Montante, Periodicidade e Mais tarde)

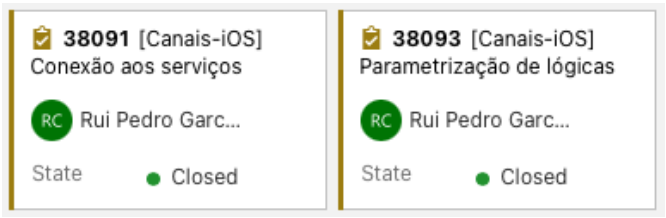
1. *Quando pretende aplicar os seus arredondamentos?*
2. *Indique quando pretende aplicar o valor acumulado dos seus arredondamentos:*
3. *Quando acumular?*
4. *O montante acumulado dos arredondamentos irá ficar no "MEALHEIRO DE ARREDONDAMENTO" até atingir o montante indicado para aplicação.*
5. *Quando pretende terminar o Programa?*
6. *Data fim do Programa* (em laranja)
7. ("Opcional" deverá estar mais visível)

Figura 26 - User Story Periodicidade

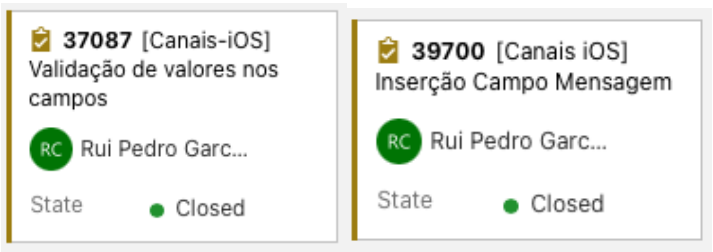
6.2. Tarefas Atribuídas

As seguintes tarefas, que me foram atribuídas nesta *sprint* são todas relativas às três *User Stories* que eu apresentei anteriormente, passo então a mostrar as tarefas por *User Story*.

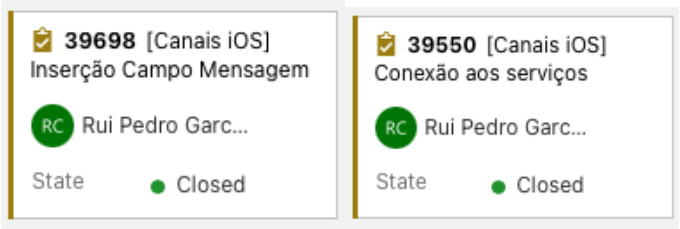
- **Distribuição pelos Meus Destinos – Barras**



- **Valor Acumulado**



- **Periodicidade**



Como é percetível, na *User Story* da distribuição pelos destinos as tarefas pelas quais eu fiquei responsável foram as que não ficaram terminadas na sprint anterior, isto é a parametrização de lógicas para o *layout* criado bem como a ligação aos serviços. Relativamente ao Valor Acumulado, foi necessário adicionar um campo, bem como validar o valor introduzido pelo utilizador. Para concluir, na Periodicidade, à semelhança do Valor Acumulado, foi necessário adicionar um campo e fazer a ligação aos serviços.

6.3. Soluções Implementadas

À semelhança do capítulo anterior, também neste subcapítulo irei explicar a minha abordagem na resolução dos problemas que me foram propostos.

6.3.1. Distribuição pelos meus destinos – Barras

Em relação à parametrização de lógicas, a minha abordagem foi a de criar um *NSMutableArray* que fosse guardar os valores dos *Sliders.* Assim, recorrendo a esses valores, seria possível perceber se a soma era igual a 100 e consequentemente habilitar o botão, se fosse inferior seria necessário mostrar o valor que faltava aplicar para chegar ao 100, e no caso de ser superior não havia nada definido pelo que o botão apenas deveria ficar desabilitado.

Em relação aos serviços, o serviço para o output dos valores das barras ainda não estava disponível pelo que a única ligação que tive de estabelecer foi para receber as poupanças anteriormente selecionadas.

```
-(void) getValue:(int)slider andIndex:(int)index
{
    [self.percents insertObject:[NSString
        stringWithFormat:@"%d", slider]atIndex:index];
    [self refreshView];
}
```

Excerto de Código 7- Leitura dos Sliders

O excerto de código anterior é executado sempre que um *Slider* é movido, e é responsável por atualizar o valor do *Slider* no *NSMutableArray* e posteriormente chama um método "*refreshView*" que irá atualizar o ecrã a partir dos novos valores do *array* de percentagens.

6.3.2. Valor acumulado

A primeira parte desta tarefa, e a mais simples, acabou por ser a de adicionar o campo mensagem, que foi uma *Label* ao layout já construído, sendo posteriormente necessário atribuir à mesma um valor devolvido pelos serviços. Em relação à

validação do valor, e tendo em conta que teria de ser apresentada uma mensagem de erro, decidi criar dois métodos, o "*belowMinAmount*" e o "*AboveMaxAmount*", sendo que a principal diferença entre eles seria a mensagem apresentada, uma vez que um iria pedir um valor superior a *x* e o outro um valor inferior a *y.*

Como tal, usei as seguintes condições para fazer a "filtragem".

```
if (self.choice == 0 && ([self.tfAmount.textField.text isEqualToString:@""] ||
[self.tfAmount.textField.text intValue] < self.minValue))
    {
        [self belowMinAmount];
        return;
    }

    if (self.choice == 0 && [self.tfAmount.textField.text intValue] > self.maxValue)
    {
        [self aboveMaxAmount];
        return;
    }
```

Excerto de Código 8 - Verificação valor superior ou inferior

A varíavel "*choice"* apenas servia para perceber se o ecrã ativo era o do montante, pelo que depois é verificado se o campo está vazio ou se o valor é inferior ao permitido para chamar o *belowMinAmount* ou então se o valor for superior ao suposto, chamar o *aboveMaxAmount*.

Ambas as condições possuem um return pois estão inseridas no *transactionManager*, que é o responsável por mostrar o próximo ecrã quando o botão é clicado, assim, se entrar numa das condições, não irá mostrar o próximo ecrã uma vez que a transição não irá acontecer.

6.3.3. Periodicidade

À semelhança do Valor Acumulado, também aqui foi necessário acrescentar não um, mas dois campos mensagem, pelo que consistia apenas na introdução de duas *Labels* no *Storyboard,* no entanto, uma vez que o *layout* implementado estava a utilizar tamanhos fixos, estava a acontecer que eu dispositivos mais pequenos alguns elementos eram cortados, pelo que foram necessárias algumas alterações no *layout* para resolver este problema, o que atrasou um bocadinho a conclusão desta tarefa.

Uma das *Labels* só é mostrada quando são escolhidos cartões de crédito, pelo que como essa funcionalidade ainda não está disponível a *Label* nunca vai ter conteúdo.

Relativamente à conexão aos serviços, foi necessário criar um objeto de acordo com a estrutura do serviço, de modo a enviar essa resposta do serviço para o próximo ecrã que seria mostrado.

6.4. Resultados

A imagem seguinte mostra o *layout* das barras nos seus três possíveis estados, quando a soma é inferior a 100, quando é igual a 100 e quando é superior a 100.

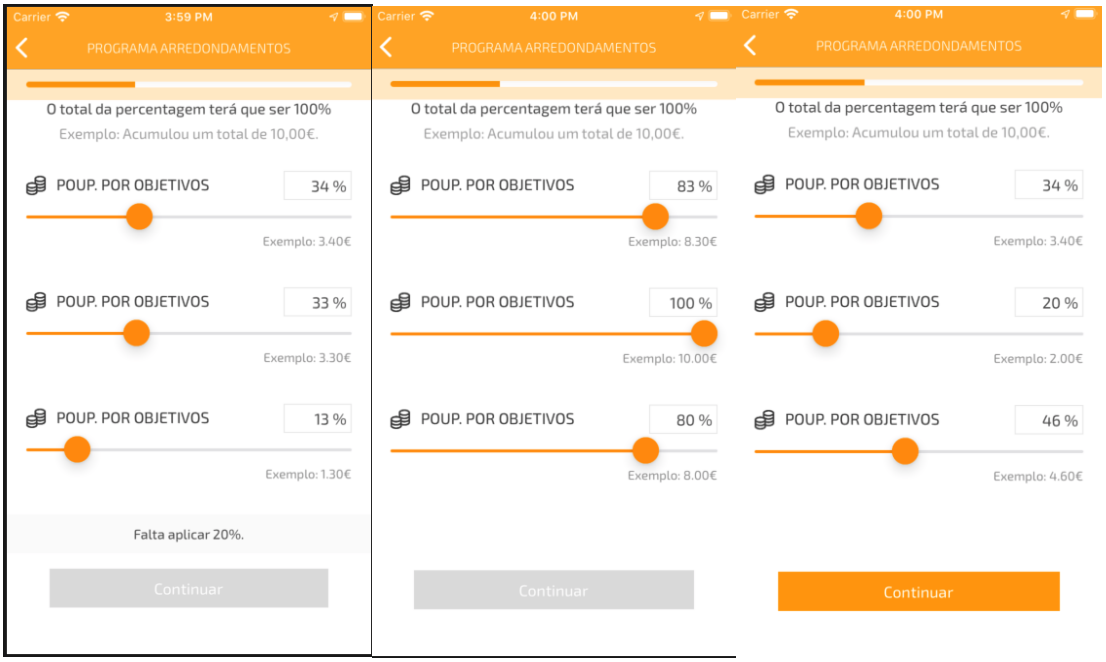


Figura 27 - Resultado Distribuição pelos meus destinos

As próximas imagens são relativas ao campo mensagem inserido, bem como às validações do valor no ecrã “Valor Acumulado”. Mostra assim o erro de quando o valor é muito pequeno, o erro quando o valor é muito elevado e o ecrã antes do clique no *Textfield.*

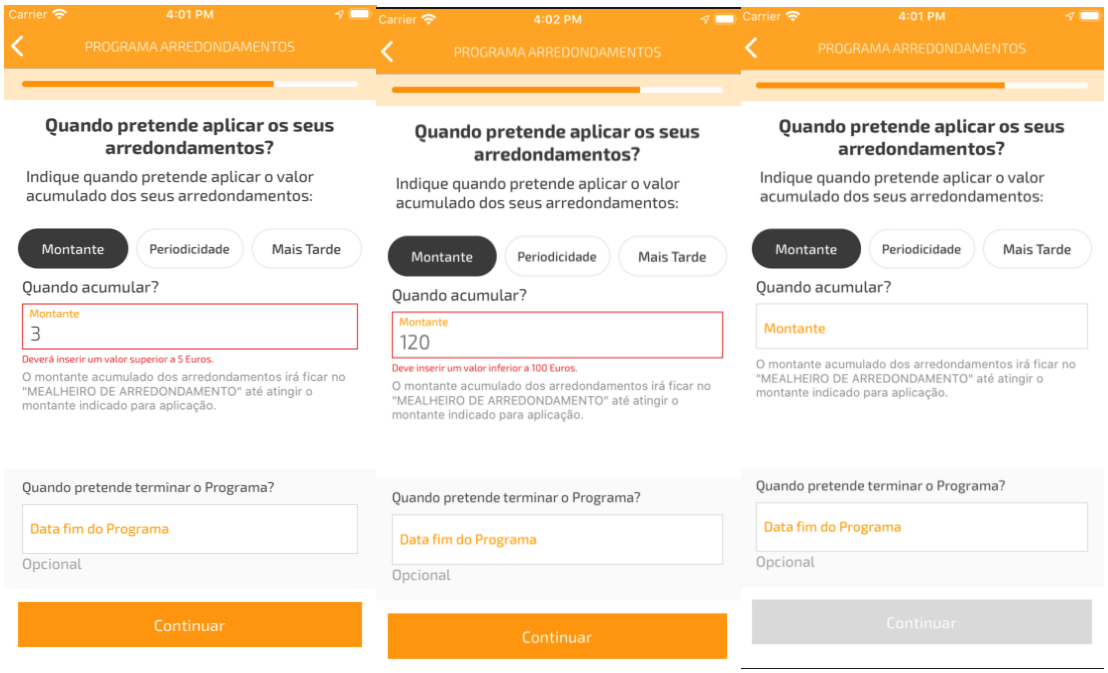


Figura 28 - Resultado Validação de valores

Finalmente fica uma imagem que mostra o campo mensagem no ecrã da Periodicidade.

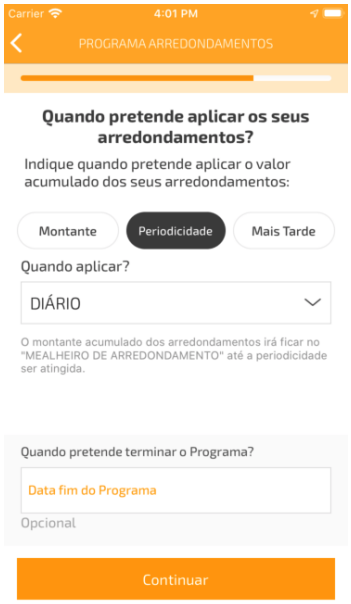


Figura 29 - Resultado Campo Mensagem (Periocidade)

7. Terceira Sprint

Nesta *sprint* não existiram muitas tarefas para o *front-end* que era onde eu estava, sendo a maior parte das tarefas relativas a tarefas da parte dos serviços. Assim, também nesta sprint eu não tive muito trabalho a nível de desenvolvimento.

7.1. User Stories

Passo então a mostrar as User Stories relativas às tarefas que me foram atribuídas na *sprint.*

- Eu, enquanto utilizador do Montepio24, quero ter a possibilidade de definir a percentagem de distribuição pelos meus destinos, através de barras de preenchimento.

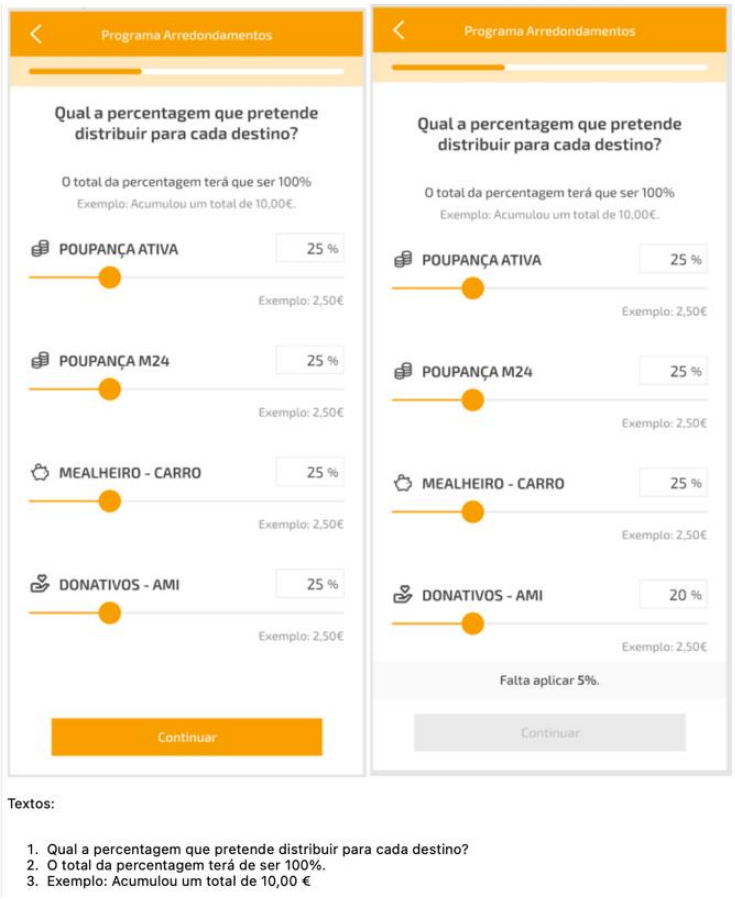


Figura 30 - User Story Distribuição pelos meus destinos - Barras

- Eu, enquanto utilizador do Montepio24, quero ter a opção de ir acumulando os meus arredondamentos até tomar a minha decisão nos destinos pretendidos.

Textos (aplicar nos demais ecrãs Montante, Periodicidade e Mais tarde)

1. *Quando pretende aplicar os seus arredondamentos?*
2. *Indique quando pretende aplicar o valor acumulado dos seus arredondamentos:*
3. *Poderá decidir mais tarde como aplicar o valor arredondado que acumular.*
4. *Quando pretende terminar o Programa?*
5. *Data fim do Programa* (em laranja)
6. ("Opcional" deverá estar mais visível)

Figura 31 - User Story Decido mais tarde

- Eu, enquanto utilizador do Montepio24, quero ser avisado em ecrã sobre a conclusão da adesão ao programa de arredondamentos.

Texto ecrã:
Parabéns! A adesão ao programa de arredondamentos está concluída.

Texto do Botão:

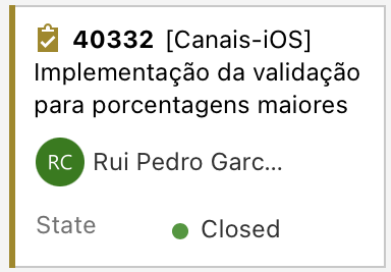
Consultar programa de arredondamentos

Figura 32 - User Story Conclusão Onboarding

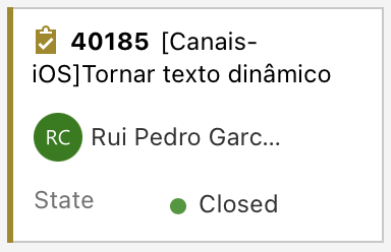
7.2. Tarefas Atribuídas

As seguintes tarefas que me foram atribuídas, são, como disse anteriormente, relativas às *User Stories* apresentadas em cima, pelo que passo a mostrar, à semelhança das *sprints* anteriores quais as tarefas que me foram associadas por *User Story.*

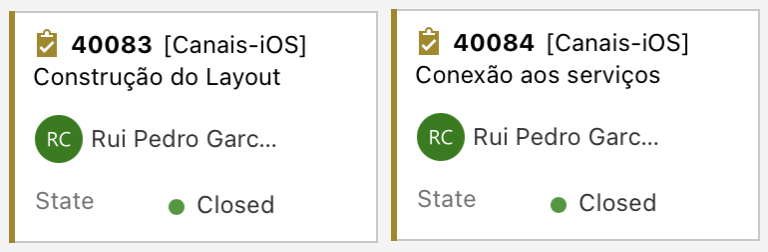
- **Distribuição pelos meus destinos – barras**



- **Decido Mais Tarde**



- **Conclusão Onboarding**



Como é percetível pelas imagens acima, fiquei responsável pela implementação para percentagens maiores na User Story “Distribuição pelos meus destinos – barras”, relativamente à “Decido Mais Tarde” apenas tive de tornar o texto dinâmico, isto é, o conteúdo da *Label* que era estático passou a ser enviado pelos serviços pelo que a minha tarefa era associar o conteúdo da resposta à *Label.* Já relativamente à “Conclusão Onboarding” tive então de construir o *Layout* bem como de o conectar aos serviços.

7.3. Soluções Implementadas

Neste subcapítulo, mais uma vez, irei passar a explicar a minha abordagem na implementação das tarefas anteriormente mostradas.

7.3.1. Distribuição pelos meus destinos – Barras

Tendo em conta que na *sprint* anterior já tinha sido implementada uma validação para percentagens inferiores a 100, ficou também já preparada a lógica para a validação das percentagens superiores, apenas foi necessário acrescentar um "*if*" na função de desabilitar o botão para que fosse então mostrada a mensagem do valor que estaria a ser aplicado em excesso.

Assim, tendo em conta que esta função só seria chamada caso o valor lido fosse diferente de 100, o seguinte excerto de código resolve o problema pedido.

```
if (self.counter < 100)
    self.missingPercentageLabel.text = [NSString stringWithFormat:
      @"Falta aplicar %d%@.", 100-self.counter, aux];
  else
    self.missingPercentageLabel.text= [NSString stringWithFormat:
        @"Aplicou %d%@ a mais.", self.counter-100, aux];
```

Excerto de Código 9 - Validação de percentagens inferiores e superiores

7.3.2. Decido mais tarde

Também esta tarefa foi de simples resolução, tendo em conta que apenas consistiu em adicionar ao objeto da resposta dos serviços um novo campo e associar o mesmo ao valor da *Label* em questão.

7.3.3. Conclusão Onboarding

Relativamente à tarefa da construção do *layout* e tendo em conta a sua simplicidade, não houve qualquer problema com a criação do mesmo. O mesmo não se pode dizer da conexão aos serviços, tendo em conta que este era o último *step* e que nós à data apenas tínhamos desenvolvido o programa até ao *step* 6 foi necessário um ajuste da parte dos serviços para que fosse possível efetuar a chamada deste último passo a partir do 6, o que demorou algum tempo e algumas tentativas até ficar

tudo funcional e conseguirmos obter a resposta correta no *front-end*, tendo esta tarefa sido fechada apenas no último dia da *sprint*.

Ainda relativamente à conexão aos serviços, e por ser um step novo, tive então de definir os objetos de *output* (passados por parâmetro aos serviços) e de *input* (recebidos na resposta dos serviços), bem como a definição do *url* do último *step*.

Após isso e utilizando o *transaction manager* da *ITSector* as chamadas aos serviços são simples.

7.4. Resultados

A seguinte imagem é relativa à validação de percentagens superiores a 100, pelo que por cima do botão é mostrado ao utilizador o valor em excesso para que o mesmo possa efetuar a distribuição de outra forma.

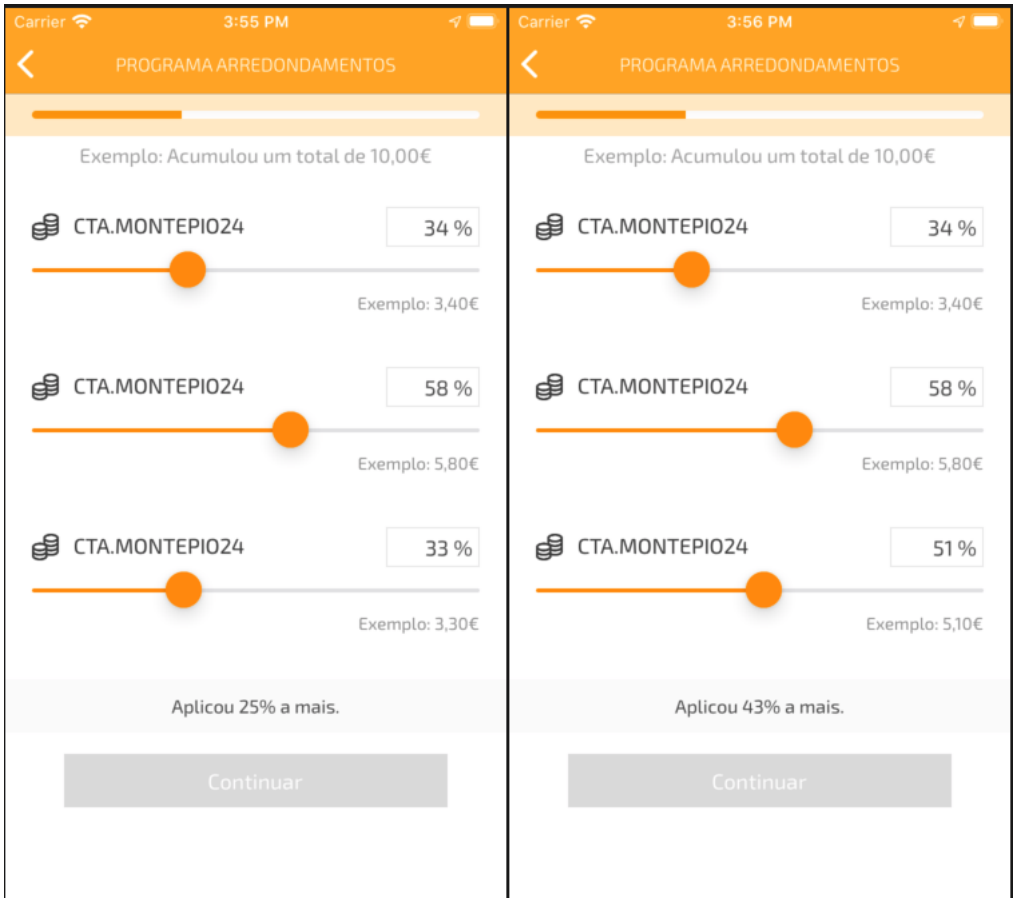


Figura 33 - Resultado Validação de percentagens maiores

A seguinte imagem mostra então o texto recebido da parte dos serviços para mostrar ao utilizador no separador “Mais Tarde”.

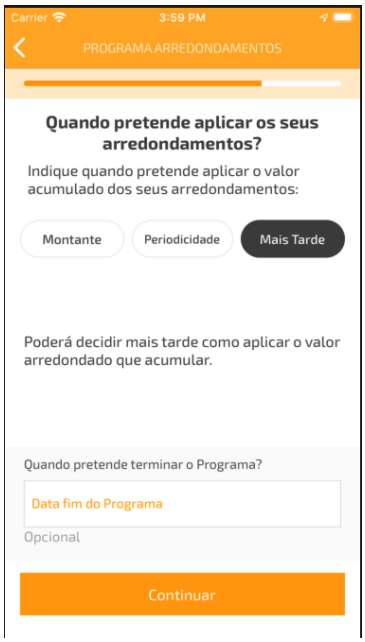


Figura 34 - Resultado texto dinâmico

Para terminar, fica então o resultado do Layout da conclusão do Onboarding, salientando que a mensagem apresentada é recebida dos serviços.

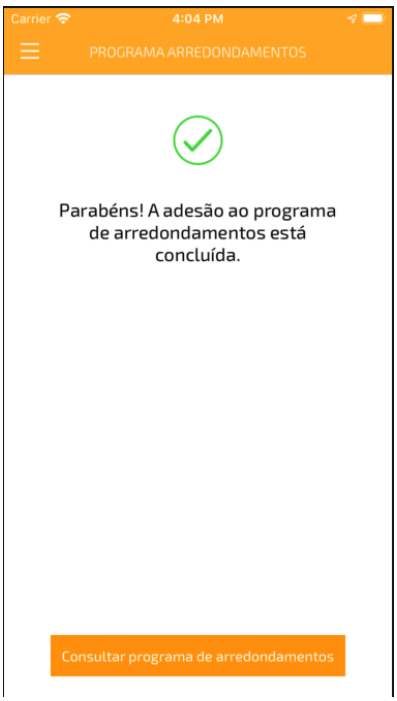


Figura 35 - Resultado Conclusão Onboarding

8. Quarta Sprint

Mais uma vez, também esta sprint foi maioritariamente relativa ao *back-end* as únicas tarefas associadas ao *front-end* foram realizadas por mim. Será de salientar que, para mim, esta sprint apenas teve a duração de um dia, uma vez que começou no meu último dia de estágio sendo que a parte da manhã foi a reunião de planeamento e na parte de tarde finalizei as tarefas que me foram propostas.

8.1. User Stories

Seguem-se as *User Stories* em que eu atuei nesta minha última *sprint*.

- Eu, enquanto utilizador do Montepio24, quero ter a opção de remover destinos que tenha definido no momento da escolha destes.

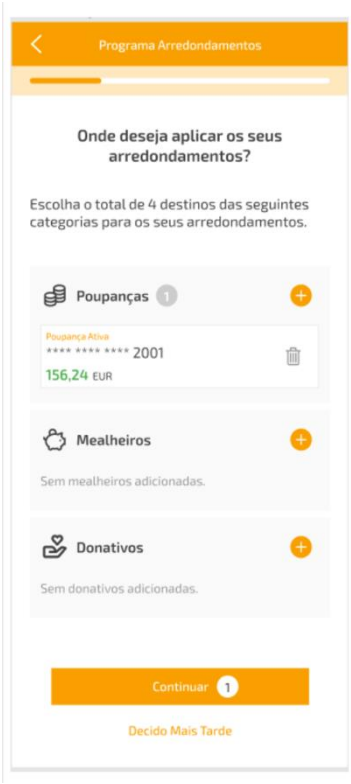


Figura 36 - User Story Remover escolhas de destinos

- Eu, enquanto utilizador do Montepio24, quero ter a opção de remover origens que tenha definido no momento da escolha destas.

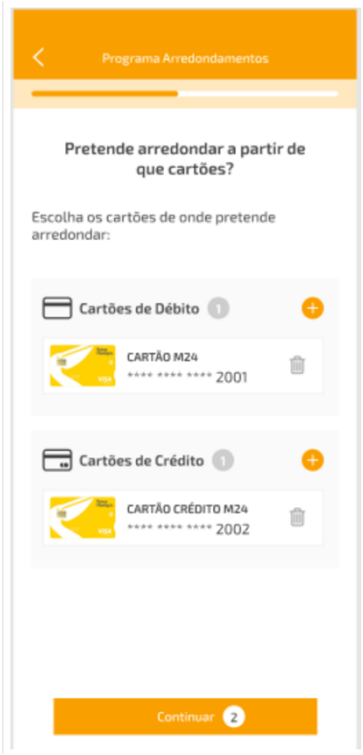


Figura 37 - User Story Remover escolhas de origens

8.2. Tarefas Atribuídas

Desta vez, tive uma tarefa relativa a cada *User Story*, sendo que o nome das *User Stories* já acaba por descrever a tarefa.

- **Remover Escolhas de Destinos**

- **Remover Escolhas de Origens**

8.3. Soluções Implementadas

Uma vez que em termos de lógica as tarefas eram muito parecidas, passo a explicar a minha abordagem, que foi idêntica para ambas as tarefas.

Assim, em ambos os casos foi necessário utilizar um *delegate* que fosse "disparado" quando o botão com a forma de caixote do lixo de cada uma das *Views* customizadas.

Em ambas as *ViewControllers* existiam dois *arrays,* um populado com as poupanças/cartões selecionados e outro com todas as poupanças/cartões existentes.

Assim sendo, a lógica de eliminar não é complexa, quando há a resposta ao *delegate* conseguimos obter a posição do cartão/poupança na lista dos selecionados. Uma vez que temos a posição, é então possível adicionar o conteúdo dessa posição ao *array* que inicialmente continha todas as poupanças/cartões, uma vez que quando algum é selecionado ele é retirado desta lista e adicionado à lista dos selecionados. Pelo que agora apenas foi necessário fazer o processo inverso, adicionar à lista "completa" e fazer a remoção da lista dos selecionados.

Dado isto apenas é necessário limpar os itens apresentados atualmente no ecrã e recarregar os mesmos baseado no novo conteúdo do array.

O seguinte excerto de código mostra toda a explicação anterior.

```
-(void) deleteSavings:(NSInteger)row
{
    [self.allSavings
addObject:self.selectedSavings[row]];
    [self.selectedSavings removeObjectAtIndex:row];
    self.choiceCounter++;
    [self cleanItens];
    [self showButtonsDestiny];
}
```

Excerto de Código 10 - Remover uma poupança

8.4. Resultados

Apesar de o resultado não ser demonstrável neste formato, o mesmo poderá ser comprovado através da execução da aplicação.

No entanto, foi conseguido o objetivo, isto é, eliminar as poupanças/cartões e as mesmas voltaram para a outra lista de forma a voltarem a poder ser selecionadas caso o utilizador assim o entenda.

9. Conclusão

Este estágio deu-me a oportunidade de conhecer novas tecnologias com as quais nunca tinha trabalhado, mas que, no entanto, gostei e as terei em consideração no meu futuro profissional.

De uma forma geral, e apesar da funcionalidade ainda não estar finalizada, nem disponível à data de escrita deste documento, sinto que consegui ser útil no seu desenvolvimento.

Para além da oportunidade de trabalhar com uma nova tecnologia, este estágio fez-me perceber, acima de tudo as dinâmicas de trabalho dentro de uma empresa, bem como das metodologias de trabalho utilizadas, neste caso, o *Scrum*.

Durante a implementação, foi também possível consolidar alguns conceitos anteriormente adquiridos em algumas unidades curriculares da licenciatura, como por exemplo a importância do *Git*, bem como a reutilização de código.

De salientar será que todo o material de código relativo à Academia *iOS* bem como este relatório se encontram disponíveis num repositório no GitHub, sendo o endereço do mesmo https://github.com/Rui-Costa26/Estagio-iOS.

Assim, e para concluir, posso afirmar que este estágio enriqueceu sem dúvida alguma quase todos os conhecimentos que me foram transmitidos durante toda a licenciatura, uma vez que os pude consolidar e aplicar num contexto real, acreditando que esses seriam os principais objetivos do estágio, creio ter sido bem-sucedido.

Bibliografia

https://www.apple.com/swift/

https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

https://www.itsector.pt/about-us

ANEXOS

Anexo A – Teste 1 iOS

Teste 1 iOS

1. Crie um novo projeto Command Line Tool com o nome TesteFizz_SeuNome.

2. Crie uma class com o nome FizzBrain.
3. Adicione à class anterior um método de classe startFizzing que quando invocado imprima na consola os números de 1 a 100, inclusive.
   Mas, para múltiplos de três, imprima "Fizz" em vez do número.
   Para os múltiplos de cinco, imprima “Buzz”.
   Para números múltiplos de três e cinco, imprima “FizzBuzz”.

   Amostra do resultado pretendido:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
....
```

4. No final crie uma pasta zipada com o projecto com o nome **TesteFizz_SeuNome**.

Anexo B – Teste 2 iOS

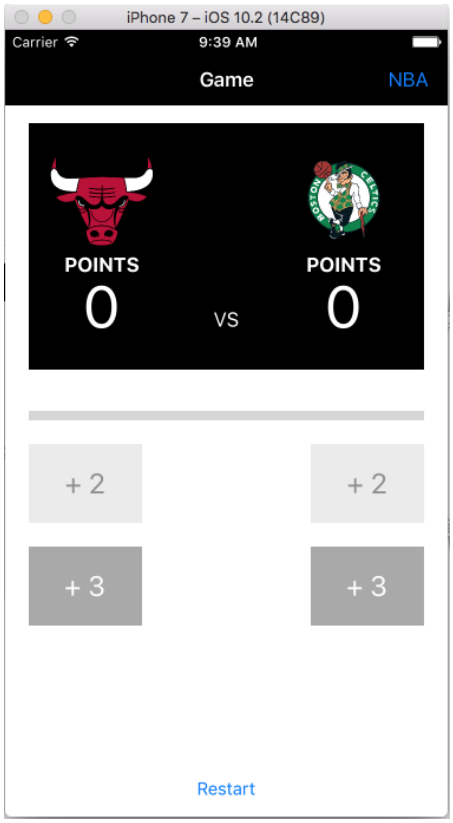
Teste 2 iOS

O objectivo da app é permitir registar os pontos de um jogo de basquetebol.

O resultado final deverá estar o mais próximo possível das imagens que o teste apresenta.

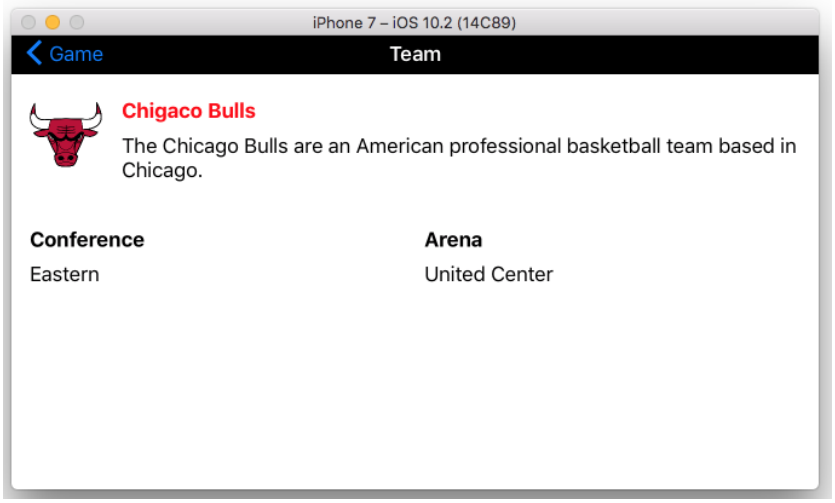
1. Crie uma nova app com o nome TesteUI_SeuNome. A app deverá estar disponível somente para iPhone.

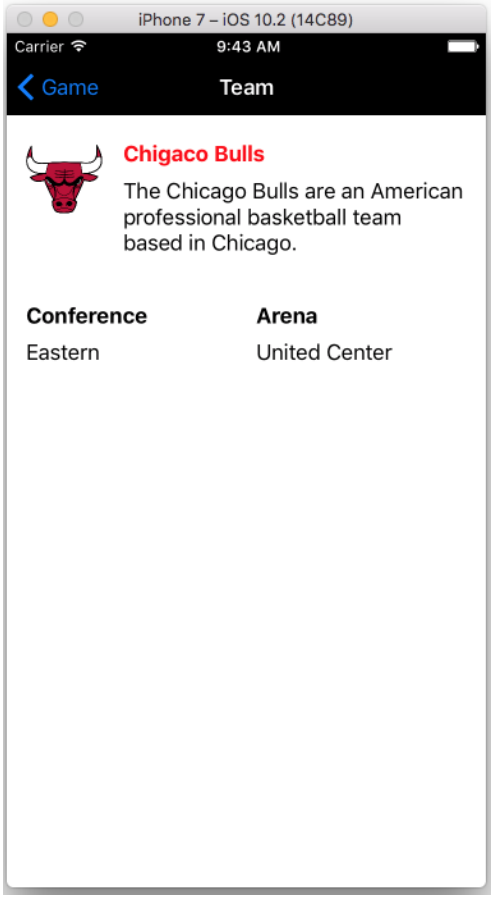
2. O ecrã inicial da app permitirá marcar os pontos para cada uma das equipas. Sempre que for pressionado o botão +2 ou +3 deverá ser incrementado o valor de pontos da equipa respectiva. Sempre que o botão Restart for pressionado, os marcadores de cada uma das equipas deverá ser reiniciado a zero.



3. Quando for pressionado o logotipo de cada uma das equipas, deverá ser apresentado um ecrã de detalhes. O ecrã deverá apresentar o aspecto semelhante às

imagens seguintes (exemplo para a equipa Bulls). Nota: Deverás adicionar constraints por forma ao layout se reajustar em Portrait e Landscape.





Os textos a apresentar para cada uma das equipas devem ser:

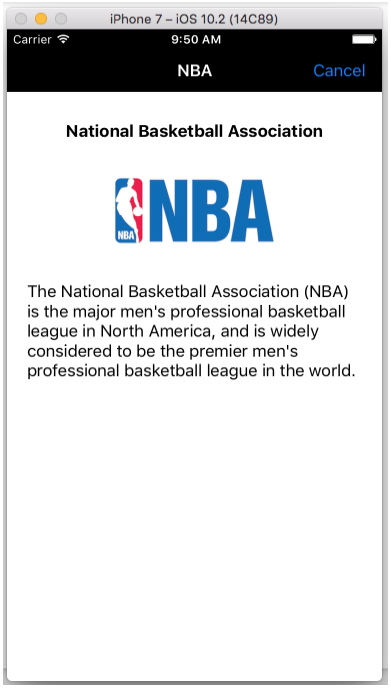
Chigaco Bulls The Chicago Bulls are an American professional basketball team based in Chicago. **Conference** Eastern **Arena** United Center	Boston Celtics The Boston Celtics are an American professional basketball team based in Boston, Massachusetts. **Conference** Eastern **Arena** TD Garden

4. Quando o botão **NBA** do primeiro ecrã for pressionado deverás apresentar o ecrã da figura seguinte. Ao pressionar o botão **Cancel** deverá voltar ao ecrã anterior.

Os textos a apresentar são:

National Basketball Association

The National Basketball Association (NBA) is the major men's professional basketball league in North America, and is widely considered to be the premier men's professional basketball league in the world.



5. Defina a navegação da aplicação como mostra a imagem seguinte:

6. No final crie uma pasta zipada com o projecto com o nome **Teste2_SeuNome**.

Anexo C – iOS Project

iOS Project: "Currency Converter app"

Goal

Build an iPhone app to allow to convert currencies

Objetives

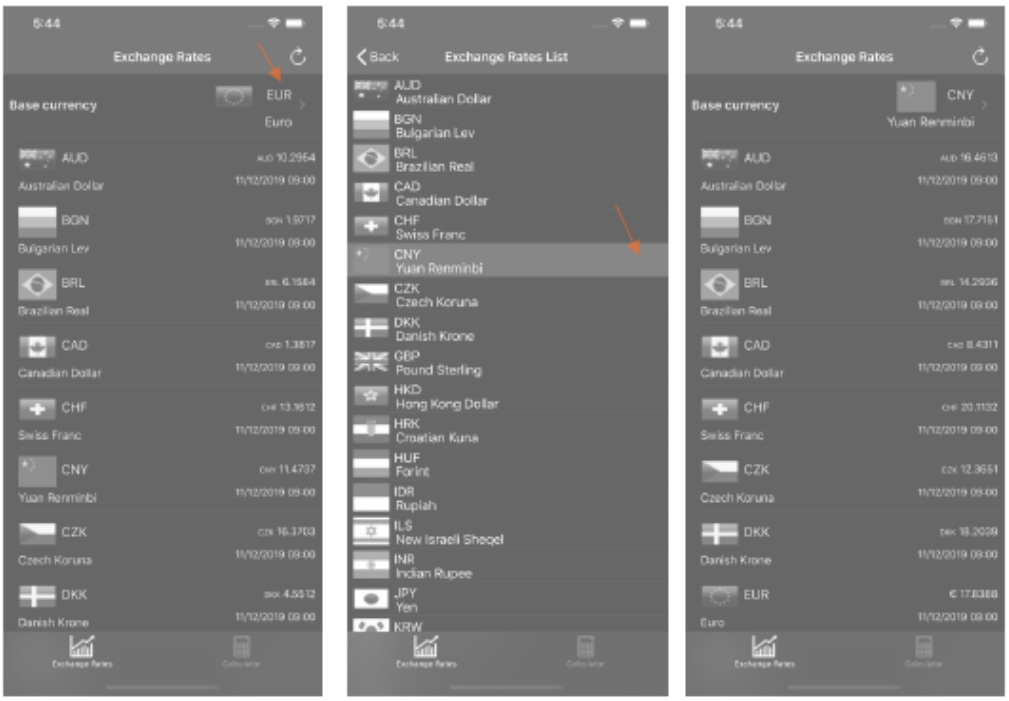
- Install Xcode and create a new iOS project
- Import images into the Xcode project
- Develop of user Interfaces with UIKit
- Edit the storyboard and add UI elements to the view controller
- Create layout constraints
- Handle button taps with actions and outlets
- Install cocoa pods
- Use a network library to get data from webservices
- Create a network layer in objective-c
- Writing code to:
    - Present a list of currencies
    - Get data from a REST API to get Currencies rates
    - Calculate currency conversion

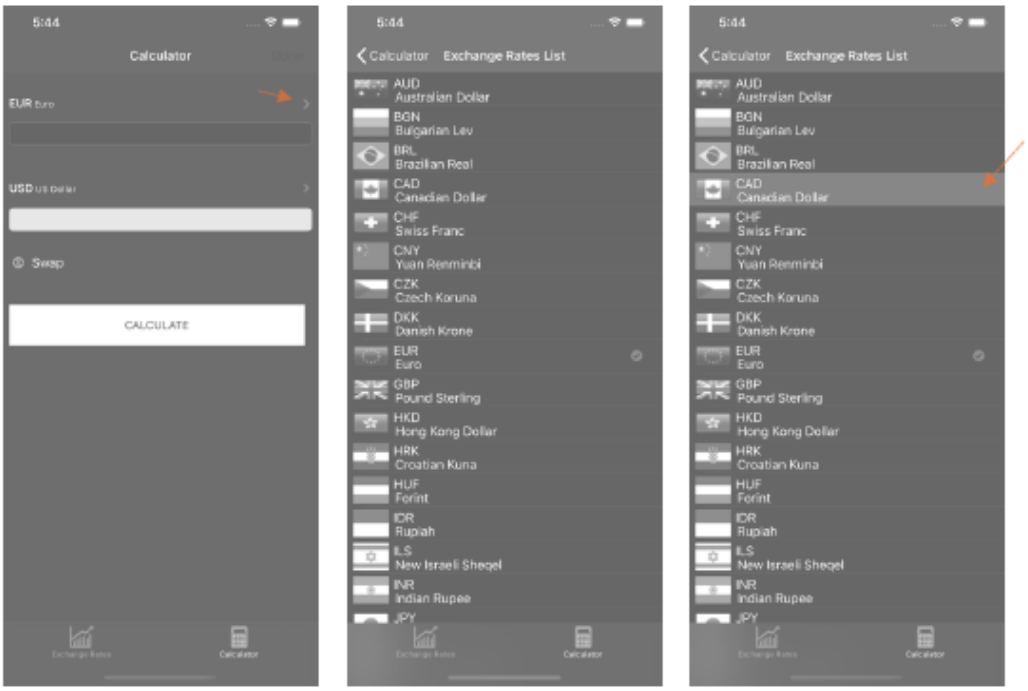
Requirements

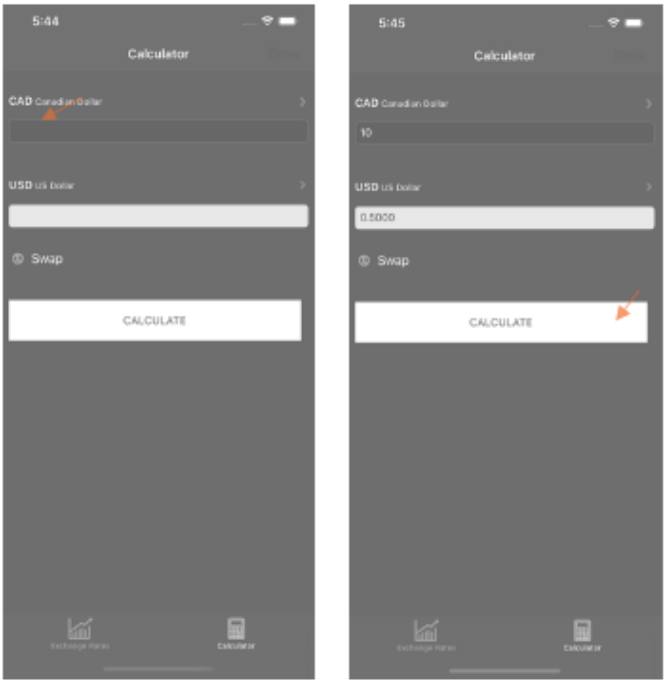
For this project Objective-c, swift and XCode knowledge is required.

To develop iOS apps for the iPhone using latest Xcode version.

App layout







Some Currencies API's where you can retrieve data

https://exchangeratesapi.io/
https://fixer.io/
https://currencylayer.com/