INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

RENAN FRANÇA MASSMANN

UM ESTUDO E APLICAÇÃO DO FRAMEWORK XAMARIN NO DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

Prof. Maikon Cismoski dos Santos

PASSO FUNDO
2018

RENAN FRANÇA MASSMANN

UM ESTUDO E APLICAÇÃO DO FRAMEWORK XAMARIN NO DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Maikon Cismoski dos Santos

PASSO FUNDO

2018

RENAN FRANÇA MASSMANN

UM ESTUDO E APLICAÇÃO DO FRAMEWORK XAMARIN NO DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Maikon Cismoski dos Santos

_____

Josué Toebe

_____

Vanessa Lago Machado

_____

Rafael Marisco Bertei

PASSO FUNDO

2018

RESUMO

A ampla adoção dos Smartphones remete cada vez mais a necessidade de migração de Sistemas Web para o cenário dos dispositivos móveis, gerando oportunidades de estudo e especialização em tecnologias de desenvolvimento para as plataformas móveis. Este projeto teve como objetivo o estudo e aplicação do framework Xamarin no desenvolvimento de uma aplicação móvel multiplataforma, o aplicativo visa facilitar a organização de jogos entre os adeptos do futebol. A aplicação foi desenvolvida para as plataformas Android e IOS e utiliza uma base de dados alocada remotamente na plataforma de serviços em nuvem da Microsoft. Com a realização deste estudo conclui-se que o Xamarin.Forms facilita o desenvolvimento para múltiplos sistemas móveis, pois permite que ocorra o compartilhamento do código base entre os projetos.

Palavras-chave: Xamarin. Xamarin.Forms. Plataformas móveis. Aplicativo, Azure aplicativos móveis

ABSTRACT

The wide adoption of smartphones brings more and more the need to migrate Web Systems to the mobile device landscape. This project aimed at the study and application of the framework Xamarin in the development of a multiplatform mobile application, the application aims to facilitate the organization of games between football fans. The application is designed for Android and IOS platforms and uses a remotely allocated database on Microsoft's cloud services platform. With the accomplishment of this study it is concluded that Xamarin.Forms facilitates the development for multiple mobile systems since it allows the sharing of the base code between the projects.

Keywords: Xamarn. Xamarin.Forms. Mobile Platforms. App. Azure Mobile Apps.

LISTA DE ABREVIATURAS E SIGLAS

IFSUL - Instituto Federal Sul-rio-grandense
TCC - Trabalho de Conclusão de Curso
SO - Sistema Operacional
API - *Application Programming Interface* (Interface de programação de aplicativos)
C# - C-Sharp
PC - *Personal Computer* (Computador Pessoal)
GUI - *Graphical User Interface* (Interface Gráfica de Usuário)
UI - *User Interface* (Interface de Usuário)
CLR - *Common Language runtime* (Ambiente de Execução Independente de Linguagem)
FCL - *Framework Class Library (*Conjunto de Bibliotecas Unificadas)
DLL - *Dynamic-link Library* (Biblioteca de Vínculo Dinâmico)
IaaS - *Infrastructure as a Service* (Infraestrutura como um serviço)
PaaS - *Platform as a Service* (Plataforma como um Serviço)
SDK - *Software development kit* (kit de Desenvolvimento de Software)
SGBD - Sistema de gerenciamento de banco de dados
REST - *Representational state transfer* (Transferência de Estado Representacional)
URI - *Uniform Resource Identifier* (Identificador Padrão de Recursos)
URL - *Uniform Resource Locator* (Localizador Padrão de Recursos)
HTTP - *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
CSS - *Cascading Style Sheets* (Folha de Estilo em Cascatas)

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A introdução dos dispositivos móveis ao longo dos últimos anos gerou uma nova necessidade na área de desenvolvimento de software: o desenvolvimento para esses dispositivos. Atualmente quase todas as pessoas possuem smartphone e o utilizam para tarefas como acesso às redes sociais, correio eletrônico, compras online, informações, notícias e consumo multimídia, ou seja, tudo o que era feito com os computadores agora pode ser feito utilizando os smartphones, tornando interessante a migração de sistemas Web para o ambiente móvel.

O mercado de dispositivos móveis é dominado por duas plataformas principais: Android e IOS. Juntos esses Sistemas Operacionais representam aproximadamente 98% do mercado a nível global (NETMARKETSHARE, 2018). No Brasil, Android domina as vendas de novos smartphones com 93% contra 5,8% do IOS (KANTARWORLDPANEL, 2018).

Para que um aplicativo seja disponibilizado para essas duas plataformas é necessário a execução de projetos distintos, uma vez que o ambiente de desenvolvimento do Android é completamente diferente do IOS. Desenvolver diferentes versões para uma mesma aplicação pode demandar um número maior de profissionais ou diferentes equipes de desenvolvimento e maior tempo despendido nos projetos, aumentando o custo e muitas vezes inviabilizando a implementação.

A fim de contornar esse problema, tecnologias foram desenvolvidas como é o caso dos frameworks de desenvolvimento multiplataforma. Dentre eles destacam- se o PhoneGap e o Ionic para aplicativos híbridos e o Xamarin para aplicativos nativos. Esses frameworks permitem que apenas uma versão da aplicação seja desenvolvida e então o framework se encarrega de compilar o projeto e gerar a aplicação para as diferentes plataformas. Essa abordagem traz vantagens e desvantagens apresentadas ao longo deste trabalho.

1.1 JUSTIFICATIVA

A dificuldade em desenvolver aplicativos móveis para múltiplas plataformas torna interessante o estudo de opções ao modo nativo de desenvolvimento. Frameworks, entre eles o Xamarin, contornam o maior problema ao desenvolver aplicativos para diferentes Sistemas Operacionais, que é a necessidade de

desenvolvimento de projetos distintos para o mesmo aplicativo, facilitando e viabilizando o seu desenvolvimento.

1.2 OBJETIVOS

O presente trabalho tem por objetivo estudar o framework Xamarin e aplicá-lo no desenvolvimento de um aplicativo para as plataformas Android e IOS, demonstrando que a utilização de um framework pode facilitar e viabilizar o desenvolvimento de aplicativos multiplataforma. Dentre os objetivos específicos estão: Realizar um estudo do *framework* Xamarin.Forms; estudar a utilização da Plataforma Azure como *back-end* de aplicativos móveis; projetar o aplicativo a ser desenvolvido; desenvolver o aplicativo para Android e IOS utilizando Xamarin.Forms, avaliar o *framework* e disponibilizar o aplicativo para realização de testes e avaliação.

1.3 ESTRUTURA DA MONOGRAFIA

Visando a melhor organização e estruturação, esta monografia foi dividida nos seguintes capítulos: Capítulo 2 (Referencial Teórico): apresenta as tecnologias e os fundamentos necessários para a construção do conhecimento aplicado no desenvolvimento do aplicativo proposto. Capítulo 3 (Metodologia): apresenta o estudo de caso para o desenvolvimento do aplicativo. Capitulo 4 (Desenvolvimento): Apresenta como foi realizado o desenvolvimento do aplicativo. Capítulo 5 (Resultados): descreve e ilustra os resultados obtidos a partir da metodologia e do desenvolvimento, avaliação do *framework* e do aplicativo. Capítulo 6 (Conclusão): Apresenta uma reflexão sobre o projeto desenvolvido, sobre o framework e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capitulo serão descritos os principais conceitos das tecnologias utilizadas no desenvolvimento de aplicativos móveis multiplataforma. Serão abordados assuntos referente ao uso atual dos dispositivos móveis, principais plataformas utilizadas e os desafios do desenvolvimento de aplicativos móveis para múltiplos Sistemas Operacionais; tecnologias da Microsoft utilizadas no desenvolvimento com Xamarin: framework .NET, linguagem de programação C#, sistema de gerenciamento de dados Microsoft SQL Server; RESTful API *Web Services*, frameworks para desenvolvimento multiplataforma; a plataforma Azure e seus serviços em nuvem[1] voltados ao desenvolvimento móvel.

2.1 DISPOSITIVOS MÓVEIS

A Indústria de Computadores Pessoais tem testemunhado uma grande mudança ao longo dos últimos anos com a introdução dos dispositivos móveis. Os tradicionais PCs (Computador Pessoal) e Notebooks ainda existem e continuarão a existir por muito tempo pois são necessários em tarefas que exigem a utilização de um teclado físico e tela grande, como por exemplo: Edição de textos e planilhas, conteúdo multimídia e programação. Porém, grande parte do conteúdo antes presente somente nos computadores, está agora disponível nos dispositivos com telas pequenas, especialmente para acesso às redes sociais, correio eletrônico, informações, notícias e consumo multimídia (PETZOLD, 2016).

De acordo com pesquisa realizada pela Fundação Getúlio Vargas (CAPELAS, 2017), até o fim de 2017, Brasil teria um smartphone por habitante; De acordo com CISCO (2018) estima-se que 563 milhões de dispositivos móveis foram acrescentados em 2016, somados aos 7.3 bilhões de dispositivos que já existiam, a nível global. Dados estes que nos dão uma noção de como se encontra o cenário atual em relação ao uso de smartphones no brasil e no mundo, e que destaca a relevância do mercado de desenvolvimento de aplicativos móveis.

[1] Refere-se à utilização da memória e da capacidade de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet.

Atualmente há duas plataformas móveis dominantes (NETMARKETSHARE, 2018):

- A plataforma Android, presente em 69.91% dos *smartphones/tablets* vendidos em maio de 2018;
- A plataforma IOS, presente nos dispositivos da Apple e representa 28,48% na pesquisa citada anteriormente.

No Brasil, Android possui 93% do *market share*[2] de março a julho de 2018, enquanto IOS possui 4,7% e Windows Phone 2,4% (KANTARWORLDPANEL, 2018).

2.2 PLATAFORMAS

Nesta seção serão apresentadas as plataformas móveis Android e IOS. Segundo KANTARWORLDPANEL (2018), estes Sistemas Operacionais móveis representam a maior participação no mercado de smartphones.

2.2.1 Android

O Android é um Sistema Operacional Móvel pertencente ao Google. Seu sistema “é baseado no kernel do Linux, que é responsável por gerenciar a memória, os processos, threads, segurança dos arquivos e pastas, além de redes e drivers” (LECHETA, 2015, p.27, tradução nossa).

A segurança do Android é baseada na segurança do Linux. Onde cada aplicação é executada em um único processo e cada processo contém uma thread dedicada. Para cada aplicação instalada no celular é criado um usuário no sistema operacional para ter acesso a sua estrutura de diretórios. Dessa forma, nenhum outro usuário pode ter acesso a essa aplicação (LECHETA, 2015).

O Android é uma plataforma para aplicações móveis completamente livre e de código aberto, permitindo que diversas empresas e desenvolvedores do mundo todo possam contribuir constantemente para sua melhora e evolução através de correções de falhas ou adicionando novas funcionalidades (LECHETA, 2015).

---

[2] Grau de participação de uma empresa no mercado em termos das vendas de um determinado produto; fração do mercado controlada por ela.

Por ter uma licença flexível, o Android permite que cada fabricante possa realizar alterações em seu código-fonte para customizar seus produtos, sem necessidade de compartilhar essas alterações com as outras empresas ou pagar por isso. Essas modificações tornam possível a utilização do Android em inúmeros dispositivos, como relógios, TVs, *Tablets*, painéis de automóveis, entre outros (LECHETA, 2015).

Atualmente, o Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android, que utiliza Java como linguagem de programação.

2.2.2 IOS

O IOS é o Sistema Operacional desenvolvido pela Apple para ser utilizado inicialmente no iPhone, mas logo se tornou o SO voltado a todos os dispositivos da empresa. Sua execução é restrita ao hardware construído pela Apple, tornando o IOS um SO exclusivo dos dispositivos da Apple (MILANI, 2012).

A primeira versão do IOS foi lançada junto ao primeiro iPhone em 2007, durante a WWDC (*Apple World Wide Developer Conference*), evento para desenvolvedores realizado pela Apple. No ano de 2008 foi lançado o kit para desenvolvimento de aplicativos (SDK) e a *App Store*, loja virtual que distribui as aplicações IOS pelo mundo (LECHETA, 2012).

O desenvolvimento de aplicativos para o IOS utiliza a linguagem de programação Objective-C. Foi criada no início dos anos 80 e é baseada na linguagem C padrão. Caracteriza-se por ser uma linguagem dinâmica, que possibilita a reutilização do código, a transmissão de mensagens entre determinadas estruturas e a criação de classes em tempo de execução (LECHETA, 2012).

O desenvolvimento de aplicativos para IOS requer a utilização de uma das três licenças de desenvolvimento da Apple. A primeira, de uso não comercial, é a mais simples, sendo a única gratuita, e habilita o programador a utilizar o Xcode[3], desenvolver as aplicações e testá-las somente no simulador do IOS, presente no Xcode, não sendo permitida a publicação dos aplicativos na *App Store*; A segunda

[3] O Xcode é uma IDE desenvolvida pela Apple que permite o desenvolvimento de projetos para seus dispositivos móveis.

licença é a Standard, que permite a instalação dos aplicativos nos dispositivos físicos e publicação na App Store; e a terceira licença é a Enterprise, que além dos benefícios da licença Standard, habilita mais de um desenvolvedor a utilizá-la (MILANI, 2012).

2.3 DESENVOLVIMENTO MULTIPLATAFORMA

A principal dificuldade em desenvolver aplicativos móveis tendo como alvo múltiplas plataformas é a diferença no modo de desenvolvimento, onde cada plataforma possui suas próprias características, impedindo que programadores especialistas em determinada plataforma utilizem seu conhecimento em outra. São apontados por Petzold (2016) quatro grandes obstáculos ao desenvolver para diferentes Sistemas:

1. *Diferentes paradigmas de interface gráfica* - Todas as plataformas apresentam particularidades quanto à interface gráfica e interação com o dispositivo através do multitoque. Cada uma possui sua própria maneira para alternar entre aplicativos e páginas, diferentes formas de abrir e mostrar menus, e ainda, diferentes maneiras de usar o toque na tela para diferentes ações. Os usuários se acostumam com o modo de interação com os aplicativos em uma plataforma específica e esperam que todos os aplicativos se comportem de maneira similar, assim, cada plataforma está associada à sua própria cultura e essas características culturais devem ser preservadas no desenvolvimento dos aplicativos de cada Sistema Operacional

2. *Diferentes ambientes de desenvolvimento* - Atualmente, os programadores estão acostumados a utilizar sofisticados softwares que facilitam a tarefa de desenvolver softwares, denominados Ambientes de Desenvolvimento Integrado (IDE). Cada Plataforma possui sua própria IDE: Xcode no Mac para desenvolvimento IOS, Android Studio em diversos SOs para desenvolvimento Android e Visual Studio no PC para desenvolvimento Windows.

3. *Diferentes Interface de programação* - As plataformas são baseadas em diferentes Sistemas Operacionais que possuem suas próprias Interface de programação de aplicativos (API). Assim, o mesmo objeto da Interface pode ter o nome diferente em cada plataforma. Por exemplo, um objeto que chamado *UISwitch* no IOS, no Android é chamado *Switch* e no Windows *ToggleSwitch*.

4. *Diferentes linguagens de Programação* - Apesar dos desenvolvedores terem certa flexibilidade na escolha da linguagem de programação para cada plataforma, em geral, cada plataforma é associada à uma linguagem de programação em particular: Objective-C para iPhone e iPad, Java para dispositivos Android e C# para Windows

Diante dessas diferenças, desenvolver aplicativos tanto para Android como para IOS geralmente exige equipes de desenvolvimento diferentes para cada projeto, cada uma especialista em uma linguagem de programação e API específica de cada plataforma. Com o objetivo de facilitar o desenvolvimento para aplicações móveis, novas alternativas foram estudadas para a criação de soluções que contornassem o problema da diferença da linguagem de programação. A utilização de uma linguagem de programação comum a todas as plataformas móveis, onde o código base poderia ser compartilhado entre os projetos foi a solução apresentada pelo Xamarin, utilizando uma linguagem de programação comum a todos os projetos, sendo C# juntamente com o framework .NET.

2.4 .NET

O .NET é um *framework* desenvolvido pela Microsoft. É uma plataforma para desenvolvimento e execução de sistemas e aplicações. Do ponto de vista dos programadores, o framework .NET é o sistema operacional. É através dele que são invocadas todas as funções necessárias ao funcionamento dos programas, em qualquer Sistema Operacional.

Com ideia semelhante à plataforma Java, o programador deixa de escrever código para um sistema ou dispositivo específico, e passa a escrever para a plataforma .NET. Aplicações escritas para ele funcionam em um ambiente de software controlado, em oposição a um ambiente de hardware, através de uma máquina virtual de aplicação (MICROSOFT, 2018a).

O .NET Framework consiste de dois componentes principais, ou seja, ela é executada sobre uma *Common Language Runtime (*CLR, traduzida como Ambiente de Execução Independente de Linguagem) interagindo com um *Framework Class Library* (FCL, traduzido como Conjunto de Bibliotecas Unificadas). Ele permite

executar diversas linguagens permitindo grande interoperabilidade[4] entre elas (MICROSOFT, 2017a).

O CLR fornece gerenciamento de memória, controle de exceção, interoperabilidade, manipulação de processamento paralelo e concorrente, reflexão, segurança, serviços de compilação para a arquitetura específica, entre outros. Já a FCL oferece APIs para Interface de Usuário, acesso a dados, conectividade com banco de dados, redes, web, criptografia, acesso aos serviços do sistema operacional, estruturas de dados e algoritmos diversos, facilidades para a linguagem e muito mais (MICROSOFT, 2018a).

Originalmente o .NET só funcionava no Windows, porém com o lançamento do .NEt Core em junho de 2016 o framework passou a funcionar também no Linux e no MacOS. Originalmente era proprietário, mas seus códigos fontes foram liberados. O .NET Core já nasceu como um projeto 100% código aberto, contando com contribuições da comunidade através da Fundação .NET (MICROSOFT, 2018a).

2.5 C# (C-Sharp)

É uma linguagem de programação desenvolvida pela Microsoft. Foi lançada em 2000, baseada em C++ e influenciada por outras linguagens como Java e Pascal, caracteriza-se por ser uma linguagem orientada a objetos, fortemente tipada[5] e imperativa[6]. Desde sua primeira versão, C# apresentou suporte a propriedades e eventos, que se revelaram tipos de membros que são particularmente adequados para a programação de *Graphical User Interfaces* (GUI, traduzido como Interface Gráficas de Usuário). Ao logo dos anos, o C# continuou sendo melhorado, o suporte à *Generics*, funções *Lambda* e operações assíncronas transformaram C# em uma linguagem de programação classificada como multiparadigma, tradicionalmente imperativa, pode ainda ser declarativa ou funcional (PETZOLD, 2016).

---

[4] Capacidade de um sistema de se comunicar de forma transparente com outro sistema.

[5] Linguagem tipada, ou linguagem tipificada, é uma linguagem de programação que usa variáveis com tipos específicos.

[6] Programação Imperativa um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa.

Desde a sua criação, C# tem sido estreitamente associado ao Microsoft .NET Framework. No nível mais baixo, o .NET fornece uma infraestrutura para os tipos de dados básicos do C# (*int, double, string* e assim por diante). Mas a extensa biblioteca de classes do framework .NET oferece suporte para muitas tarefas comuns encontradas em muitos tipos diferentes de programação. Que incluem: Operações Matemáticas, Depuração do código, Reflexões, Coleções, Globalização, Manipulação de arquivos, Rede, Segurança, *Threading*[7], Serviços Web, Manipulação de dados, XML e leitura e escrita JSON (PETZOLD, 2016).

A sintaxe da linguagem C# é semelhante à de outras linguagens baseadas no C, como C++ e Java (MICROSOFT, 2018b). Em particular:

- O ponto e vírgula (;) é usada para indicar o final de uma declaração.
- As chaves são usadas para declarações de grupo. As declarações são geralmente agrupadas em métodos (funções), métodos em classes e classes em *namespaces*.
- As variáveis são atribuídas usando um sinal de igual, mas comparadas usando dois sinais de iguais consecutivos.
- Os colchetes [] são usados com vetores, ambos para declará-los e obter um valor em um determinado índice em um deles.

2.6 MICROSOFT SQL SERVER

O Microsoft SQL SERVER é um Sistema de Gerenciamento de Banco e Dados (SGBD) para banco de dados relacionais desenvolvido pela Microsoft. Sua função é armazenar e recuperar dados solicitados por outras aplicações localmente ou remotamente através da rede. Há diferentes versões do SQL Server disponíveis ao público, que suportam diferentes cargas de trabalho, desde soluções básicas como por exemplo um banco de dados local, como soluções complexas para grandes corporações que possuem aplicações com milhares de usuários que acessam suas bases de dados ao mesmo tempo utilizando a internet (MICROSOFT, 2018c).

Algumas funções do MS-SQL Server são: Gatilhos, Funções*, Stored Procedures* e *Extended Stored Procedures*. Sua atual versão é o Microsoft SQL Server

[7] É forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas Concorrencialmente.

2017, sendo disponibilizado em diferentes edições com diferentes recursos para diferentes necessidades dos usuários, sendo as edições *Interprise* e *Standard* pagas e as edições *Developer e Express* gratuitas. Ainda, a Microsoft disponibiliza uma versão baseada em nuvem[8] do Microsoft SQL Server, apresentada como *Platform as a Service* (PaaS, traduzido como Plataforma como uma Oferta de Serviços) no Microsoft Azure (MICROSOFT, 2018c).

2.6.1 Armazenamento de Dados

Um banco de dados é uma coleção de tabelas com colunas digitadas. O SQL Server suporta diferentes tipos de dados, incluindo tipos principais, como *Integer*, *Float, Decimal, Char* (incluindo cadeias de caracteres), *Varchar* (cadeias de caracteres de comprimento variável), binário (para *blobs* de dados não estruturados), Texto (para dados de texto) entre outros (MICROSOFT, 20187c).

2.6.2 Recuperação de Dados

O modo principal de recuperação de dados de uma base de dados do SQL Server é através de consultas. A consulta é expressa usando uma variante de SQL chamada T-SQL. A consulta especifica declarativamente o que deve ser recuperado. É processado pelo processador de consulta, que calcula a sequência de etapas que serão necessárias para recuperar os dados solicitados. A sequência de ações necessárias para executar uma consulta é chamada de plano de consulta. O SQL Server escolhe o plano que deverá produzir os resultados no menor tempo possível. Isso é chamado de otimização de consulta e é executado pelo próprio processador de consulta (DELANEY, 2006).

2.7 RESTful API Web Services

[8] Refere-se à utilização da memória e da capacidade de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet.

Uma *Application Programming Interface (*API, traduzido como Interface de Programação de Aplicativos) é um conjunto de rotinas com função de realizar a comunicação entre dois softwares. A *Representational State Transfer* (REST, em português Transferência de Estado Representacional) é um estilo arquitetônico baseado em como a Web funciona, por exemplo: o cliente faz requisições ao servidor e recebe uma resposta usando o protocolo HTTP[9].

O termo REST foi introduzido pela primeira vez por Roy Thomas Fielding no Capítulo 5 de sua tese de doutorado. A tese é uma explicação retrospectiva da arquitetura escolhido para desenvolver a Web. Em sua tese, Fielding examina as partes da *World Wide Web* que funcionam muito bem e extrai princípios de design que poderiam fazer qualquer outro sistema de hipermídia distribuído, seja relacionado à Web ou não, tão eficiente quanto. A motivação original para o desenvolvimento de REST foi criar um modelo arquitetônico da Web. Fielding também é um dos autores da especificação HTTP, por isso não é surpreendente que o HTTP se encaixe tão bem no design da arquitetura que ele descreveu em sua dissertação (GONÇALVEZ, 2013).

As arquiteturas RESTful se tornaram populares rapidamente devido a utilização do HTTP, que é um protocolo de transporte muito robusto. Os serviços web RESTful reduzem o acoplamento cliente/servidor, tornando muito mais fácil a evolução de uma interface REST ao longo do tempo sem que clientes existentes parem de funcionar. Os serviços Web RESTful são *stateless*[10] e podem fazer uso de cache HTTP e servidores proxy para ajudá-lo a lidar com alta carga e escala muito melhor (GONÇALVEZ, 2013).

Para modelar um Serviço Web RESTful é preciso conhecer o protocolo HTTP e URIs[11], que são praticamente links da internet. Cada URL única é a representação de um objeto, que pode ser manipulado usando HTTP GET, DELETE, POST ou PUT

---

[9] Protocolo de Transferência de Hipertexto. É um protocolo de comunicação (na camada de aplicação segundo o Modelo OSI) utilizado para sistemas de informação de hipermídia, distribuídos e colaborativos. Ele é a base para a comunicação de dados da Internet

[10] O protocolo de comunicação considera cada requisição como uma transação independente que não está relacionada a qualquer requisição anterior, de forma que a comunicação consista de pares de *requisição e resposta* independentes.

[11] É uma cadeia de caracteres compacta usada para identificar ou denominar um recurso na Internet.

para, respectivamente, recuperar, excluir, criar ou atualizar se conteúdo (GONÇALVEZ, 2013).

Os recursos são o centro da arquitetura REST. Esses recursos são o conteúdo com o qual o cliente deseja interagir ou referenciar, ou alguma informação relevante para ser referenciada por um hyperlink (um produto, um resultado de uma busca, uma tabela com conteúdo, etc.). Os recursos podem estar armazenados em um banco de dados, em um arquivo ou algum outro lugar em que possam ser endereçados por uma URI (GONCALVES, 2013).

Ao acessar um recurso, o cliente nunca interage de maneira direta com este recurso. Ele, na verdade, interage com uma representação deste recurso que permanece no servidor. Esta representação pode estar em formato de texto, JSON, XML, PDF, JPG ou algum outro. Um mesmo recurso pode possuir mais de um tipo de representação (GONCALVES, 2013).

O protocolo HTTP funciona baseado nas requisições e respostas trocadas entre um cliente e um servidor. Dessa forma, na arquitetura REST, quando um cliente deseja acessar um recurso do servidor, ele envia uma requisição HTTP para o servidor por meio de uma URI, que por sua vez responde enviando uma representação do recurso acessado (GONCALVES, 2013).

2.8 FRAMEWORKS

Nesta seção são apresentados os principais frameworks para desenvolvimento móvel multiplataforma utilizados atualmente, com maior ênfase no Xamarin que é o objeto de estudo deste trabalho. Ao fim, no tópico "Discussão", será feito uma reflexão sobre os frameworks apresentados.

2.8.1 PhoneGap

O Apache Cordova é um *framework* de desenvolvimento móvel de código aberto. Ele permite a utilização de padrões de tecnologias web, como HTML5, CSS3 e JavaScript para desenvolvimento multiplataforma, evitando o desenvolvimento nativo de cada plataforma móvel (PHONEGAP, 2017).

O PhoneGap foi originalmente criado pela Empresa Nitobi Software, que veio a ser adquirida pela Adobe em 2011, mudou seu nome para PhoneGap, e mais tarde,

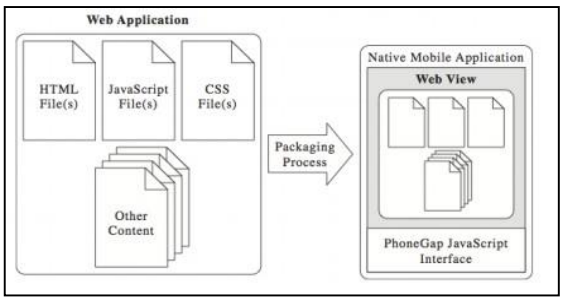
lançou uma versão de código aberto chamado Apache Cordova que foi doado para a Apache Software Foundation[12]. O código base por trás do PhoneGap e do Cordova é o mesmo, a diferença é que o PhoneGap faz uso de serviços criadas pela Adobe que permite que os aplicativos sejam testados em dispositivos móveis, entre outros.

Os aplicativos desenvolvidos usando o PhoneGap são chamados aplicativos híbridos, pois utilizam HTML/JavaScript juntamente com componentes nativos de cada plataforma. Partes do aplicativo, principalmente a Interface do Usuário, a lógica da aplicação e a comunicação com um servidor, utiliza HTML/JavaScript. A parte que se comunica e controla o dispositivo utiliza a linguagem nativa da plataforma. O PhoneGap fornece uma ponte de ligação entre o JavaScript e a linguagem nativa da plataforma, permitindo que a API do JavaScript tenha acesso as funcionalidades específicas do dispositivo, como câmera, GPS, informações do dispositivo, entre outras (GHATOL e PATEL, 2012).

Como o PhoneGap é um framework que não possui nenhuma IDE de desenvolvimento específica, fica a cargo do desenvolvedor escolher a IDE que suporte o desenvolvimento utilizando o PhoneGap. Atualmente, o PhoneGap suporta as seguintes plataformas: Android, Tizen, Blackberry10, IOS, Ubuntu, Windows Phone 8, Windows (8.1, 10, Phone 8.1) e macOS X.

O desenvolvimento de aplicações com o framework ocorre da seguinte maneira: O desenvolvedor cria a aplicação para dispositivos móveis utilizando HTML, CSS e JavaScript, então o PhoneGap, através de um conjunto de ferramentas, encapsula esta aplicação dentro de uma aplicação nativa da plataforma móvel específica (WARGO, 2012). Este processo é ilustrado na Figura 1.

Figura 1 – Encapsulamento de uma aplicação PhoneGap



Fonte: WARGO, 2012.

[12] É uma organização sem fins lucrativos criada para suportar os projetos de código aberto

A renderização do conteúdo web dentro de uma tela do dispositivo é feito pelo componente chamado *Web View*. Esta *Web View* ocupará todo o espaço da tela do dispositivo e quando a aplicação for iniciada, a página inicial da aplicação web será carregada para dentro desta *Web View*, permitindo a interação do usuário com a aplicação web previamente criada. Isso possibilita que o usuário, através de links ou códigos JavaScript, tenha acesso a outros conteúdos que foram encapsulados nesta aplicação ou consiga acesso à internet e possa utilizar os serviços web, por exemplo. (WARGO, 2012).

A aplicação gerada após o encapsulamento funciona exatamente como qualquer outra aplicação web. Seu layout é construído baseado em fontes, cores, espaçamentos e outras propriedades adicionadas ao HTML com o uso do CSS. Ela pode acessar outras páginas HTML sejam elas locais ou de um servidor web. A construção da lógica da aplicação é realizada com a linguagem JavaScript, que implementa dinamismo no conteúdo das páginas HTML, habilita o envio ou recebimento de conteúdo de servidores remotos, entre outras funcionalidades (WARGO, 2012).

Como aplicações web não tem capacidade de acesso aos componentes nativos dos dispositivos e hoje em dia a maioria dos aplicativos precisam desse acesso, O PhoneGap proporciona que a aplicação web contida dentro da aplicação PhoneGap tenha acesso a estes componentes nativos que não fazem parte do contexto web. Isso ocorre devido ao conjunto de APIs e bibliotecas JavaScript do PhoneGap encarregados desta tarefa (WARGO, 2012).

Os aplicativos criados com o PhoneGap mantém a mesma aparência entre as plataformas, ou seja, os elementos da interface de usuário não terão aparência nativa, mas sim a aparência criada com uso de HTML e CSS. Este problema pode ser contornado com uso de frameworks para criação da interface de usuário e que se encarregam de manter a aparência nativa nos aplicativos criados com o PhoneGap.

2.8.2 Ionic

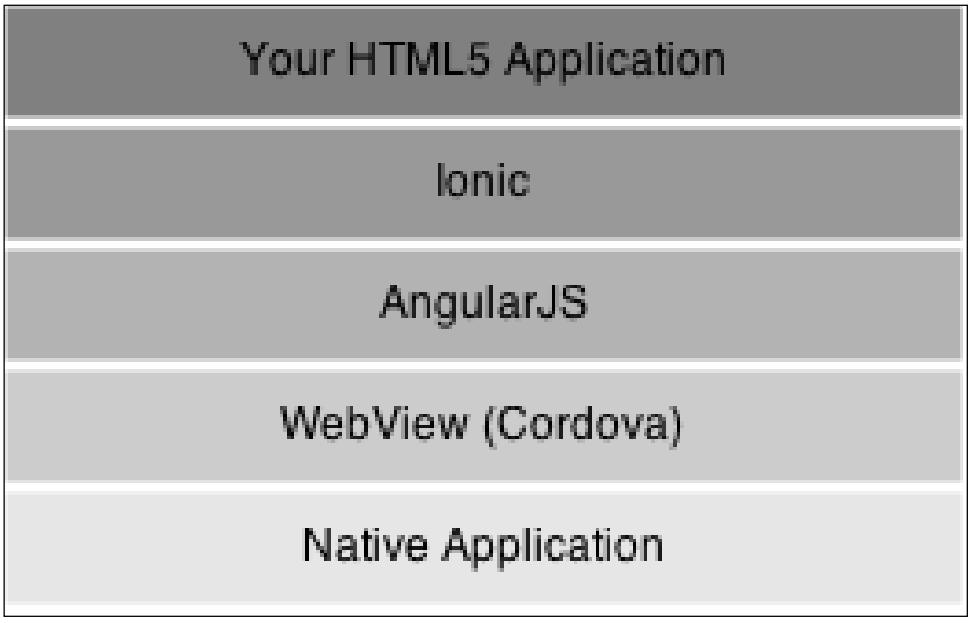
Ionic é um framework de *User Interface* (UI, traduzido como Interface de Usuário) de aplicativos móveis. Ionic é responsável pela aparência e interações da interface de usuário em aplicativos híbridos desenvolvidos utilizando o PhoneGap (IONIC, 2017). Devido à utilização do PhoneGap pelo Ionic, tudo o que foi escrito

anteriormente sobre o desenvolvimento de aplicativos híbridos utilizando PhoneGap se aplica para os aplicativos utilizando Ionic, respeitadas as diferenças na parte gráfica, que é onde o Ionic atua.

Ionic fornece elementos e layouts nativos para a Interface de Usuário de aplicativos móveis iguais aos fornecidos pelos SDKs nativos do IOS e Android, mas que não estão acessíveis pelas aplicações web. Dessa forma, Ionic juntamente com o Cordova permite que aplicativos híbridos mantenham a aparência e comportamento semelhante aos aplicativos desenvolvidos nativamente (IONIC, 2017).

Na Figura 2 é ilustrado as camadas da arquitetura de uma aplicação utilizando Ionic. No topo está a aplicação desenvolvida utilizando HTML5, seguido pelas camadas do framework Ionic que implementa os elementos da Interface Gráfica, estes são manipulados pelo AngularJS[13]; O Cordova, então, encapsula a aplicação em uma *Web View* que é inserida na aplicação nativa de cada plataforma gerada pelo framework (HAVLENA, 2015).

Figura 2 – Camadas da Arquitetura do Ionic



Fonte: HAVLENA, 2015.

[13] É um framework JavaScript de código aberto, mantido pelo Google, que auxilia na execução de *single-page applications*.

2.8.3 Xamarin

Xamarin é uma plataforma de desenvolvimento de aplicativos móveis multiplataforma que permite que aplicativos sejam criados utilizando uma única linguagem de programação (C#) no código base, permitindo o compartilhamento de código entre os projetos das diferentes plataformas, e assim, amenizar a necessidade de desenvolver códigos separados para cada sistema móvel existente no mercado.

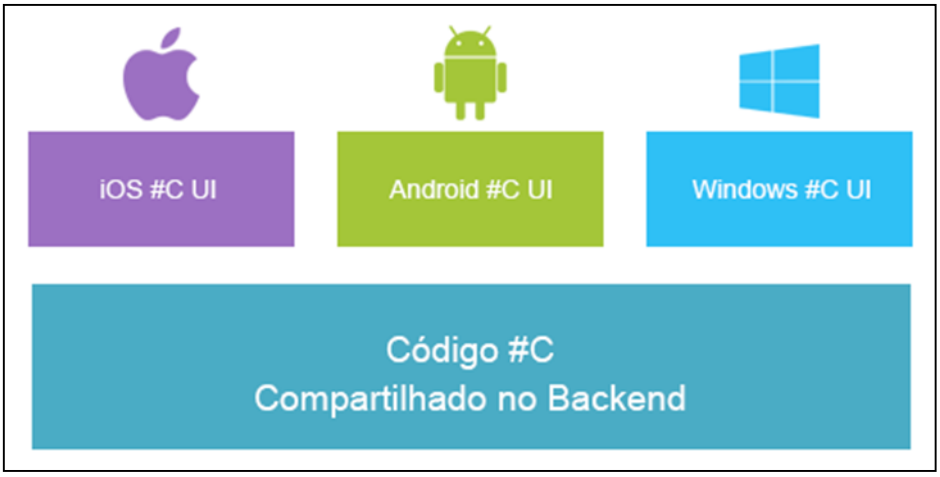
Após o lançamento do .NET em junho de 2000, a empresa Ximian iniciou o desenvolvimento do Mono, um projeto de código aberto com objetivo de criar uma implementação alternativa do compilador do C# e do Framework .NET que pudesse rodar no Linux. Em 2011, os fundadores do Ximian (Miguel de Icaza e Nat Friedman) fundaram o Xamarin, que ainda contribui para o projeto Mono mas com a proposta de adaptar o Mono para ser a base para soluções móveis multiplataforma (PETZOLD, 2016).

Em 2014, a Microsoft disponibilizou uma versão de código aberto do compilador do C#, chamado de .NET *Compiler Platform* (codinome “Roslyn”), juntamente com o anúncio da Fundação .NET se comprometendo em participar ativamente e contribuir para tecnologias .NET de código aberto, onde Xamarin atua principalmente. Com o objetivo de levar o desenvolvimento móvel multiplataforma para a comunidade de desenvolvedores Microsoft, em março de 2016 Xamarin foi comprada pela Microsoft que tornou a sua utilização gratuita para todos os usuários do Visual Studio (PETZOLD, 2016).

Inicialmente, Xamarin focou principalmente em tecnologias de compilação e em três conjuntos básicos de Bibliotecas .NET trazidas do projeto mono: Xamarin.Mac, Xamarin.IOS e Xamarin.Android. Juntas essas três bibliotecas formam a Plataforma Xamarin. As bibliotecas consistem de versões .NET de APIs nativas do Mac, IOS e Android. Permitindo, dessa forma, que utilizando essas bibliotecas desenvolvam aplicativos em C# com acesso as APIs nativas dessas três plataformas, e também com acesso à biblioteca de classes do .NET Framework (PETZOLD, 2016).

A principal vantagem em criar aplicativos para múltiplas plataformas utilizando uma única linguagem de programação é possibilidade de compartilhamento de código entre as aplicações. Como pode ser visto na Figura 3, toda a parte lógica do aplicativo é compartilhado entre os projetos, ficando apenas a parte da interface de usuário a ser desenvolvido utilizando as APIs específicas de SO.

Figura 3 – Esquema da plataforma Xamarin



Fonte: Adaptado de (XAMARIN, 2017a)

O código comum entre os projetos das diferentes plataformas é isolado em um outro projeto, podendo ser um *Portable Class Library* (PCL, traduzido como Biblioteca de classes portáteis), que coloca todo o código compartilhado em uma *Dynamic-link Library* (DLL, traduzido como Biblioteca de Link Dinâmico) que pode ser referenciada a partir de outros projetos, ou *Systems, Applications & Products* (SAP, traduzido como Projeto de Ativos Compartilhados), que são arquivos e códigos compartilhados entre os projetos. Em ambos os casos, o código compartilhado tem acesso à biblioteca de classes do framework .NET (PETZOLD, 2016).

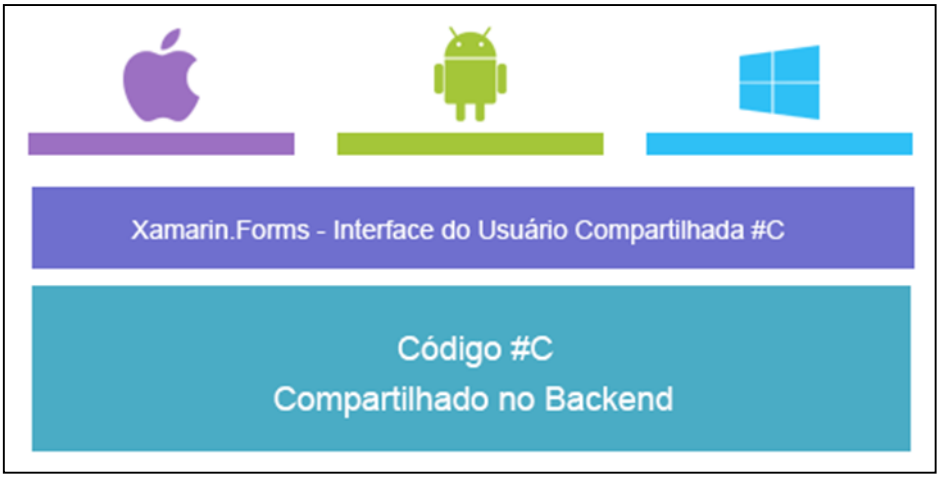
Os aplicativos criados usando o Xamarin são nativos, já que é possível explorar todo o potencial da linguagem de desenvolvimento específica, mesmo que não seja a mesma usada pela plataforma Xamarin. Quando o aplicativo IOS é compilado, o compilador C# do Xamarin gera uma linguagem intermediaria (LI) do C#, que então faz uso do compilador da Apple no Mac para gerar código nativo da máquina IOS, assim como o compilador do Objective-C. Para o aplicativo Android, o compilador C# do Xamarin gera a LI, que é executado em uma versão do Mono no dispositivo ao lado do motor Java, mas as chamadas da API do aplicativo são praticamente as mesmas como se o aplicativo fosse escrito em Java (PETZOLD, 2016).

Com o intuito de amenizar a necessidade de criar uma interface de usuário específica de cada plataforma, Xamarin apresentou o Xamarin.Forms, que consiste da utilização de bibliotecas para a implementação da API Xamarin.Forms. Essas bibliotecas são basicamente uma coleção de classes chamadas *renderers*, que

transformam os objetos da interface de usuário do Xamarin.Forms em objetos específicos de cada Sistema operacional (PETZOLD, 2016).

A Figura 4 ilustra a estrutura do Xamarin juntamente com o Xamarin.Forms, que representa o código da parte gráfica do aplicativo compartilhada entre as plataformas. Dessa forma, Xamarin permite um aumento significante no porcentual de código compartilhado entre os projetos das diferentes plataformas. A camada superior representa os recursos de interface específicos de cada sistema que não fazem parte do Xamarin.Forms, que ainda podem ser utilizados através do uso de suas APIs nativas.

Figura 4 – Xamarin + Xamarin.Forms



Fonte: XAMARIN, 2017a

Principais características do Xamarin.Forms:

- Fornece abstração dos elementos visuais específicos de cada plataforma, através de um modelo comum para todas as plataformas;
- Utiliza componentes nativos no momento da execução, mantendo a aparência nativa de cada plataforma;
- Componentes específicos de cada plataforma continuam acessíveis. Xamarin.Forms interage com os recursos nativos de cada plataforma, garantindo assim, que seus aplicativos possam ter acesso a todos recursos de um aplicativo desenvolvido nativamente;
- Facilita o compartilhamento de código entre as plataformas. Simplifica a utilização de código referente à uma plataforma específica juntamente com o código que é compartilhado entre as plataformas;

Aplicativos criados com o Xamarin.Forms são nativos, assim cada plataforma possui seu próprio projeto contendo seu código específico e faz uso de um projeto que é compartilhado, neste projeto é mantido o código comum a todas as plataformas.

Segue abaixo um exemplo de uma tela gerada para três plataformas (Figura 6) distintas a partir de um código XAML da imagem abaixo:

Figura 5 – Exemplo de código XAML

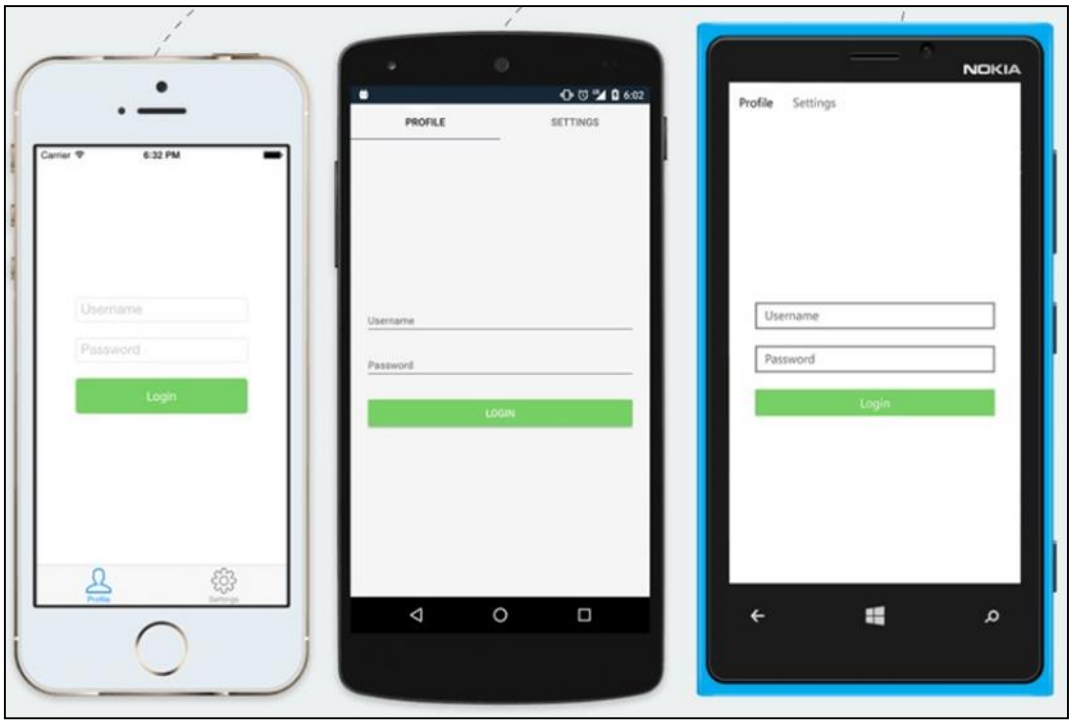
```
<?xml version="1.0" encoding="UTF-8"?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="MyApp.MainPage">
    <TabbedPage.Children>
        <ContentPage Title="Profile" Icon="Profile.png">
            <StackLayout Spacing="20" Padding="20"
                         VerticalOptions="Center">
                <Entry Placeholder="Username"
                       Text="{Binding Username}"/>
                <Entry Placeholder="Password"
                       Text="{Binding Password}"
                       IsPassword="true"/>
                <Button Text="Login" TextColor="White"
                        BackgroundColor="#77D065"
                        Command="{Binding LoginCommand}"/>
            </StackLayout>
        </ContentPage>
        <ContentPage Title="Settings" Icon="Settings.png">
            <!-- Settings -->
        </ContentPage>
    </TabbedPage.Children>
</TabbedPage>
```

C# XAML

Fonte:XAMARIN, 2018b

Tela gerada após a compilação dos projetos utilizando o Xamarin.Forms contendo o código da imagem acima:

Figura 6 – Tela do aplicativo em diferentes plataformas



Fonte: XAMARIN, 2018b

Uma desvantagem do Xamarin.Forms é que ele não é indicado para aplicações que contenham muitos elementos específicos de cada plataforma, isto é, aplicativos que ao serem disponibilizados para diferentes Sistemas Operacionais tenham aparência, comportamento e usabilidade diferentes, pois nestes casos o principal benefício da utilização do Xamarin.Forms, que é o compartilhamento de código, é prejudicado. Para que se atinja alto porcentual de compartilhamento de código entre os projetos é importante que aplicativos mantenham similaridade quando disponibilizados para os diferentes Sistemas Operacionais, utilizando os mesmos objetos de Interface de usuário e que não utilizem muitos recursos exclusivos de determinada plataforma.

2.8.4 Discussão

Como opção ao desenvolvimento nativo de aplicações móveis, foram apresentadas alternativas que permitem o desenvolvimento multiplataforma de duas maneiras distintas utilizando diferentes tecnologias: A utilização da plataforma Xamarin e a utilização do *framework* PhoneGap juntamente com o Ionic.

A escolha pelo desenvolvimento com Xamarin ou com PhoneGap + Ionic vai depender muito da preferência do desenvolvedor, pois a principal diferença entre eles está na forma como os as aplicações são desenvolvidas e as tecnologias utilizadas. Xamarin utiliza a linguagem de programação C# e está diretamente ligado às tecnologias da Microsoft, enquanto PhoneGap utiliza tecnologias ligadas ao desenvolvimento web, como HTML, CSS e JavaScript.

Pôde ser observado que tanto aplicações desenvolvidas com Xamarin como as desenvolvidos com Ionic mantém a aparência nativa, descartando este quesito como um diferencial de um ou de outro.

Sempre que há comparações entre desenvolvimento nativo e o desenvolvimento híbrido são levantadas questões referentes ao desempenho e fluidez das aplicações, onde aplicações nativas tendem a ter melhor desempenho e maior fluidez sobre as aplicações híbridas (WODEHOUSE, 2018).

Neste trabalho será estudado o Xamarin pelo fato deste utilizar a tecnologias da Microsoft, utilizadas no desenvolvimento com o Xamarin, e por compilar e gerar aplicativos nativos.

2.9 SERVIÇO DE APLICATIVOS MÓVEIS MICROSOFT

Azure é a plataforma de serviços em nuvem da Microsoft. Consiste de um conjunto abrangente de serviços em nuvem utilizados por desenvolvedores e profissionais de TI para criar, implantar e gerenciar aplicativos através de uma rede global de datacenters. Possui uma variedade de produtos que facilitam a construção de forma eficiente de aplicativos móveis simples até soluções de escala de internet (AZURE, 2018b).

Utilizando os serviços da Azure é possível hospedar aplicativos existentes, simplificar o desenvolvimento de novos aplicativos e ainda aprimorar aplicativos locais. O Azure integra os serviços de nuvem necessários para desenvolver, testar, implantar e gerenciar. O Azure fornece infraestrutura como um serviço, plataforma como um serviço e ainda serviço para hospedagem de código (AZURE, 2018).

O Serviço de Aplicativos Móveis no Azure é uma oferta de Plataforma como Serviço. Oferece um conjunto de recursos para o desenvolvimento de Aplicativos Móveis que permitem a criação de aplicativos nativos e multiplataforma através da utilização dos *Software Development Kits* (SDK, em português kit de Desenvolvimento

de Software) disponibilizados (AZURE, 2017). A imagem abaixo mostra uma visão geral do Serviço.

Figura 7 – Recursos do Serviço de Aplicativos Móveis

Fonte: AZURE, 2018

A comunicação entre o aplicativo e o serviço é feito através da utilização da API disponibilizada. O serviço pode ser utilizado com as principais plataformas móveis utilizadas atualmente, como mostra a Figura 6. Segue abaixo a descrição dos principais recursos:

- Armazenamento de Dados em Nuvem - são ofertados os serviços para que os aplicativos utilizem um banco de dados remoto, facilitando a sincronização de dados em tempo real entre os diferentes dispositivos e os dispositivos de diferentes plataformas
- Autenticação de Usuário - é um recurso que oferece uma maneira para seu aplicativo conectar usuários utilizando provedores de identidade de terceiros: Azure *Active Directory*, Facebook, Google, Conta da Microsoft e Twitter. O aplicativo se baseia nas informações de identidade do provedor para que o aplicativo não tenha de armazenar essas informações por conta própria (AZURE, 2018).
- Notificações por *Push* - Facilita a implementação de Notificações por *Push* pois disponibiliza a infraestrutura necessária para a sua utilização.

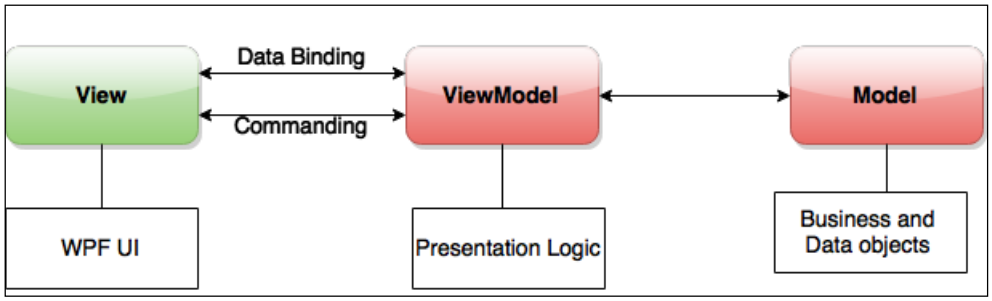
A utilização do serviço de aplicativos móveis visa facilitar o desenvolvimento de soluções para os itens citados acima através da implementação de um *back-end* comum a todas as plataformas e que forneça os recursos necessários para sua

implementação, deixando que o desenvolvedor se concentre em desenvolver apenas lado cliente da aplicação e então apenas configure os recursos no lado servidor (AZURE, 2018c).

2.10 MVVM

MVVM é um padrão de desenvolvimento de projetos introduzido em 2005 pela Microsoft. Visa estabelecer a separação entre a lógica de apresentação, lógica de negócio e a lógica de *back-end*. A comunicação entre as camadas de Interface e da camada de Lógica de Apresentação se dá pelo mecanismo de *Binding* (Figura 8), detalhado nas subseções a seguir. Essa separação entre camadas permite que o código tenha algumas características desejáveis como: estabilidade, modularidade, manutenibilidade e flexibilidade (PROCEDI, 2016).

Figura 8 – Padrão da arquitetura MVVM



Fonte: DOTNETPATTERN,2018

O MVVM é composto por três componentes: *View, ViewModel* e *Model*, abaixo está a definição de cada um deles.

2.10.1 View

É a interface do usuário desenvolvido na linguagem de marcação XAML, ou seja, as *Views* são as telas do sistema. É responsável por apresentar informações na tela ou executar as ações do usuário. A comunicação entre a View e a View *Model* se dá pela propriedade *DataBinding* definida na View (Figura 9). Esta propriedade armazena uma referência da *ViewModel* que através da propriedade *Binding* se

comunica com a *ViewModel*, passando e recebendo dados, enviando comandos e recebendo notificações da ViewModel (PROCEDI, 2016).

Figura 9 – Exemplo de utilização da propriedade *BindingContext*

```
10    namespace jogada.Views.Match
11    {
12        [XamlCompilation(XamlCompilationOptions.Compile)]
13        public partial class MatchPage : ContentPage
14        {
15            private Models.Match match;
16
17            public MatchPage ()
18            {
19                BindingContext = new MatchViewModel();
20                InitializeComponent ();
21            }
```

Fonte - Do Autor (2018)

2.10.2 ViewModel

É o mediador entre a *View* e o *Model*. É responsável por manipular a Lógica de Apresentação, fornecendo os dados para a View através da exposição de suas propriedades e do Binding feito pela View. É dessa forma que a View é notificada das alterações nos dados da *ViewModel* e também que a ViewModel recebe as modificações feitas na View. A *ViewModel* se comunica diretamente com a Model tendo acesso as suas propriedades e métodos (PROCEDI, 2016).

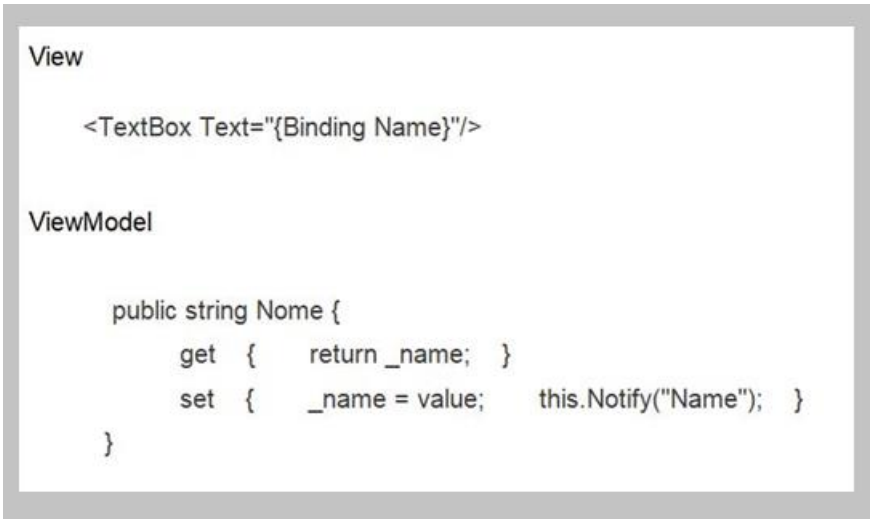
2.10.3 Model

São as classes que representam os objetos do *backend* (tabelas do banco de dados), utilizadas para recuperar e salvar os dados.

2.10.4 Binding

Binding é o modo que uma *View* usa para se comunicar com a *ViewModel*, é através dessa associação que os dados da View refletem em alterações nas propriedades da *ViewModel* e que a *View* é notificada das alterações das propriedades da *ViewModel* ocorrendo a sincronização entre ambas instantaneamente (PROCEDI, 2016). Exemplo da comunicação entre a *View* e a *ViewModel* pode ser visto na imagem a seguir.

Figura 10 – Exemplo de Código



Fonte - Do Autor (2018)

3 METODOLOGIA

Este capítulo apresenta a metodologia empregada no desenvolvimento de uma aplicação móvel utilizando a plataforma Xamarin e as tecnologias descritas anteriormente. A seguir é apresentado a descrição do aplicativo, modelagem do sistema, diagramas de Caso de Uso e de Classes e ambiente de desenvolvimento.

3.1 DESCRIÇÃO DO APLICATIVO

A dificuldade em organizar partidas de futebol entre amigos remete à necessidade de desenvolver um aplicativo para esse fim. Para que um jogo ocorra é necessário um número mínimo de jogadores. O problema é que muitas vezes os jogos são cancelados devido à falta de jogadores disponíveis para determinados horários e locais. Isso acontece porque o meio de comunicação entre os jogadores é ineficiente e ineficaz, pois a falta de um aplicativo específico para a organização dos jogos leva a utilização, por parte dos jogadores, de aplicativos genéricos sem recursos que facilitem essa tarefa.

O aplicativo a ser desenvolvido visa facilitar a comunicação entre os jogadores, que utilizando o aplicativo, poderão interagir entre si com o objetivo de organizar os jogos. Os principais recursos do sistema são:

- Autenticação e Autorização - A primeira tela do sistema é a tela de *login*. Somente usuários autenticados terão acesso ao aplicativo, e através da implementação de autorização será garantida a integridade e segurança dos dados da conta do usuário. Cada usuário poderá visualizar o perfil de seus amigos;
- Adicionar Amigos - Os usuários poderão adicionar outros usuários, criando assim suas redes de contatos que será utilizado para enviar convites para as partidas criadas;
- Gerenciar partidas de Futebol: Os usuários poderão criar os jogos e enviar convites aos seus amigos, estes ao receberem o convite da partida terão a opção de aceitar ou recusar o convite para o jogo.
- Visualizar Jogos - Será o local onde os usuários terão acesso as informações de todos jogos. Os jogos poderão ser criados, alterados e excluídos. As informações serão atualizadas conforme o criador do jogo o alterar, neste momento, todos os usuários poderão visualizar as alterações nesta tela.

- Recursos extras - Serviço utilizado para troca de mensagens em tempo real entre usuários, como ocorre no Facebook, por exemplo, e serviço de autenticação que permite o *login* no aplicativo utilizando mecanismos externos, como por exemplo, sua conta no Facebook, Google e Microsoft, estes recursos serão implementados apenas se houver tempo hábil disponível.

3.2 MODELAGEM DO SISTEMA

Através do levantamento de requisitos foi realizado a modelagem do aplicativo que será desenvolvido. Este será desenvolvido para Android e IOS e terá como objetivo principal gerenciar as partidas de futebol criadas pelos usuários.

3.2.1 Requisitos Funcionais e Não Funcionais

Requisitos funcionais especificam funções que o sistema deverá executar. Requisitos não funcionais descrevem como o sistema fará determinado tarefa, estão relacionados com desempenho, usabilidade e tecnologias utilizadas pelo sistema.

Requisitos Funcionais:

- Manter usuários;
- Manter amigos;
- Criar partidas de futebol;
- Manter convites;
- Exibir informações das partidas;

Requisitos Não Funcionais:

- O aplicativo deve ser multiplataforma;
- O Aplicativo só funcionará quando o usuário possuir conexão com a internet;
- O aplicativo deverá utilizar um *back-end* na nuvem com suporte para banco de dados;

3.2.2 Diagrama de Casos de Uso

Este diagrama tem por objetivo demonstrar a interação do sistema com o usuário. Permite uma visualização resumida das ações que ocorrerão no sistema e dos atores do envolvidos em cada caso de uso.

Figura 11 – Diagrama de Casos de Uso

Fonte: Do Autor (2018)

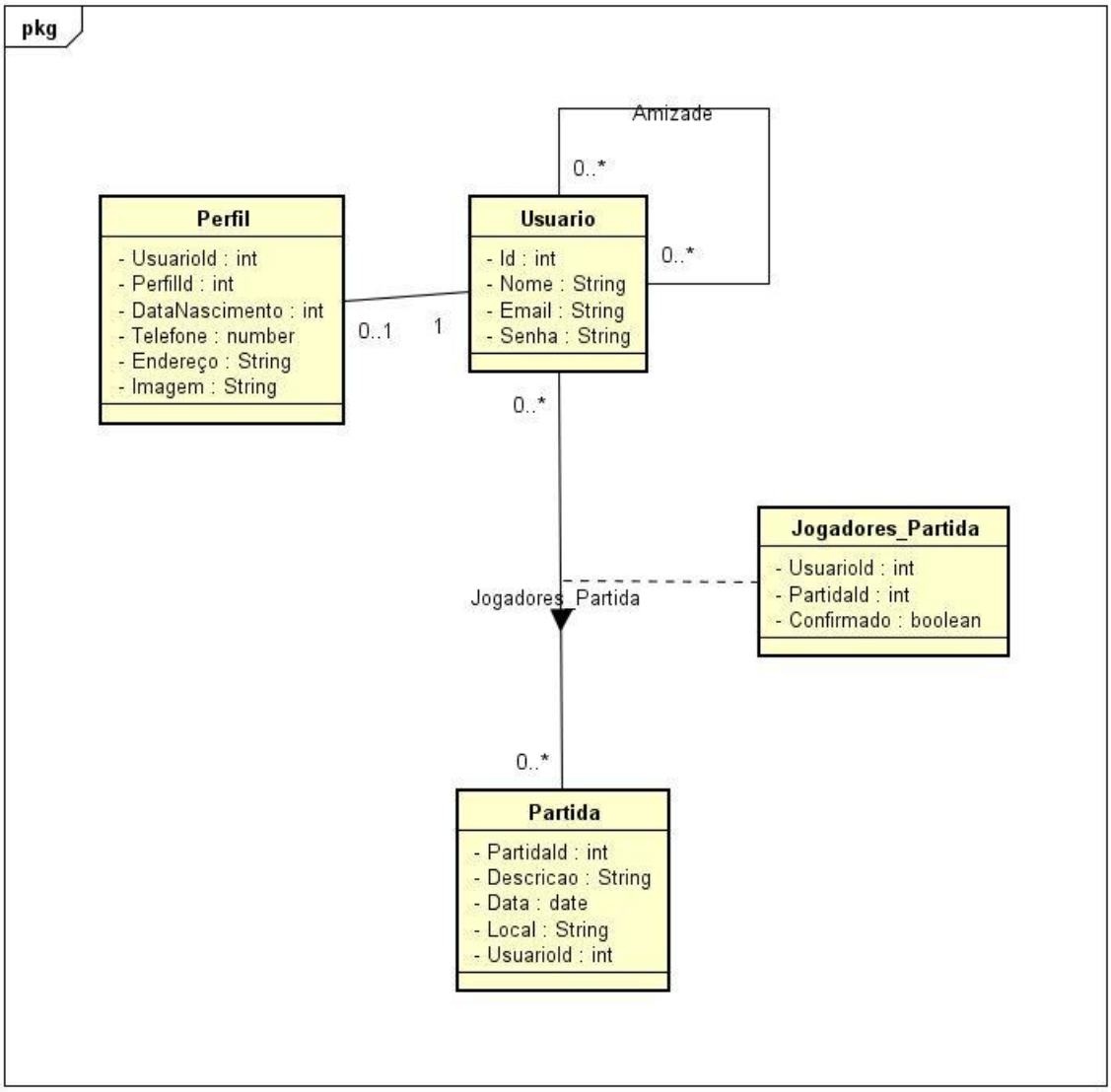
O detalhamento do diagrama de casos de uso encontra-se no Apêndice A da seção Apêndices deste trabalho.

3.2.3 Diagrama de Classes

O diagrama de classes visa demonstrar como é a organização das classes dentro do sistema. A Figura 12 apresenta um diagrama com foco na persistência de

dados. Nele se encontram as classes, seus atributos, relacionamentos e multiplicidades.

Figura 12 – Diagrama de Classes



Fonte: Do autor (2018)

A classe *Usuário* possui os atributos referentes ao usuário do aplicativo, os atributos e-mail e senha serão utilizado como *login*. Esta classe possui um auto relacionamento consigo mesma, pois cada usuário poderá adicionar outros usuários que serão seus amigos no aplicativo.

A classe *Perfil* mantém informações extras referentes a cada usuário. No momento em que é criado um usuário, este é automaticamente associado à um perfil em branco, que poderá ser modificado posteriormente.

A classe *Partida* armazena informações dos jogos criados pelos usuários, esta classe se relaciona com usuários gerando um relacionamento muitos para muitos, uma vez que um usuário pode criar várias partidas e cada partida contém vários jogadores, gerando a tabela associativa*Jogadores_Partida.*

Na classe *Jogadores_Partida* há o atributo "Confirmado", sendo do tipo booleano e será usado para informar se o usuário está confirmado ou não para o jogo.

3.2.4 Ambiente de Desenvolvimento do Aplicativo

A IDE utilizada é o Visual Studio Community 2017 e o tipo do projeto é Cross-Platform Xamarin.Forms. Seu *back-end* utiliza o Serviço de Aplicativos Móveis da plataforma Azure, contendo o banco de dados relacional SQL Server. O aplicativo implementará ainda o recurso de *Storage* presente no Azure que será o repositório de imagens do aplicativo.

4 DESENVOLVIMENTO

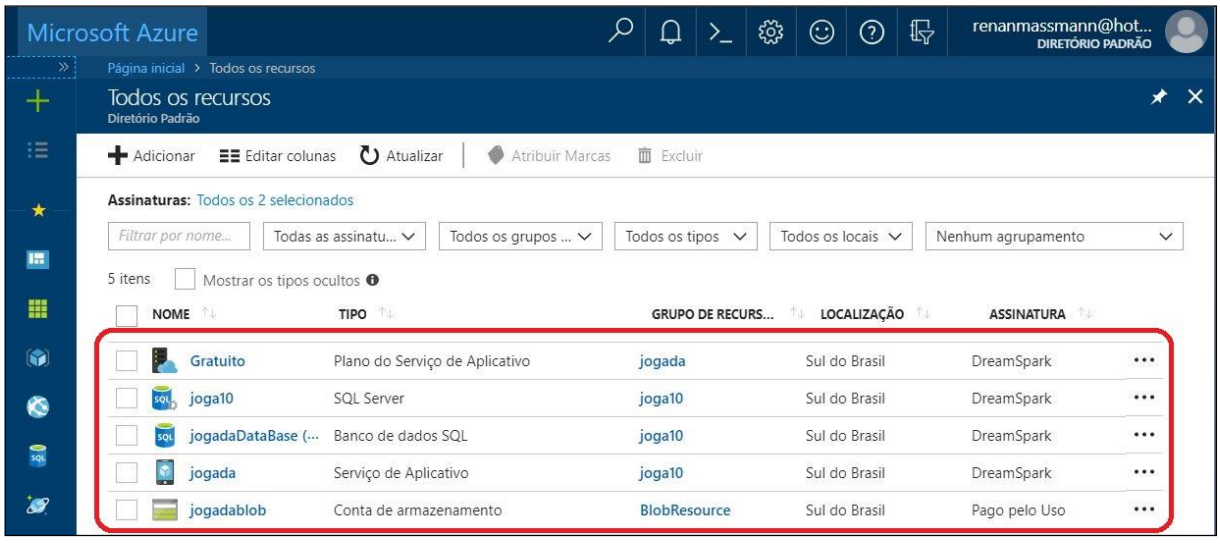
Este capítulo demonstra como foi realizado o desenvolvimento do projeto utilizando o framework Xamarin no Visual Studio, juntamente com os recurso de Banco de Dados e Armazenamento de Blobs[14] do Serviço de Aplicativos Móveis da plataforma Azure, consumidos pelo aplicativo.

4.1 BASE DE DADOS NO AZURE

No Portal do Azure foi criado o Serviço de Aplicativos Móveis onde foi configurado informações pertinentes ao serviço e também o banco de dados que o aplicativo iria usar, no caso foi necessário criar um Servidor SQL Server e então criar o banco de dados nele; Para o armazenamento de imagens foi criado o recurso de *Storage* para armazenar *Blobs* de imagens.

A Figura 13 exibe, em destaque, os recursos criados, necessários para utilizar o Azure como base de dados do aplicativo. Cada um desses recursos podem ser configurados de acordo com as necessidades do cliente.

Figura 13 – Recursos criados no Azure para o projeto



Fonte - Do Autor (2018)

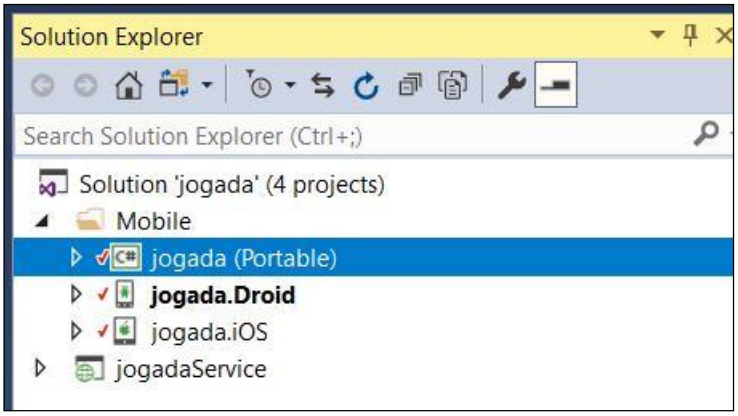
[14] É uma estrutura de dados binários.

O jogada é o serviço de aplicativo que foi criado, ao abri-lo é apresentado um menu com opções de configurações do aplicativo, monitoramento, ferramentas de desenvolvimento, plano do serviço, implementação e segurança. Em implementação na opção início rápido foi baixado a solução para um projeto Xamarin.Forms juntamente com o projeto do tipo ASP.NET Web API Project (.NET *backend)*, explicados no item a seguir. O Servidor SQL Server é o joga10, que possui o banco de dados **jogadaDataBase** utilizado pelo jogada, os três estão contidos no plano de Serviços de Aplicativo Gratuito (plano básico). Já o **jogadablob** é um recurso à parte criado apenas para servir como repositório de imagens para o aplicativo.

4.2 ESTRUTURA DO PROJETO

A *Solução* gerado pelo Serviço de Aplicativo do Azure contém quatro projetos (Figura 14), onde os projetos da pasta *Mobile* pertencem a um Projeto do tipo *Cross-Platform* Xamarin.Forms e o jogadaService é um Projeto do tipo ASP.NET Web API Project (*.NET backend*), sendo a instância de serviço móvel que suporta o aplicativo. A vantagem em baixar a solução diretamente do Azure é que o projeto jogada vem pré-configurado para consumir os recursos do projeto jogadaService, como por exemplo o acesso ao banco de dados.

Figura 14 – Estrutura do Projeto

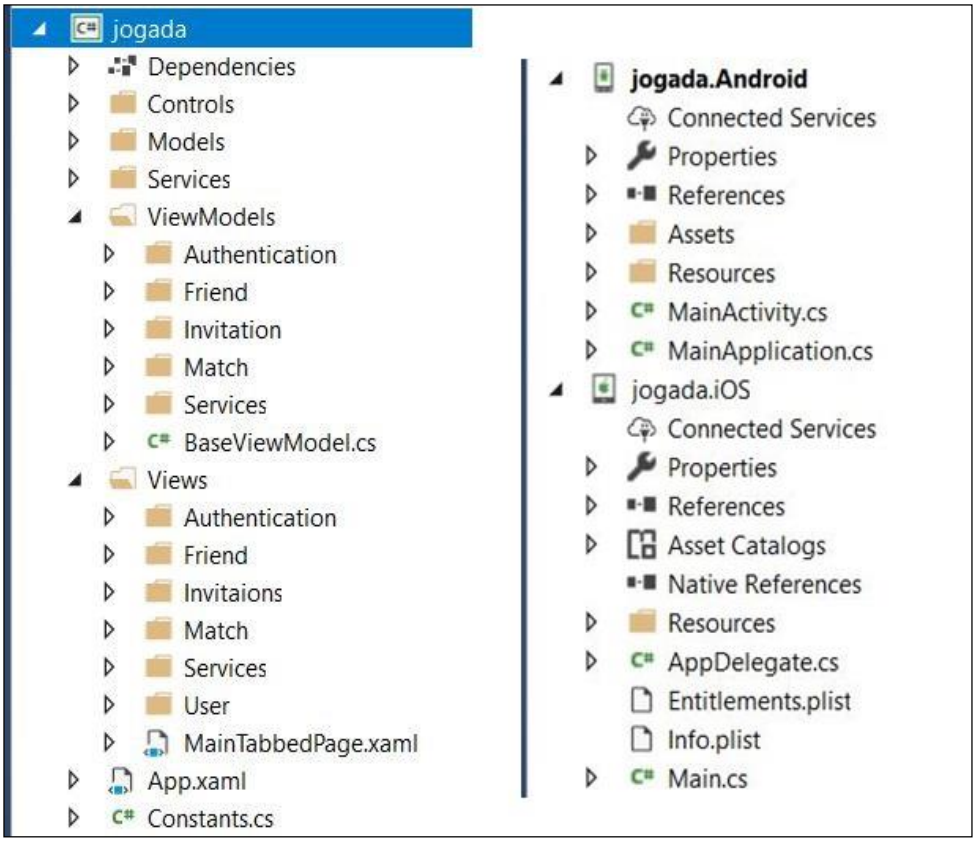


Fonte - Do Autor (2018)

4.2.1 Projeto jogada

Composto pelos projetos **jogada, jogada.Droid e jogada.IOS**. O projeto jogada contém tudo o que é compartilhado por todos os projetos da *Solução*. Os dois projetos restantes são específicos de cada plataforma, Android e IOS, nesses projetos só deve ser implementado o código específico da plataforma, e que irá complementar o código base (compartilhado). A Figura 15 exibe a estrutura dos projetos, note que praticamente 100% do conteúdo criado no desenvolvimento do aplicativo está no projeto jogada e que este segue o padrão MVVM.

Figura 15 – Detalhamento da estrutura do projeto Xamarin.Forms



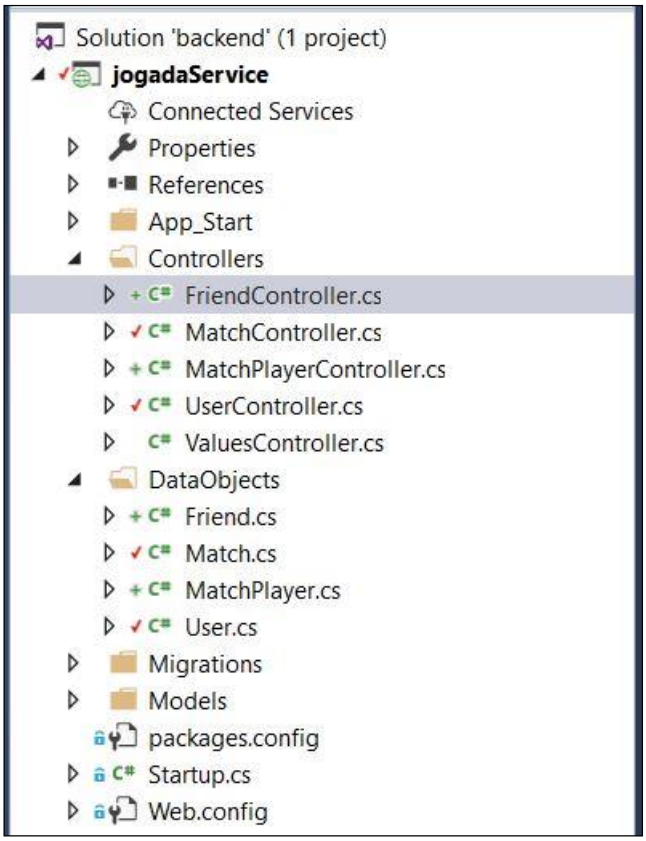
Fonte - Do Autor (2018)

4.2.2 Projeto JogadaService

O JogadaService é *o* Serviço do projeto jogada. A Figura 16 exibe sua estrutura, onde a pasta *DataObjects* contém as classes dos objetos que através do recurso

Migrations do *Entity Framework*[15], refletem as tabelas no banco de dados. *Migrations* também são utilizadas para atualizações do banco quando as classes são modificadas. A pasta *Controllers* contém controles do tipo Azure Mobile *Table Controller*, que são *Web APIs* para a criação das tabelas no Azure usando o *Entity Framework*. Este projeto é publicado no Azure no link[16] do Serviço de Aplicativo, tornando o Serviço disponível para ser consumido pelos aplicativos clientes.

Figura 16 – Detalhamento da estrutura do *back-end* do Projeto



Fonte - Do Autor (2018)

4.2.3 Operações CRUD

O acesso aos dados do banco de dados remoto para realizações das operações CRUD (consultas, inserções, exclusões e alterações) é feito através da

---

[15] Ferramenta de mapeamento objeto relacional (*Object Relational Management* - ORM), que permite aos desenvolvedores trabalhar com classes (entidades) que correspondem a tabelas em um banco de dados.

[16] http://jogada.azurewebsites.net

utilização dos métodos da Interface disponibilizada pelo Serviço. Esta interface contém métodos, sendo os mais utilizados neste projeto os seguintes*: InsertAsync(), UpdateAsync(), DeleteAsync(), ReadAsync() e LookupAsync()*. Um exemplo de leitura de dados da tabela *Friends* pode ser visto na Figura 17, onde a classe *Constants* contém a variável *friendTable* que é a referência da tabela *Friend* no banco de dados do Azure, que então é utilizada no método *GetFriends* para chamar o método *ReadAsync()* e retornar a lista de amigos.

Figura 17 – Exemplo de Código Consulta

```
public static class Constants
{
    // Replace strings with your Azure Mobile App endpoint.
    public static string ApplicationURL = @"https://jogada.azurewebsites.net";
    public static MobileServiceClient azClient = new MobileServiceClient(ApplicationURL);
    public static IMobileServiceTable<Friend> friendTable = azClient.GetTable<Friend>();
}

public void GetFriends()
{
    var friends = Constants.friendTable.ReadAsync();
}
```

Fonte: Do Autor (2018)

5 RESULTADOS

Este capítulo apresenta o aplicativo desenvolvido de acordo com a metodologia proposta no capítulo 3 e do desenvolvimento explicado no capítulo 4, exibindo suas telas e descrevendo suas funcionalidades, bem como sua avaliação pelos usuários. Como o aplicativo foi desenvolvido para as plataformas Android e IOS, por padrão serão utilizadas as imagens da tela do Android quando as interfaces forem iguais em ambos os Sistemas Operacionais, havendo diferença, as telas das duas plataformas serão apresentadas.

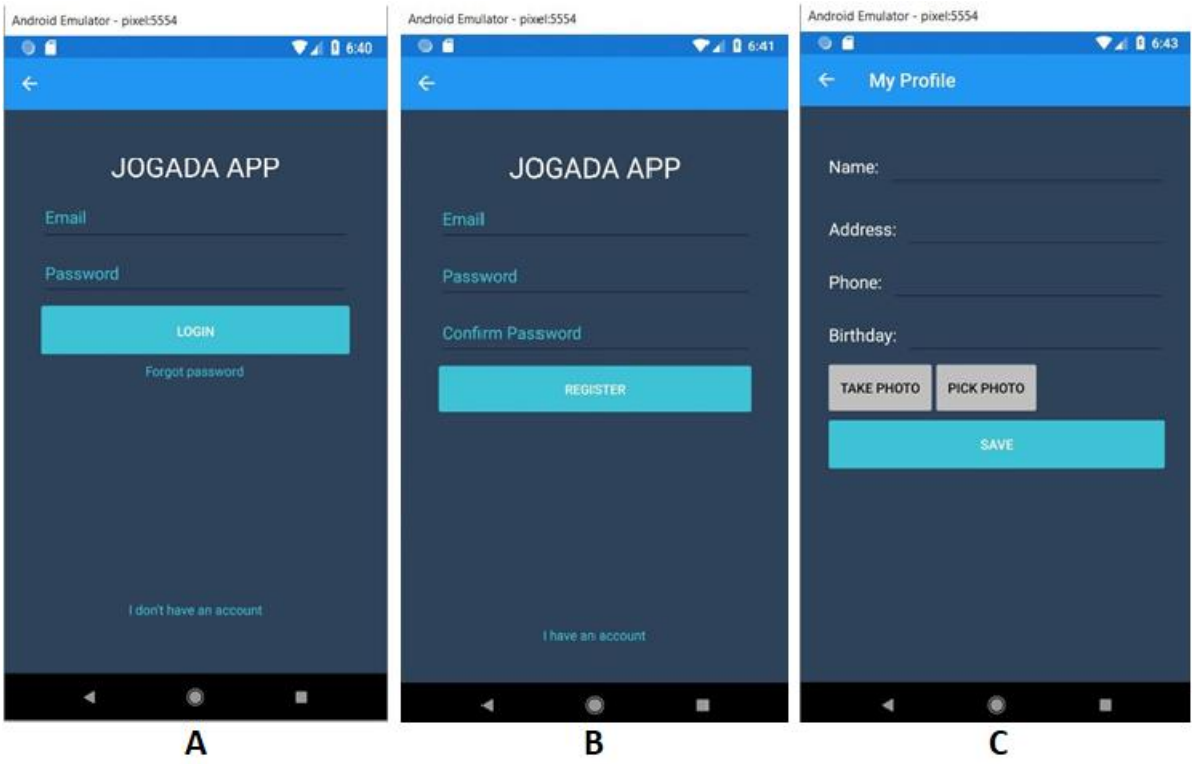
5.1 SISTEMA DESENVOLVIDO

Nessa seção são descritas as funcionalidades do aplicativo e são ilustradas as suas telas. O Item 5.1.1 é descreve como funciona o sistema de autenticação; O Item 5.1.2 apresenta a Tela Principal do aplicativo; O Item 5.1.3 detalha a relação de amizades entre os usuários; O Item 5.1.4 descreve as funcionalidades relacionadas aos jogos; O Item 5.1.5 descreve o funcionamento dos convites de jogos recebidos pelos usuários.

5.1.1 Cadastro de Usuários e Login

O processo de cadastro e login no aplicativo é ilustrado na Figura 18. Ao instalar o aplicativo e executá-lo pela primeira vez, é exibido a tela de login (Figura 18A). Caso o usuário não possua cadastro deverá alternar para a tela de Registro (Figura 18B) e informar os dados para criação da conta, e então direcionado para a tela de Perfil (Figura 18C). Na tela de perfil o usuário preenche o formulário com os detalhes da sua conta e adicionar uma imagem, finalizando criação da conta e sendo direcionado para a tela principal do aplicativo.

Figura 18 – Tela de login, registro e perfil

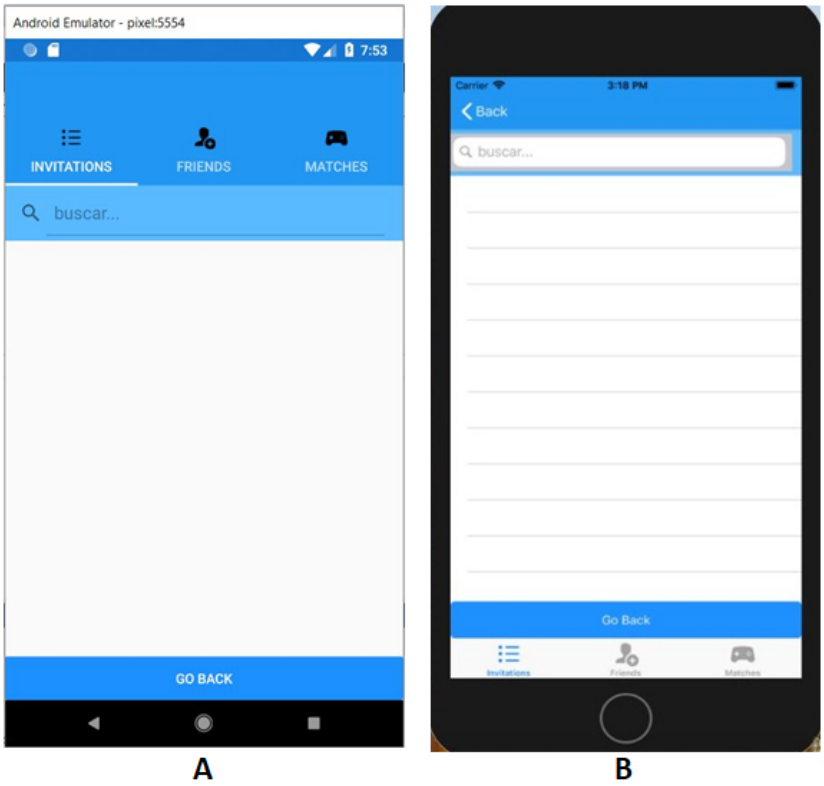


Fonte - Do Autor(2018)

5.1.2 Tela Principal

A Figura 19 apresenta a tela principal exibida no Android (Figura 19A) e no IOS (Figura 19B), note que por padrão as abas são apresentadas na parte superior da tela no Android e na parte inferior no IOS. A tela principal é composta por três abas: Convites, Amigos e Jogos, as quais são detalhadas a seguir.

Figura 29 – Tela Principal do Aplicativo



Fonte - Do Autor (2018)

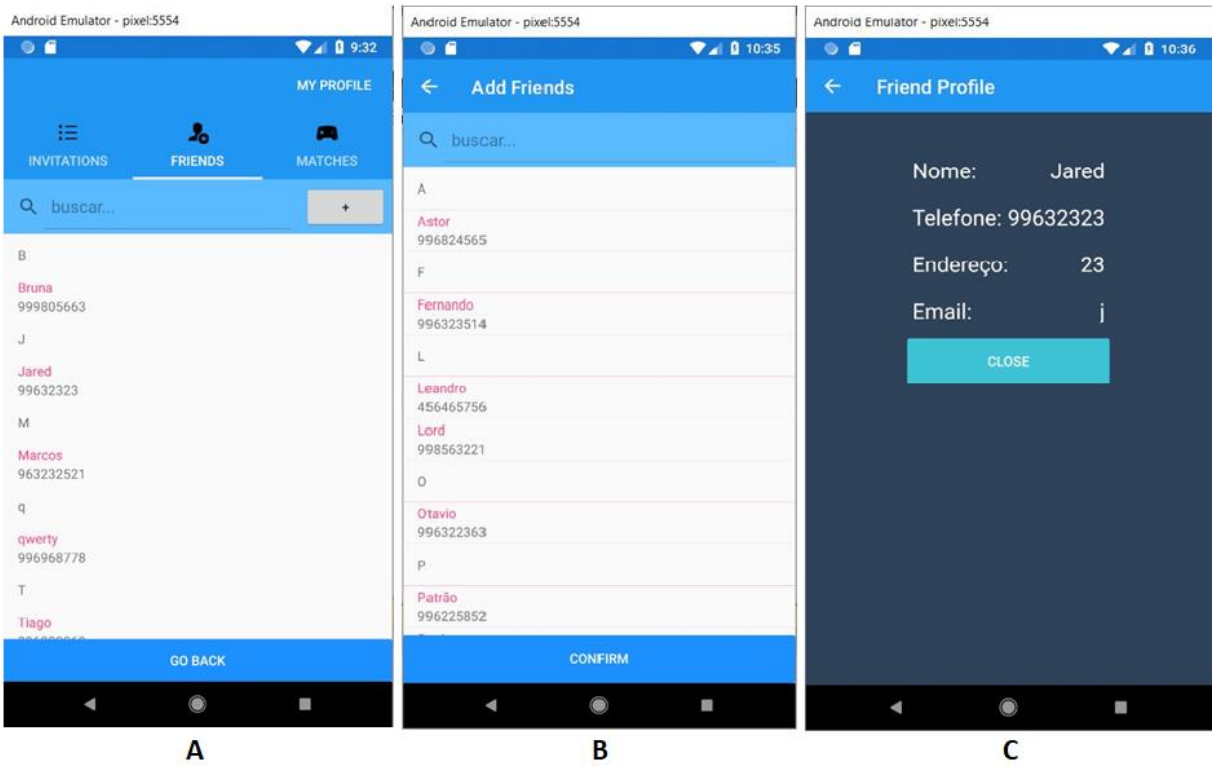
5.1.3 Manutenção de Amigos

A Figura 20 apresenta as telas referentes a manutenção de amigos. A aba Amigos (Figura 20A) carrega a página com a lista de amigos, e ao clicar em um nome o perfil é apresentado (Figura 20B). Ainda, esta página contém um campo de busca usado para filtrar amigos pelo nome e um botão para adicionar novos amigos.

A tela de adição de amigos (Figura 20C) apresenta a lista com os usuários cadastrados no aplicativo, estes podem ser filtrados através do campo de busca. Quando o usuário clica (toque) no nome que quer adicionar, este é removido desta lista e adicionado na lista de amigos, estas operações são persistidas no banco de dados.

A lista de amigos aparece nos jogadores disponíveis a serem adicionados para um jogo criado pelo usuário, ou seja, ao criar um jogo o usuário poderá convidar apenas seus amigos para o jogo.

Figura 20 – Telas Manutenção de Amigos

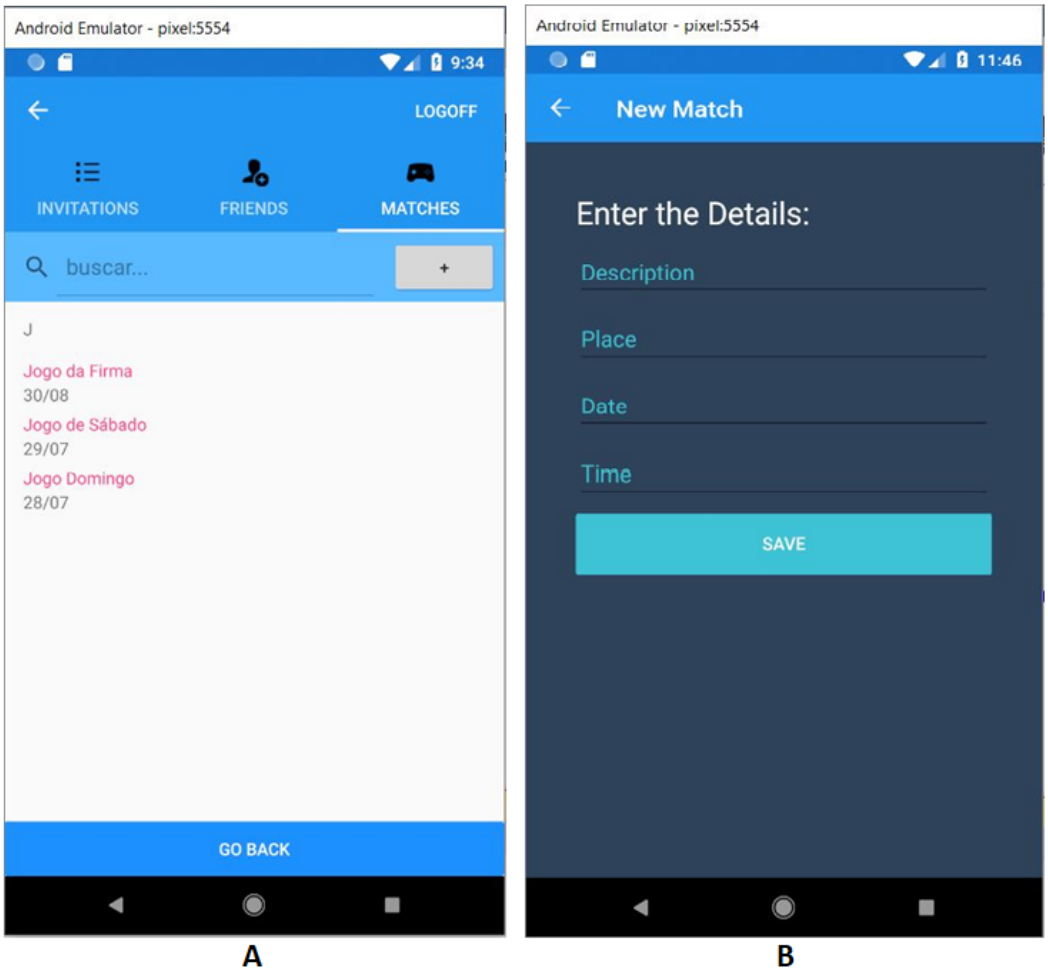


Fonte - Do Autor (2018)

5.1.4 Manutenção de Jogos

A Figura 21 exibe a tela de jogos e o formulário de um novo jogo. A tela de jogos (Figura 21A) carrega a lista de jogos contendo a descrição e a data do jogo, bastando clicar em um jogo para visualizar seus detalhes. Ainda, esta página contém um campo de busca usado para filtrar os jogos pela descrição e um botão para adicionar um novo jogo, que abre a tela com o formulário do novo jogo (Figura 21B).

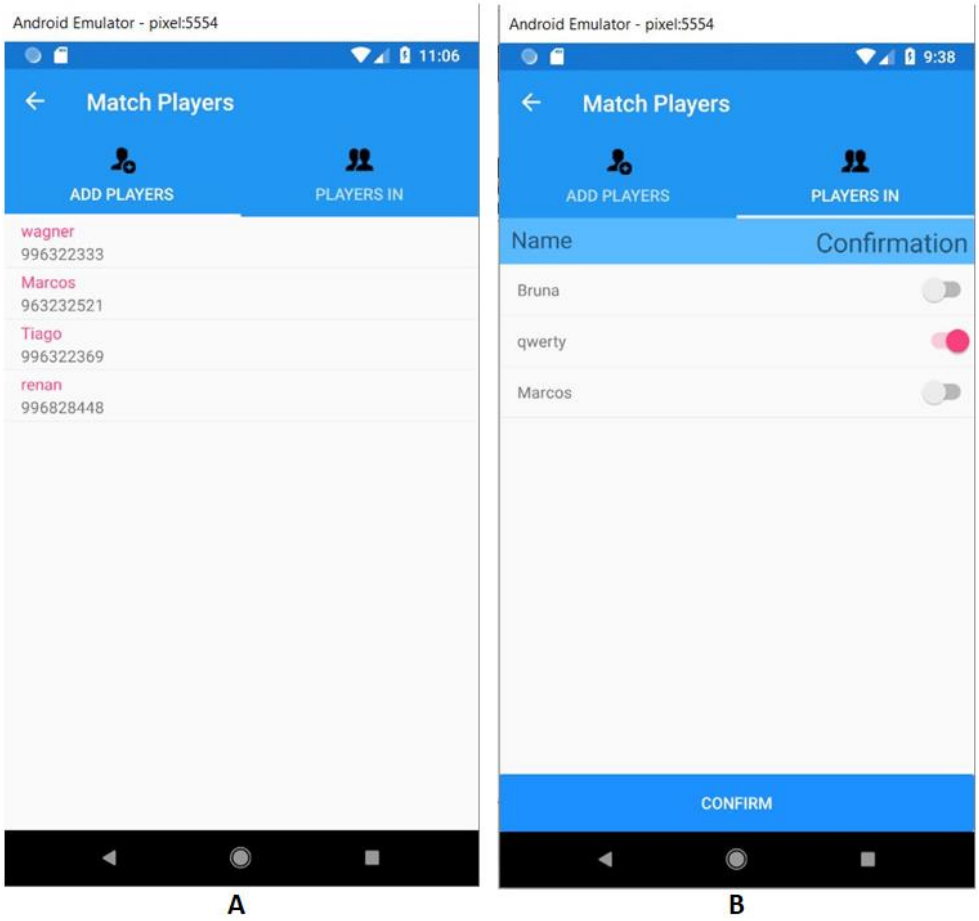
Figura 23 – Tela de Jogos e Formulário de novo Jogo



Fonte - Do Autor (2018)

Ao salvar o novo jogo, a tela de adição de jogador ao jogo é aberta (Figura 22). Esta tela é composta por duas abas, uma carrega a tela de adição de jogador (Figura 22A) e a outra mostra os jogadores adicionados (Figura 22B). A lista de jogadores disponíveis é composta pelo usuário e seus amigos, para adicionar um jogador basta clicar em seu nome. A tela de Jogadores do Jogo permite que o Usuário inicie o jogador como confirmado ou não para o jogo, contudo cada jogador tem a opção de aceitar ou recusar o convite do jogo quando este aparecer na sua lista de convites.

Figura 22 – Tela de Adição de Jogadores para o jogo

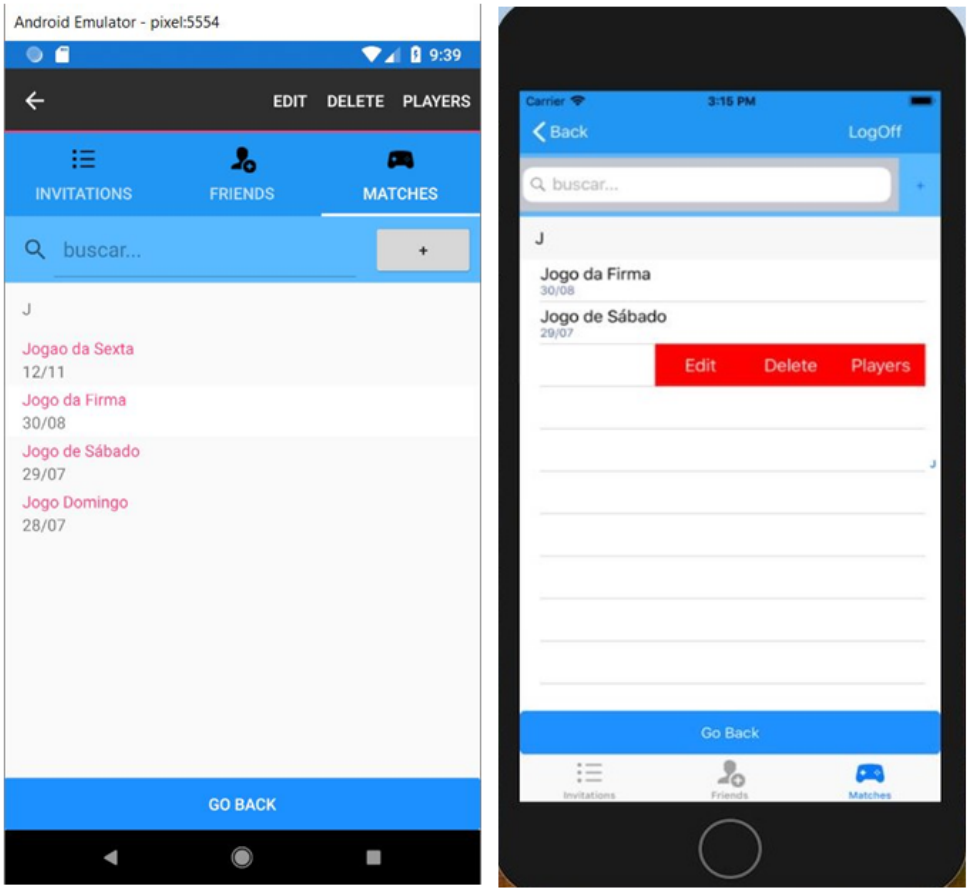


Fonte - Do Autor (2018)

As opções de cada jogo pode ser visto na Figura 23. Manter o toque por um momento (no Android) ou clicar e arrastar (no IOS) uma célula da lista, executa a ação chamada *ContextActions,* que exibe opções para a célula e, nesse caso, habilita as opções de deletar, editar e visualizar jogadores do jogo.

Note na imagem a seguir a diferença da interface no Android (Figura 23A) e no IOS (Figura 23B) para a ação explicada anteriormente.

Figura 23 – Opções para cada jogo da lista

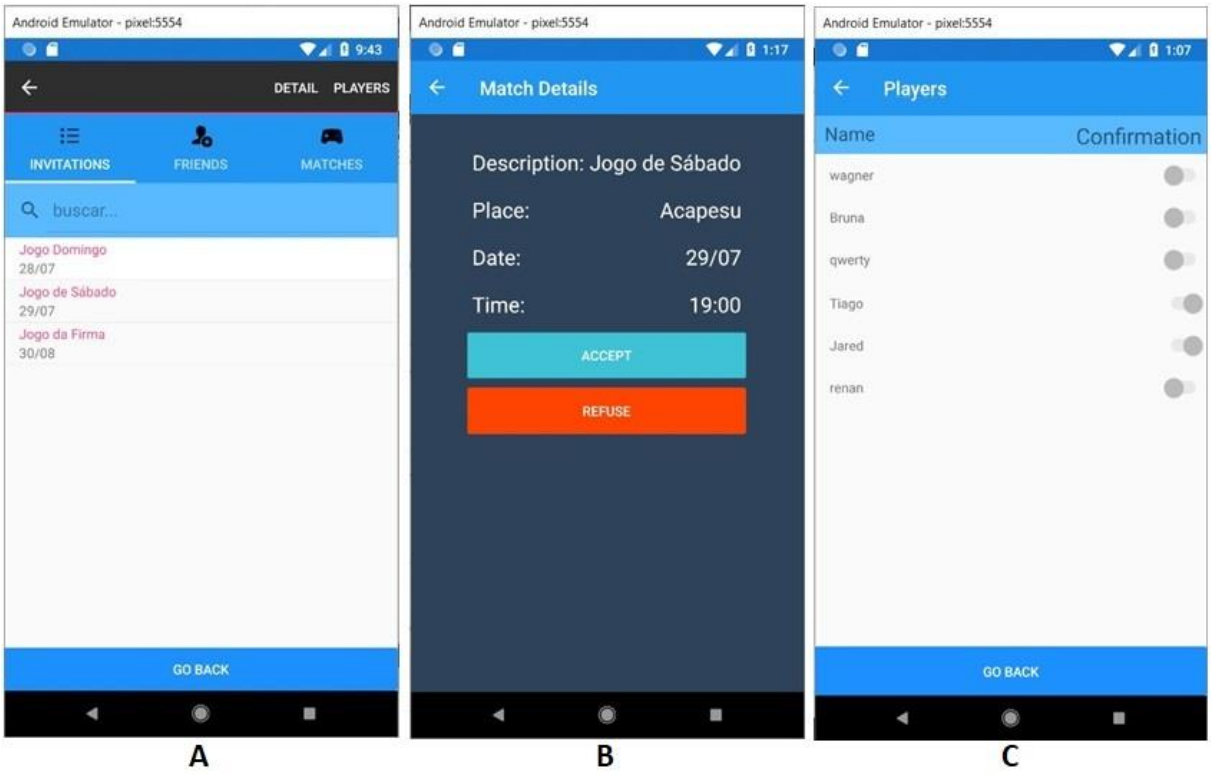


Fonte - Do Autor (2018)

5.1.5 Manutenção de Convites

Na tela de convites (Figura 24A) é mostrado a lista de jogos em que o usuário foi convidado, ou seja, jogos criados por outros amigos e que o adicionaram na lista de jogadores. As células desta lista são similares as da lista de jogos e se comportam da mesma maneira, habilitando as opções de Detalhes e Jogadores. Ao clicar em Detalhes a tela contendo as informações pertinentes ao jogo é mostrada (Figura 24B), contendo dois botões utilizados para que o usuário aceite ou recuse o convite do jogo. Após aceitar ou recusar o convite a tela de jogadores desta partida é mostrada (Figura 24C), contendo o nome dos jogadores e status de confirmação para a participação na partida.

Figura 24 – Telas de Manutenção de Convites



Fonte: Do Autor (2018)

5.2 AVALIAÇÃO DO APLICATIVO PELOS USUÁRIOS

Para fazer uma avaliação e validar o aplicativo desenvolvido, identificando possíveis melhorias a serem realizadas posteriormente, foi realizado uma pesquisa através da disponibilização do aplicativo para um seis usuários que testaram o aplicativo em seus smartphones. Os usuários realizaram as tarefas contidas em um questionário e então responderam as questões de múltipla escolha utilizadas para a avaliação do aplicativo e apresentadas neste trabalho.

O questionário foi construído com base no questionário presente na monografia de (PROCEDI, 2016). Ele é composto por tarefas que simulam o uso do aplicativo e questões de avaliação do tipo múltipla escolha.

Para avaliação do aplicativo pelos usuários foram definidas as seguintes tarefas:

- Tarefa 1: Criar um Usuário e Logar no Aplicativo.
- Tarefa 2: Atualizar o Perfil inserindo seus dados no formulário.
- Tarefa 3: Adicionar alguns amigos.

• Tarefa 4: Visualizar o perfil de algum amigo.

• Tarefa 5: Criar um jogo.

• Tarefa 6: Receber resposta dos amigos convidados para o jogo.

• Tarefa 7: Alterar data e local do jogo.

• Tarefa 8: Visualizar informações do jogo para saber quem aceitou e recusou o convite.

Com base nas tarefas anteriores os usuários responderam o seguinte questionário:

Para cada questão o usuário deverá escolher uma opção variando entre (1) Discordo totalmente e (5) Concordo totalmente

1. Foi fácil encontrar o caminho para executar as tarefas solicitadas

   1( ) 2( ) 3( ) 4( ) 5 ( )

2. Foi fácil realizar o *login* no sistema?

   1( ) 2( ) 3( ) 4( ) 5 ( )

3. Adicionar amigos é rápido e fácil

   1( ) 2( ) 3( ) 4( ) 5 ( )

4. A inserção dos dados na tela de Perfil se deu de forma tranquila

   1( ) 2( ) 3( ) 4( ) 5 ( )

5. A criação e gerenciamento de um jogo se deu de forma prática e intuitiva

   1( ) 2( ) 3( ) 4( ) 5 ( )

6. A tela de visualização de jogos é intuitiva

   1( ) 2( ) 3( ) 4( ) 5 ( )

7. Este aplicativo facilita a tarefa de organizar partidas de futebol entre amigos

   1( ) 2( ) 3( ) 4( ) 5 ( )

8.Você recomendaria este aplicativo para seus amigos

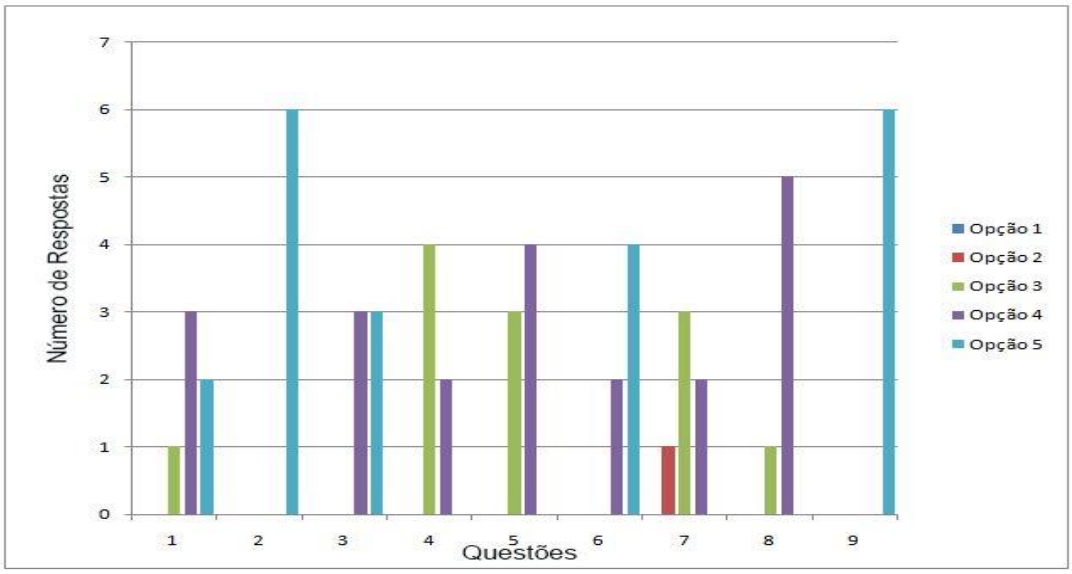
   1( ) 2( ) 3( ) 4( ) 5 ( )

9. O desempenho do aplicativo foi satisfatório

   1( ) 2( ) 3( ) 4( ) 5 ( )

O aplicativo foi testado por seis usuários e suas respostas foram compiladas resultando no gráfico da imagem a seguir.

Figura 25 – Gráfico de avalição do aplicativo



Fonte - Do Autor (2018)

Conforme os resultados apresentados no gráfico acima pode-se dizer que o aplicativo teve uma boa aceitação, obtendo em quase todas as questões respostas marcadas nas opções 3, 4 ou 5.

Na facilidade de encontrar o caminho para executar as tarefas solicitas (Questão 1) 50% dos usuários marcaram Concordo. Quanto a facilidade em realizar o login no aplicativo (Questão 2) todos marcaram Concordo totalmente. Inserir amigos foi fácil e rápido (Questão 3) as respostas foram divididas entre Concordo e Concordo totalmente. Preencher o formulário do Perfil foi tranquilo (Questão 4) quatro usuários escolheram a opção 3, acredito que pelas validações dos campos. A criação de jogos se deu de forma prática e intuitiva (Questão 5) ficou entre Concordo e Concordo totalmente. A Questão 6 referente a tela de visualização de jogos obteve quatro votos em Concordo totalmente. Sobre se o aplicativo facilita a organização de jogos (Questão 7) 50% dos usuários marcaram a opção 3, talvez pelo fato de o aplicativo não contar com a possibilidade de trocas de mensagens entre os usuários. Perguntados se recomendariam o aplicativo para os amigos (Questão 8) 5 usuários

marcaram Concordo. Sobre o desempenho do aplicativo (Questão 9) todos os usuários marcaram Concordo totalmente.

5.3 AVALIAÇÃO DO FRAMEWORK

Com a utilização do Xamarin foi possível desenvolver uma aplicação móvel disponibilizada para as duas principais plataformas móveis atuais (Android e IOS). A aplicação desenvolvida utiliza 100% de código compartilhado, ou seja, todo o desenvolvimento se deu no projeto que contém o código compartilhado entre as plataformas, deixando para os projetos específicos apenas alterações nos arquivos de configurações.

Durante o desenvolvimento da aplicação foi possível contar com a ajuda da comunidade de desenvolvedores Xamarin, o que facilita na resolução de problemas através dos fóruns de discussões. Além disso, Xamarin conta também com uma documentação contendo exemplos de como usar os recursos do framework, facilitando o entendimento.

Pequenos ajustes específicos de plataforma são necessários nos arquivos da interface do aplicativo, pois em alguns casos o resultado do layout gerado pelo framework pode não ser o desejado em uma das plataformas, como por exemplo, a cor da fonte nos formulários no IOS foi alterada para contrastar com a cor de fundo. Assim, aplicações em que há muitas diferenças entre as interfaces das plataformas, não é indicado o uso do Xamarin, pois nesse caso seriam necessários muitos ajustes.

A utilização do C# como linguagem de programação única agiliza e facilita o desenvolvimento. A utilização dos recursos do C# e do .NET possibilita que o que é feito em Java (Android) e em Objective-C (IOS) possa ser feito utilizando C#, permitindo o compartilhamento de código entre os projetos.

Portanto, o framework Xamarin se mostrou uma ótima ferramenta no desenvolvimento móvel multiplataforma, sendo uma alternativa ao desenvolvimento nativo. Dentre os seus pontos fortes destacam-se: Desenvolvimento multiplataforma; utilização de uma única linguagem de programação, compartilhamento de código e geração de aplicativos nativos.

6 CONSIDERAÇÕES FINAIS

O propósito do trabalho foi realizar um estudo do framework Xamarin e aplicá-lo no desenvolvimento de uma aplicação móvel multiplataforma, com intuito de verificar se sua utilização contorna o maior problema enfrentado no modo tradicional de desenvolvimento, que é necessidade de desenvolvimento de projetos distintos (Android, IOS, Windows Phone) para o mesmo aplicativo.

Como objetivo de estudo foi desenvolvido um aplicativo para as plataformas Android e IOS, integrado com uma base de dados remota. O objetivo do aplicativo é facilitar a organização de partidas de futebol entre amigos. O aplicativo tem formato de rede social, onde os usuários podem adicionar amigos, gerenciar partidas de futebol e participarem das partidas criadas por seus amigos. Para que os usuários de ambas as plataformas interajam entre si, a aplicação utiliza a mesma base de dados, disponível na plataforma de serviços em nuvem Azure.

Foram observados durante o desenvolvimento do aplicativos os pontos fortes e fracos do framework Xamarin. Destacam-se entre os pontos fortes o grande compartilhamento de código, diminuindo o tempo de desenvolvimento das aplicações multiplataforma; utilização do C# como linguagem de programação única no desenvolvimento, facilitando o desenvolvimento; documentação online e com exemplos de implementação. Como ponto franco pode ser citado o fato de o Xamarin ainda não possuir todos os controles de interface disponíveis no desenvolvimento nativo.

Como trabalho futuro podem ser citadas as funcionalidades a serem acrescentadas na aplicação, tais como: Sistema de notificações por *Push*, importante para que os usuários sejam notificados sobre os jogos criados; Sincronização *offline*, o que permitiria que os usuários visualizassem as informações de jogos passados mesmo estando *offline;* Chat em tempo real, proporcionando a comunicação entre os usuário de um jogo, por exemplo.

REFERÊNCIAS

AZURE. **Aplicativos Móveis**. Disponível em: <https://azure.microsoft.com/pt-br/services/app-service/mobile/>. Acesso em 21 de junho de 2018.

AZURE. **O que é o Azure?.** Disponível em: <https://azure.microsoft.com/pt-br/overview/what-is-azure/>. Acesso em 21 junho de 2018.

CAPELAS, Bruno. **Até o fim de 2017, Brasil terá um smartphone por habitante, diz FGV**. Disponível em: <http://link.estadao.com.br/noticias/gadget,ate-o-fim-de-2017-brasil-tera-um-smartphone-por-habitante-diz-pesquisa-da-fgv,70001744407>. Acesso em 21 de junho 2018.

CISCO. **Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper**. Disponível em: < https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html >. Acesso em 20 de junho de 2018.

DELANEY,Kalen. **Inside Microsoft SQL Server 2005: The Storage Engine**. Microsoft Press, 2006.

DOTNETPATTERN. **WPF MVVM Introduction**. Disponível em: < http://dotnetpattern.com/wpf-mvvm-introduction>. Acesso em 19 de junho de 2018.

GHATOL, Rohit; PATEL, Yogesh. **Beginning PhoneGap.** New York: Apress Media, 2012.

GONCALVES, **Antonio. Beginning Java EE 7. Apress**, 2013.

HAVLENA, Matouš. **Ionic thrives to redefine development of mobile hybrid applications.** Disponível em: http://www.havlena.net/en/mobile-development/ionic-thrives-to-redefine-development-of-mobile-hybrid-applications/. Acesso em 21 de junho de 2018.

IONIC. **All About Ionic.** Disponível em: < https://ionicframework.com/about>. Acesso em 21 de junho de 2018.

KANTARWORLDPANEL. **Smartphone OS sales Market share.** Disponível em: <https://www.kantarworldpanel.com/global/smartphone-os-market-share/>. Acesso em 12 de out. de 2017.

LECHETA, Ricardo R**. Desenvolvendo para iPhone e iPad**. Novatec Editora, 2012.

LECHETA, Ricardo R. **Google Android-4a Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. Novatec Editora, 2015.

MICROSOFT. **Documentação do SQL Server.** Disponível em: <https://docs.microsoft.com/pt-br/sql/sql-server/sql-server-technical-documentation>. Acesso em 15de nov. 2017.

MICROSOFT. **Introdução à linguagem C# e ao .NET Framework**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>. Acesso em 21 de junho de 2018.

MICROSOFT. **Stored Procedure Basics.** Disponível em:https://msdn.microsoft.com/en-us/library/ms191436.aspx. Acesso em 15 de nov. 2017

MICROSOFT. **Welcome to .NET**. Disponível em: <https://docs.microsoft.com/en-us/dotnet/welcome>. Acesso em 21 de junho de 2018.

MILANI, André. **Programando para iPhone e iPad: Aprenda a construir aplicativos para o iOS**. Novatec Editora, 2012.

NETMARKETSHARE. **Mobile/Tablet Operating System Market Share**. Disponível em: <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8& qpcustomd=1>. Acesso em 21 de junho de 2018.

PETZOLD, Charles. **Creating Mobile Apps with Xamarin.Forms: Cross-platform C# programming for iOS, Android, and Windows**. Microsoft Press, 2016.

PHONEGAP. **PhoneGap Documentation Overview**. Disponível em: <http://docs.phonegap.com/en/3.5.0/guide_overview_index.md.html#Overview>. Acesso em 21 de out. de 2017.

WARGO, John M. **PhoneGap Essentials: Building Cross-Platform Mobile Apps**. Addison-Wesley, 2012.

WODEHOUSE, **Carey. Xamarin vs. PhoneGap: Which Cross-Platform Mobile App Software is Right for You?.** Disponível em: < https://www.upwork.com/hiring/mobile/xamarin-vs-phonegap-which-cross-platform-mobile-app-software-is-right-for-you/>. Acesso em 21 de junho de 2018.

XAMARIN. **Deliver native Android, iOS, and Windows apps, using existing skills, teams, and code.** Disponível em: <https://www.xamarin.com/platform>. Acesso em: 14 de nov. de 2017.

XAMARIN. **Xamarin.Forms.**Disponível em: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/ >. Acesso em 21 de junho de 2018.

APÊNDICES

Apêndice A - Descrição dos Casos de Uso

Aqui serão detalhados os casos de uso, exibindo como os atores externos interagem com o sistema, fluxos das atividades, fluxos alternativos e condições para que cada caso de uso seja executado.

Caso de Uso Efetuar *Login*

Nome do Caso de Uso	Efetuar *Login*
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	WebService
Resumo	Este caso de uso descreve o processo de *login* de um usuário no aplicativo.
Pré-Condições	Conexão com a internet
Pós-Condições	*Login* efetuado ou não
Fluxo principal	
Ações do Ator	Ações do Sistema
1. Informar seu e-mail e senha	
	2. Consultar o banco de dados remoto por meio do Web Service.
	3. Caso o usuário ou senha estiverem errados é emitida uma mensagem de erro.
Restrições/Validações/Regras de Negócio	1. O usuário precisa possuir um cadastro no banco de dados remoto.
Fluxo Alternativo I – Usuário ainda não cadastrado	
Ações do Ator	Ações do Sistema
	1. Executar Caso de Uso Criar Usuário

Caso de Uso Criar Usuário

Nome do Caso de Uso	Criar usuário
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	WebService
Resumo	Este caso de uso descreve o processo de criação de um novo usuário na aplicação.
Pré-Condições	Conexão com a internet
Pós-Condições	Usuário criado
Fluxo principal	
Ações do Ator	Ações do Sistema
1. Solicitar a criação de um novo usuário.	
	2. Exibir uma tela de preenchimento de dados de cadastro.
3. Informar os dados necessários para o cadastro	
	4. Acessar o Web Service e registar o usuário no banco de dados remoto.
Restrições/Validações/Regras de Negócio	1. Não pode haver um usuário cadastrado no banco de dados remoto com o mesmo e-mail
	2. O nome do usuário, e-mail e senha são campos obrigatórios
	3. O *login* do usuário deve ser um e-mail no formato válido
	4. A senha deve possuir no mínimo oito caracteres
Estrutura dos Dados	
1. Nome do Usuário	
2. E-mail do Usuário (Utilizado para *login*)	
3. Senha do usuário	

Caso de Uso Visualizar Jogos

Nome do Caso de Uso	Visualizar Jogos
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	WebService
Resumo	Este caso de uso descreve o processo de visualização de Jogos em que o usuário está participando.
Pré-Condições	Conexão com a internet
Pós-Condições	Lista de Jogos é exibida
Fluxo principal	
Ações do Ator	Ações do Sistema
1. Solicitar a visualização de jogos pendentes.	
	2. Acessar o Web Service que executará a função de buscar no banco de dados remoto os jogos em que o usuário se encontra com status ativo.
	3. O Web Service retorna o resultado que é apresentado ao usuário
4. O Usuário tem acesso a lista de jogos podendo acessar seus detalhes.	
Restrições/Validações/Regras de Negócio	1. O usuário deve estar participando de algum jogo
	2. O usuário pode escolher entre jogos em aberto, encerrados ou por período de tempo
Fluxo Alternativo I – Usuário ainda não cadastrou operação financeira	
Ações do ator	Ações do Sistema

	1. O sistema exibe uma mensagem que nenhum jogo foi encontrado.

Caso de Uso Manter Partidas

Nome do Caso de Uso	Manter Partidas
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	Web Service
Resumo	Este caso de uso descreve as possíveis atividades de manutenção do cadastro de Partidas. Permite visualizar, cadastrar, alterar ou excluir partidas de futebol na aplicação.
Pré-Condições	Conexão com a internet, estar autenticado no aplicativo
Pós-Condições	A partida é criada, visualizada, alterada e excluída
Fluxo principal	
Ações do Ator	Ações do Sistema
1. Solicitar a criação de uma nova partida	
	2. Exibir a tela com o formulário de criação de uma partida.
3. O usuário preenche o formulário com os dados da partida a ser criada.	
	4. Chamar a validação de regras de negócio e utilizar o Web Service para persistir a nova partida no banco de dados remoto.

Restrições/Validações/Regras de Negócio	1. Os dados inseridos para a criação da partida devem atender às validações de requisitos aplicado ao formulário
Estrutura dos Dados	
1.Descrição da Partida	
2. Local da Partida	
3. Data	
Fluxo Alternativo I – Excluir Partida	
Ações do ator	Ações do Sistema
1. Selecionar a opção de Visualização de Partidas	
	2. Utilizar o Web Service para consultar o banco de dados e retornar as partidas que o usuário tem cadastradas.
	4. Exibir as partidas encontradas
5. Selecionar a partida que deseja excluir.	
	6. Utilizar o Web Service para realizar a operação de exclusão no banco de dados remoto.
Fluxo Alternativo II – Alterar Partida	
Ações do ator	Ações do Sistema
1. Selecionar a opção de Visualização de Partidas	
	2. Utilizar o Web Service para consultar o banco de dados e retornar as partidas que o usuário tem cadastradas.
	4. Exibir as partidas encontradas

5. Selecionar a partida que deseja alterar.	
	6. Utilizar o Web Service para realizar a operação de alteração no banco de dados remoto.
Fluxo Alternativo III – Visualizar Partidas	
Ações do ator	Ações do Sistema
1. Selecionar a opção de Visualização de Partidas	
	2. Utilizar o Web Service para consultar o banco de dados e retornar as partidas que o usuário tem cadastradas.
	4. Exibir as partidas encontradas
5. Selecionar a partida que deseja visualizar.	
	6. Exibe a tela com os detalhes da partida.

Caso de Uso Manter Amigos

Nome do Caso de Uso	Manter Amigos
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	Web Service
Resumo	Este caso de uso descreve as possíveis atividades de manutenção do cadastro de amigos. Permite visualizar, cadastrar, alterar ou excluir amigos na aplicação.
Pré-Condições	Conexão com a internet, estar autenticado no aplicativo, possuir o e-

	mail do amigo que será adicionado, o amigo deve possuir uma conta no aplicativo
Pós-Condições	A amigo é adicionado e excluído da lista de contatos do usuário
Fluxo principal	
Ações do Ator	Ações do Sistema
1. Solicitar a adição de um novo amigo	
	2. Exibir a tela com o formulário de adição de amigos
3. O usuário preenche o formulário com o e-mail do amigo a ser adicionado	
	4. Chamar a validação de regras de negócio e utilizar o Web Service para verificar se o e-mail está cadastrado no sistema.
Restrições/Validações/Regras de Negócio	1. Os dados inseridos para a criação da partida devem atender às validações de requisitos aplicado ao formulário
Fluxo Alternativo I – O amigo não possui uma conta	
Ações do ator	Ações do Sistema
	O Sistema apresenta uma mensagem de erro.
Fluxo Alternativo II – Remover Amigo	
Ações do ator	Ações do Sistema
1. Selecionar a opção de Visualização de Amigos	
	2. Utilizar o Web Service para consultar o banco de dados e retornar

	os amigos que o usuário tem cadastradas.
	4. Exibir os amigos encontradas
5. Selecionar o amigo que deseja excluir dos contatos.	
	6. Utilizar o Web Service para realizar a operação de exclusão no banco de dados remoto.
Fluxo Alternativo III – Visualizar Amigos	
Ações do ator	Ações do Sistema
1. Selecionar a opção de Visualização de Amigos	
	2. Utilizar o Web Service para consultar o banco de dados e retornar os amigos que o usuário tem cadastradas.
	4. Exibir os amigos encontradas
5. Selecionar ao amigo que deseja visualizar.	
	6. Exibe a tela do perfil do amigo.

Caso de Uso Efetuar Convites

Nome do Caso de Uso	Efetuar Convites
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	Web Service
Resumo	Este caso de uso descreve o processo de envio de convites para participar de determinada partida

Pré-Condições	Conexão com a internet, estar autenticado no aplicativo, possuir amigos previamente adicionados, possuir a partida previamente criada.
Pós-Condições	O convite é enviado
Fluxo principal	
Ações do Ator	Ações do Sistema
1. Na tela da partida, selecionar *convidar amigos*	
	2. Utilizar o Web Service para verificar no banco de dados os amigos do usuário
	3. retornar a lista de amigos encontrados
4. Selecionar os amigos que receberão o convite do jogo	
	5. Utilizar o Web Service para enviar o convite do jogo para os amigos selecionados pelo usuário.
Restrições/Validações/Regras de Negócio	
Fluxo Alternativo I – O amigo está *offline* no momento do envio	
Ações do ator	Ações do Sistema
	Como o banco de dados do aplicativo é remoto, quando o usuário estiver online o aplicativo será atualizado.