UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
Departamento de Computação
Bacharelado em Sistemas de Informação

GRAPPHIA: INTEGRANDO APLICATIVO COM O REALTIME DATABASE DA PLATAFORMA FIREBASE

Pedro Henrique Cerqueira Estanislau

Diamantina
2018

Pedro Henrique Cerqueira Estanislau

GRAPPHIA: INTEGRANDO APLICATIVO COM O REALTIME DATABASE DA PLATAFORMA FIREBASE

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Departamento de Computação, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Luciana Pereira de Assis

Diamantina
2018

Pedro Henrique Cerqueira Estanislau

GRAPPHIA: INTEGRANDO APLICATIVO COM O REALTIME DATABASE DA PLATAFORMA FIREBASE

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Departamento de Computação, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Profa. Dra. Luciana Pereira de Assis

Data de aprovação 02/08/18.

Profa. Dra. Luciana Pereira de Assis
Departamento de Computação - UFVJM

Prof. Dr. Alessandro Vivas Andrade
Departamento de Computação - UFVJM

Profa. Dra. Josiane Magalhães Teixeira
Departamento De Matemática e Estatística - UFVJM

Diamantina
2018

Dedico este trabalho aos meus pais que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa da minha vida.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus.

Agradeço aos meus pais, Ana Lúcia e Valter, pelo amor, incentivo e apoio incondicional, que me possibilitaram a oportunidade de estudar fora. Agradeço ao Gerson pelos conselhos sábios, incentivo e apoio.

Agradeço à minha orientadora Profa. Dra. Luciana Pereira de Assis pela orientação, apoio, confiança e paciência durante a realização deste trabalho.

Devo expressar a minha gratidão à uma pessoa que conheci este ano e vem mudando aos poucos a minha vida, a Izabela. Sem o seu contínuo apoio e incentivo, com certeza, este trabalho não seria concluído.

Agradeço aos meus amigos da família "БОЛАДОС" pelas partidas de futebol (sdds), brincadeiras, festas de aniversário e acolhimento.

Agradeço profundamente ao Paulo por sempre estar ao meu lado, me incentivando, me tolerando e sempre me orientado a por os pés no chão. E agradeço à família Cordeiro Repolês/Khouri por me acolherem como um membro. A conclusão da parte de programação deste trabalho, e do curso, teria sido ainda mais difícil se não fosse pelo apoio e amizade fornecido por Lucas "Tevez". Por isto meu imenso agradecimento. Agradeço à Yohana por me apoiar emocionalmente por grande parte da minha graduação e pelo carinho. Agradeço ao Gabriel "Rufs" pelo carinho e companheirismo.

Agradeço imensamente aos meus amigos de Diamantina, especialmente o Luíz "Lulu", o Pedro "Psiu" e o Pedro Orlando, pela paciência, companheirismo, irmandade e por sempre me ajudarem com o que era necessário. Agradeço à meninas da República Amsterdam por me acolherem e pelas conversas aleatórias.

Agradeço ao meu primo Wagner por me acolher, me ajudar a estabelecer em Diamantina e pela companhia aos jogos sofridos do Galo. Agradeço à Cláudia, minha primeira "Chefe", por todo amor, incentivo, apoio e sempre me ajudando dentro do possível.

Agradeço aos membros e ex-membros do projeto Grapphia e aos discentes do Departamento de Computação pelas ajudas. Agradeço a A.A.A.S.I. The Bug pelas festas e por me proporcionar novas experiências.

I like to thank Tia Loca and Luiza for always giving me uncondtional help, and many thanks to Miss Hannah for always helping me with whatever was necessary in my hard times and with whatever was troubling me. Love you all! I would also like to thank the Prud'-hommes for the many years of friendship and uncoditonal love. Go Habs!

Agradeço à todos os meus professores por me proporcionarem o conhecimento não apenas racional, mas, a manifestação do caráter e afetividade da educação, no processo de formação profissional. E a todos que direta ou indiretamente, fizeram parte da minha formação, o meu muito obrigado!

A persistência é o caminho do êxito.
-CHARLES CHAPLIN

RESUMO

O Grapphia é um jogo digital educacional que auxilia no ensino da ortografia da Língua Portuguesa (LP). Atualmente, análise de dados do aplicativo Grapphia, é feita através da extração manual dos dados de cada dispositivo móvel dos usuários. Esse procedimento rudimentar se deve ao fato do sistema atual de gerenciamento de banco de dados não possuir conexão com um servidor, o que dificulta a obtenção e análise dos dados. O objetivo deste trabalho é apresentar uma solução para integrar o aplicativo, desenvolvido pelo software Unity, com um banco de dados não relacional (NoSQL) na nuvem, denominado Firebase Realtime Database. O uso de um banco de dados não relacional é devido a sua flexibilidade e alta escalabilidade. Assim sendo, este trabalho descreve e apresenta uma ferramenta para auxiliar na integração de aplicativos móveis e banco de dados não relacional, que permite o armazenamento e manipulação dos dados em nuvem. Durante a implementação do sistema, o Firebase Realtime Database demonstrou sua efetividade e facilidade para armazenar e manipular os dados do usuário do aplicativo.

Palavras-chave: Jogo Digitais. Jogos Digitais Educacionais. Banco de dados não relacional. NoSQL. Firebase. Firebase Realtime Database. Unity.

ABSTRACT

Grapphia is a digital educational game that assists in teaching the ortography of the Portuguese Language. Currently, data analysis of the Grapphia application is done by manually extracting data from each user's mobile device. This rudimentary procedure is due to the fact that the current database management system does not have a connection to a server, which makes it difficult to obtain and analyze the data. The objective of this work is to present a solution to integrate the application, developed by Unity software, with a non-relational database (NoSQL) in the cloud, called Firebase Realtime Database. The use of a non-relational database is due to its flexibility and high scalability. Thus, this paper describes and presents a tool to assist in the integration of mobile applications and non-relational database, which allows the storage and manipulation of the data in the cloud. During system implementation, the Firebase Realtime Database demonstrated its effectiveness and ease of storing and manipulating the application's user data.

Keywords: Digital Game. Educational Digital Games. Non-relational database. NoSQL. Firebase. Firebase Realtime Database. Unity.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - Application Programming Interface

BaaS - Backend as a Service

JSON - JavaScript Object Notation

NoSQL - Non SQL

SDK - Software Development Kit

SGBD - Sistema de Gerenciamento de Banco de Dados

UFVJM - Universidade Federal dos Vales do Jequitinhonha e Mucuri

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho propõe integrar o aplicativo Grapphia (ASSIS *et al.*, 2017), criado através da ferramenta Unity (UNITY, 2017), com um banco de dados baseado em nuvem denominado Firebase Realtime Database (FIREBASE, 2018b), que armazena e sincroniza todos os dados em tempo real e permite funcionalidade offline. Essa integração tem como objetivo facilitar a agrupação e a análise dos dados após o uso do aplicativo Grapphia. Este capítulo tem por finalidade apresentar o problema situado no contexto, evidenciando a motivação e os objetivos propostos.

Grapphia é um aplicativo digital, para dispositivos móveis, que visa auxiliar no ensino da Língua Portuguesa, em especial a ortografia. O aplicativo é direcionado para crianças alfabetizadas de 8 a 10 anos, que podem apresentar dúvidas quanto ao nível ortográfico. O objetivo do sistema é promover a interação entre o ensino da ortografia e o uso de recursos didáticos tecnológicos (ASSIS *et al.*, 2017). O ensino de ortografia no aplicativo se dá através de atividades de complementação de palavras, a partir da reflexão e escolha entre duas letras, apresentadas na tela de um dispositivo móvel.

O desenvolvimento do aplicativo exigiu a criação de um banco de dados com palavras que são frequentes no vocabulário infantil e que apresentam uma possível dificuldade ortográfica, devido ao fato de possuir mais de uma representação gráfica para um mesmo som (ASSIS *et al.*, 2017). O seu desenvolvimento se dá através da utilização do *game engine* (motor de jogo[1]) e do ambiente de desenvolvimento integrado, o software Unity (UNITY, 2017). Essa ferramenta une as animações, os aúdios, os *scripts* e o uso das redes, gerando um aplicativo. Posteriormente, é escolhida a plataforma no qual o aplicativo será gerado, dentre elas: Android, iOS, PS3, PS4, Mac, Linux, Windows, etc.

O banco de dados relacional utilizado para a criação da aplicação foi o SQLite, uma ferramenta de pequeno porte. Esta ferramenta é *open source*[2] e *serverless*[3]. Como esta ferramenta não possui comunicação com uma rede externa, a extração dos dados de cada dispositivo móvel é feita manualmente, através da remoção do arquivo interno e depois este arquivo é transferido para um computador para a análise dos dados. Devido ao fato desse processo ser operoso e lento, foi proposto a integração do aplicativo com um servidor.

Como o banco de dados SQLite é *serverless*, não seria possível implementar esta proposta. Outra proposta, seria a utilização de outros bancos de dados *open source*, como por exemplo, o MySQL. Conforme a maior utilização do Grapphia, o volume de dados cresce rapidamente e a performance de um banco de dados relacional se degrada (NAYAK; PORIY;

---

1 O software que fornece aos criadores de jogos, o conjunto necessário de recursos, para criar jogos de maneira rápida e eficiente.

2 Código fonte disponibilizado com licença de código aberto no qual o direito autoral permite estudar, modificar e distribuir o software de graça para qualquer um.

3 Não tem comunicação com qualquer servidor ou rede externa.

POOJARY, 2013). O melhor caminho então, seria um banco de dados não relacional, devido a sua alta flexibilidade[4] e fácil escalabilidade[5] (NAYAK; PORIY; POOJARY, 2013). Assim sendo, a proposta deste trabalho é a integração do Grapphia, desenvolvido pelo Unity com o Firebase Realtime Database.

Firebase é uma plataforma do tipo *Backend as a Service* (BaaS)(FIREBASE, 2018b). Este modelo, BaaS, de acordo com Sareen (2013), é um meio de facilitar a integração de aplicativos com um armazenamento em nuvem de *back-end*[6], e, ao mesmo tempo, fornecer serviços de autenticação do usuário, notificações de *push* e integração com as redes sociais.

Um dos serviços oferecidos pela plataforma Firebase é o Firebase Realtime Database, um banco de dados não relacional NoSQL, hospedado na nuvem. Todos os dados são armazenados e sincronizados em tempo real com todos os clientes associados. Usando as ferramentas do Firebase e independente da plataforma em qual a aplicação foi criada, atualizações feitas no banco de dados são compartilhadas com todos os clientes, em tempo real (FIREBASE, 2018d).

O Firebase permite uma integração com a ferramenta Unity, através do uso dos pacotes do Firebase para o Unity e adicionado ao projeto do Unity (FIREBASE, 2018g). Um desses pacotes é o banco de dados Firebase Realtime Database. Através de alguns procedimentos básicos de instalação e algumas linhas de código, é possível fazer a inserção do banco de dados do Firebase ao projeto do software Unity. Uma das características do Realtime Dabatase é a continuação do seu funcionamento mesmo que o aplicativo esteja offline e atualizando uma versão interna. Após uma conexão com a rede, a versão interna do cliente é sincronizada com a versão do servidor remoto. (FIREBASE, 2018e).

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho consiste na integração do aplicativo Grapphia, desenvolvido com uso do software Unity com o Firebase Realtime Database, para facilitar o processo de recuperação e análise dos dados dos usuários do aplicativo Grapphia.

1.1.2 Objetivos Específicos

- Estudar as ferramentas existentes que possibilitam a integração de aplicativo a um servidor externo;
- Estudar o banco de dados NoSQL;
- Facilitar a análise dos dados gerados pelo aplicativo Grapphia;

[4] Não tem restrições em estruturar o banco de dados.
[5] Capacidade de um sistema expandir sem que o seu desempenho piore.
[6] A parte de um sistema de computador ou parte de um software, onde os dados são armazenados ou processados em vez das partes que são usadas pelo usuário (THESAURUS, 2018a).

1.2 Estrutura do Trabalho

Este trabalho possui 5 capítulos, o Capítulo 1, o atual capítulo, estabeleceu os atuais problemas referentes ao aplicativo Grapphia, possuir um banco de dados sem conexão com um servidor e a dificuldade de análise dos dados.

No capítulo 2 são apresentados uma revisão dos conceitos básicos para entendimento do trabalho, dentre eles: os dispositivos móveis e a plataforma Android. Além disso, o referido capítulo apresenta o histórico do aplicativo Grapphia que é uma evolução do jogo digital Laça Palavras. Posteriormente, as principais ferramentas utilizadas para desenvolvimento do Grapphia são apresentadas (Unity e SQLite). O capítulo também apresenta uma comparação entre banco de dados relacional e não relacional, em específico o Firebase, que é a proposta deste trabalho para integração do aplicativo Grapphia a um serviço de computação em nuvem. Por fim, é apresentado o software Exploratory, responsável por analisar os dados extraidos do banco de dados.

O Capítulo 3 apresenta os métodos para que a integração entre o Firebase e aplicativo móvel, desenvolvido com o software Unity, seja possível. Dentre esses métodos estão: a instalação do SDK do Firebase para Unity, inserção, remoção, estrutura e extração dos dados.

No Capítulo 4 são apresentados os testes feitos para mostrar as inserções, remoções e a extração dos dados do banco de dados e posteriormente a análise dos mesmos.

O Capítulo 5 apresenta as conclusões e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo tem como objetivo apresentar conceitos básicos para o entendimento do trabalho. A seção 2.1 apresenta os conceitos sobre dispositivos móveis e a subseção 2.1.1 sobre a plataforma Android. As subseções 2.2.1 e 2.2.2 apresentam a evolução do jogo digital *Laça Palavras* para o Grapphia. A seção 2.3 apresenta o software Unity e suas subseções 2.3.1 e 2.3.2 sobre os *scripts* e a linguagem utilizada no software Unity, C#, respectivamente. A seção 2.4 apresenta a ferramenta SQLite. A seção 2.5 aborda a diferença entre um banco de dados relacional e um banco de dados não relacional. A seção 2.6 e sua subseção 2.6.1 abordam sobre a plataforma Firebase e o Firebase Realtime Database, respectivamente. Por último, a seção 2.7 apresenta o software para análise de dados, Exploratory.

2.1 Dispositivos Móveis

Os dispositivos móveis são dispositivos de comunicação de longo alcance, portáteis e sem fio. Alguns anos atrás, quando os telefones celulares não eram tão comuns, os dispositivos e os custos de comunicação eram caros para os usuários. Porém, nos últimos anos, à medida que o uso de dispositivos aumentou, seu custo diminuiu consideravelmente e esse fator ajudou a torná-los disponíveis para todos. Os telefones celulares atualmente são baratos, fáceis de usar, confortáveis e equipados com uma variedade de recursos.

Dispositivos móveis estão hoje nas mãos de usuários de todas as idades. Atualmente, as pessoas estão equipadas com os mais recentes modelos, e, constantemente, surgem no mercado novos modelos de celulares a fim de substituir os antigos e atrair os olhares dos usuários. Os recursos comumente fornecidos nos aparelhos mais recentes são: GPS, TV digital, câmera fotográfica, Wi-Fi, lanterna, calculadora, rádio, gravador, etc. Devido ao volume de recursos armazenados em um pequeno aparelho os torna, para muitos usuários, um item indispensável no cotidiano.

Não há dúvida de que os dispositivos móveis tornaram a vida mais fácil e confortável para todos. Os usuários estão constantemente conectados a seus familiares, amigos e outros conhecidos. De acordo com GOLLNER (2014, p. 1), a posse de um dispositivo móvel “representa praticidade, conexão a qualquer hora e em qualquer lugar e, ainda, confere ao seu portador *status* e uma identidade que reflete um sujeito moderno e conectado ao mundo.”.

De acordo com o IBGE (2016), existe um telefone móvel celular em 92,6% dos domicílios do país. Dentre a amostra de 48.070 domicílios no Brasil, 97,2% tinham um telefone móvel celular com acesso à Internet. Ainda segundo a mesma pesquisa, dados apontam que o celular foi o equipamento mais utilizado para fins de uso da internet, por indivíduos dentro da faixa etária de 10 anos ou mais, durante um período de 3 meses. Das 116.073 mil pessoas pesquisadas, 94,6% (109.818 mil pessoas) utilizaram o aparelho celular como meio para acesso

a internet.

No mercado existem vários tipos de sistemas operacionais para dispositivos móveis, por exemplo: Android, iOS, Windows Phone, Blackberry OS, dentre outros. O Grapphia foi desenvolvido somente para a plataforma Android, através do software Unity.

A subseção 2.1.1 descreve o sistema operacional Android. Este sistema operacional é frequentemente encontrado nos dispositivos móveis e para o qual o Grapphia foi desenvolvido.

2.1.1 Android

Em parceira com a *Open Handset Alliance* (OHA), a Google desenvolveu a plataforma Android. A OHA é um grupo formado pelos gigantes do mercado das industrias de dispositivos móveis, tais como: Motolora, HTC, LG, Samsung, ASUS, entre outras. Assim, sua estrategia é conseguir alcançar o máximo de usuários.

A plataforma Android tem aumentado o interesse da indústria de dispositivos móveis, devido a dois aspectos principais: sua natureza de código aberto[1] e seu modelo de arquitetura (GANDHEWAR; SHEIKH, 2010). Por ser código aberto, o Android permite aos desenvolvedores compreender melhor o seu funcionamento e seus recursos, a correção de *bugs* e melhorias em novas funcionalidades, para atender as tendências do mercado (LECHETA, 2013).

Como o seu modelo de arquitetura é baseado no Linux, de acordo com Gandhewar e Sheikh (2010, p. 12, tradução nossa)[2], isto "adiciona o uso do Linux à indústria móvel, possibilitando aproveitar o conhecimento e os recursos oferecidos pelo Linux". Qualquer usuário pode desenvolver um aplicativo para a plataforma Android, facilitando a obtenção de mais usuários. Para iniciar este desenvolvimentos de aplicações é necessário instalar o Android SDK, que contém um conjunto de ferramentas de desenvolvimento e emulador. Sua linguagem principal de programação é Java. A disponibilização para os usuários se dá através, principalmente, pela loja virtual do Android, a *Play Store*.

Com o fácil acesso e constantes atualizações, o Android é a plataforma mais utilizada nos dispositivos móveis. A empresa de consultoria Gartner constatou que o Android está presente em 85,9% das vendas de dispositivos móveis em todo o mundo, só no primeiro trimestre de 2018 (GARTNER, 2018), enquanto o iOS está presente em 14,1% das vendas.

2.2 Jogos Digitais

O presente capítulo apresenta um pouco da história e objetivo dos jogos digitais desenvolvidos pelos discentes do Departamento de Computação da UFVJM. A subseção 2.2.1 apresenta o jogo digital *Laça Palavras*, produto de trabalho de conclusão de curso de dois

[1] Código fonte disponibilizado com licença de código aberto no qual o direito autoral permite estudar, modificar e distribuir o software de graça para qualquer um.

[2] Adds the use of Linux to the mobile industry, allowing to take advantage of the knowledge and features offered by Linux.

discentes. A subseção 2.2.2 apresenta o jogo digital Grapphia, uma evolução do jogo *Laça Palavras*.

2.2.1 *Laça Palavras*

O Grapphia se origina dos autores Santos (2016) e Leal (2016), com a ideia de apresentar um jogo digital ortográfico para auxiliar a aprendizagem de crianças. Em conjunto, os dois autores criaram o aplicativo *Laça Palavras* para dispositivos Android, usando o software Unity. A tela inicial deste aplicativo é mostrada na Figura 2.1. De acordo com Leal (2016, p. 93), o aplicativo "tem como objetivo auxílio nos treinos ortográficos de palavras da língua portuguesa que possuem a característica da concorrência, que consiste em que duas letras estão aptas a representar o mesmo som, no mesmo contexto".

Figura 2.1 – Tela Inicial do aplicativo *Laça Palavras*

Fonte: Leal (2016)

A fim de "balancear dinamicamente a dificuldade do jogo"(SANTOS, 2016, p. 33), cada autor usou uma técnica de aprendizagem por reforço. O Santos (2016) usou SARSA e o Leal (2016) usou Q-Learning. Aprendizagem de reforço, de acordo com Kaelbling, Littman e Moore (1996, p. 237, tradução nossa)[3] é "o problema enfrentado por um agente que deve aprender o comportamento por meio de interações de tentativa e erro com um ambiente dinâmico".

Segundo Watkins e Dayan (1992, p. 55, tradução nossa)[4], Q-Learning é quando "um agente toma uma ação em um determinado estado, e avalia as suas consequências em termos de recompensa imediata ou pena que recebe e a sua estimativa do valor do estado para o qual ele é levado". O método SARSA (*State-Action-Reward-State-Action*), que significa Estado-Ação-Recompensa-Estado-Ação é uma modificação do Q-Learning, que utiliza um mecanismo de iteração de política.

[3] Reinforcement learning is the proble faced by an agent that must learn behavior through trial-and-erros interactions with a dynamic environment.

[4] An agent tries an action at a particular state, and evaluates its consequences in terms of the immediate reward or penalty it receives and its estimate of the value of the state to which it is taken.

Visando uma melhoria estética, aumentar os cenários de jogabilidade e simplificação de código, o *Laça Palavras* se transformou no Grapphia.

2.2.2 Grapphia

Na procura de novos métodos de aprendizagem, o Grapphia foi desenvolvido para suprir a falta de aplicativos no auxílio de ensino da ortografia da Língua Portuguesa (LP) (ASSIS *et al.*, 2017). A ortografia é parte integrante do processo de aprendizagem de Língua Portuguesa, entretanto, seu conteúdo apresenta grande dificuldade de assimilação por parte dos educandos, por conter uma categoria de palavras que apresentam o mesmo som porém com duas escritas diferentes, como apresentado por Lemle (1988):

> " É o caso da letra S e da letra Z, que são usadas, ora uma, ora outra, para representar o mesmo som [z] entre duas vogais. Temos MESA, mas também REZA. Temos AZAR, mas também CASAR. Do mesmo tipo é a rivalidade entre C-Ç e SS, usados entre vogais para representar aquilo que é sempre o mesmo som [s]: POSSEIRO e ROCEIRO, ASSENTO e ACENTO, PASSO e LAÇO, CAÇADO e CASSADO. Da mesma maneira, o CH e o X competem na representação da fricativa palatal surda (taxa, racha) e o G e o J rivalizam no privilégio de representar a fricativa palatal sonora (jeito, gente, sujeira, bagageiro)"(LEMLE, 1988, p. 23).

Visando solucionar essa dificuldade na aprendizagem dessa categoria de palavras, os autores Assis *et al.* (2017) basearam-se na metodologia de ensino proposta por Alvarenga (1995), em que o processo de aprendizagem de palavras irregulares se dá através do exercício de memorização. Para tanto, esse processo ocorre de maneira individual.

Baseando-se em tal metodologia ativa, a utilização de jogos didáticos como ferramenta no processo de aprendizagem proporciona a apropriação do conteúdo de forma mais satisfatória, de acordo com o dicionário e de forma lúdica, o que torna a aprendizagem mais atrativa.

Conforme apontado por Gaspar, Oliveira e Oliveira (2015), o uso de dispositivos móveis no processo de aprendizagem da LP ainda se encontra com diminutos estudos e divulgações a respeito. Além disso, a abordagem do conteúdo de ortografia com o uso de dispositivos móveis ainda não apresentam publicações.

O aplicativo Grapphia, assim como o *Laça Palavras*, foi desenvolvido com o uso do software Unity através da junção de animações, imagens, *scprits* e aúdios. A Tela Inicial do Grapphia é mostrada na Figura 2.2. Comparando a Tela Inicial do *Laça Palavras* e do Grapphia, nota-se a evolução na estética do jogo. As figuras 2.3 e 2.4 ilustram as telas de cadastro dos aplicativos *Laça Palavras* e Grapphia, respectivamente.

A fim de incentivar os usuários a se interessarem pelo jogo, os desenvolvedores do Grapphia, dividiram os grupos de palavras em livros com determinadas temáticas. Cada livro

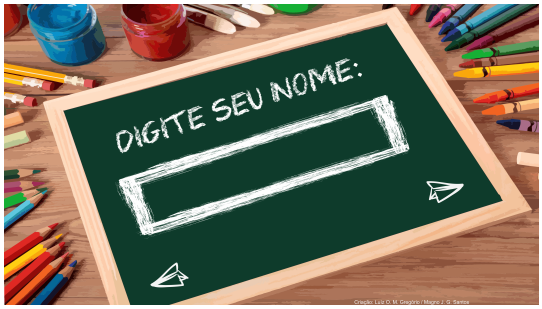
Figura 2.2 – Tela Inicial do aplicativo Grapphia

Fonte: Assis *et al.* (2017)

Figura 2.3 – Tela cadastrar usuário - *Laça Palavras*

Fonte: Leal (2016)

Figura 2.4 – Tela cadastrar usuário - Grapphia



Fonte: Assis *et al.* (2017)

apresenta uma história que contêm as palavras dos grupos. A figura 2.5 expõe ao usuário os livros que podem ser acessados. Estes livros são: “A Fazenda”, “Passeio no Zoo” e “Sol de Verão”.

O livro “A Fazenda” contém o grupo de palavras S e Z. A figura 2.6 ilustra a capa do livro “A Fazenda”. Para acessar a história do livro “A Fazenda” o usuário necessita trocar as páginas arrastando o dedo do canto inferior direito ao canto inferior esquerdo. Após a leitura da história, o usuário toca no botão “Jogar” e escolhe o personagem que deseja utilizar. A figura 2.7 mostra ao usuário a primeira palavra escolhida aleatoriamente, *Perigosa*. Nota-se que a letra *s* da palavra é ocultada a fim de que o usuário possa pensar na ortografia da palavra. Para dar continuidade ao jogo, o usuário precisa optar por uma das letras apresentadas na tela para completar a palavra. Caso o usuário acerte, sua pontuação é incrementada.

O livro “Sol de Verão”, mostrado na figura 2.8, contém o grupo de palavras L e U. O procedimento para acessar a história do livro “Sol de Verão” é a mesma do livro “A Fazenda”. A figura 2.9 mostra a palavra “Almanaque” do livro “Sol de Verão.”

O livro “Passeio no Zoo”, demonstrado na figura 2.10, contém o grupo de palavras

Figura 2.5 – Estante de Livros do Grapphia

Fonte: Assis *et al.* (2017)

Figura 2.6 – Capa do livro *A Fazenda*

Fonte: Assis *et al.* (2017)

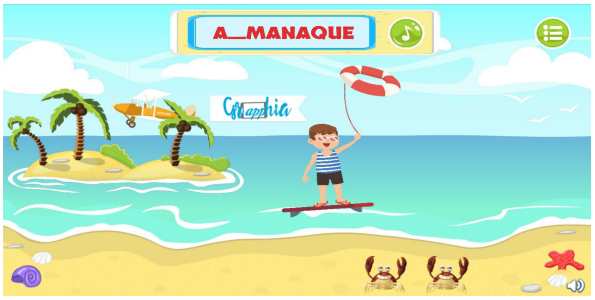
Figura 2.7 – Palavra do livro *A Fazenda*

Fonte: Assis *et al.* (2017)

Figura 2.8 – Capa do Livro Sol de Verão

Fonte: Assis *et al.* (2017)

Figura 2.9 – Palavra do livro Sol de Verão



Fonte: Assis *et al.* (2017)

G e J. Este livro ainda está em fase de construção.

Figura 2.10 – Capa do livro Passeio no Zoo

Fonte: Assis *et al.* (2017)

Como dito anteriormente, o Grapphia foi desenvolvido com o uso do software Unity e, portanto, ela será apresentado na próxima seção.

2.3 Software Unity

O mercado de criação de jogos móveis vem crescendo, desde os primeiros lançamentos de aplicativos de dispositivos móveis. Com a nova tecnologia lançada no mercado, a demanda por jogos com melhor qualidade de produção, jogabilidade, narrativa e executivo se elevam cada vez mais. Para desenvolver esses jogos, alguém com uma ótima ideia, precisa encontrar uma ótima ferramenta, para atender às necessidades do mercado, de maneira mais rápida e fácil.

O software Unity (comumente conhecido como Unity3D) é um *game engine* (motor de jogo) e um ambiente de desenvolvimento integrado (IDE) para criar mídia interativa, principalmente, videogames. É um software comercialmente distribuído pela empresa Unity Techonologies. "Motor de jogo é o software que fornece aos criadores de jogos, o conjunto necessário de recursos, para criar jogos de maneira rápida e eficiente" (UNITY, 2018a).

O software Unity tem sido uma ferramenta popular entre desenvolvedores independentes e equipes pequenas nos últimos anos, por ser fácil de aprender a utilizar. Devido a essa popularidade, existem uma quantidade enorme de tutoriais e documentações espalhados pela internet, inclusive no site do Unity. Além de tutoriais, existem cursos profissionalizantes que capacitam desenvolvedores com esta ferramenta.

Devido a utilização de linguagens de script, C# e Javascript, os jogos desenvolvidos nesta ferramenta podem ser gerados para várias plataformas presentes no mercado, dentre elas:

Android, iOS, PS3, PS4, Mac, Linux, Windows, Nitendo, Xbox One e Steam. Além disso, os jogos podem ser executados na Web, instalando um plug-in. A ferramenta em si pode ser instalada nos Sistemas Operacionais Windows, Mac OS X e Linux.

Outra característica do software Unity é a utilização de recursos de jogos e objetos em forma de pacotes desenvolvidos por outras ferramentas e outros projetos. Isto melhora bastante a eficiência dos jogos (XIE, 2012).

Existem outros *game engines* no mercado que desenvolvem jogos digitais, tais como: CryEngine, Cocos2D, Cube, MonoGame, id Tech 3, dentre outras. A versão do software Unity utilizado neste trabalho é a 5.3.

2.3.1 *Scripts*

Scripts são listas de comandos que automatizam processos em linguagens de programação. No software Unity, para desenvolver os próprios recursos de jogabilidade, é necessário criar componentes característicos, através do uso de *scripts*.

De acordo com Unity (2018b), o uso de *scripts* é essencial em qualquer tipo de jogo, desde os jogos simples aos mais complexos. A cada interação do jogador é necessário um *script* para que eventos se desencadeiem e os objetivos sejam cumpridos. *Script* é usado para controlar o comportamento físico dos objetos nos jogos e gerar efeitos gráficos. Para movimentar uma simples bola, por exemplo, é necessário a criação de um *script* para que a bola mova sozinha ou de acordo com a interação do usuário.

No software Unity a classe base da qual deriva todos os *scripts* é *MonoBehaviour*. Quando se usa a linguaguem C#, é necessário derivar-se desta classe. Todos os *scripts* precisam apresentar método *Start()*, que é chamado no início da execução do *script*.

2.3.2 C#

O C# (pronuncia-se C Sharp) é uma linguagem de programação orientada a objeto desenvolvida pela Microsoft como parte da plataforma .NET. O foco desta plataforma é o desenvolvimento de Serviços WEB XML, ou também conhecido como *Web Services*. O princípio do *Web Services* é "permitir que as aplicações, sejam elas da Web ou Desktop, ou ainda middleware, se comuniquem e troquem dados de forma simples e transparente, independente do sistema operacional ou da linguagem de programação"(LIMA; REIS, 2002, p. 3).

Esta plataforma é composta de várias ferramentas que apoiam desenvolvimentos de aplicações. Estas ferramentas são compostas por linguagens de programação, compiladores, modelo de objetos e outros. Dentre as linguagens de programação está a linguagem C#. Segundo Lima e Reis (2002, p. 3-4), esta linguagem "surge como uma linguagem simples, robusta, orientada a objetos, fortemente tipada e altamente escalável". Toda a documentação sobre esta linguagem está disponível no *site* da Microsoft [5].

[5] Documentação da linguagem C# disponível no link https://docs.microsoft.com/pt-br/dotnet/csharp/

De acordo com Lima e Reis (2002), as seguintes características são essenciais para a linguagem C#:

- Simplicidade;
- Completamente orientada a objetos: qualquer variável necessita fazer parte de uma classe;
- Fortemente tipada: evita erros por manipulação imprópria do desenvolvedor;
- Flexibilidade: pode-se usar ponteiros, caso o desenvolvedor necessite, mas com o custo de desenvolver código com não-gerenciado (o código não é gerenciado por um tempo de execução);
- Suporte a código legado (antigos): pode comunicar-se com códigos antigos de objetos COM e DLLs escritas em uma linguagem não-gerenciada;

2.4 SQLite

O sistema de gerenciamento de banco de dados (SGBD) relacional utilizado para a criação da aplicação foi o SQLite, uma ferramenta *open source*, de pequeno porte, embutida, e *serverless*. SQLite disponibiliza transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) (SQLITE, 2017b). Transações ACID serão abordadas na seção 2.5. Além disto, Owens (2006) cita outros motivos para utilizar o SQLite, dentre elas: configuração zero, as configurações necessárias são feitas somente pelo programador e não de ajuda externa; portabilidade, executado em diferentes sistemas operacionais, tais como Windows, Mac OS X e Linux; compacidade, leve em armazenamento e auto-contido, nenhum servidor de banco de dados é necessário; simplicidade; flexibilidade; confiabilidade, códigos bem documentados e elaborados.

Outra vantagem de se usar o SQLite é o fato da configuração de rede não ser necessária. Todas as alterações no banco de dados são feitas no próprio disco rígido. De acordo com Owens (2006, p. 1, tradução nossa)[6] "isso reduz a sobrecarga relacionada às chamadas de rede, simplifica a administração do banco de dados e mais fácil implantar seu aplicativo. Tudo que você precisa é compilado diretamente no seu programa". Essa vantagem, porém, é uma limitação ao mesmo tempo. Pelo fato de ser *serverles*[7], especificamente *serverless* clássico, significa que não segue o modelo cliente/servidor, ou seja, é interna e não tem comunicação com qualquer servidor ou rede externa (SQLITE, 2017a).

Outras limitações, de acordo com Owens (2006), são: concorrência, é permitido várias leituras, porém, uma leitura de cada vez; tamanho do banco de dados, a cada 1 Megabyte, são necessários 256 bytes de RAM, afetando as taxas de transações. Portanto, SQLite não é recomendado a ser utilizado em projetos de grande porte ou que tem projeção de crescimento.

[6] This reduces overhead related to network calls, simplifies database administration, and makes it easier to deploy your application.

[7] Não tem comunicação com qualquer servidor ou rede externa.

O SQLite é utilizado por várias empresas consideradas gigantes no mercado, dentre elas a Apple. O sistema operacional Mac OS X, da Apple, utiliza o SQLite em um dos seus *frameworks* (OWENS, 2006). SQLite pode ser utilizado em várias linguagens, dentre elas, o C#, C++, PHP, Java, etc. Neste trabalho foi utilizado a linguagem C#.

2.5 Banco de Dados Relacional *versus* Banco de Dados Não Relacional

O que diferencia um banco de dados relacional de um não relacional, NoSQL, é a forma como os dados são organizados e armazenados. Em um banco de dados relacional, os dados são armazenados em formas de tabelas. Dentro de cada tabela, existe colunas e em cada coluna, contém um tipo de dado (*strings* e inteiros). Para que o dado seja inserido em seu devido lugar, o banco de dados precisa ser esboçado previamente e os esquemas das tabela precisam estar bem definidos.

O banco de dados NoSQL reduz o planejamento da modelagem de dados e, assim, é ideal para armazenar dados não estruturados. Os dados são agrupados em um registro e, então, não há necessidade de ter relacionamentos entre os dados para ser formada a informação. Os autores Sharma e Dave (2012) apresentam quatro diferentes tipos de armazenamentos de dados no NoSQL: banco de dados *key-value*, banco de dados de armazenamento de documentos, bancos de dados colunares e banco de dados de grafo. Além destes quatro tipos, Nayak, Poriy e Poojary (2013) apresenta um quinto tipo: banco de dados orientado a objetos.

Os dados, em um banco de dados do tipo *key-value*, "consistem em duas partes, uma *string* que representa a chave e os dados reais que devem ser referidos como valor, criando assim um par de '*key-value*'"(NAYAK; PORIY; POOJARY, 2013, p. 16, tradução nossa)[8]. Dados salvos neste tipo de banco de dados podem ser usados em situações onde é necessário salvar a sessão ou salvar itens favoritos de um usuário. O banco de dados de armazenamento de documentos é constituído de um par de chaves e valores, que são quase iguais ao banco de dados do tipo *key-value* (SHARMA; DAVE, 2012). A única diferença é que os valores armazenados (chamados de "documentos") fornecem alguma estrutura e codificação dos dados gerenciados.

Para Nayak, Poriy e Poojary (2013, p. 16, tradução nossa)[9] armazenamentos de colunares, "cada chave é associada a um ou mais atributos (colunas). [...] Os dados armazenados no banco de dados são baseados na ordem de classificação da família de colunas". Basicamente, os dados são salvos em uma tabela onde nela possui várias famílias de colunas, e dentro destas famílias, outras colunas onde estão as propriedades. Banco de dados de grafo é baseado na teoria dos grafos. Em geral, um grafo é composto por nós, arestas e propriedades. Em um banco de dados de grafo os nós representa as entidades, propriedades representa os atributos e cada aresta a relação (SHARMA; DAVE, 2012). Banco de dados de grafo pode ser utilizado em várias

[8] The data consists of two parts, a string which represents the key and the actual data which is to be referred as value thus creating a "*key-value*" pair.

[9] [...] each key is associated with one or more attributes (columns). [...] The data which is stored in the database is based on the sort order of the column family.

aplicações tais como aplicações de redes sociais, bioinformática, gerenciamento de conteúdo, segurança e controle de acesso, gerenciamento de rede e nuvem. “Um banco de dados orientado a objetos é um banco de dados no qual os dados ou as informações a serem armazenadas são representados como um objeto (semelhante a um objeto usado no conceito de linguagem de programação orientada a objetos)”(NAYAK; PORIY; POOJARY, 2013, p. 18, tradução nossa)[10].

O uso do NoSQL vem aumentando de acordo com o seu potencial de alta escalabilidade[11] e flexibilidade[12] (FOTACHE; COGEAN, 2013). De acordo com Nayak, Poriy e Poojary (2013), o banco de dados NoSQL, em relação ao modelo relacional, tem evoluído em um ritmo muito alto e fornece uma ampla escolha de modelos de dados para escolher. Em contra partida, os mesmos autores acreditam que o NoSQL, em relação ao modelo relacional, é imaturo, a manutenção é difícil e que alguns bancos NoSQL não são compatíveis com as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), mas, garante as propriedades BASE (Basicamente disponível, estado leve e eventualmente consistente).

Apresentado por Brewer (2000), o teorema do CAP (Consistência, disponibilidade e tolerância a particionamento), afirma que um sistema de compartilhamento só pode garantir duas propriedades dentre as três seguintes:

- Consistência: quando qualquer dado é gravado, qualquer um que lê do banco de dados sempre verá a ultima versão do dado;
- Disponibilidade: cada operação é esperada terminar com uma resposta intencional;
- Tolerância de Partição: o banco de dados pode ser lido e gravado mesmo quando partes são inacessíveis.

A propriedade BASE foca na tolerância de partição e disponibilidade do teorema do CAP. Segundo Pokorny (2013) esta propriedade, na qual o NoSQL se baseia, garante as seguintes semânticas:

- Basicamente disponível: o banco de dados aparece funcionar na maioria do tempo;
- Estado leve: o estado do sistema deve se mudar ao decorrer do tempo, mesmo sem gravação de dados;
- Eventualmente consistente: o sistema será consistente ao longo do tempo.

A propriedade ACID, que foca na consistência e disponibilidade do teorema do CAP, providencia as seguintes garantias, segundo Pritchett (2008):

- Atomicidade: todas as operações nas transações serão realizadas ou não;

[10] An object oriented database is a database in which the data or the information to be stored is represented as an object (similar to an object used in the concept of object oriented programming language).
[11] Capacidade de um sistema expandir sem que o seu desempenho piore.
[12] Não tem restrições em estruturar o banco de dados.

- Consistência: o estado do banco de dados será mantido quando a transação iniciar e terminar;
- Isolamento: a transação daquele instante será tratada como se fosse a única transação sendo executada no baco de dados;
- Durabilidade: a operação não será desfeita assim que a transação for completada.

2.6 Firebase

O Firebase é uma plataforma do tipo BaaS (*Backend as a Service*)[13], com funcionalidade em tempo real de desenvolvimento de aplicativos para dispositivos móveis e Web, que fornece aos desenvolvedores ferramentas e serviços para ajudá-los a construir aplicativos de alta qualidade e aumentar sua base de usuários. Estas ferramentas e serviços giram em torno dos serviços da Google *Cloud Platform*, permitindo que os usuários salvem e recuperem dados para serem acessados de qualquer dispositivo ou navegador.

O Firebase foca nos elementos *back-end*[14] dos aplicativos permitindo o desenvolvedor focar na interface e nas funcionalidades *front-end*[15]. As ferramentas e serviços são providenciadas por meio de vários SDK com APIs simples de usar e excelente integração com várias plataformas de aplicativos móveis e web no mercado, dentre elas: Android Studio, Xcode, Aplicativos Javascipt e a ferramenta utilizada neste trabalho, Unity (FIREBASE, 2018b). Isso elimina a necessidade de criar seu próprio *script* do lado do servidor usando PHP e MySQL, ou qualquer outra configuração semelhante.

Os serviços do Firebase podem ser divido em dois grupos (FIREBASE, 2018b):

- Desenvolver e testar seu aplicativo:
  - Banco de Dados em tempo real: Banco de dados que sincroniza os dados com os dispositivos em tempo real. Regras de segurança podem ser configuradas para definir quem tem acesso a quais dados;
  - Cloud Firestore: Banco de dados de documentos NoSQL que permite armazenar, sincronizar e consultar dados facilmente para seus aplicativos para dispositivos móveis e da Web, em escala global;
  - Autenticação: Possibilita autenticação através de contas do Google, Facebook, Twitter, Github ou um sistema de contas próprio;

[13] Este modelo é um meio de facilitar a integração de aplicativos com um armazenamento em nuvem de *backend*, e, ao mesmo tempo, fornecer serviços de autenticação do usuário, notificações de *push* e integração com as redes sociais (SAREEN, 2013).

[14] A parte de um sistema de computador ou parte de um software, onde os dados são armazenados ou processados em vez das partes que são usadas pelo usuário (THESAURUS, 2018a).

[15] As partes de um computador, peça de software ou website que são vistas e usadas diretamente pelo usuário (THESAURUS, 2018b).

  - *Cloud Storage*: Armazena dados do usuário através de uma conexão segura e permite o compartilhamento dos mesmos;
  - Monitoramento de desempenho: Produz *insights* sobre o desempenho do aplicativo do ponto de vista dos usuários, com rastreamento de desempenho automático e personalizado;
  - *Crashlytics*: priorize problemas e corrija-os com relatórios de erros avançados e em tempo real;
  - *Crash Reporting*: Coleta informações de falhas que os usuários estão experienciando no aplicativo;
  - Hospedagem: Serviço para hospedagem de sites com certificado SSL;
  - *Test Lab* para Android: Serviço para testar o aplicativo em diferentes tipos de dispositivos.

- Expandir e envolver o público:
  - *Analytics* para Firebase: Ferramenta de análise, que produz insights sobre sobre o comportamento dos usuários e o aplicativo;
  - *Dynamic Links*: Usado para criar links que executam determinadas ações no aplicativo;
  - *Invites*: Utiliza o Dynamic Links para criar convites personalizados para o aplicativo, que podem ser enviados pelo usuário para outras pessoas;
  - *Cloud Messaging*: Permite enviar mensagens para os usuários através do aplicativo. É possível definir para quais grupos de usuários a mensagem será enviada;
  - Google *AdWords*: Ferramenta para publicar anúncios do aplicativo no Google, YouTube ou Play Store;
  - Configuração Remota: Permite que versões diferentes do aplicativo sejam publicadas para diferentes usuários;
  - Previsões: Aplica o aprendizado de máquina aos dados de análise para criar grupos dinâmicos de usuários com base no comportamento previsto;
  - *App Indexing*: Permite que o aplicativo seja encontrado e pesquisas no Google Search, caso o assunto que o usuário procura seja relacionado com o app;
  - *AdMob*: Facilita a monetização do aplicativo, colocando anúncios que encaixem no design do mesmo.

Um dos serviços mais utilizados da plataforma da Google é o Firebase *Realtime Database* (banco de dados em tempo real). A plataforma Firebase oferece planos de uso para os seus serviços, sendo 3 planos diferentes: *Spark*, *Flame*, *Blaze*. Todos os 3 planos inclui

gratuitamente os seguintes serviços: *Analytics*, *App Indexing*, Autenticação (exceto *Phone Auth*),*Cloud Messaging*, *Crashlytics*, *Dynamic Links*, *Invites*, Configuração Remota e Previsões.

O plano *Spark* é recomendável para desenvolvedores amadores e é gratuito. Para o serviço Realtime Database existe um limite de 100 conexões simultâneas, 1GB de armazenamento e 10GB por mês de *download*. O plano *Flame* já é recomendável para aplicativos em expansão e tem o preço de US$ 25 por mês. O serviços Realtime Database tem o limite de 100 mil conexões simultâneas, 2,5GB de armazenamento e 20GB por mês de *download*.

O plano *Blaze* é um plano para aplicativos já estabelecidos no mercado e que precisam de mais recursos, sendo o pagamento por utilização. O serviço Realtime Database, neste plano, permite vários bancos de dados para o mesmo projeto, tem um limite de 100 mil conexões por banco de dados, o preço de US$ 5 por GB de armazenamento e US$1 por GB de *download*.

Existem outras plataformas hospedadas na nuvem com banco de dados NoSQL, dentre elas: Azure da Microsoft, MongoDB, CouchDB. Apesar da disponibilidade destas outras plataformas, o Firebase é a que mais fornece uma integração mais sólida com o software Unity e, por isso, foi a opção utilizada neste trabalho.

2.6.1 *Firebase Realtime Database*

O banco de dados em tempo real do Firebase é um banco de dados NoSQL (tipo de armazenamento *key-value*), hospedado na nuvem, de baixa latência[16] e, por isso, tem otimizações e funcionalidades diferentes de um banco de dados relacional. Por ser um banco de dados NoSQL, o Firebase Realtime Database se baseia nas propriedades BASE, apresentado na seção 2.5, e quando se reflete no teorema do CAP o tipo de armazenamento *key-value* garante mais as propriedades disponibilidade e tolerância de partição do que a propriedade consistência. “O Realtime Database oferece uma linguagem de regras declarativas que permite que você defina como os dados devem ser estruturados, como devem ser indexados e quando podem ser lidos e gravados” (FIREBASE, 2018g).

A API do Realtime Database foi projetada para permitir apenas operações que podem ser executadas rapidamente. De acordo com a documentação do Firebase (2018d), “isso permite que você crie uma excelente experiência em tempo real que pode atender a milhões de usuários sem comprometer a capacidade de resposta. Por isso, é importante pensar em como os usuários precisam acessar seus dados e, em seguida, estruturá-los de acordo”. Os dados são armazenados em formato JSON, por ser um banco de dados NoSQL (FIREBASE, 2018d). JSON (JavaScript Object Notation) é um formato de texto para armazenar e trocar dados. A subseção 2.6.1.1 irá explicar mais sobre este formato.

Toda vez que os dados são alterados, todos os clientes em todas as plataformas conectados, recebem as atualizações em milissegundos e isso elimina requisições HTTPS, que normalmente gastam mais tempo. Cada vez que o usuário usa o aplicativo, o SDK do Firebase

[16] Tempo gasto do envio dos dados de um ponto ao outro na rede.

Realtime Database preserva os dados em disco e, ao estabelecer uma conexão com uma rede, sincroniza os dados com o estado atual do servidor, permitindo usuários utilizarem os aplicativos mesmo off-line (FIREBASE, 2018d).

2.6.1.1 JSON

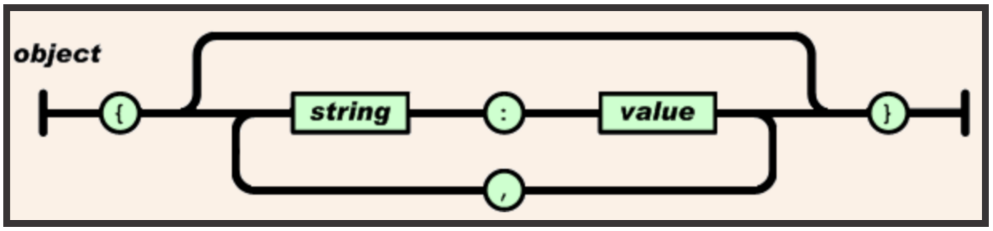
JSON (JavaScript Object Notation) é um formato de texto para armazenar e trocar dados (entre servidores e aplicações web). O JSON usa a sintaxe do JavaScript, mas o formato JSON é somente texto. Portanto, JSON é independente de qualquer linguagem, ou seja, pode ser lido e usado como formato de dados por qualquer linguagem de programação, dentre elas: C++, C#, Java, JavaScript, Perl, Python, dentre outras (JSON, 1999).

JSON pode ser divida em duas estruturas (JSON, 1999):

- Uma coleção de pares nome/valor. Em várias linguagens, isto é caracterizado como um *object*, *record*, *struct*, dicionário, *hash table*, *keyed list*, ou *arrays* associativas;
- Uma lista ordenada de valores. Na maioria das linguagens, isto é caracterizado como uma *array*, vetor, lista ou sequência.

Em JSON, os dados são armazenados conforme figura 2.11. Um objeto JSON inicia e termina com uma chave de abertura e fechamento ( ). Dentro dessa estrutura encontra-se o nome, no formato de uma *string*, o sinal de pontuação dois pontos (:) e, em seguida, o seu valor. Havendo outros elementos, estes são separados por vírgula e seguem o mesmo formato.

Figura 2.11 – Estrutura de armazenamento JSON



Fonte: JSON (1999)

O banco de dados não relacional NoSQL possui um tipo de armazenamento chave-valor, assim, a utilização do JSON é a mais adequada por possuir também o mesmo tipo de armazenamento. No Firebase Realtime Database, todos os dados são armazenados como objetos JSON. No banco de dados do Firebase cada chave na árvore é chamado de *node* ou de filho e cada tem um valor.

Segundo o Firebase (2018c), seu banco de dados é como uma árvore JSON hospedada na nuvem e ao adicionar novos dados "à árvore JSON, eles se tornam nodes na estrutura JSON existente com a chave associada. Você pode fornecer suas próprias chaves, como códigos do usuário ou nomes semânticos, ou obtê-los usando o método *Push()*". Caso o desenvolvedor crie as próprias chaves, é necessário não conter determinados caracteres, sendo estes: ., $, #,

[,], / nem os caracteres 0 a 31 ou 127 de ASCII de controle. Também, as chaves precisam ser codificadas em UTF-8 e só podem ter até 768 bytes.

2.6.1.2 Regras de Segurança do Firebase Realtime Database

Não há necessidade de um servidor de aplicativos para dispositivos móveis ou navegadores da Web acessarem o Firebase Realtime Database diretamente. Assim que os dados são lidos ou gravados, são executados as regras baseadas em expressões (Regras de Segurança do Firebase Realtime Database) garantindo segurança e a validação dos dados. De acordo com a documentação do banco de dados do Firebase (2018d), "por meio da integração com o Firebase Autenticação, os desenvolvedores podem definir quem tem acesso, a quais dados e como esses dados podem ser acessados".

O acesso de leitura e gravação ao banco de dados do Firebase é restrito por diretrizes. Apenas usuários autenticados podem alterar o banco de dados e a autenticação dos usuários ocorre através do recurso Firebase Authentication. Caso este recurso não seja utilizado, deve-se definir as Regras de Segurança do Firebase Realtime Database para público ou customizar as regras para que o acesso seja restrito. O guia das Regras de Segurança [17] está disponível nos documentos do Firebase Database.

As regras do Firebase Database tem quatro tipos:

- .read: permite quando os dados podem ser lidos pelos usuários;
- .write: permite quando os dados podem ser gravados pelos os usuários;
- .validate: garante que os dados que estão sendo escritos estejam de acordo com um padrão específico, por exemplo, se os dados inseridos tem um determinado tamanho, respeita um intervalo, se é *string*, se é *bool*, se é numero, se é um e-mail e se estes dados contém *nodes* filhos;
- .indexOn: permite fazer consultas específicas em uma coleção de *nodes* usando qualquer chave filha em comum. Um exemplo seria ordenar os usuários, chaves, pela o valor das suas alturas, seus *nodes* filhos.

A configuração das regras são de acordo com as necessidades do projeto. Como padrão, leitura e gravação é somente permitido para usuários autenticados pelo serviço Firebase Authentication. A figura 2.12 demonstra o código padrão do Firebase Database.

A figura 2.13 ilustra como são as regras para qualquer pessoa ter permissão de ler ou gravar no banco de dados. Estas regras não são recomendáveis, por deixar o banco de dados vulnerável. Porém é útil para a fase de desenvolvimento. A figura 2.14 descreve regras mais seguras, pois desativa completamente a leitura e gravação do banco de dados, porém impossibilita o acesso aos dados.

[17] Guia para entender as regras do Firebase Realtime Databse. Disponível em https://firebase.google.com/docs/database/security/.

Figura 2.12 – Regra padrão para o Firebase Database

Fonte: Firebase (2018f)

Figura 2.13 – Regras públicas para o Firebase Database

Fonte: Firebase (2018f)

Figura 2.14 – Regras privadas para o Firebase Database

Fonte: Firebase (2018f)

2.7 Exploratory

Com o intuito de analisar os dados do banco de dados, o software Exploratory foi utilizado neste trabalho para este fim. O software analisa e envolve conjuntos de dados a fim de resumir as características principais, executada pela linguagem R.

O software Exploratory é um produto de código fechado[18], apesar de usar a linguagem R. Para utilizar este software é necessário pagar uma mensalidade de acordo com o uso. O uso pessoal é US$39 por mês e uma conta corporativa tem o preço de US$79. Entretanto, há disponibilidade de um plano comunitário para estudantes ou professores. É importante ter um e-mail institucional para cadastrar no site do software Exploratory [19]. Após a criação da conta, é preciso baixar o software, instalá-lo e seguir os passos contidos na documentação [20].

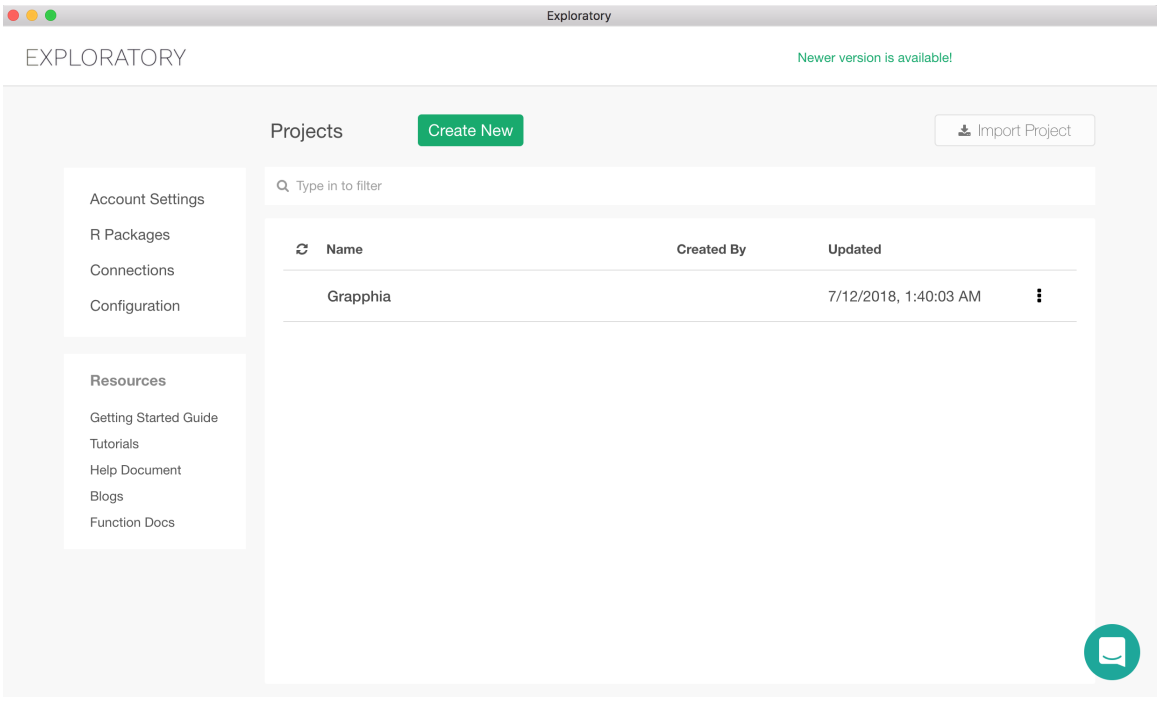
Outras alternativas para este software seria a importação dos dados ao software Microsoft Excel ou a análise manual dos dados.

A tela inicial deste software é apresentada pela figura 2.15. A figura 2.16 ilustra uma das funcionalidades deste software, Inspecionar Dados com Visualização de Resumo. Os dados usados nesta figura representam os atrasos de voo dos EUA. Ao clicar na aba *Summary*, essa funcionalidade apresenta resumidamente os dados importados ao software. Assim é possível fazer um análise rápida dos dados. O retângulo verde em destaque na figura mostra as *carriers*

[18] Antônimo de código aberto, ou seja, código fonte é suprimido.
[19] Criar conta com e-mail institucional. Disponível em: https://exploratory.io/plan?plan=Community.
[20] Documentação de como utilizar o software Exploratory e importar dados JSON disponível em https://docs.exploratory.io.

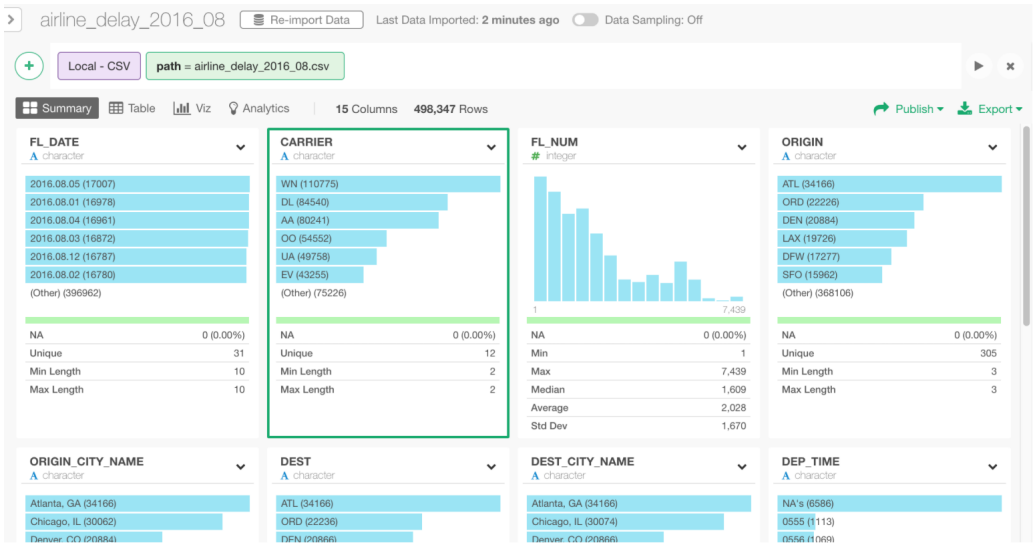
(empresas aéreas) que mais se atrasam. Nesta mesma figura há o resumo de outros tipos de dados tais como as origens e os destinos dos voos, os números dos voos e as datas dos voos.

Figura 2.15 – Tela Inicial do software Exploratory



Fonte: Elaborada pelo autor

Figura 2.16 – Inspecionar dados com visualização de resumo



Fonte: Exploratory (2018)

3 MÉTODOS

O presente capítulo objetiva abordar os passos necessários para que a integração do aplicativo Grapphia com o serviço Firebase Realtime Database fosse possível.

A seção 3.1 apresenta a atual estrutura das tabelas do banco de dados do Grapphia. A seção 3.2 aborda a integração do software Unity com o Firebase Realtime Database. A subseção 3.2.1 aborda a escolha da plataforma para o desenvolvimento do Grapphia. A subseção 3.2.2 apresenta os passos para criar um projeto na plataforma Firebase e como fazer a conexão com o software Unity. As subseções 3.2.3 e 3.2.4, apresentam as orientações para inserir o SDK do Firebase ao software Unity e como configurar o SDK no software Unity, respectivamente. A subseção 3.2.5 apresenta as recomendações do Firebase para estruturar os dados a serem salvos.

A subseção 3.2.6 e suas subseções apresentam os códigos necessários para diferenciar dispositivos móveis, como salvar e remover os dados em cada *node* da árvore JSON e a definição das regras de segurança. A subseção 3.2.7 apresenta como exportar e analisar os dados.

3.1 Estrutura do Banco de Dados SQLite

Inicialmente o único SGBD utilizado no Grapphia era o SQLite, um banco de dados relacional. A base consistia de 3 tabelas: *users* (Tabela 1), *palavraOpcao* (Tabela 2) e *palavraAcertoUser* (Tabela 3). Os atributos e os seus tipos de cada tabela estão descritos a seguir:

Tabela 1 – Tabela *users*

users	
Atributo	Tipo
ID (chave primária)	integer
Name	varchar
Score	integer
Erros	integer
Nivel	integer
scoreDitado	integer
erroDitado	integer

A integração do SGBD SQLite com o software Unity foi feita pelos autores Santos (2016) e Leal (2016). Para que a utilização acontecesse no software Unity, a inclusão da biblioteca *SQLite4Unity3d* foi necessária. A figura 3.1 mostra o código necessário para estabelecer a conexão entre o SQLite e o aplicativo Grapphia.

Durante a integração do software Unity com o Firebase, foi necessário a criação de uma tabela com o nome de *keyPhone*. Esta tabela contém somente um atributo do tipo *string*. É explicado o mótivo da criação desta tabela na subseção 3.2.6.1.

Tabela 2 – Tabela *palavraOpcao*

palavraOpcao	
Atributo	Tipo
ID (chave primária)	integer
palavra_completa	varchar
palavra	varchar
opcao1	varchar
opcao2	varchar
nivel	integer
nome_audio_menino	varchar
nome_audio_menino	varchar
nome_audio_ditado	varchar

Tabela 3 – Tabela *palavraAcertoUser*

palavraAcertoUser	
Atributo	Tipo
ID (chave primária)	integer
idUser	varchar
idPalavra	bool
acerto	varchar
nivelPalvra	varchar

Figura 3.1 – Conexão com o banco de dados e criação das tabelas

```
_connection = new SQLiteConnection(dbPath, SQLiteOpenFlags.ReadWrite | SQLiteOpenFlags.Create);

Debug.Log("Final PATH: " + dbPath);
//Criando tabela usuário e tabela opção!
_connection.CreateTable<user>();
_connection.CreateTable<palavraOpcao>();
_connection.CreateTable<palavraAcertoUser>();
```

Fonte: Elaborada pelo autor

3.2 Integração do Unity com Firebase Realtime Database

A necessidade da integração do Grapphia com um banco de dados com conexão à um servidor surgiu a partir da observação que a extração dos dados seria penosa e constante. Até então, a extração era feita transferindo a base de dados dos dispositivos móveis para o computador. Assim, faz-se necessário ter acesso manual aos dispositivos dos usuários para efetuar a captura dos dados. Com esses dados armazenados no computador, realizava-se o agrupamento dos dados para posterior análise. Como o SQLite não faz conexão com um servidor, e mudar o SGBD seria um processo árduo e prolongado, então este trabalho propõe a utilização do Firebase Realtime Database, um banco de dados não relacional hospedado na nuvem.

Este trabalho fez uso da versão gratuita (Plano *Spark*), pois, além de gratuito, possui

fácil integração com o software Unity e por ser um banco de dados flexível[1], de baixa latência[2] e facilmente escalável[3]. A subseção 2.6.1 apresenta os detalhes do Firebase Realtime Database.

A integração do Firebase e do Firebase Realtime Database com o software Unity foi baseada na documentação disponibilizada por Firebase (2018a) e Firebase (2018g), respectivamente.

3.2.1 Escolha da Plataforma

A plataforma Android foi escolhida para o desenvolvimento do Grapphia. A configuração exigida para a construção do aplicativo nesta plataforma é o software Unity com a versão 5.3 ou mais recente.

O Android foi a alternativa deste trabalho por ser gratuito e de fácil disseminação para os fins da pesquisa do Grapphia. Para executar o aplicativo, é necessário que o dispositivo móvel contenha a versão 4.1 do Android ou superior. Versões anteriores não suportam algumas funcionalidades do sistema. A subseção 2.1.1 demonstra os detalhes sobre esta plataforma.

3.2.2 Inserção do Firebase ao Unity

Para a utilização da ferramenta do Firebase é necessário ter uma conta da Google. Assim, o usuário terá permissão para utilizar os serviços fornecidos pela plataforma e configuração do banco de dados do Firebase.

Para estabelecer uma conexão do Firebase ao projeto no Unity, são necessários um *project* e um arquivo de configuração da plataforma. O *project* é criado no Firebase *console*, exibido pela figura 3.2, clicando em *Add project*, em português, “Adicionar projeto”.

Ao realizar esta etapa, o site direcionará à uma nova aba, apresentada na figura 3.3. Nesta tela deve-se preencher os dados do projeto, tais como: nome, País/região e, por fim, ler e indicar se aceita o termo de compromisso. Após confirmação, basta clicar no botão *Create project*, “Criar Projeto”, em português.

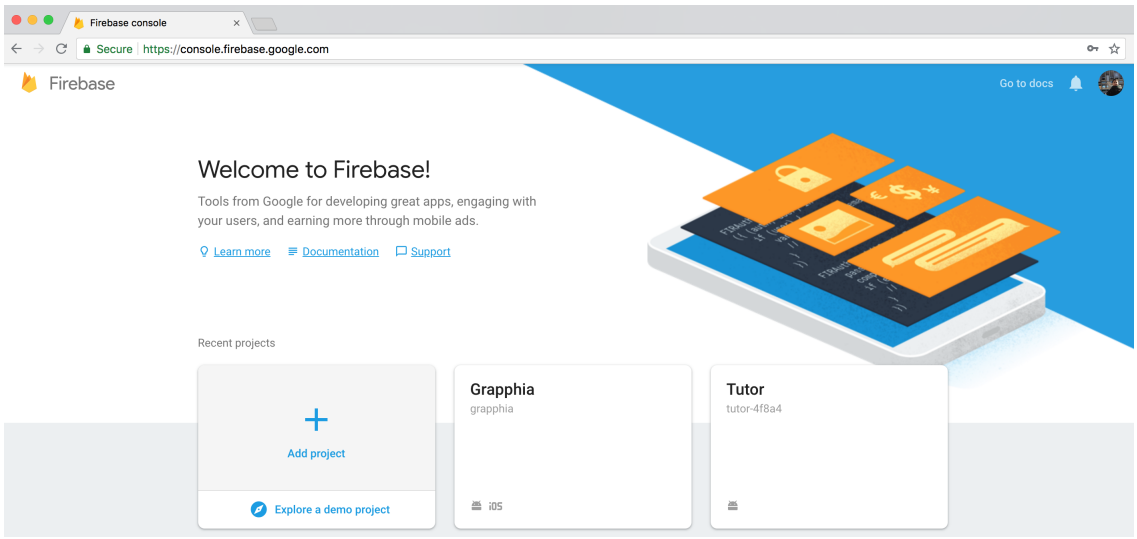
Após a criação do projeto, a plataforma conduz para a página de Visão Geral do projeto, onde é possível configurar e gerenciar o mesmo. Em seguida, deve-se clicar em *Add Firebase to your Android app*, “Adicionar o Firebase ao seu aplicativo Android”, e seguir as orientações. Caso o interesse seja adicionar o Firebase ao aplicativo na plataforma iOS ou Web, clica-se em uma das opções de acordo com a plataforma. A figura 3.4 mostra as opções possíveis de plataforma para o desenvolvedor escolher.

Durante o processo de adição do Firebase ao aplicativo do Android, é necessário registrar o aplicativo, entretanto, o nome do pacote do Android a ser inserido deve estar de acordo com o nome do pacote fornecido no software Unity.

---

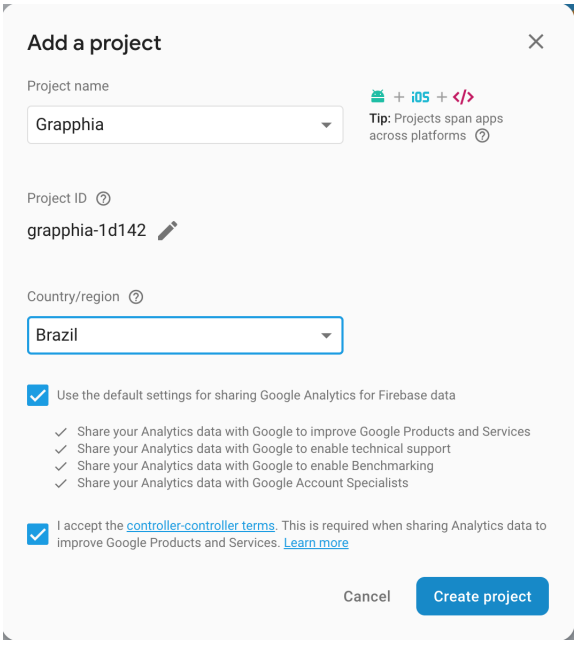
1 Não tem restrições em estuturar o banco de dados.
2 Tempo gasto do envio dos dados de um ponto ao outro da rede.
3 Capacidade de um sistema expandir sem que o seu desempenho piore.

Figura 3.2 – *Console* Firebase



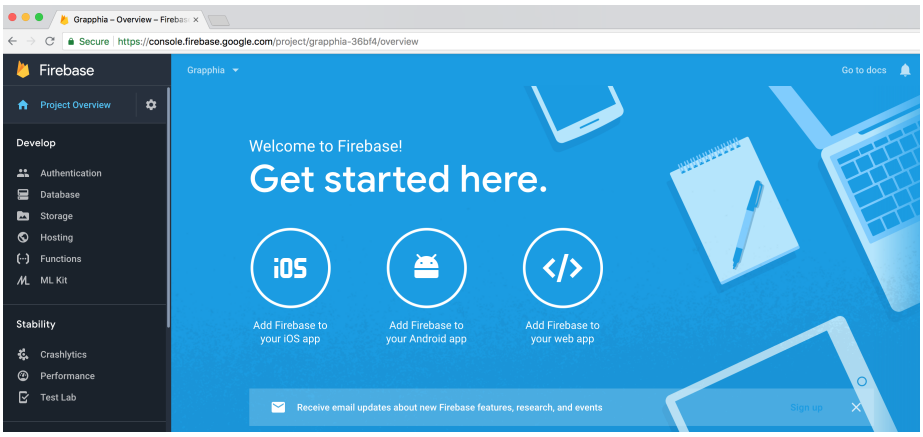
Fonte: Elaborada pelo autor

Figura 3.3 – Adicionar Projeto



Fonte: Elaborada pelo autor

Figura 3.4 – Adicionar o Firebase ao seu aplicativo Android

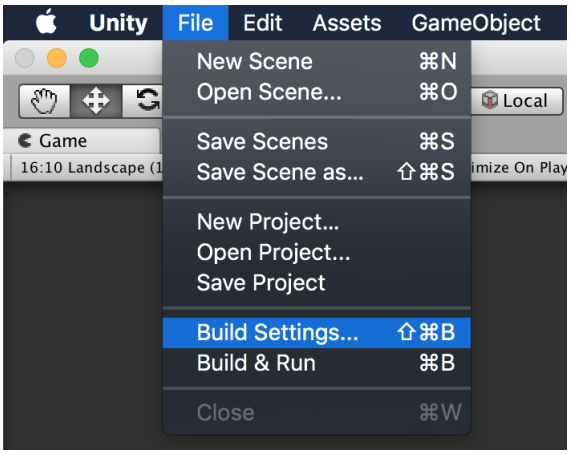


Fonte: Elaborada pelo autor

É possível descobrir o nome do pacote no software Unity, de acordo com os passos a seguir:

1. Abrir o projeto do Unity;
2. Selecionar o item do menu *File*;
3. Clicar em *Build Settings*, representado pela figura 3.5;

Figura 3.5 – Item do Menu File - *Build Settings*

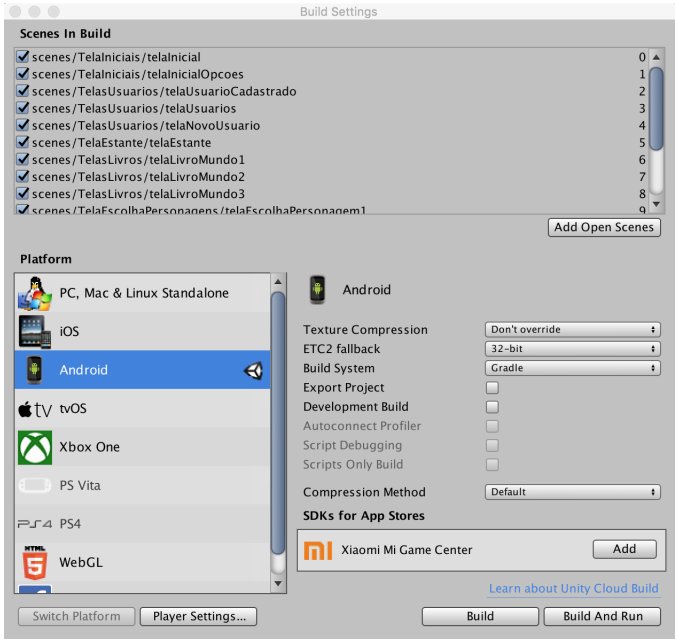


Fonte: Elaborada pelo autor

4. Clicar em *Player Settings* na nova janela exibida, figura 3.6;
5. Copiar o nome do pacote, *Package Name*, fornecido pelo *Inspector*, localizado no canto direito da tela, abaixo de *Other Settings* e *Identification*, destacado em azul na figura 3.7.

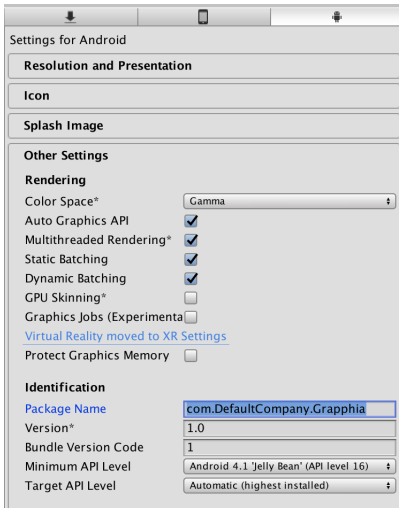
Depois de copiado, é necessário inserir o nome do pacote e clicar em "Registrar aplicativo", apontado pela figura 3.8. Posteriormente, é disponibilizado um arquivo no formato JSON para baixar, "google-services.json", indicado pela figura 3.9.

Figura 3.6 – Nova janela exibida com a opção *Player Settings*



Fonte: Elaborada pelo autor

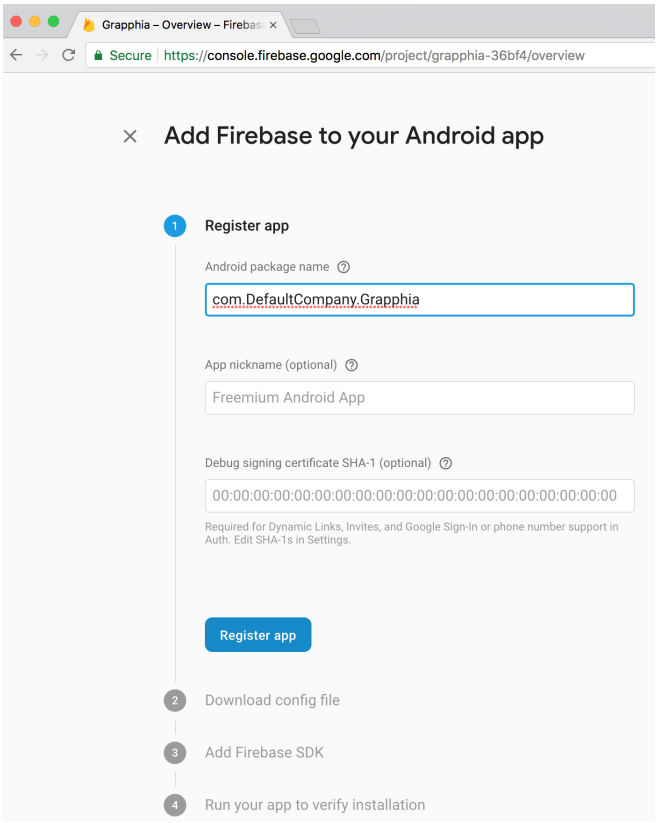
Figura 3.7 – Nome do pacote destacado em azul



Fonte: Elaborada pelo autor

O arquivo gerado de acordo com a plataforma escolhida precisa ser adicionado em qualquer lugar da pasta *Assests* do projeto Unity, exibido pela figura 3.10, por se tratar de arquivo de configuração que fará a comunicação do Unity com o projeto criado no *console* do Firebase.

Figura 3.8 – Registrar o aplicativo



Fonte: Elaborada pelo autor

3.2.3 Inserção do SDK do Firebase ao Unity

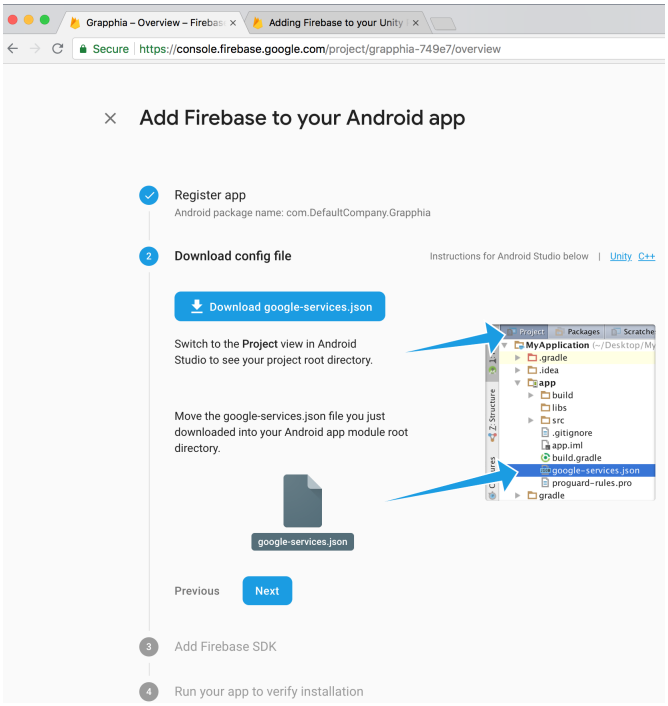
O SDK do Firebase é um conjunto de ferramentas para utilizar os serviços já mencionados na seção 2.6, dentre elas, o Firebase Realtime Database. Com o aplicativo registrado e o arquivo inserido na pasta *Assets*, é obrigatório instalar o SDK do Firebase no projeto do Unity. Para que isto aconteça, é fundamental baixar o SDK do Firebase para o software Unity na página de documentação do Firebase [4]. Após baixar o SDK, basta descompacta-lo num local mais conveniente.

A instalação do SDK no projeto é feita ao abrir o projeto no Unity. Para importar qualquer *plug-in* de qualquer recurso do Firebase, no caso deste trabalho o Firebase Realtime Database, deve-se seguir os seguintes passos:

1. Selecionar o item de menu *Assests*;

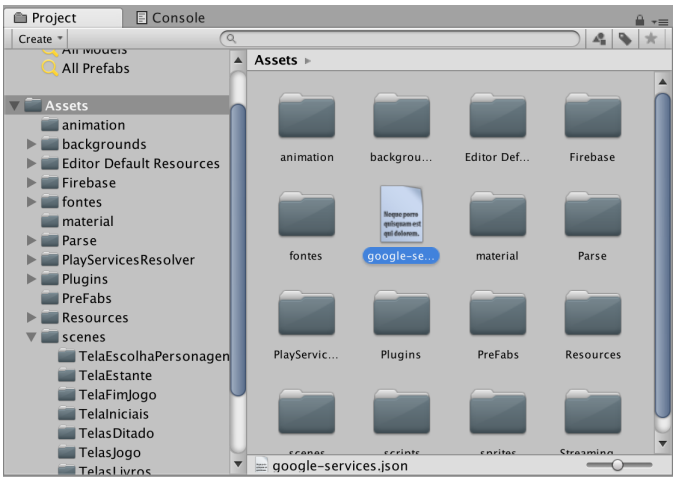
[4] Site oficial do Firebase. SDK disponível em https://firebase.google.com/docs/unity/setup

Figura 3.9 – Baixar arquivo JSON



Fonte: Elaborada pelo autor

Figura 3.10 – Inserindo arquivo JSON na pasta *Assets*

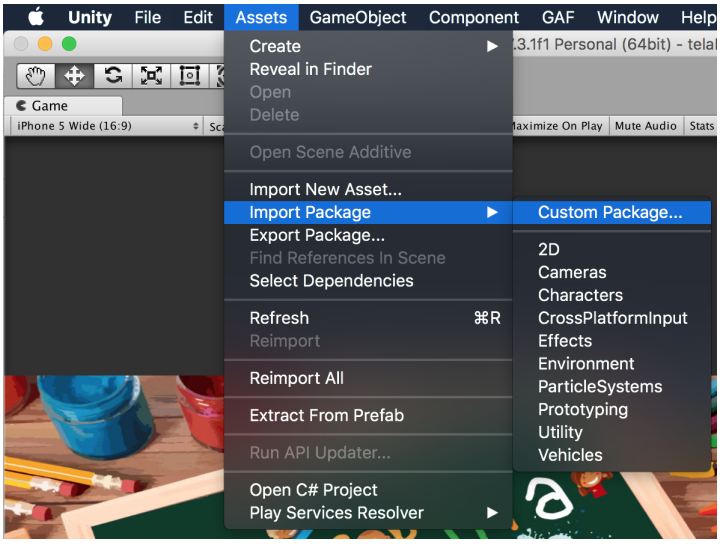


Fonte: Elaborada pelo autor

2. Selecionar *Import Package*;

3. Selecionar *Custom Package*, como representado pela figura 3.11;

Figura 3.11 – Selecionando *Custom Package*



Fonte: Elaborada pelo autor

4. Localize a pasta do SDK descompactada e selecione “FirebaseDatabase.unitypackage”;

5. Clique em *Import* na janela *Import Unity Package* exibida, como demonstra a figura a 3.12.

3.2.4 Configurar o SDK para o Unity Editor

Ao criar o projeto no Firebase, a plataforma gera um URL do banco de dados que, posteriormente, deve ser inserido no código do projeto do software Unity para que o SDK, instalado no projeto do software Unity, comunique adequadamente com o banco de dados. Antes de acessar a instância do banco de dados, é necessário chamar a função *SetEditorDatabaseUrl()*, passando o URL do banco de dados por referência. Esta função deve ser chamada dentro do método *Start()* devido este método ser chamado no início da execução do *script*.

Os passos necessários para encontrar o URL do banco de dados são os seguintes:

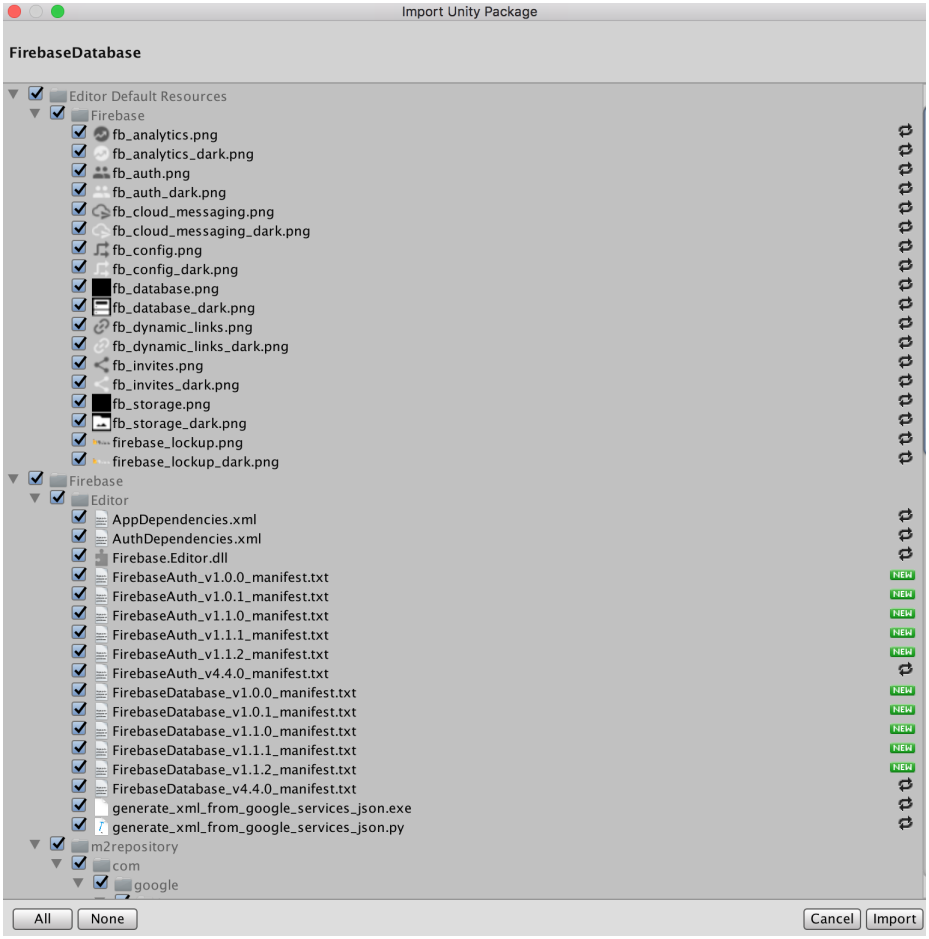
1. Acessar o *console* do Firebase;

2. No menu *Develop*, clicar em Database, representado pela figura 3.13;

3. Clicar em Realtime Database;

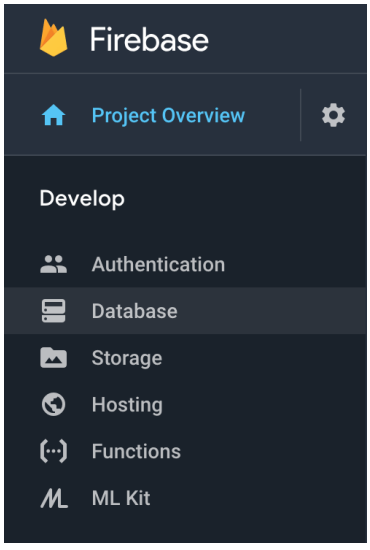
4. O URL do banco encontra-se na próxima página, como mostra a figura 3.14

Figura 3.12 – Clicando em *Import*



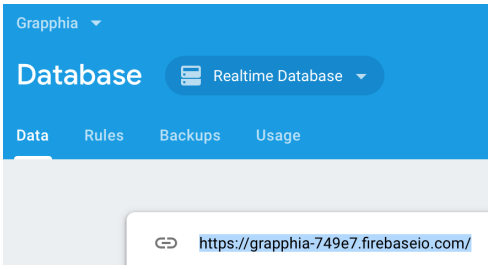
Fonte: Elaborada pelo autor

Figura 3.13 – Acessar o bando de dados



Fonte: Elaborada pelo autor

Figura 3.14 – URL do banco de dados



Fonte: Elaborada pelo autor

A figura 3.15 apresenta a função desenvolvida, *InitizializeFirebase()*, inserida no método *Start()* para inicializar a conexão com o banco de dados na nuvem. Na linha 20 da figura 3.15, a aplicação de objeto, *FirebaseApp* comunica com todos os serviços do Firebase usado por um aplicação. *DefaultInstance* referencia para o aplicativo de padrão no *console* do Firebase. A função *SetEditorDatabaseUrl()* permite inserir o URL do banco de dados do projeto.

Figura 3.15 – Inicializar a conexão com Firebase

```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using System.IO;
4  using SQLite4Unity3d;
5  using System.Collections;
6  using Firebase;
7  using Firebase.Unity.Editor;
8  using Firebase.Database;
9
10 //Classe da tela inicial!
11 public class telaInicial : MonoBehaviour {
12
13     public GameObject somOn; // opção ativar e desativar som!
14     public GameObject somOff;
15     // Inicialização da tela início!
16
17     DatabaseReference reference;
18
19     void InitializeFirebase() {
20         FirebaseApp.DefaultInstance.SetEditorDatabaseUrl("https://grapphia.firebaseio.com/");
21         reference = FirebaseDatabase.DefaultInstance.RootReference;
22     }
23     void Start () {
24
25         InitializeFirebase ();
26
```

Fonte: Elaborada pelo autor

A função *InitizializeFirebase()* é também declarada nos *scripts*: *createUser.cs*, *gameController.cs*, *telaInicial.cs* e *usuariosJogo.cs*, e posteriormente incluída na função *Start()*.

3.2.5 Estruturar Dados

Os dados armazenados no Firebase Realtime Database são armazenados na forma de texto JSON, como mencionado na subseção 2.6.1.1. Uma das práticas recomendadas pelo Firebase (2018c) é evitar o aninhamento dos dados. Esta recomendação se deve ao modo de como é executado o procedimento de recuperação dos dados. Quando se busca um dado no banco de dados, todos os nodes filhos são recuperados. De acordo com Firebase (2018c), "quando concede a um usuário o acesso de leitura ou gravação a um node do banco de dados, ele também pode acessar todos os dados desse node".

3.2.6 Salvar Dados

A inserção de dados no Firebase Realtime Database se dá através de algumas linhas de códigos. Para iniciar a gravação dos dados no banco é necessário acessar uma instância de *DatabaseReference*.

Na linha da 17 da figura 3.15, está sendo declarado, dentro da classe pública *telaInicial*, a variável *reference* para a classe *DatabaseReference*. Posteriormente na linha 21, dentro da função *InitizializeFirebase()*, a mesma variável recebe o ponto de entrada para o acesso com o Firebase Database, *FirebaseDatabase*. A propriedade *DefaultInstance* recebe a instância do aplicativo padrão do Firebase. A propriedade *RootReference* referencia a localização do *node* raiz do banco de dados.

Para inserir qualquer dado na árvore JSON, é preciso utilizar a variável *reference* que indica o local onde o dado será armazenado. Para alcançar esta localização, utiliza-se a função pública *Child()* da classe *DatabaseReference*. Caso exista um *node* dentro de um outro *node*, é fundamental utilizar a mesma função novamente ou digitar o caminho, usando *string* entre aspas, no parâmetro da função *Child()*. Para separar *nodes* é imprescindível utilizar uma barra comum (*/*).

No final de cada variável *reference* é vital acrescentar a função *SetValueAsync()*, uma *thread* assíncrona, com o dado (um objeto) passando como parâmetro. Uma *thread* é uma série de comandos que existe na forma de uma unidade de trabalho e, quando é assíncrona, significa que uma outra *thread* pode ser executa mesmo que a atual não termine sua execução.

3.2.6.1 Diferenciando dispositivos móveis

Os dados do aplicativo Grapphia estão sendo salvos em um banco de dados relacional, SQLite. Como o SQLite está sendo utilizado em paralelo, é aproveitado a sua consistência de não repetir os nomes no banco de dados. Entretanto, como vários dispositivos móveis serão utilizados, será necessário um forma de separar os mesmos e manter consistência dos dados no banco de dados do Firebase. Dentro da biblioteca do Firebase, existe um método que, automaticamente, gera chaves de acordo com o *timestamp*[5], o método *Push().Key*.

Sempre que um novo *node* filho for adicionado em um determinado lugar do Firebase, o método *Push()* gera uma chave exclusiva, apontando para uma nova localização na árvore JSON. Como esta chave é gerada de acordo com um *timestamp*, os dados são inseridos automaticamente em ordem cronológica. Assim, vários usuários podem incluir dados simultaneamente em um determinado *node* evitando conflitos de gravação. Para que a chave gerada seja retornada para uma referência *Push()* é necessário chamar a propriedade *Key* e armazenar em uma variável *string*, como mostra na linha 60 da figura 3.16. A chave gerada neste exemplo está sendo referenciada para o *node users*.

O método *Push().Key* é chamado toda vez que a função *Start()*, do *script telaInicial.cs* é chamada. Para não criar vários *nodes* na árvore JSON, a variável *string* recebe a chave gerada pelo método citado somente quando não existe um banco de dados, ou seja, somente quando o aplicativo é executado pela primeira vez. Vale ressaltar que essa função também é chamada

[5] Registro de data e hora em um determinado ponto no tempo independente de qualquer fuso horário ou calendário, representado os segundos e frações de segundos na resolução de nanossegundos.

Figura 3.16 – Inserindo chave gerada na tabela *keyPhone*

```
56          //Se não tiver nenhuma palavra no banco você chama a função e preenche as palavras
57          if(bancoPalavras.Instance.total_palavras_geral < 1 ){
58              bancoPalavras.Instance.salvar_palavras_no_banco();
59
60              string pushKey = reference.Child("users").Push().Key;
61
62              var k = new keyPhone{
63                  keyFireBase = pushKey,
64              };
65
66              _connection.Insert(k);
67          }
```

Fonte: Elaborada pelo autor

mesmo quando o dispositivo estiver off-line e quando a conexão é estabelecida com o servidor, a chave é inserida em seu devido lugar no banco de dados.

A chave gerada precisa ser armazenada em um determinado lugar para que todos os dados sejam salvos debaixo do mesmo *node* no Firebase Realtime Database. Então, a Tabela *keyPhone* foi criada no banco de dados SQLite para que a chave gerada pelo método *Push().Key* seja armazenada toda vez que o aplicativo roda pela primeira vez. O único atributo, keyFirebase, e seu tipo está representado pela Tabela 4. Esta chave também é armazenada nas Tabelas *users* e *palavraAcertoUser* para garantir consistência dos dados.

Tabela 4 – Tabela *user*

keyPhone	
Atributo	Tipo
keyFirebase	string

3.2.6.2 Salvar dados da Tabela *users*

No aplicativo Grapphia, a função para criar usuário *create()* se encontra no *script CreateUser.cs*. Esta função, primeiramente, verifica a existência de algum usuário já cadastrado com o nome apresentado e, caso isso ocorra, o usuário é advertido para inserir outro nome. Caso não exista um usuário cadastrado, é chamada a função *data.CreateUser()*, do *script usuarioJogo.cs*, passando por parâmetro os valores iniciais para os objetos nome, *score*, nível, *scoreDitado* e *key* (chave gerada pelo Firebase). O objeto nome é inicializado com o texto que o usuário digitou e o objeto *key*, inicializado com a chave gerada pelo Firebase. Os objetos *score*, nível e *scoreDitado* são inicializados com o valor 0.

Na função *data.CreateUser()* é criada uma nova instância da classe *user* na variável *p*, recebendo os valores dos objetos passados por parâmetro. Portanto, essa nova variável contém os objetos com os seus valores da classe *user*, tais como: o nome, *score*, erros, nível, acertos do ditado, erros do ditado e chave gerada pelo Firebase. Para acessar estes objetos basta digitar a variável *p*, seguida por um ponto e nome do objeto. Por exemplo, digitando *p.Nome*, você acessa o objeto nome e para acessar o objeto *score* basta digitar *p.Score*.

A figura 3.17 demonstra a implementação dos códigos que insere os dados do usuário no *node users* no banco de dados do Firebase.

Figura 3.17 – Código que insere dados do usuário no banco de dados do Firebase

```
string path = keyFireBase + "/" + p.Name;

reference.Child("users").Child(path + "/Nome").SetValueAsync(p.Name);
reference.Child("users").Child(path + "/Score").SetValueAsync(p.Score);
reference.Child("users").Child(path + "/Erros").SetValueAsync(p.Erros);
reference.Child("users").Child(path + "/Nivel").SetValueAsync(p.Nivel);
reference.Child("users").Child(path + "/Score Ditado").SetValueAsync(p.scoreDitado);
reference.Child("users").Child(path + "/Erro Ditado").SetValueAsync(p.erroDitado);
reference.Child("users").Child(path + "/Date & Time").SetValueAsync(System.DateTime.UtcNow.ToString("HH:mm dd MMMM, yyyy"));
```

Fonte: Elaborada pelo autor

A variável *path* do tipo *string* recebe parte do caminho em que o dado será armazenado, a *string keyFirebase* (chave gerada pelo método *Push().Key*), uma barra e o nome do usuário. Assim, ao inserir o nível do usuário, por exemplo, o endereço de armazenamento será */users/keyFirebase/Nome/Nível*. No final de cada variável *reference* tem a função *SetValueAsync()* recebendo como parâmetro o objeto.

Os valores das variáveis *Score* e *Erros*, são atualizados nas funções *pressedButtonLetter1()* e *pressedButtonLetter2()* dos *scripts gameController.cs* e *gameController2.cs*. Dentro de cada uma destas funções, os valores são atualizados de acordo com o acerto e o erro do usuário em tempo real. O motivo para salvar estas variáveis em tempo real é que estas serão úteis para a pesquisa do Grapphia para analisar o ambiente dos usuários e as instruções dadas aos usuários.

3.2.6.3 Salvar dados da Tabela *palavrasAcertoUser*

A função *salvar_palavrasAcertoUser()*, que encontra-se no *script comandosBasicos.cs* é responsável por inserir os dados sobre quais palavras foram acertadas ou erradas na Tabela *palavrasAcertoUser*. A figura 3.18 ilustra o código responsável por adicionar os dados no *node palavrasAcertoUser* do banco de dados do Firebase. Nas linhas 191 e 192 dessa figura, o código atualiza no banco de dados do Firebase quando palavras são apresentadas novamente para o usuário. Nas linhas 200 e 201 dessa figura, o código insere os dados no banco de dados do Firebase quando as palavras são apresentadas pela primeira vez para o usuário.

Na tabela *palavraAcertoUser* do SQLite, cada acerto ou erro do usuário é identificado por um *ID*. No projeto do Grapphia e, consequentemente, no Firebase, estas informações são armazenadas em uma estrutura de dados *array* (vetor). Cada posição deste vetor contém uma tupla da tabela.

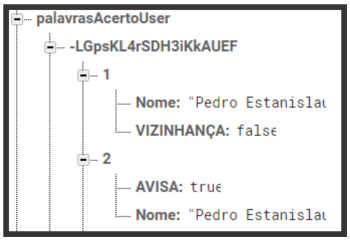
A figura 3.19, ilustra a organização dos dados no *node palavrasAcertouser*. Neste *node* será salvo um *node* filho, representado pela chave gerada pelo método *Push().Key* (que representa o identificador geral dos usuários daquele dispositivo em específico). Em seguida será salvo os *nodes* filhos com o valor do *ID* de cada transação. Posteriormente, será salvo os dados contidos no vetor. Os únicos dados salvos serão o valor booleano do acerto e o nome do usuário.

Figura 3.18 – Código que insere dados das palavras que o usuário acertou ou errou no banco de dados do Firebase

```
181
182         string pathFireBase = "palavrasAcertoUser/" + dadosJogo.Instance.currentUser.key + "/";
183
184         for(int i=0; i<palavrasAcerto.Length; ++i){
185
186             if (palavrasAcerto [i].idPalavra > 0 && palavrasAcerto [i].Id > 0) {
187                 _connection.Update (palavrasAcerto [i]);
188
189                 auxPalavra = bancoPalavras.Instance.palavras[palavrasAcerto[i].idPalavra - 1].palavra_completa;
190
191                 reference.Child (pathFireBase + palavrasAcerto [i].Id + "/"+auxPalavra).SetValueAsync (palavrasAcerto[i].acerto);
192                 reference.Child (pathFireBase + palavrasAcerto [i].Id + "/"+ "Nome").SetValueAsync (dadosJogo.Instance.currentUser.Name);
193
194
195             } else if (palavrasAcerto [i].idPalavra > 0) {
196                 _connection.Insert (palavrasAcerto [i]);
197
198                 auxPalavra = bancoPalavras.Instance.palavras[palavrasAcerto [i].idPalavra - 1].palavra_completa;
199
200                 reference.Child (pathFireBase + palavrasAcerto [i].Id + "/"+auxPalavra).SetValueAsync (palavrasAcerto[i].acerto);
201                 reference.Child (pathFireBase + palavrasAcerto [i].Id + "/"+ "Nome").SetValueAsync (dadosJogo.Instance.currentUser.Name);
202             }
203         }
```

Fonte: Elaborada pelo autor

Figura 3.19 – Estrutura do *node palavrasAcertoUser*



Fonte: Elaborada pelo autor

3.2.6.4 Salvar dados da Tabela *palavraOpcao*

As palavras escolhidas para o aplicativo Grapphia são inicialmente inseridas em um vetor e posteriormente são inseridas na Tabela *palavraOpcao*. Situada no *script comandosBasico.cs*, a função *salvar_palavras_no_banco()* é responsável por inserir as palavras na Tabela *palavraOpcao* do SQLite.

Aproveitando esta inserção, as palavras e suas informações são inseridas no banco de dados do Firebase de acordo com o código da figura 3.20. As palavras serão ordenadas de acordo com o atributo *ID* da tabela *palavraOpcao*.

Figura 3.20 – Código que insere palavras no banco de dados do Firebase

```
for (int i = 0; i < total_palavras_geral; ++i){
    _connection.Insert (palavrasBanco[i]);

    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/palavra").SetValueAsync (palavrasBanco[i].palavra);
    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/letra_correta").SetValueAsync (palavrasBanco[i].letra_correta);
    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/opcao1").SetValueAsync (palavrasBanco[i].opcao1);
    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/palavra_completa").SetValueAsync (palavrasBanco[i].palavra_completa);
    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/nivel").SetValueAsync (palavrasBanco[i].nivel);
    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/nome_audio_menino").SetValueAsync (palavrasBanco[i].nome_audio_menino);
    reference.Child ("palavras").Child (palavrasBanco[i].Id + "/nome_audio_menina").SetValueAsync (palavrasBanco[i].nome_audio_menina);
}
```

Fonte: Elaborada pelo autor

3.2.6.5 Remoção de dados

No aplicativo Grapphia existe a opção de apagar qualquer usuário cadastrado no banco de dados SQLite. No Firebase Realtime Database para excluir os dados do banco de dados basta executar a função pública *RemoveValueAsync()* referenciando o local dos dados na árvore.

Na figura 3.21, o código na linha 232 representa a remoção do *node* do usuário escolhido. Primeiro referencia a localização dos dados do usuário no banco de dados e posteriormente executa-se a função *RemoveValueSync()*. Os dados não são removidos do *node palavrasAcertoUser*, no Firebase, por serem relevantes na pesquisa do Grapphia. A remoção ocorre apenas no banco de dados interno (SQLite) do dispositivo móvel.

Figura 3.21 – Código que remove dados do banco de dados Firebase

```
219
220     // função para remover usuários!
221     public void removeUser()
222     {
223         DataService data = new DataService(_connection);
224        //data.EstabeleceConexao ("grapphia");
225
226         var users = _connection.Table<user>().Where(x => x.Name == Users.captionText.text);
227         var key = bancoPalavras.Instance._connection.Table<keyPhone> ().FirstOrDefault().keyFireBase;
228
229         foreach (var user in users){
230
231             data.removeUser(user.Id);
232             reference.Child ("users/" + key).Child ("/" + user.Name).RemoveValueAsync ();
233         }
234             Users.options.RemoveAt(Users.value);
235
236             Users.captionText.text = "";
237     }
```

Fonte: Elaborada pelo autor

3.2.6.6 Definição das Regras de Segurança

Conforme mencionado na subseção 2.6.1, as regras de segurança do Firebase por padrão são restritas somente a usuários autenticados pelo serviço Firebase Authentication. Como este serviço não é utilizado neste trabalho, as regras de segurança foram alteradas de acordo com o uso do aplicativo.

Como neste trabalho somente está sendo gravado dados no Firebase Realtime Database e não tendo consultas no mesmo, as regras definidas são uma união das regras do tipo pública e das regras do tipo privada. A figura 3.22 apresenta essa junção. A leitura dos dados seguiu o exemplo de regras privadas e a gravação dos dados seguiu o exemplo de regras públicas.

3.2.7 Análise dos Dados

Através do software Exploratory é possível fazer uma análise dos dados exportados do banco de dados do Firebase. Em qualquer ponto da pesquisa do Grapphia os dados podem ser exportados em um arquivo no formato JSON. Para exportar os dados, é necessário acessar a seção do banco de dados no *Console*. Quando acessada a página do banco de dados, deve-se

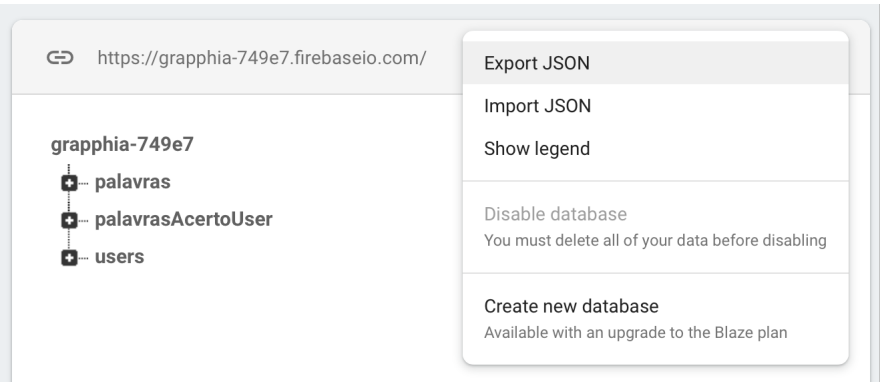
Figura 3.22 – Regras para o banco de dados Firebase do aplicativo Grapphia

```
{
  "rules": {
    ".read":false,
    ".write": true,
  }
}
```

Fonte: Elaborada pelo autor

clicar no ícone do menu de 3 (três) pontos, na parte superior direita do painel de dados. A figura 3.23 ilustra o procedimento de como exportar os dados do Firebase. Ao clicar em *Export JSON*, no menu, inicializa-se o *download* do arquivo.

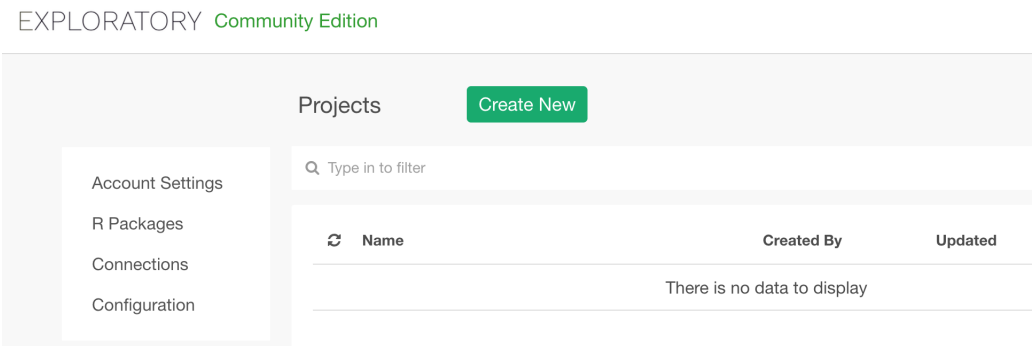
Figura 3.23 – Exportando dados do banco de dados do Firebase



Fonte: Elaborada pelo autor

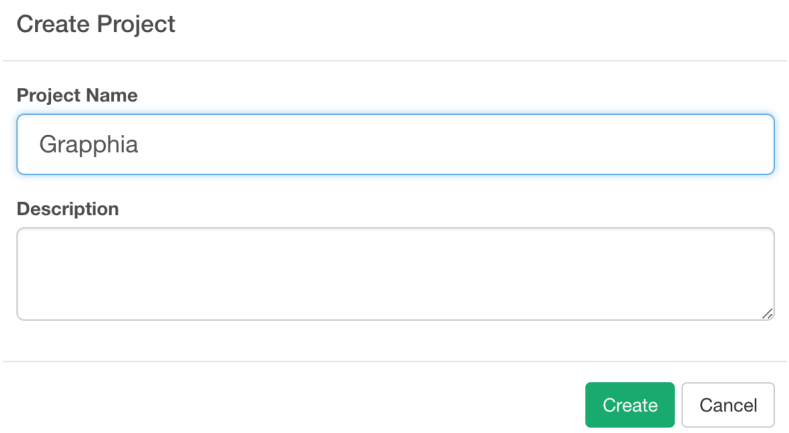
Posteriormente, o arquivo JSON pode ser importado no software Exploratory, para que a análise dos dados seja feita. Para fazer a análise dos dados no Exploratory, é necessário criar um novo projeto. A figura 3.24 mostra o botão para criar um novo projeto. Em seguida, uma nova aba é apresentada (figura 3.25) com um campo para inserção do nome do projeto e um botão para criar o projeto. Os passos para importação do arquivo JSON serão descritas na seção 3.2.7.

Figura 3.24 – Criando projeto no software Exploratory



Fonte: Elaborada pelo autor

Figura 3.25 – Aba para criar novo projeto



Fonte: Elaborada pelo autor

4 TESTES E RESULTADOS

O presente capítulo tem como objetivo mostrar a inserção e remoção de dados no Firebase Realtime Database através de 7 testes. Os testes são compostos de inserção de dados nos *nodes palavras*, apresentado na seção 4.1, *users*, apresentado na seção 4.2, e *palavraAcertoUser*, apresentado na seção 4.4, remoção dos dados no *node users*, apresentado na seção 4.5, quando o dispositivo móvel está *online* e *off-line*, apresentado na seção 4.6. Após os testes com o Firebase Realtime Database, é feito uma análise dos dados exportados do banco de dados do Firebase, através do uso do software Exploratory, mostrado na seção 4.7.

4.1 Inserção de dados no *node palavras*

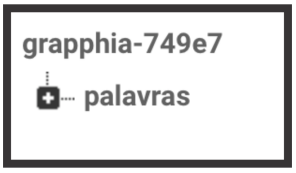
No momento em que cria-se o projeto no Firebase, o banco de dados está vazio. O *node* parente, *grapphia-749e7*, está recebendo o valor *null* (sem valor), como ilustra a figura 4.1.

Figura 4.1 – Banco de dados vazio

Fonte: Elaborada pelo autor

Quando pela primeira vez o aplicativo é executado e as palavras são preenchidas no banco de dados SQLite, ocorre a inserção das palavras no banco de dados do Firebase. A figura 4.2 demonstra o *node palavras*, como um filho do *node* parente, *grapphia-749e7*, criado assim que o aplicativo é executado em um dispositivo móvel.

Figura 4.2 – Palavras inseridas no banco de dados

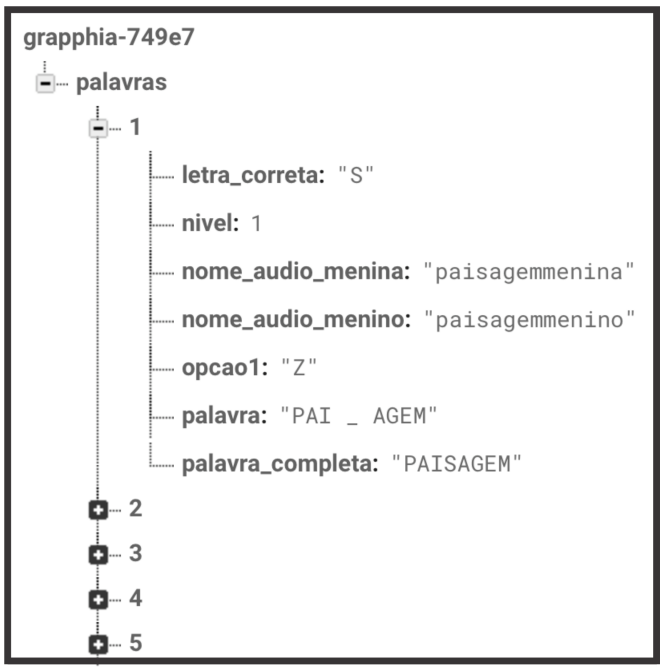


Fonte: Elaborada pelo autor

A figura 4.3 expõe quando o *node palavras* é expandido e revela os dados armazenados de acordo com a ordem do *ID* das palavras. Abaixo desse *node* tem o filho com o *ID* da palavra e abaixo deste filho tem os dados de cada palavra. Entre esses dados estão a palavra sem

a letra, a letra correta que o usuário precisa escolher para acertar a palavra, o nível da palavra e a palavra completa. Essa estrutura se repete para todas as outras palavras.

Figura 4.3 – *Node palavras* expandido

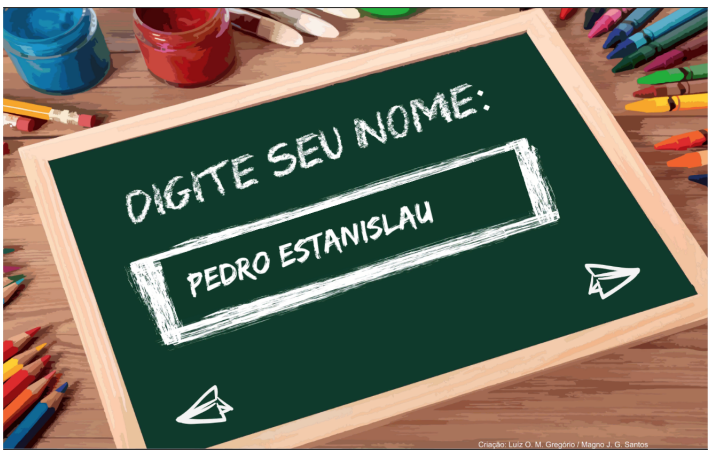


Fonte: Elaborada pelo autor

4.2 Inserção de dados no *node users*

No aplicativo Grapphia, é necessário criar um usuário para utilizar o jogo. A figura 4.4 ilustra o momento em que o jogador deve digitar um nome para criar o seu usuário.

Figura 4.4 – Criando o usuário

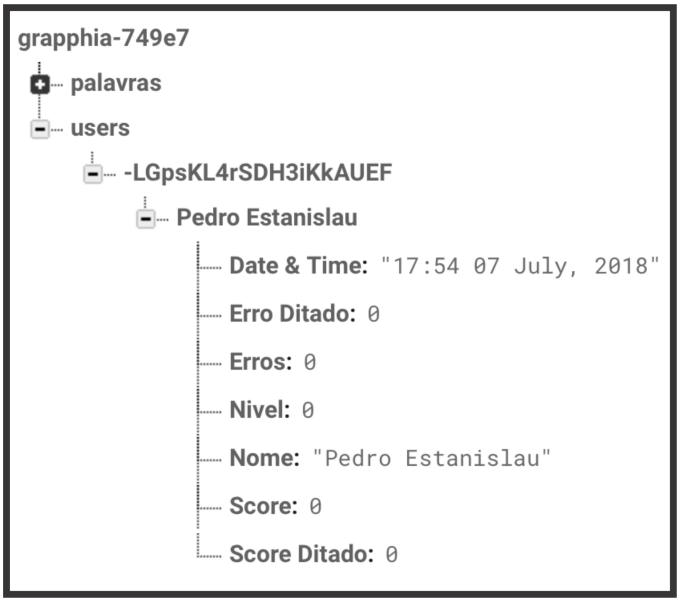


Fonte: Elaborada pelo autor

A figura 4.5 demonstra a inserção dos dados do usuário no banco de dados do Firebase. Nessa figura, primeiro aparece o *node* parente, *grapphia*, depois o *node palavras*, depois o *node* filho *users*, depois o *node $key*, gerado pelo método *Push().Key*.

Abaxo do *node $key*, está o *node* com o nome do usuário. Os filhos do *node* do nome do usuário contém a data e hora que os dados foram inseridos, a quantidade de erros acumulados no jogo, a quantidade de erros do ditado, o nível que o usuário se encontra, o nome, os acertos (representado por *Score*) e os acertos do ditado (representado por *Score Ditado*).

Figura 4.5 – Criando o usuário



Fonte: Elaborada pelo autor

4.3 Atualização das variáveis *Score* e *Erros*

Após a seleção do usuário, do livro e do personagem, o aplicativo inicia a parte do jogo, no qual é apresentada uma palavra para que o usuário selecione a letra para completar a palavra. A figura 4.6 mostra quando o usuário acerta a palavra “Avisa” e o sistema indica que o usuário acertou. A figura 4.7 mostra o instante que a variável *Score* é atualizada no banco de dados do Firebase assim que o usuário acerta a palavra.

A figura 4.8 exemplifica a situação em que o usuário erra a palavra “Vizinhança” e é mostrado a ele que errou. A figura 4.9 demonstra a atualização do banco de dados do Firebase assim que o jogador erra a palavra. O texto destacado em amarelo, tanto na figura 4.7 e 4.9, é o caminho da árvore onde as variáveis são atualizadas.

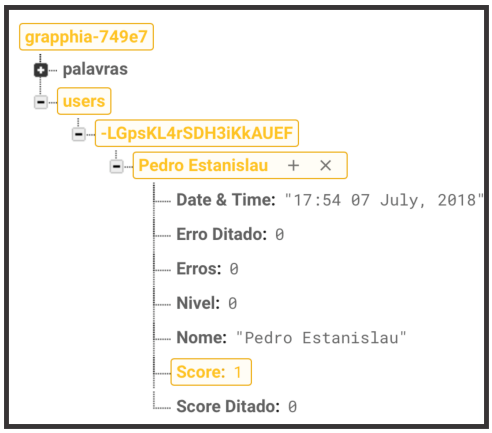
4.4 Inserção de dados no *node palavrasAcertoUser*

O *node palavrasAcertoUser* é atualizada quando o jogador quiser retornar ao *Menu Inicial*, quando o mesmo queira sair do jogo ou quando termina de mostrar todas as palavras. A

Figura 4.6 – Acerto da palavra Avisa

Fonte: Elaborada pelo autor

Figura 4.7 – Atualizando variável *Score*

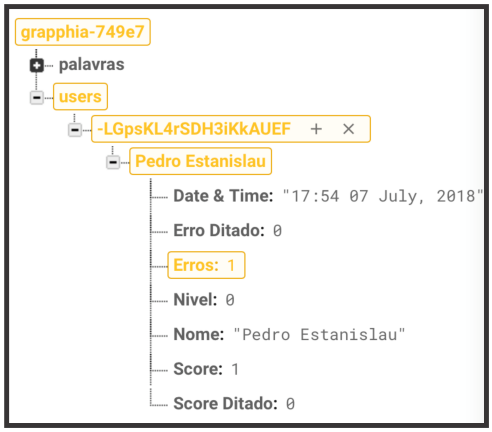


Fonte: Elaborada pelo autor

Figura 4.8 – Erro da palavra Vizinhança

Fonte: Elaborada pelo autor

Figura 4.9 – Atualizando variável *Erros*



Fonte: Elaborada pelo autor

figura 4.10 revela quando o *node palavrasAcertoUser* é criado e expandido, mostrando os dados inserido neste *node*.

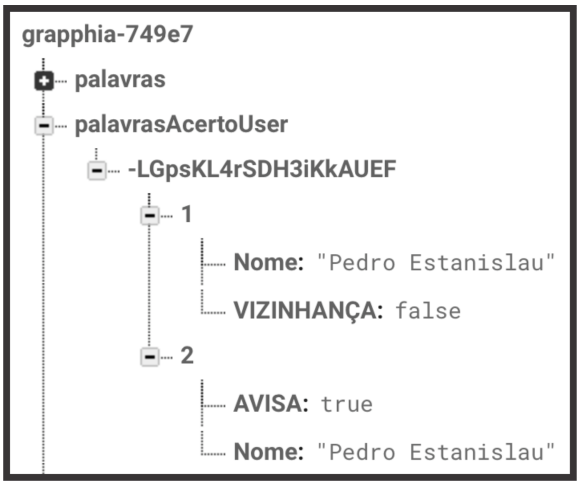
4.5 Remoção de dados no Firebase Realtime Database

Quando o jogador quiser remover um perfil de usuário do jogo Grapphia, o mesmo aperta o botão “Continuar” no *Menu Inicial* e na próxima tela seleciona o perfil que deseja apagar. Assim que o perfil é selecionado, como mostra a figura 4.11, o usuário aperta o botão “Apagar”. A figura 4.12 mostra, destacado em vermelho, o instante em que os dados do usuário são removido. Os dados no *node palavrasAcertouser* são mantidos por serem úteis para a pesquisa do Grapphia.

4.6 Inserção de dados com o dispositivo móvel sem conexão

O Firebase Realtime Database permite inserir dados mesmo como o dispositivo estando sem conexão com o servidor, ou seja, sem conexão com a internet. O SDK armazena os dados no dispositivo móvel e quando a conexão é restabelecida, os mesmo são enviados para o

Figura 4.10 – Inserindo o *node palavraAcertoUser*

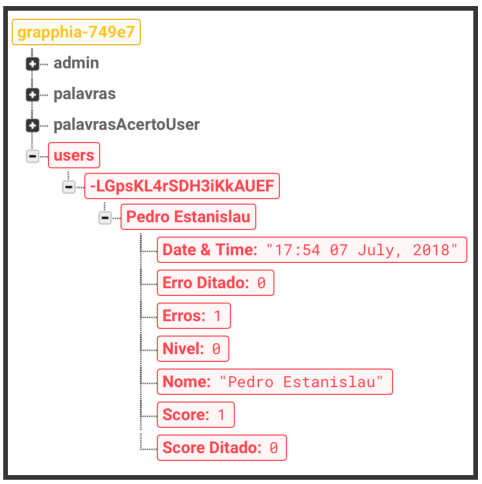


Fonte: Elaborada pelo autor

Figura 4.11 – Remover usuário no Grapphia

Fonte: Elaborada pelo autor

Figura 4.12 – Remoção dos dados do Usuário no banco de dados do Firebase



Fonte: Elaborada pelo autor

banco de dados. A fim de testar a funcionalidade de gravar dados sem conexão, foi utilizado um dispositivo móvel com a plataforma Android em Modo Avião[1].

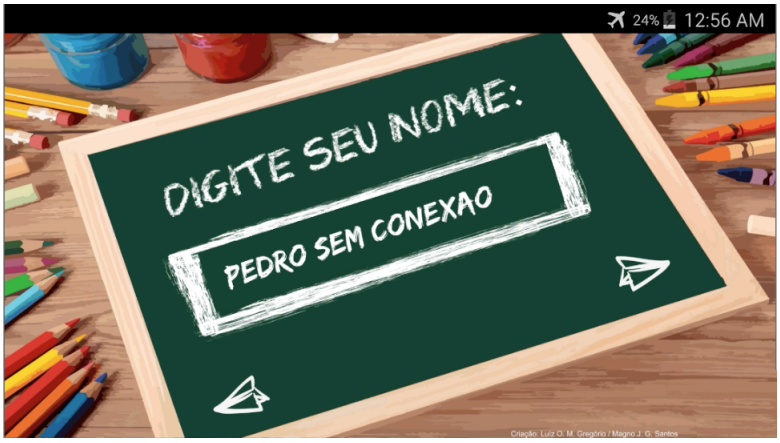
A figura 4.13 mostra o momento em que o jogador digita um nome no campo para criar o seu usuário com o dispositivo móvel sem conexão com qualquer rede. No canto superior direito percebe-se o ícone do Modo Avião.

As figuras 4.14 e 4.15 demonstram o acerto e erro das palavras Casa e Avise, respectivamente, com o dispositivo em Modo Avião.

A figura 4.16 ilustra o momento exato que os dados, destacados em amarelo, referente ao teste sem conexão são inseridos no banco de dados do Firebase. A figura 4.17 mostra os nodes recém adicionados expandidos, expondo os dados inseridos.

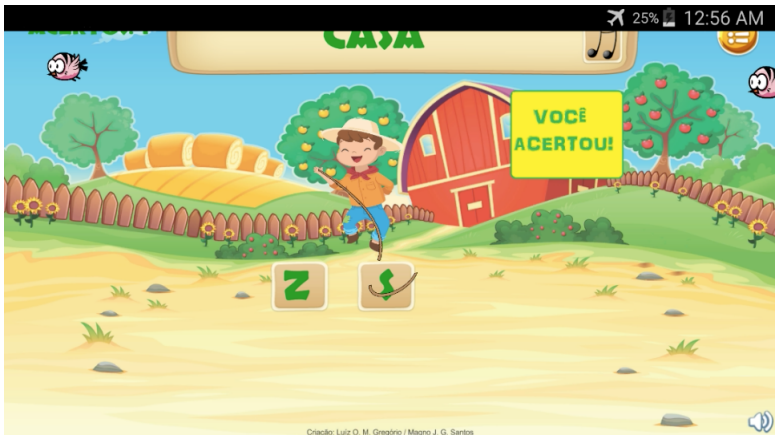
[1] Desativa todos os recursos sem fio do dispositivo.

Figura 4.13 – Adicionando usuário sem conexão com servidor



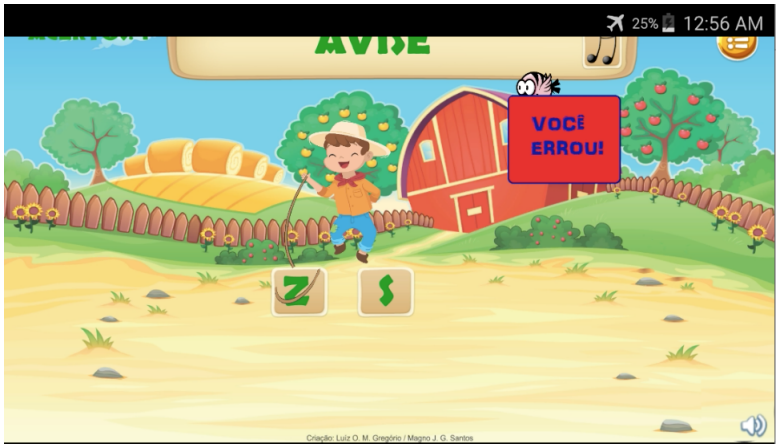
Fonte: Elaborada pelo autor

Figura 4.14 – Acerto da palavra Casa



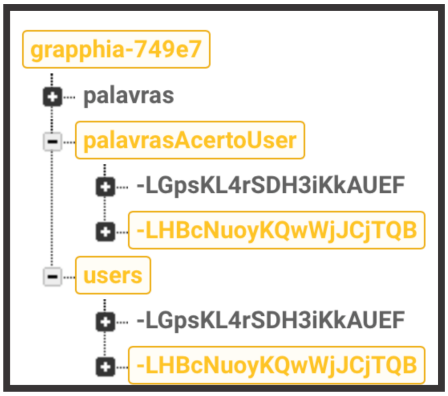
Fonte: Elaborada pelo autor

Figura 4.15 – Erro da palavra Avise



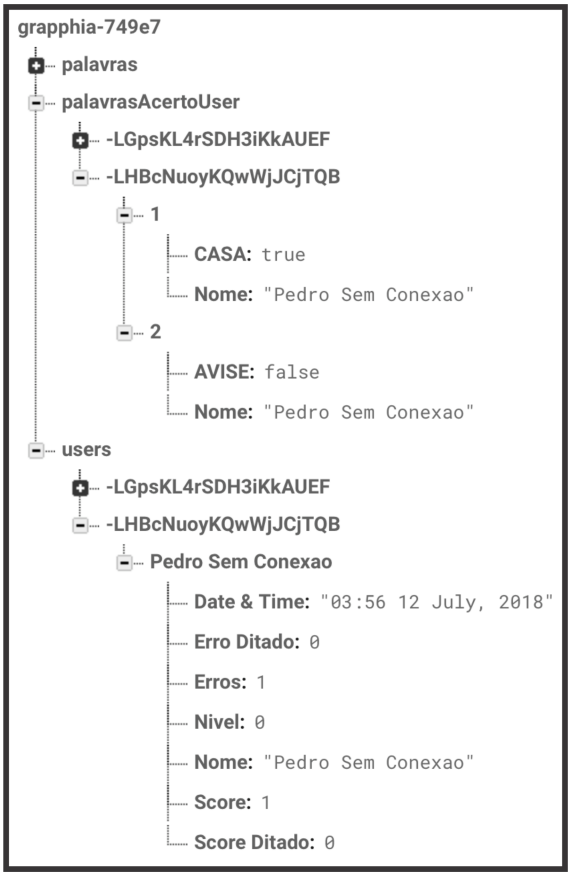
Fonte: Elaborada pelo autor

Figura 4.16 – Instante que conexão é restabelecida com internet no dispositivo móvel



Fonte: Elaborada pelo autor

Figura 4.17 – *Nodes* expandidos



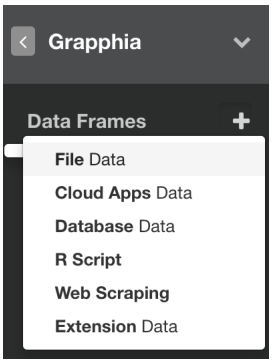
Fonte: Elaborada pelo autor

4.7 Análise dos dados do *node palavrasAcertoUser*

Para analisar os dados contidos no banco de dados do Firebase, é preciso exportá-los no formato JSON e importá-los no software Exploratory. Para importar o arquivo JSON é preciso seguir os seguintes passos:

1. Clicar no botão com o símbolo '+' do lado do *Data Frames* e seleciona *File Data*, como ilustra a figura 4.18;

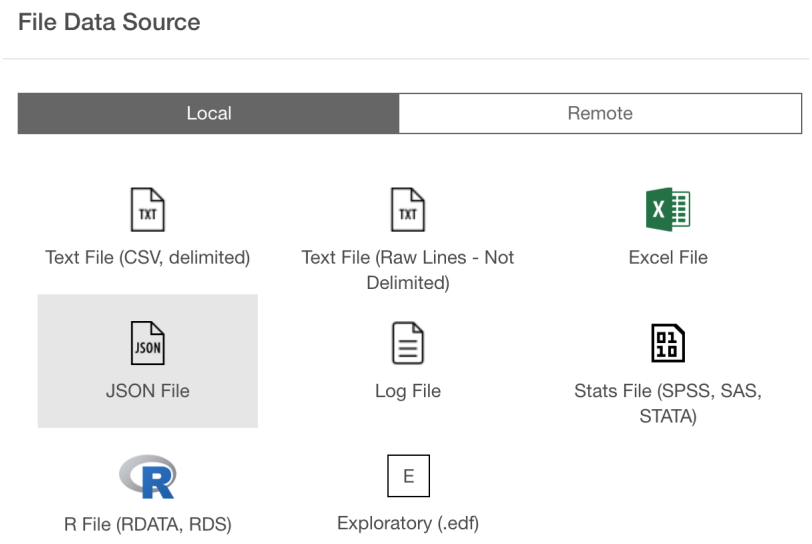
Figura 4.18 – Importar dados ao software Exploratory



Fonte: Elaborada pelo autor

2. Clique na aba *Local*, caso o arquivo JSON esteja no computador, ou a aba *Remote*, caso o arquivo esteja em um servidor remoto, e depois aperte em *JSON File*, mostrado na figura 4.19;

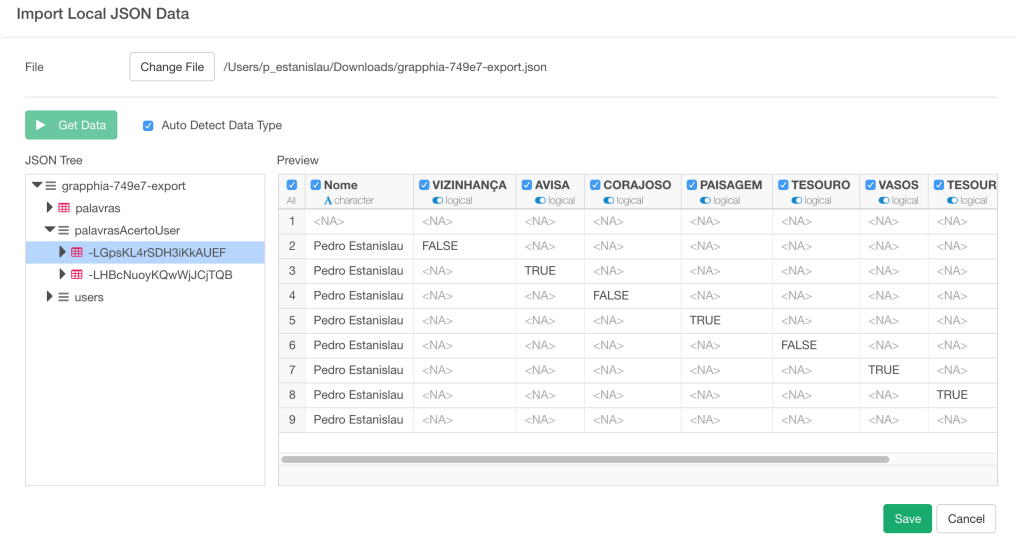
Figura 4.19 – Selecionar *JSON File*



Fonte: Elaborada pelo autor

3. Selecionar o *node* (cada *node* representa um dispositivo móvel) no menu da árvore JSON e selecionar as colunas no *preview* e depois clicar em *Save*, como mostra a figura 4.20;

Figura 4.20 – Selecionar *node* e colunas



Fonte: Elaborada pelo autor

4. Repetir este processo para os outros *nodes*, ou seja, os outros dispositivos móveis.

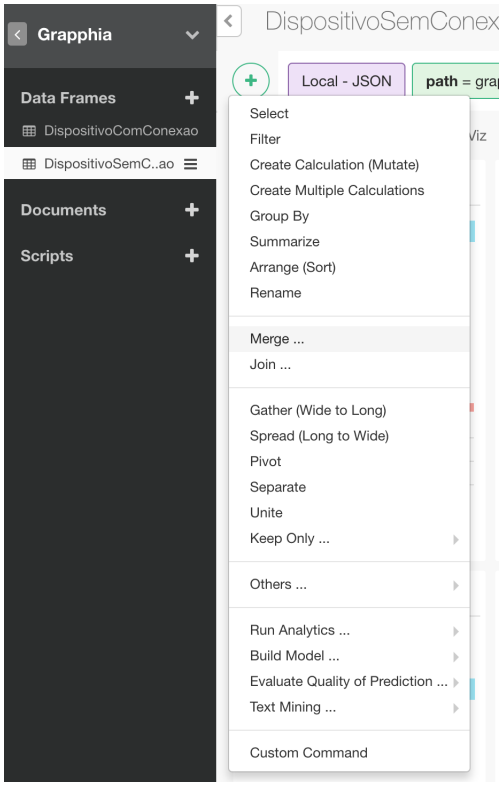
Após inserir todos os dispositivos móveis, a junção dos seus dados será útil para gerar informação para a pesquisa do Grapphia. Quantas vezes determinada palavra surgiu e as taxas de erros e acertos, são informações que podem ser extraídas da análise utilizando o software Exploratory. A junção dos dados é feita seguindo os seguintes passos:

1. Seleciona um dos *Data Frames* (dispositivos móveis) criados e aperta o símbolo '+' e seleciona a opção *Merge*, como ilustra a figura 4.21.

2. Escolher o tipo de operação para *Bind Rows (Merge all rows)* (fundir todas linhas) e incluir os outros *Data Frames* (dispositivo móveis), como mostra a figura 4.22;

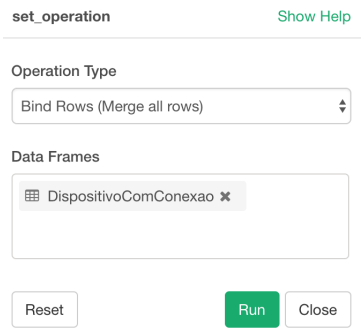
A figura 4.23 ilustra os dados após a junção dos *Data Frame*, ou dados dos dispositivos móveis.

Figura 4.21 – Selecionar *Merge*



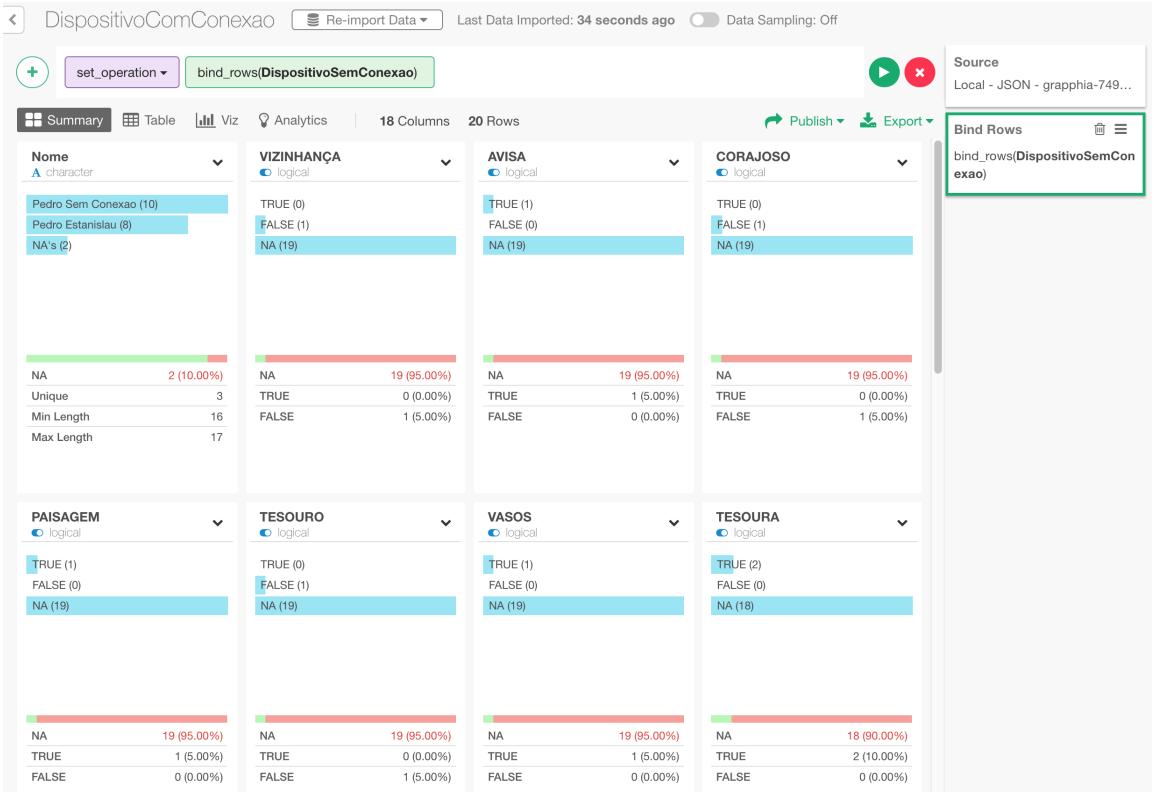
Fonte: Elaborada pelo autor

Figura 4.22 – Escolher tipo de operação e incluir *Data Frames*



Fonte: Elaborada pelo autor

Figura 4.23 – Dados exibidos após junção dos dados



Fonte: Elaborada pelo autor

5 CONCLUSÃO E TRABALHOS FUTUROS

Durante o processo de ensino-aprendizagem de ortografia da Língua Portuguesa, é recorrente a apresentação de dúvidas por parte dos educandos, principalmente, no que diz respeito ao grupo de palavras irregulares. Visando auxiliar nesse processo, foi desenvolvido o aplicativo para dispositivos móveis, Grapphia.

O presente trabalho objetivou integrar o aplicativo Grapphia, criado pelo software Unity, com o Firebase Realtime Database, um banco de dados hospedado na nuvem. Essa inserção, além de permitir a extração de dados, também possibilita a análise futura dos dados obtidos, com a ajuda de um software.

A integração da plataforma Firebase foi feita através da leitura dos guias e documentação disponibilizadas pelo Firebase. Após esta leitura, foi criado o projeto Grapphia no *Console* do Firebase e depois inserido o SDK para o software Unity do Firebase ao projeto do Grapphia no software Unity. Posteriormente, foi feita a configuração do SDK para o software Unity, sendo inseridas as bibliotecas, classes e linhas de códigos nos *scripts*. Depois dessas inserções foram feitos os testes de inserção e remoção dos dados e, consequentemente, a extração e análise dos mesmos.

Considerando o objetivo proposto, conclui-se que a integração da plataforma Firebase com o software Unity foi realizada com sucesso, além disso, a análise dos dados foi facilitada pela utilização do software Exploratory.

Para que a integração do software Unity com a plataforma Firebase não fique restringida somente ao serviço do Firebase Realtime Database, os possíveis trabalhos futuros são apresentados a seguir:

- Desenvolver código para utilizar a funcionalidade de recuperação de dados do banco de dados do Firebase e assim futuramente eliminando o uso de banco de dados interno;
- Integrar o serviço Firebase Authentication para aumentar a segurança do banco de dados;
- Integrar o serviço Firebase Analytics;
- Desenvolver o aplicativo Grapphia para a plataforma iOS e usufruir a plataforma Firebase em conjunto.

REFERÊNCIAS

ALVARENGA, D. Análise de variações ortográficas. **Revista Presença Pedagógica**, 1995.

ASSIS, L.; BODOLAY, A.; GREGÓRIO, L.; SANTOS, M.; VIVAS, A.; PITANGUI, C.; BANDEIRA, D. Grapphia: Aplicativo para dispositivos móveis para auxiliar o ensino da ortografia. **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**, v. 6, n. 1, p. 609, 10 2017.

BREWER, E. A. Towards robust distributed systems. In: **PODC**. [S.l.: s.n.], 2000. v. 7.

EXPLORATORY. **Getting Started Guide: Inspect Data with Summary View**. 2018. Acesso em: 20 jul. 2018. Disponível em: ⟨https://docs.exploratory.io/tutorials/quick-start.html⟩.

FIREBASE. **Como adicionar o Firebase ao seu projeto do Unity**. 2018. Acesso em: 01 mai. 2018. Disponível em: ⟨https://firebase.google.com/docs/unity/setup?authuser=0⟩.

FIREBASE. **Documentação do Firebase**. 2018. Acesso em: 01 mai. 2018. Disponível em: ⟨https://firebase.google.com/docs/⟩.

FIREBASE. **Estruturar o banco de dados**. 2018. Acesso em: 27 jun. 2018. Disponível em: ⟨https://firebase.google.com/docs/database/unity/structure-data⟩.

FIREBASE. **Firebase Realtime Database**. 2018. Acesso em: 01 mai. 2018. Disponível em: ⟨https://firebase.google.com/docs/database/⟩.

FIREBASE. **Gravar dados off-line**. 2018. Acesso em: 01 mai. 2018. Disponível em: ⟨https://firebase.google.com/docs/database/unity/save-data#write_data_offline⟩.

FIREBASE. **Primeiros passos com as regras de bancos de dados**. 2018. Acesso em: 27 jun. 2018. Disponível em: ⟨https://firebase.google.com/docs/database/security/quickstart⟩.

FIREBASE. **Primeiros passos com o Firebase Realtime Database para Unity**. 2018. Acesso em: 01 mai. 2018. Disponível em: ⟨https://firebase.google.com/docs/database/unity/start⟩.

FOTACHE, M.; COGEAN, D. Nosql and sql databases for mobile applications. case study: Mongodb versus postgresql. **Informatica Economica**, INFOREC Association, v. 17, n. 2, p. 41, 2013.

GANDHEWAR, N.; SHEIKH, R. Google android: An emerging software platform for mobile devices. **International Journal on Computer Science and Engineering**, v. 1, n. 1, p. 12–17, 2010.

GARTNER. **Gartner Says Worldwide Sales of Smartphones Returned to Growth in First Quarter of 2018**. 2018. Acesso em: 19 jun. 2018. Disponível em: ⟨https://www.gartner.com/newsroom/id/3876865⟩.

GASPAR, W.; OLIVEIRA, E. H. T.; OLIVEIRA, K. M. T. Aprendizagem da língua portuguesa com dispositivos móveis: Um mapeamento sistemático da literatura. In: **Anais do XXVI Simpósio Brasileiro de Informática na Educação**. [S.l.: s.n.], 2015.

GOLLNER, A. P. Comunicação mercadológica em dispositivos móveis: proposições teóricas e aplicações contemporâneas. **XIX Congresso de Ciências da Comunicação na Região Sudeste**, 2014.

IBGE. **Acesso à Internet e à Televisão e Posse de Telefone Móvel Celular para Uso Pessoal**. 2016. Acesso em: 18 jun. 2018. Disponível em: ⟨ftp://ftp.ibge.gov.br/Trabalho_e_Rendimento/Pesquisa_Nacional_por_Amostra_de_Domicilios_continua/Anual/Acesso_Internet_Televisao_e_Posse_Telefone_Movel_2016/Analise_dos_Resultados.pdf⟩.

JSON. **Introducing JSON**. 1999. Acesso em: 27 jun. 2018. Disponível em: ⟨https://www.json.org/json-pt.html⟩.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237–285, 1996.

LEAL, A. S. **Aplicação do método de aprendizagem por reforço q-learning na adaptatividade dinâmica de dificuldade de um jogo digital ortográfico**. Trabalho de Conclusão de Curso, 2016.

LECHETA, R. R. **Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. [S.l.]: Novatec Editora, 2013.

LEMLE, M. **Guia Teórico do Alfabetizador**. [S.l.]: Editora Ática, 1988.

LIMA, E.; REIS, E. C# e. net–guia do desenvolvedor. **Rio de Janeiro: Campus**, 2002.

NAYAK, A.; PORIY, A.; POOJARY, D. Type of nosql databases and its comparison with relational databases. **International Journal of Applied Information Systems**, v. 5, n. 4, p. 16–19, 2013.

OWENS, M. **The definitive guide to SQLite**. [S.l.], 2006.

POKORNY, J. Nosql databases: a step to database scalability in web environment. **International Journal of Web Information Systems**, Emerald Group Publishing Limited, v. 9, n. 1, p. 69–82, 2013.

PRITCHETT, D. Base: An acid alternative. **Queue**, ACM, v. 6, n. 3, p. 48–55, 2008.

SANTOS, V. C. **Aplicação do algoritmo SARSA no balanceamento dinâmico de dificuldade de um jogo digital ortográfico**. Trabalho de conclusão de curso, 2016.

SAREEN, P. Cloud computing: types, architecture, applications, concerns, virtualization and role of it governance in cloud. **International Journal of Advanced Research in Computer Science and Software Engineering**, v. 3, n. 3, 2013.

SHARMA, V.; DAVE, M. Sql and nosql databases. **International Journal of Advanced Research in Computer Science and Software Engineering**, v. 2, n. 8, 2012.

SQLITE. **SQLIte is Serveless**. 2017. Acesso em: 21 mai. 2018. Disponível em: ⟨https://www.sqlite.org/serverless.html⟩.

SQLITE. **SQLite is Transactional**. 2017. Acesso em: 21 mai. 2018. Disponível em: ⟨https://www.sqlite.org/transactional.html⟩.

THESAURUS, C. A. L. D. . **Back-end**. Cambridge University Press, 2018. Acesso em: 17 jul. 2018. Disponível em: ⟨https://dictionary.cambridge.org/dictionary/english/back-end⟩.

THESAURUS, C. A. L. D. . **Front-end**. Cambridge University Press, 2018. Acesso em: 17 jul. 2018. Disponível em: ⟨https://dictionary.cambridge.org/dictionary/english/front-end⟩.

UNITY. 2017. Www.unity3d.com.

UNITY. **Game engines—how do they work?** 2018. Acesso em: 11 jun. 2018. Disponível em: ⟨https://unity3d.com/what-is-a-game-engine⟩.

UNITY. **Scripting**. 2018. Acesso em: 11 jun. 2018. Disponível em: ⟨https://docs.unity3d.com/Manual/ScriptingSection.html⟩.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992.

XIE, J. Research on key technologies base unity3d game engine. In: IEEE. **Computer Science & Education (ICCSE), 2012 7th International Conference on**. [S.l.], 2012. p. 695–699.