CENTRO UNIVERSITÁRIO FACVEST
CURSO DE CIÊNCIA DA COMPUTAÇÂO
ELISON NILSON COELHO

ARQUITETURA MICROSSERVIÇOS APLICADA A APLICATIVOS MÓVEIS

LAGES
2020

ELISON NILSON COELHO

ARQUITETURA MICROSSERVIÇOS APLICADA A APLICATIVOS MÓVEIS

Projeto apresentado à Banca Examinadora do Trabalho de Conclusão de Curso II de Ciência da Computação para análise e aprovação.

Orientador: Prof. Willen Leolatto Carneiro

Coorientador: Prof. Igor Muzeka

LAGES
2020

ELISON NILSON COELHO

ARQUITETURA MICROSSERVIÇOS APLICADA A APLICATIVOS MÓVEIS

Trabalho de Conclusão de Curso de Ciência da Computação apresentado ao Centro Universitário UNIFACVEST como parte dos requisitos para obtenção do título de bacharel em Ciência da Computação.

Orientador: Prof. Willen Leolatto Carneiro

Coorientador: Prof. Igor Muzeka

Lages, SC ___/___/2020.

Nota_____

_____

Márcio José Sembay

LAGES
2020

AGRADECIMENTOS

Agradeço primeiramente a Deus por todas as oportunidades na vida e por ter iluminado o meu caminho.

Aos meus pais por todo o apoio, confiança e incentivo durante todos os meus estudos até agora.

Aos professores e orientadores por ter mostrado o caminho da informação, e ensinar todo o necessário.

E a todos os meus amigos por terem ajudado desde o início nos momentos de dificuldade.

RESUMO

Este projeto tem como objetivo criar uma arquitetura de desenvolvimento para aplicativos móveis que rodam em sistemas operacionais Android e iOS, unindo a tecnologia React Native com padrões de Microsserviços e *Micro Frontends*, afim de permitir a reutilização de código e funcionalidades entre projetos. A pesquisa foi realizada com as metodologias de pesquisa exploratória, tecnológica e bibliográfica, demonstrando como funciona a arquitetura monólito, que é uma forma comumente usada para desenvolvimento de aplicações, as dificuldades que este modelo traz e as soluções para a resolução dos problemas trazidos por esta arquitetura. O projeto proposto é uma aplicação protótipo que tem como objetivo demonstrar o funcionamento da arquitetura criada, observando também os benefícios trazidos pela junção destas tecnologias. A pesquisa busca resolver os problemas gerados pela arquitetura monólito, criando uma forma de desenvolvimento aplicável em qualquer empresa que queira otimizar o desenvolvimento das suas aplicações móveis, permitindo o compartilhamento de funcionalidades, melhora da manutenção do código, atualizações através da rede, e agilidade ao processo de desenvolvimento.

Palavras-chave: Arquitetura, Microsserviços, Aplicativos Móveis.

ABSTRACT

This project aims to create a development architecture for mobile applications that run on Android and iOS operating systems, combining React Native technology with Microservices and Micro Frontends standards, in order to allow the reuse of code and functionality between projects. The research was carried out using exploratory, technological and bibliographic research methodologies, demonstrating how monolith architecture works, which is a commonly used form for application development, the difficulties that this model brings and the solutions to solve the problems brought by this architecture. The proposed project is a prototype application that aims to demonstrate the functioning of the architecture created, also observing the benefits brought by the combination of these technologies. The research seeks to solve the problems generated by the monolith architecture, creating a form of development applicable in any company that wants to optimize the development of its mobile applications, allowing the sharing of functionalities, improvement of code maintenance, updates through the network, and agility to the development process.

Keywords: *Architecture, Micro Services, Mobile Apps.*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	*Application Programming Interface*
ARM	*Advanced* RISC *Machine*
CSS	*Cascading Style Sheet*
GPU	*Graphics Processing Unit*
HTML	*Hypertext Markup Language*
IDE	*Integrated Development Environment*
iOS	Sistema Operacional Móvel do iPhone
iPadOS	Sistema Operacional Móvel do iPad
JSON	*JavaScript Object Notation*
NoSQL	Banco de dados não relacional.
NFC	*Near Field Communication*
PIN	*Personal Identification Number*
PWA	*Progressive Web App*
REST	*Representational State Transfer*
tvOS	Sistema Operacional Móvel do Apple TV
UML	*Unified Modeling Language*
watchOS	Sistema Operacional Móvel do Apple Watch
WEB	*World English Bible*

SUMÁRIO

1 INTRODUÇÃO

No desenvolvimento de aplicativos móveis (Android e iOS), a arquitetura monólito quando utilizada em grandes projetos, começa a apresentar aos desenvolvedores e empresas, algumas dificuldades que prejudicam a velocidade com que as aplicações são mantidas e desenvolvidas, como dificuldade de escala, aproveitamento de código, entre outros (PONCE, MÁRQUEZ, ASTUDILLO 2019).

Esta arquitetura é definida pelo autor Newman (2019, p.12) como sendo um único e grande bloco de código executado no mesmo processo tendo toda a escrita centralizada a um único banco de dados, sendo uma forma comum de criação de sistemas.

Muitas empresas buscam encontrar uma forma de solucionar as dificuldades trazidas pelo monólito. Existem algumas formas de solucionar estes problemas, uma delas é com o React Native, que é uma ferramenta de desenvolvimento multiplataforma onde Dabit (2019), afirma que esta metodologia permite compartilhar uma mesma base de código para criar aplicações Android e iOS, e outra forma que traz muitos benefícios é a arquitetura de microsserviços, que por sua vez traz uma forma de deixar o sistema modular separando o código de acordo com a sua responsabilidade (PONCE, MÁRQUEZ, ASTUDILLO 2019).

Combinando estas técnicas, será possível criar um novo padrão de desenvolvimento que trará muitos benefícios e promete alavancar o desenvolvimento de aplicações móveis e solucionar todos ou a maioria dos problemas citados acima.

1.1 Justificativa

Este estudo terá vários benefícios que o padrão que aqui será criado vem a proporcionar, partindo de uma vontade pessoal de se aprofundar no estudo e aprendizado no desenvolvimento de aplicações para dispositivos móveis de uma forma eficiente, pois criar um padrão que solucione os problemas de escala, compartilhamento de código, manutenção, entre outros, trará grandes ganhos em todo o processo de desenvolvimento de diversas empresas.

Este protótipo justifica-se também por já possuir um primeiro local de testes que irá utiliza-lo como base para o desenvolvimento de aplicações móveis, a NDDigital[1], que é uma empresa que passa pelas dificuldades da utilização da arquitetura monólito em seus sistemas de

[1] A NDDigital é uma empresa situada em Lages – Santa Catarina, que desde 2004 está no mercado de *softwares*, são especialistas no desenvolvimento e na implantação de *softwares* para gestão de outsourcing de impressão e documentos fiscais eletrônicos.

outras tecnologias e que busca iniciar no desenvolvimento de soluções móveis já começando com uma forma otimizada, reduzir o desperdício no desenvolvimento dos seus aplicativos móveis e escalável para novas funcionalidades.

1.2 Importância

Tendo o primeiro local de testes como exemplo, a NDDigital, não existe hoje algo que traga os benefícios aqui procurados, a arquitetura de microsserviços que traz várias vantagens é utilizada apenas em seus sistemas *web,* e desejando aplicar este modelo nas suas soluções para dispositivos móveis, busca-se um estudo como este para identificar as dificuldades de implantar este padrão e os seus desafios.

Com isto em consideração, além de todo o aprendizado que é possível obter com as tecnologias aqui empregadas, realizar um estudo que solucione problemas como acoplamento de código, bases de código separadas para manter, uma para Android e outra para iOS, dificuldade de compartilhamento entre equipes e dificuldade de manutenção de código em arquiteturas monólito, trará grandes ganhos para alavancar a produção de aplicativos móveis em times de desenvolvimento.

2 OBJETIVOS

2.1 Objetivo Geral

Criar uma arquitetura padrão de desenvolvimento de aplicativos móveis para homogeneizar o desenvolvimento entre vários times e possibilitar escala.

2.2 Objetivos Específicos

a) Identificar como a arquitetura de microsserviços pode ser aproveitada para criação de aplicativos móveis.
b) Criar uma arquitetura base que possibilite a reutilização de código e de funcionalidades, para que times de desenvolvimento possam reduzir desperdício, tempo de desenvolvimento, entre outros.
c) Desenvolver um aplicativo protótipo utilizando a arquitetura definida, afim de mostrar o seu funcionamento e provar a sua viabilidade.

3 REVISÂO DE LITERATURA

Para o entendimento de como funciona a arquitetura de microsserviços e os seus conceitos, primeiramente será explicado um pouco sobre a arquitetura que ela se propõe a ser uma alternativa: o monólito.

3.1 Arquitetura Monólito

O monólito é uma arquitetura clássica onde toda a aplicação é instalada como uma só, ou seja, uma instância da aplicação contém todo o código que faz a leitura e a escrita em um único banco de dados, sendo este um exemplo clássico onde a maioria dos sistemas se enquadram (NEWMAN, 2019).

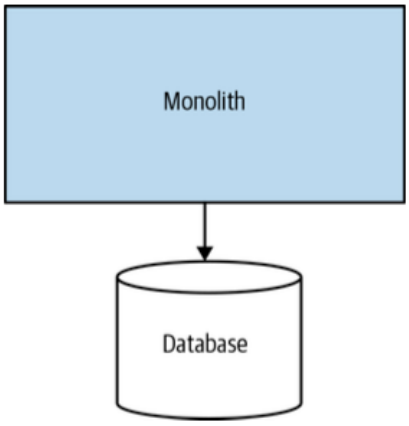

Figura 1: Diagrama de uma arquitetura monólito clássica.
Fonte: NEWMAN, 2019, p.12

A partir da arquitetura clássica do monólito, foram criadas algumas variações, afim de criar uma estrutura mais organizada e reduzir as amarrações, como explica Newman (2019, p.13) que onde uma das variações, é o monólito modular, que para muitas organizações, pode ser uma escolha melhor que a monólito clássica, pois se feito com um bom isolamento entre os módulos, pode se chegar a um bom nível de trabalho paralelo, mas que mesmo assim, não consegue ter todas as vantagens de uma arquitetura de microsserviços, tendo ainda a desvantagem de que o banco de dados continua sendo único, gerando dificuldades ao remover ou adicionar novos módulos.

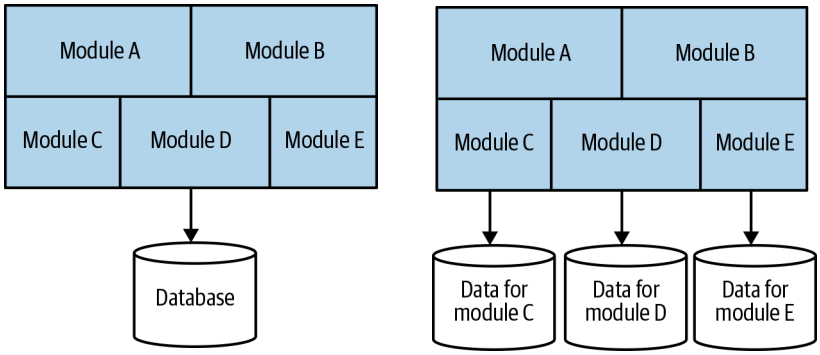

Figura 2: Estratégia de banco de dados único ou divididos.
Fonte: NEWMAN, 2019, p.13 e p.14

Newman (2019, p.14) ainda comenta que existe uma variação que busca contornar a dificuldade do único banco de dados citado acima, que é a criação de bancos de dados específicos para cada módulo, mas comenta também que essa arquitetura acaba criando mais dificuldades e herdando os problemas do monólito clássico, e do modular.

Essas arquiteturas e variações, dão a impressão de estarem mais organizadas, mas continuam dando abertura para amarrações entre os módulos, e para grandes aplicações corporativas com um grande time ou com mais de um time trabalhando na mesma solução, duas ou mais pessoas podem acabar alterando o mesmo código, por isso, esta pode não ser a melhor estratégia.

Olhando para o lado positivo da arquitetura, descobre-se que esta arquitetura funciona bem em pequenos projetos que não irão crescer e que não tem uma projeção de escala para o futuro, Richardson (2019, p.4) explica que em projetos pequenos, observa-se os seguintes benefícios:

- Facilita o desenvolvimento, no sentido de não ser necessário se preocupar em separar as funcionalidades em módulos.
- Facilidade de testar, é possível criar um teste automatizado que vai do início ao fim da execução do sistema.
- O sistema também fica simples de instalar, gerando um único pacote.

Richardson (2019, p.4) explica também que a grande limitação do monólito começa a se apresentar quando o sistema começa a crescer de tamanho, impulsionado pelo seu sucesso inicial, onde novas funcionalidades começam a ser inseridas e o tamanho do time e do projeto aumenta, o desenvolvimento começa então a se tornar lento e difícil.

Ponce, Márquez, Astudillo (2019) citam que as aplicações que tem um grande crescimento e seguem o padrão monólito, costumam a apresentar as seguintes dificuldades:

- Dificuldade de entender e de modificar, tornando o desenvolvimento mais lento.
- Prejudica a melhoria contínua, pois uma alteração em um ponto, requer que todo o sistema tenha que ser recompilado e distribuído.
- É uma arquitetura que compromete a escalabilidade
- Dificuldade em mudar a tecnologia do projeto, precisando reescrever tudo do zero, ou seja, requer o comprometimento com a mesma tecnologia do início ao fim.

Pode se determinar com os pontos acima que que a arquitetura de monólito pode apresentar ao desenvolvedor dificuldades no desenvolvimento e inclusão de novas funcionalidades e consequentemente aumentando a incidência de defeitos no código.

3.2 Arquitetura de Microsserviços

Tendo em vista os problemas apresentados pelo desenvolvimento em monólitos, foi necessário encontrar uma outra forma de criar aplicações sem as dificuldades acima e de forma mais organizada e escalável, a solução em potencial é a arquitetura de microsserviços, onde Ponce, Márquez, Astudillo (2019, p.1) definem esta arquitetura da seguinte forma:

> A arquitetura de Microsserviços (MAS) é uma forma de desenvolver uma única aplicação como um conjunto de pequenos serviços, cada um rodando em seu próprio processo e se comunicando com mecanismo leve, por exemplo, uma *API HTTP* (PONCE, MÁRQUEZ, ASTUDILLO (2019, p.1).

Logo é possível entender que desta forma a aplicação é dividida em várias partes em que cada uma delas funciona de forma independente uma das outras.

Os autores explicam que para acompanhar a rápida e grande demanda do mercado por novas funcionalidades, acaba por ser necessário algumas evoluções na forma de desenvolver aplicações, que são necessárias para atender a demanda, uma delas é o baixo acoplamento de código e alta escalabilidade do sistema, e a outra é remover dependências de time e ter uma rápida distribuição.

Ponce, Márquez, Astudillo (2019) explica que a Arquitetura de microsserviços resolve estes pontos, pois como cada funcionalidade fica isolada em seu próprio módulo, não é feito um acoplamento de código em pontos desnecessários entre elas, o que naturalmente acaba

melhorando a escalabilidade, pois adicionar e remover funcionalidades se torna muito mais fácil, não precisando alterar o que já está funcionado.

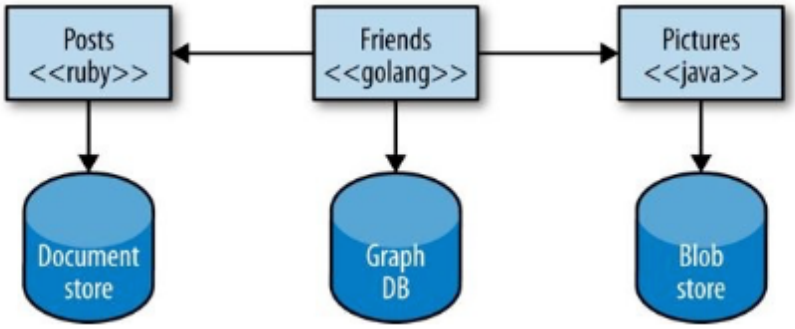

Figura 3: Microsserviços usando tecnologias diferentes em cada módulo.
Fonte: NEWMAN, 2015, p.6

Os Autores finalizam a explicação comentando que esta arquitetura também possibilita remover dependências de time, pois como segue uma arquitetura modular e previsível, um desenvolvedor de outro time não precisa conhecer o código da solução completa para adicionar uma nova funcionalidade, o que reduz consideravelmente o tempo de distribuição.

Estes pontos positivos são muito importantes e times com aplicações com alta demanda claramente teriam benefício com ela.

Entretanto existem também alguns pontos negativos nesta abordagem que são necessários levar em consideração, Ghofrani, Lübke (2018) descobriu em seu estudo algumas dificuldades em se trabalhar desta forma, que são oriundas da própria natureza de distribuição de módulos.

Ghofrani, Lübke (2018) apontou em primeiro lugar o aumento do número de repositórios para manter, visto que cada módulo é um projeto diferente, onde cada um deles necessita de manutenção, outra dificuldade mencionada é a comunicação e o compartilhamento de dados entre os módulos, que por natureza se torna mais restrito, visto que a comunicação entre os módulos fica mais pontual, necessitando de uma estrutura bem definida, também foi encontrado em alguns momentos uma certa complexidade na depuração de dois módulos em conjunto e por ultimo a própria habilidade técnica e conhecimento da arquitetura em si.

A partir dos argumentos dos autores acima, observa-se também que visto que esta é uma arquitetura amplamente utilizada em desenvolvimento *Web*, existe também o grande desafio de por em prática esta forma de desenvolvimento para dispositivos móveis, onde este trabalho ajudará a encontrar formas de tirar proveito desta forma de desenvolvimento.

3.3 Arquitetura de *Micro Frontends*.

A arquitetura de Microsserviços é comumente utilizada no *backend* da aplicação, modularizando o código que roda no lado do servidor, Jackson (2019) explica que ainda existem empresas construindo monólitos na camada da interface, ou seja, o *backend* é modularizado seguindo os padrões do microsserviços mas ainda existindo um grande acoplamento no lado da interface gerando dificuldades que através destas surgiu a arquitetura de *micro frontends*, onde Jackson (2019) define como sendo uma arquitetura que permite a entrega de aplicações *frontend* de forma independente, agregando a funcionalidades existentes no lado da interface do usuário.

Geers (2020) exemplifica muito bem os cenários encontrados utilizando *frontends* que utilizam monólitos com a figura 4 abaixo.

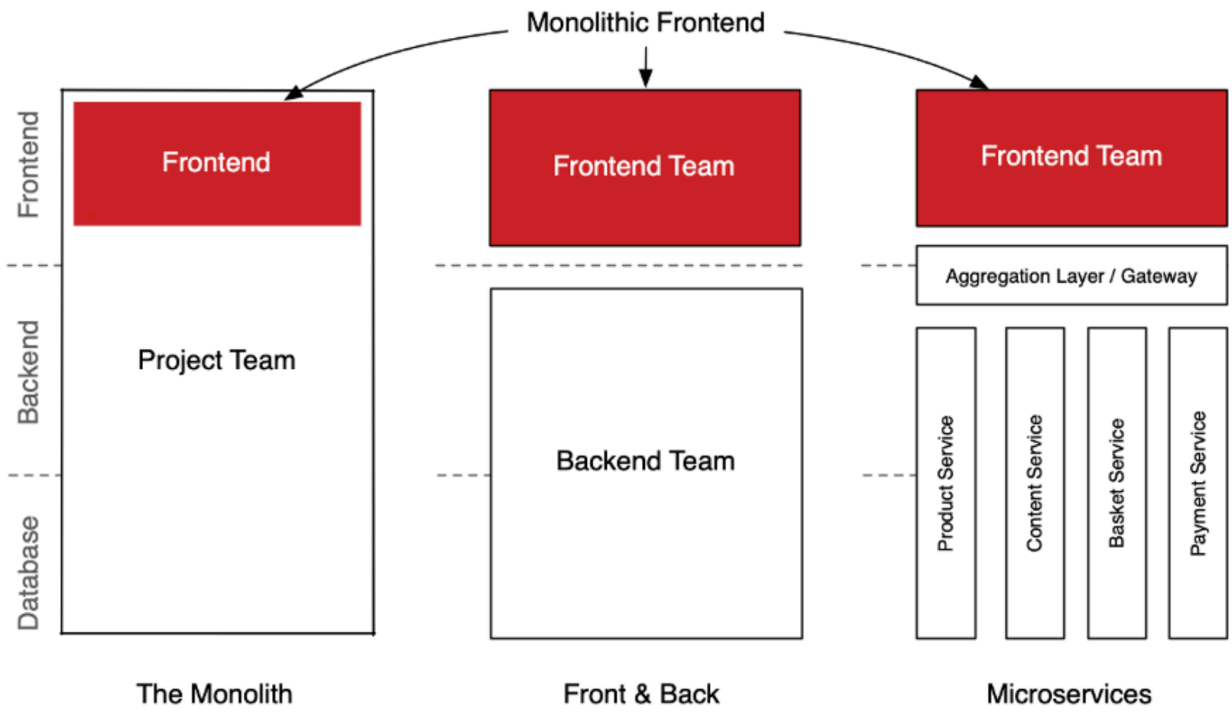

Figura 4: Cenários de arquiteturas com monólitos no *Frontend*.

Fonte: GEERS, 2020

Geers (2020) continua explicando que aplicando a arquitetura de *micro frontends*, no lugar dos monólitos, possibilita delegar cada *micro frontends* criado para um time diferente, o que gera vários benefícios, como a entrega de forma independente, isolamento de risco para apenas o escopo daquele *micro frontend*, e como é feito a quebra em pequenas partes, facilita a manutenção para o time pois este tem menos código para manter.

Unindo as implementações de *micro frontends* com microsserviços, é possível criar uma abordagem de desenvolvimento chamada de *End-To-End Teams,* onde um time consegue cuidar do frontend até o backend.

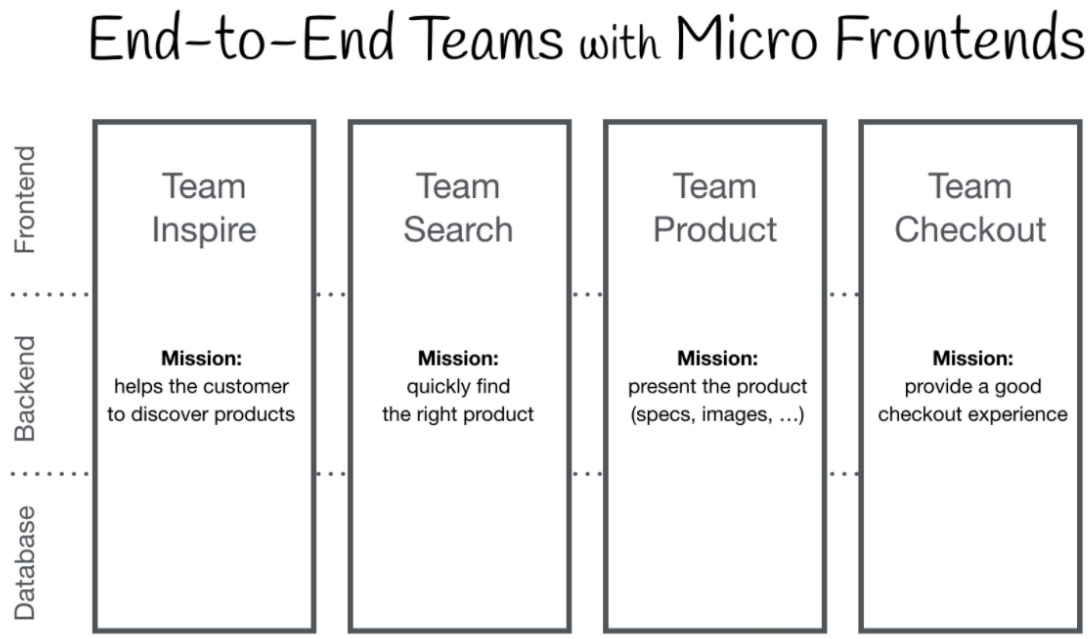

Figura 5: Visualização da arquitetura *End-to-End Teams*.
Fonte: GEERS, 2020

Resultando então uma separação completa de cada funcionalidade, que pode ser produzida por times diferentes e entregue em tempos diferentes de forma independente uma das outras, possibilitando o compartilhamento de funcionalidades entre sistemas e melhor organização geral do sistema.

3.4 Desenvolvimento Multiplataforma

Uma parte muito importante para alcançar uma maior velocidade de desenvolvimento e menos desperdício de código, é a utilização de uma ferramenta de desenvolvimento multiplataforma, onde um dos seus principais objetivos, é ter uma única base de código e compilar para Android e iOS.

Existem várias ferramentas para utilização hoje, algumas são mais eficientes e geram uma aplicação de maior qualidade no final, aqui será descrito as principais.

De acordo com Payne (2019), podemos classificar estas ferramentas em 3 categorias, *Progressive Web Apps* (PWA), hibridas, e soluções que compilam para código nativo (Nativo), nas primeira categoria, existe ferramentas como, HTML/CSS, React, Angular e Vue, o autor explica que estas ferramentas não geram uma aplicação real, elas são basicamente páginas *web* que tem uma aparência de um aplicativo e rodam através de um navegador, não sendo instaladas no aparelho, estas tem como deficiência a falta de acesso as funcionalidades físicas do aparelho, como alguns sensores, Bluetooth, NFC, entre outros, estas ferramentas, porém, tem a vantagem de serem fáceis de criar.

Na segunda categoria, na categoria Hibrida, existe as ferramentas PhoneGap, Cordova, Sencha e Ionic, estas por sua vez geram um aplicativo instalável, mas continuam ainda contidas dentro de uma *WebView*, ou seja, são componentes *Web* encapsulados no corpo de uma aplicação.

Na terceira categoria, o autor finaliza comentando que nesta, se encontram as ferramentas React Native, NativeScript, Flutter e Xamarin, que são as ferramentas que geram aplicações nativas, ou seja, na compilação, geram código nativo, utilizando componentes nativos do Sistema Operacional, o que resulta em maior velocidade de execução e qualidade, em comparação as ferramentas das categorias citadas acima, porém, são ferramentas que tem uma curva de aprendizado maior.

3.5 Ferramentas do projeto

As ferramentas utilizadas para o desenvolvimento do protótipo serão descritas neste capítulo.

3.5.1 React Native

O React Native é uma ferramenta de desenvolvimento de aplicativos móveis usando a linguagem JavaScript e biblioteca React JavaScript, que compila para componentes nativos, criado pelo Facebook tem como objetivo ajudar os desenvolvedores a criar aplicações com experiências nativas para Android e iOS a partir de uma única base de código, contando com uma interface gráfica fluida e bonita (DABIT, 2019).

Eisenman (2017) e Dabit (2019), explicam alguns benefícios que a ferramenta traz consigo, como a capacidade de utilizar a própria renderização do aparelho em que está sendo

executado, diferindo da maioria das outras ferramentas, como Cordova ou Ionic, que usam utilizam *WebViews* gerando uma perca de desempenho.

O React Native possui código fonte aberto, ou seja, não é necessário aguardar o time de desenvolvimento do Facebook para realizar correções quando é encontrado algum problema, a própria comunidade pode ajudar a realizar correções e a moldar rapidamente conforme o que os usuários do React Native precisam.

Outra característica positiva do React Native é que o mesmo tem acesso os componentes de interface também de forma nativa, enquanto outras ferramentas criam componentes que apenas imitam o visual e comportamento dos elementos de interface.

Tendo os comentários dos autores acima, entende se que se trata de uma ferramenta flexível, capaz de criar aplicações para Android e iOS, o que trás agilidade e redução no tempo de entrega para o cliente, e que compila para código nativo, ou seja, tem uma velocidade de execução de aplicativos nativos criados especificamente para iOS ou para Android, tudo isto com o Facebook dando suporte, logo, esta ferramenta também será muito importante para este projeto.

Os motivos que levaram a escolha do React native no lugar das outras ferramentas que compilam para código nativo, é descrito muito bem por Eisenman (2017), que comenta que como esta ferramenta utiliza JavaScript como linguagem, é muito fácil de aproveitar o código de outros sistemas que já utilizam esta linguagem, como produtos feitos para a Web, e também facilita a adaptação de desenvolvedores Web quando vão trabalhar com o React Native, pois já estão acostumados com o JavaScript, reduzindo a curva de aprendizado.

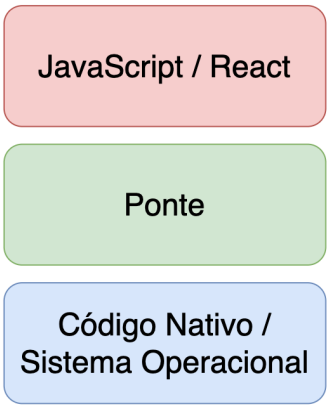

Figura 6: Arquitetura do React Native em resumo.

Fonte: Diagrama Próprio Autor.

3.5.2 Expo

Uma problema comum que ocorre no ambiente de desenvolvimento por exemplo, é quando a aplicação apresenta uma falha em produção, sendo preciso liberar uma atualização, porém para que esta atualização chegue ao usuário é preciso passar por um processo de validação e aprovação que as lojas App Store e Google Play requerem, e que de acordo com Google (2020) e Apple Inc. (2020) podem levar de 24 horas até sete dias para revisão, o que pode afetar negativamente a experiência do usuário, pois este processo irá acrescentar o tempo que o usuário final precisa esperar para receber a correção.

Para resolver este problema é utilizado o framework Expo, que possibilita entre outras funcionalidades, atualizações *Over The Air* ou OTA, que é uma forma de atualizar o código de produção sem passar pelo processo de revisão das lojas, ou seja, o usuário recebe a atualização assim que ela fica pronta.

O objetivo principal destas atualizações são correções de defeitos de código, mais especificamente a parte JavaScript da aplicação e respeitando as suas limitações, pois não é possível alterar versões de SDK, bibliotecas ou o código nativo do sistema operacional. (EXPO, 2020).

3.5.3 JavaScript

JavaScript é a uma linguagem de programação grandemente difundida e utilizada nas mais diversas plataformas incluindo o React Native, é uma linguagem leve orientada a objetos que utiliza o padrão ECMAScript, que é um padrão que da base para a linguagem e que os navegadores de internet hoje seguem, mas que não necessariamente precise de um navegador para funcionar. (MOZZILA, 2019)

```
1  const para = document.querySelector('p');
2
3  para.addEventListener('click', atualizarNome);
4
5  function atualizarNome() {
6    var nome = prompt('Insira um novo nome');
7    para.textContent = 'Jogador 1: ' + nome;
8  }
```

Figura 7: Exemplo de código da Linguagem de programação JavaScript.

Fonte: MOZZILA, 2019.

3.5.4 Firebase

Esta aplicação consumirá dados de uma REST API proveniente do Firebase, que é uma plataforma da Google que possuí diversas funcionalidades como banco de dados em tempo real, serviços que monitoração da qualidade do aplicativo, como monitoração de erros que venham a acontecer na aplicação, diferentes métricas para análise e acompanhamento de aplicativos, serviços de autenticação, entre outras funcionalidades. (GOOGLE, 2020).

3.5.5 Android

O Android é um sistema operacional baseado em Linux que funciona em *tablets*, *smartphones* e outros dispositivos móveis, seus aplicativos são escritos na linguagem de programação Java e/ou Kotlin, e compilados em ferramentas como o Eclipse ou Android Studio (DARWIN, 2012).

3.5.6 iOS

O iOS é o sistema operacional produzido pela Apple para o o iPhone, é um sistema operacional avançado que deu base a novas variações de sistemas operacionais como o iPadOS para o iPad, tvOS para o Apple TV, e watchOS para o Apple Watch, a interface gráfica destes sistemas se baseia no SwiftUI, que é uma nova camada de interface gráfica criada para facilitar o desenvolvimento de aplicações para todos os dispositivos da Apple, incluindo computadores Mac, seus aplicativos são escritos na linguagem de programação Objective-C e Swift (GAUCHAT, 2020).

3.5.7 Microsoft Visual Studio Code

A IDE escolhida para o desenvolvimento desta aplicação será o Visual Studio Code, que é uma IDE gratuita e extensível produzida pela Microsoft para ser utilizado por desenvolvedores independentes, estudantes, pequenos ou grandes times de programação, é uma ferramenta rápida e que roda nos mais diversos ambientes de desenvolvimento (MICROSOFT, 2020).

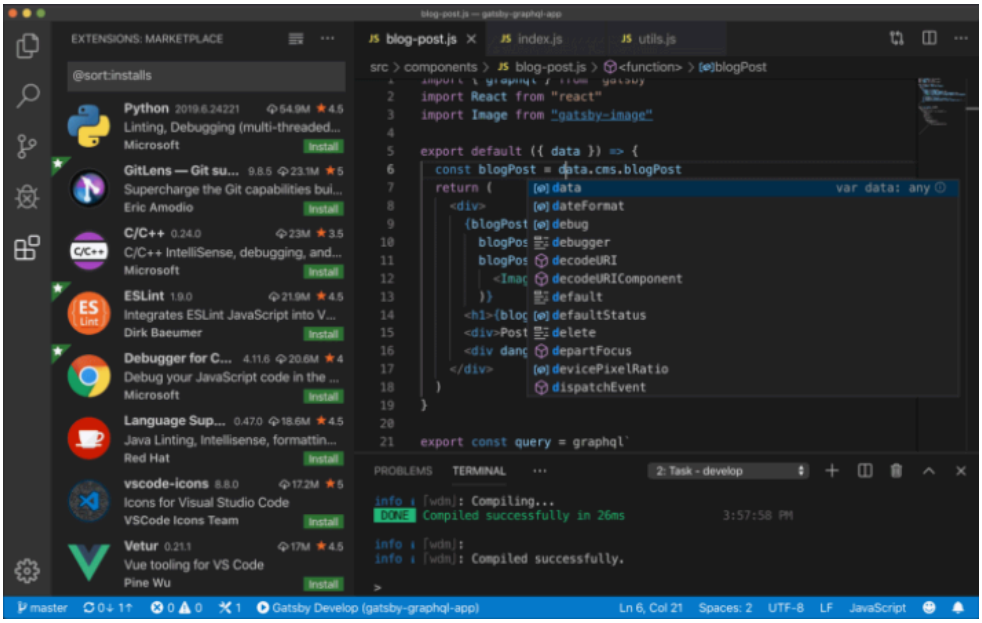

Figura 8: Tela da IDE Visual Studio Code e suas Extensões.

Fonte: Microsoft, 2020

3.5.8 Diagrams.net

Todos os diagramas deste trabalho, foram criados a partir do site Diagrams.net, que é uma ferramenta gratuita, *online* e *open-source* para criação de diagramas para várias finalidades, como UML, diagrama de Rede, fluxo, entre outros. (DIAGRAMS.NET, 2020)

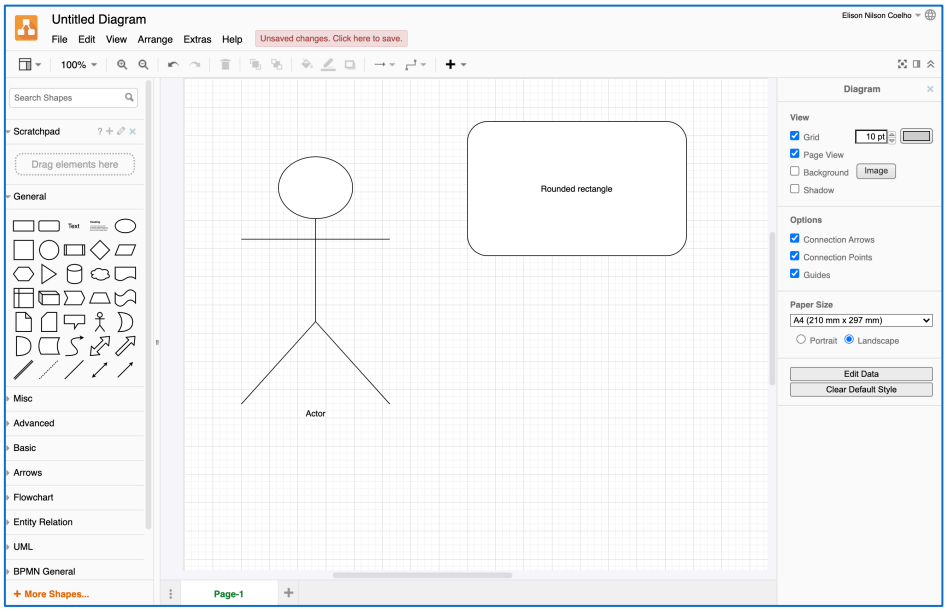

Figura 9: Tela do aplicativo online Diagrams.net.

Fonte: Diagrams.net, 2020

4 METODOLOGIA DA PESQUISA

A metodologia de pesquisa deste trabalho é de cunho tecnológico, e para que este trabalho pudesse ser realizado, foi necessário fazer pesquisas sobre o assunto abordado, por isto, baseando-se no método de pesquisa utilizado poderá ser este classificado em algumas áreas da Metodologia, que de acordo com Gil (2002, p. 41), estas classificações podem se encaixar em um dos três grandes grupos, exploratória, descritivas e explicativas.

4.1 Classificação da pesquisa

Para Gil (2002, p. 44) "A pesquisa bibliográfica é desenvolvida com base em material já elaborado, constituído principalmente de livros e artigos científicos", sendo a informação aqui contida, encontrada e entendia através de artigos científicos e livros, logo é bibliográfica, e também documental.

Esta pesquisa também se classifica como tecnológica, pois utiliza métodos e tecnologias já existentes de *software*, adicionando um tema para criar algo inovador (JUNG, 2003).

De acordo com as tecnologias escolhidas, tendo como base a arquitetura de microsserviços e o desenvolvimento multiplataforma, e também com a pesquisa, foi possível identificar como pode ser feito a união desses dois temas e aproveitar os benefícios dos dois, na produção de aplicativos móveis.

Gil (2002, p. 41), explica que uma pesquisa exploratória busca aperfeiçoar, ou aprimorar uma ideia, e tendo em vista os objetivos desta pesquisa, este trabalho se classifica como exploratória.

Tendo como resultado então, uma forma diferente de desenvolvimento, trazendo diversos benefícios no desenvolvimento de aplicativos móveis.

4.2 Limitações da pesquisa

a) Aplicações para dispositivos móveis são compiladas gerando um único pacote final para distribuição, mantendo a tecnologia homogenia por toda a aplicação, por isto uma das característica da arquitetura de Microsserviços não pode ser aplicada, que é a possibilidade de cada serviço rodar em uma tecnologia totalmente diferente uma da outra, porém esta limitação não irá afetar os principais benefícios que esta pesquisa procura.

5 CRONOGRAMA

Quadro 1 – Cronograma do projeto

Atividade	Fev.	Mar.	Abr.	Mai.	Jun.	Jul.	Ago.	Set.	Out.	Nov.	Dez.
Pesquisa do tema											
Busca de Livros e Artigos											
Escrita e Produção do Desenvolvimento											
Primeira Entrega: Desenvolvimento											
Escrita e Produção da Segunda Entrega											
Segunda Entrega											
Terceira Entrega											
Ajustes do projeto											
Desenvolvimento do protótipo											
Atualização da escrita											
Ajustes finais e Entrega do trabalho											
Defesa na Banca											

6 PROJETO

6.1 Introdução

Este projeto é constituído por um modelo de arquitetura que possibilita a criação de aplicações de forma escalável, descrito neste capítulo, e de um aplicativo genérico que implementa este modelo, sendo este um protótipo com funcionalidades comuns encontradas em outras aplicações, com o objetivo de demonstrar como é feita a implementação.

O aplicativo também demonstra a adição de uma nova funcionalidade ao projeto, afim de verificar como o sistema se comporta e também provar a capacidade da arquitetura de adicionar novas funcionalidades de forma fácil.

O aplicativo funciona em dois sistemas operacionais diferentes, Android e iOS, e para o seu desenvolvimento foi utilizado o React Native, que é uma tecnologia recente para possibilitar o desenvolvimento de aplicativos multiplataforma, trazendo consigo grandes vantagens na reutilização de código e velocidade de entrega.

Este aplicativo foi implementado seguindo os conceitos da arquitetura de microsserviços, resultando em um padrão de *micro frontends* adequado a realidade das aplicações móveis, tendo um desafio na descoberta de como esta arquitetura pôde ser utilizada em aplicações móveis.

Foi utilizado o JavaScript para a programação do sistema, e o Visual Studio Code para a IDE, sendo que esta é uma IDE que possui várias extensões para desenvolvimento de diversas aplicações, inclusive tendo uma específica para o React Native, que possibilita o desenvolvimento do mesmo.

Esta aplicação consulta os seus dados em bancos de dados mantidos em uma plataforma REST API hospedado na nuvem do Firebase, e como forma de armazenamento local, o *Async Storage,* que é um banco de dados simples de chave e valor rápido e confiável.

6.2 Estruturação da arquitetura

Observando a arquitetura de Microsserviços e Micro Frontends, e as traduzindo para dentro de uma aplicação React Native, teve-se como resultado a seguinte estruturação da figura 10 abaixo, no primeiro nível, existe um centralizador chamado de Pacotes, e dentro dele todas as funcionalidades da aplicação, criadas no projeto ou importadas de outras soluções, sendo que cada funcionalidade tem a sua API de comunicação, e as suas pastas da própria funcionalidade, como telas, componentes utilizados, a camada de modelo, repositório de comunicação com banco de dados e arquivos de configuração de dependências.

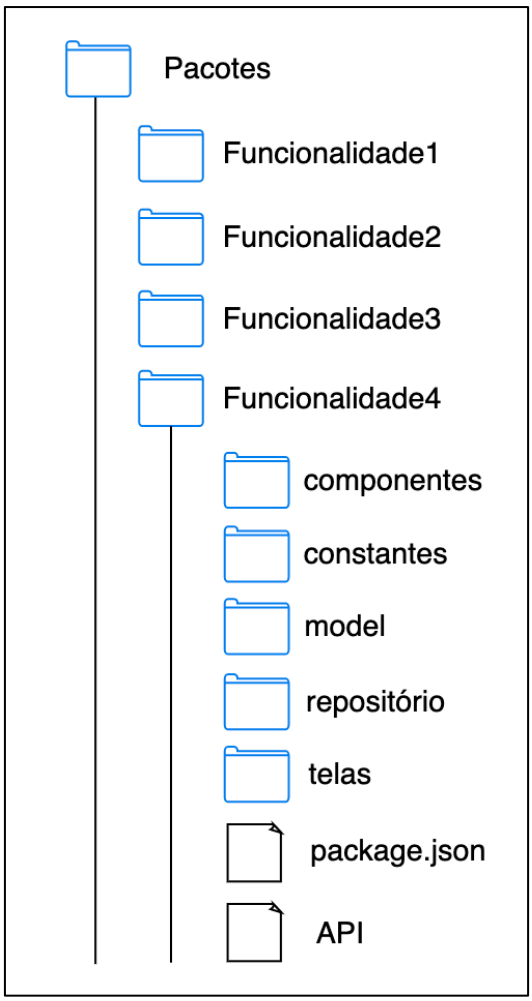

Figura 10: Exemplo de organização das funcionalidades e seus componentes.

Fonte: Próprio Autor

As pastas e componentes de cada funcionalidade são flexíveis, podendo variar de acordo com a necessidade de cada funcionalidade, apenas a API que não muda, pois descreve para as

outras funcionalidades os seus métodos que podem ser acessados, a API consiste de métodos que podem ou não ser assíncronos, e seguem a palavra chave *export* para sinalizar que o método pode ser importado em outra funcionalidade.

```
import User from "./model/User";
import { fetchUser, storeUser } from "./repository/database";

export async function authenticateUser(pin) {
  //implementacão
}

export async function getUserAuthenticated() {
//implementacão
}

export async function clearUserData() {
  //implementacão
}
```

Figura 11: Representação de uma API com os seus métodos *export*.

Fonte: Próprio Autor

6.3 Atualizações *Over-The-Air*

Para que a aplicação atualize o seu código JavaScript verificando a disponibilidade de uma código JavaScript mais recente a cada inicialização, conforme a documentação de Expo (2020) explica, é necessário instalar a biblioteca "*expo-updates*", e implementar na API da autenticação por exemplo, o método que busca e aplica as atualizações, que chama a validação do método *Updates.checkForUpdateAsync()* onde irá proceder para a verificação de atualizações de código, que fica hospedado no próprio servidor da Expo, após o cadastro e *upload* do mesmo, para então carregar a versão mais recente na inicialização do aplicativo, exemplificado na figura 12 abaixo.

```
import * as Updates from "expo-updates";

export async function checkForUpdates() {
  const { isAvailable } = await Updates.checkForUpdateAsync();
  if (isAvailable) {
    await Updates.fetchUpdateAsync();
    await Updates.reloadAsync();
  }
}
```

Figura 12: Implementação para atualizações de código JavaScript.

Fonte: Próprio Autor

6.4 Arquitetura sendo implementada.

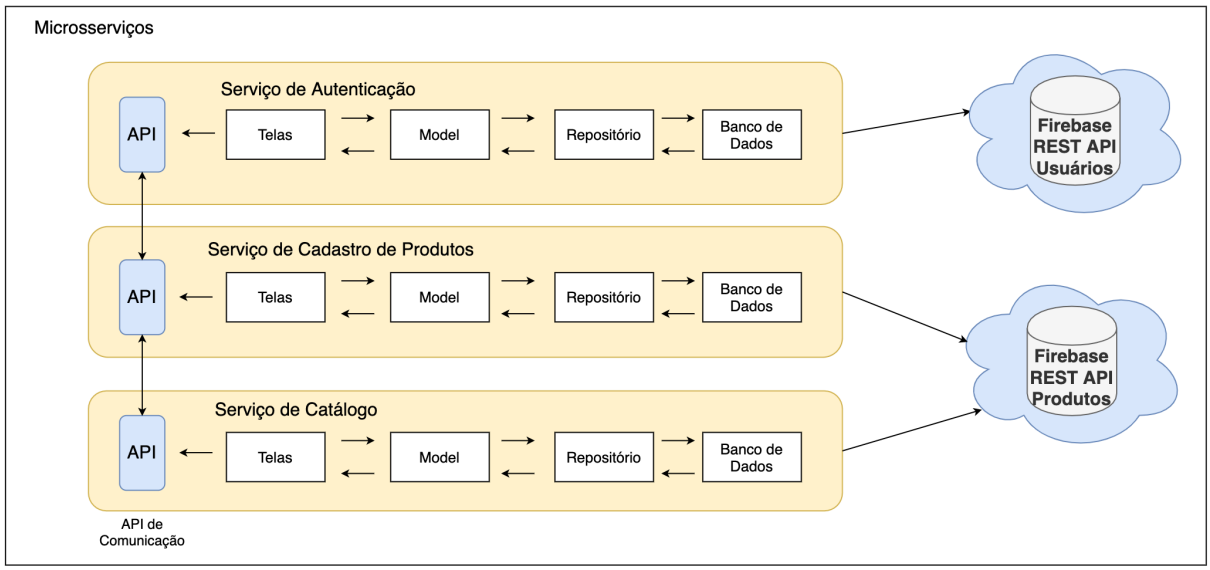

Figura 13: Arquitetura proposta para o protótipo.

Fonte: Próprio Autor

O aplicativo é implementado seguindo o novo modelo descrito, representado na figura 13 acima, mostrando uma aplicação que possuí três serviços que se comunicam entre si através de suas respectivas API's, criando o isolamento necessário para atingir o padrão de microsserviços.

Cada funcionalidade ou pacote, tem uma estrutura interna que consiste nas telas da aplicação, a lógica e regra de negócios na camada *Model*, o repositório, e a camada de banco de dados que tem a possibilidade de consumir dados de seus respectivo banco de dados interno Async Storage ou fazer chamadas REST API para consultar uma versão de seu banco que estará espelhado e disponibilizado na plataforma do Firebase.

Na sequência, é demonstrado a adição de uma nova funcionalidade, que consiste na visualização das informações do usuário autenticado, destacado na cor verde, que tem o propósito de exemplificar a adição novas funcionalidades ao sistema, ficando então da seguinte maneira como mostra a figura 14 abaixo.

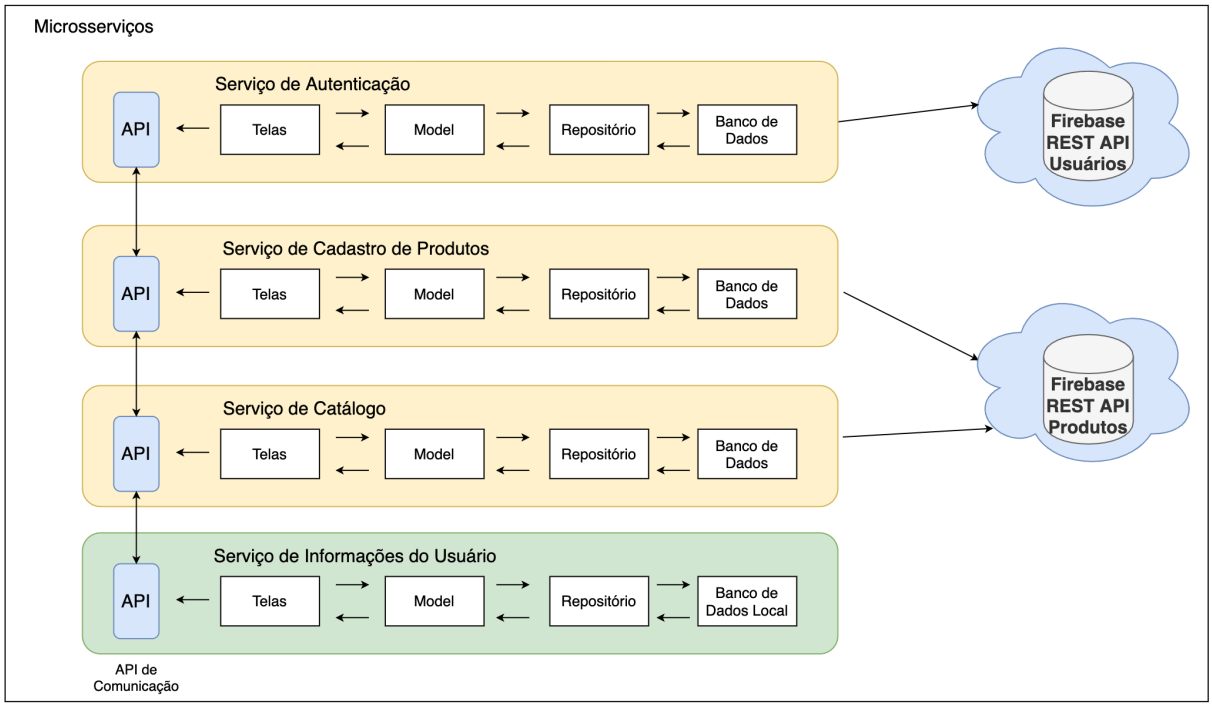

Figura 14: Arquitetura com uma nova funcionalidade adicionada.

Fonte: Próprio Autor

Com a arquitetura implementada desta maneira, é possível adicionar novas funcionalidades de forma que não é feito muitas amarras de código, o que facilita a troca ou adição de funcionalidades de outras aplicações, além exemplificar os fluxos que são encontrados em grande parte dos aplicativos hoje, como a navegação das telas.

6.5 Diagramas UML e telas do protótipo.

Neste sistema foi criado um fluxo para demonstrar a transição e interação entre cada um dos serviços, constituído pela tela de autenticação, cadastro de produtos, visualização de produtos e visualização das informações do usuário.

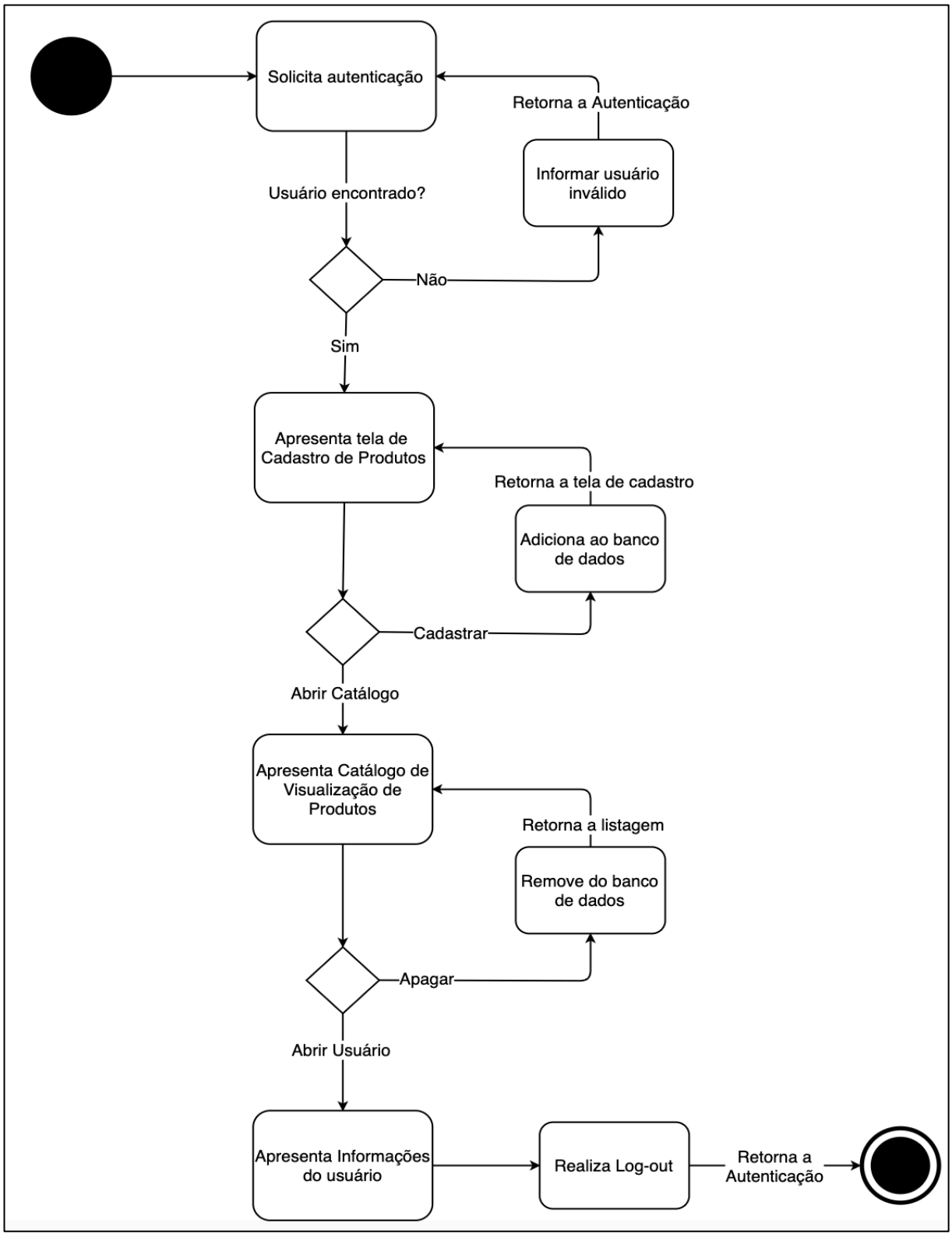

Figura 15: Diagrama de atividade do protótipo.

Fonte: Próprio Autor

O diagrama de atividade acima representa um fluxo completo do sistema, seguindo o fluxo de execução de cada atividade realizada.

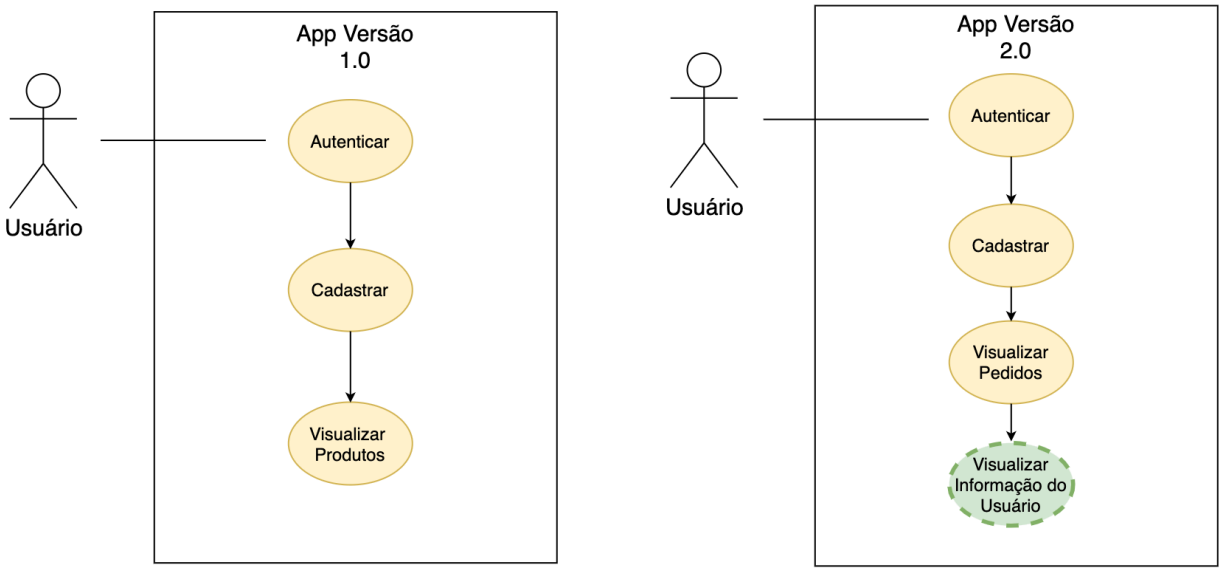

Figura 16: Diagrama de caso de uso da versão com e sem a nova funcionalidade.

Fonte: Próprio Autor

O diagrama de caso de uso representa a interação do usuário com o sistema, realizando as ações na ordem comum da interação.

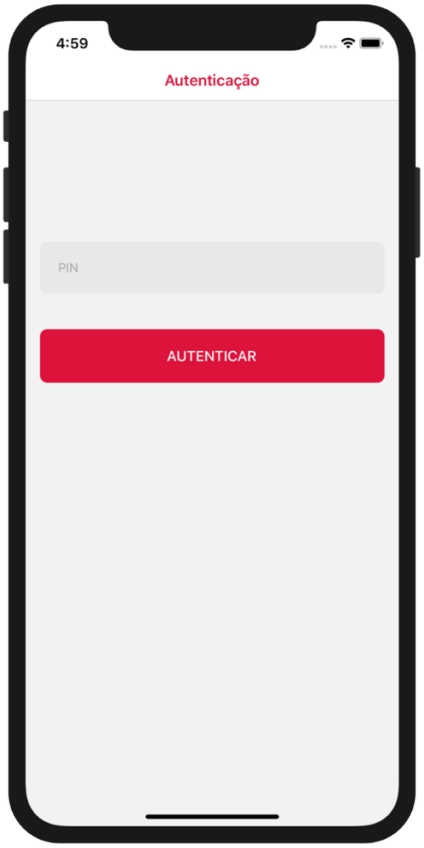

Figura 17: Tela autenticação, usando PIN.

Fonte: Próprio Autor

A tela de autenticação acima na figura 17 usa o método de autenticação por PIN, apresentando uma mensagem de usuário inválido caso seja feito a tentativa de autenticação com um usuário inexistente.

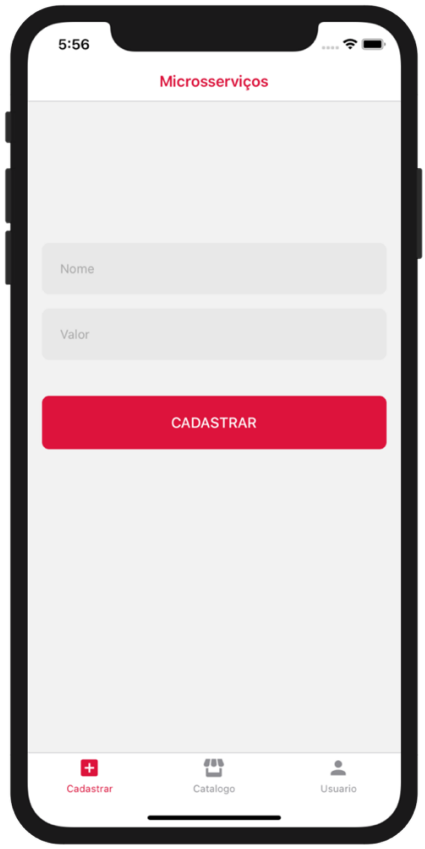

Figura 18: Tela de cadastro de produtos.

Fonte: Próprio Autor

A tela de cadastro de produtos é a primeira tela a ser apresentada após a autenticação, permitindo o cadastro de um novo produto, que possui um nome e um valor.

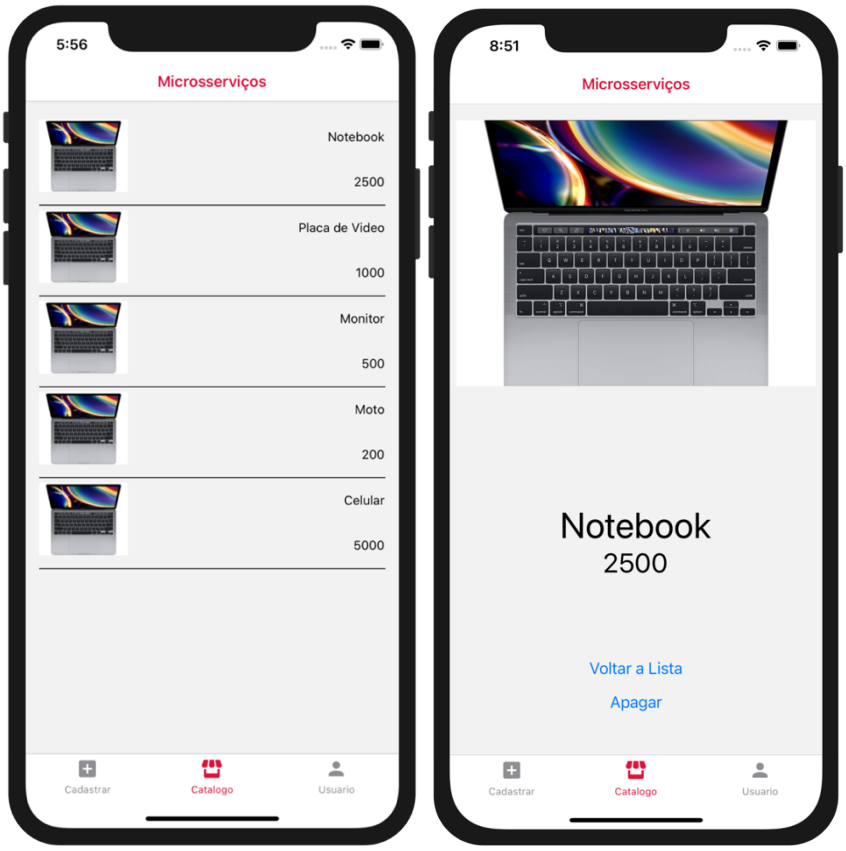

Figura 19: Telas de visualização dos produtos cadastrados

Fonte: Próprio Autor

A tela de visualização dos produtos possui uma listagem com todos os produtos cadastrados, mostrando uma imagem representativa, nome e valor do produto, tendo a possibilidade de abrir um produto e realizar a exclusão do mesmo do banco de dados.

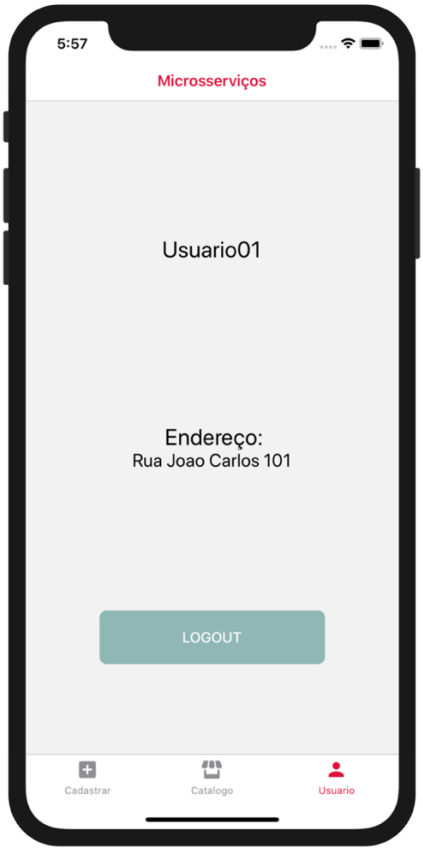

Figura 20: Tela do usuário, funcionalidade incluída posteriormente.

Fonte: Próprio Autor

A tela de informações do usuário apresenta o usuário que está autenticado no momento, uma informação de endereço e uma ação para realizar o *logout* e retornar para a tela de autenticação.

6.6 Organização do Banco de Dados do Firebase

Além do banco de dados local *Async Storage* para armazenar informações pertinentes a sessão, foram utilizados 2 banco de dados para persistência de dados, um para o serviço de autenticação de usuários e outro para o de produtos, que são mantidos na nuvem do Firebase, seguindo uma abordagem de banco de dados NoSQL, ou seja, não relacional, estruturado no padrão JSON, ficando então para este protótipo a estrutura das figuras 21 e 22 abaixo.

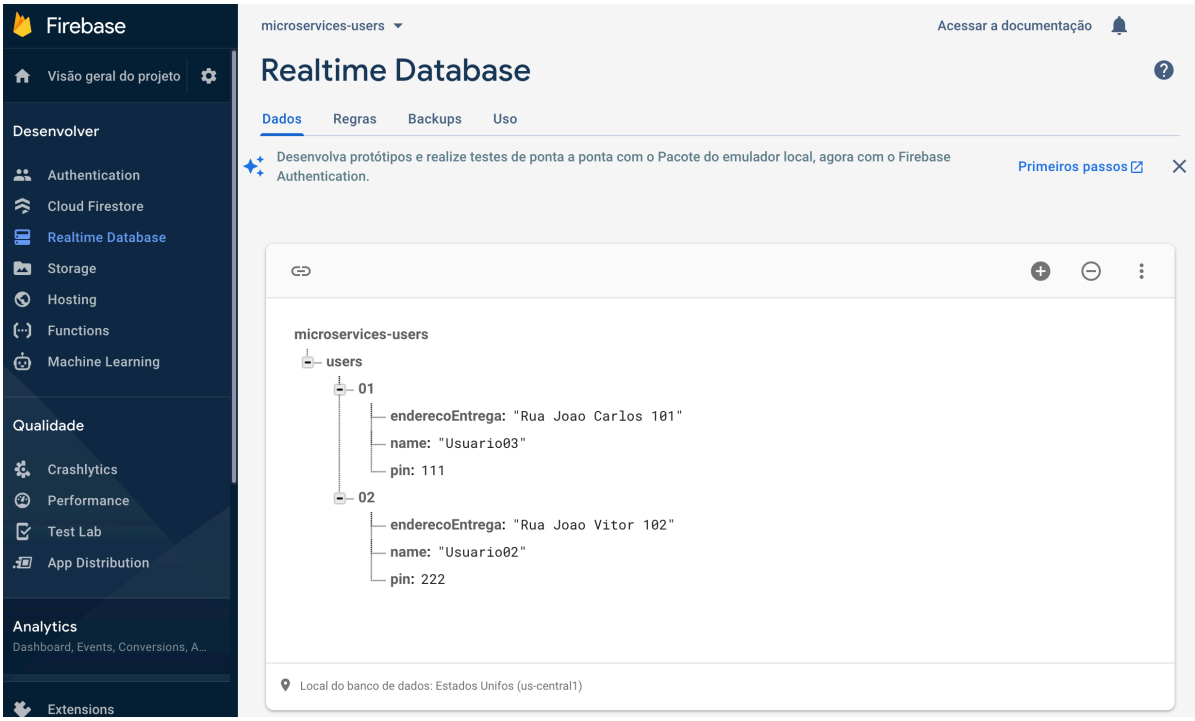

Figura 21: Estrutura do banco de dados de usuário do Firebase.

Fonte: Próprio Autor

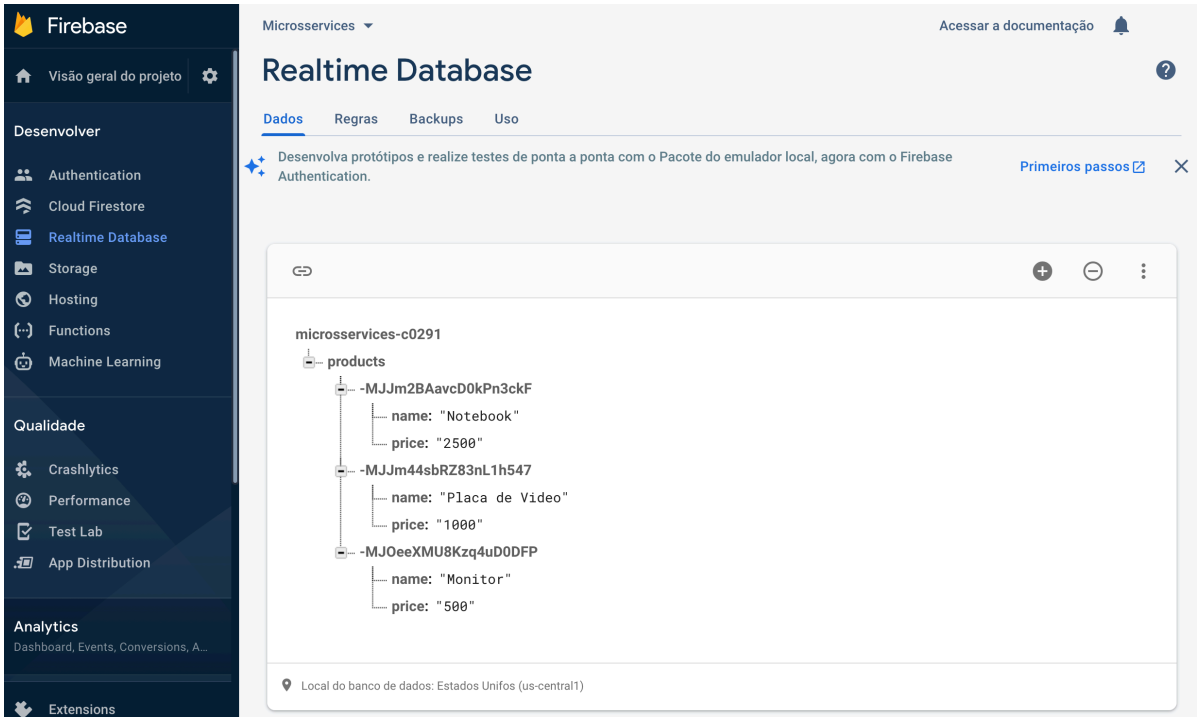

Figura 22: Estrutura do banco de dados de produtos do Firebase.

Fonte: Próprio Autor

7 CONCLUSÂO

Tendo a aprovação deste trabalho, este seguirá para a implantação no seu local de testes, para as próximas validações e recolhimento de *feedbacks*, afim de realizar melhorias futuras e se adequando a novas necessidades.

Para este trabalho, foi realizado um protótipo que reúne o essencial da arquitetura dentro das limitações encontradas, sendo possível neste momento a obtenção dos seguintes resultados conforme identificado durante a pesquisa.

Compartilhamento de código e funcionalidades entre aplicações distintas; devido a arquitetura aqui apresentada, traz uma grande agilidade no desenvolvimento, onde não é mais necessário começar uma nova aplicação do estado zero, pois permite o aproveitamento de funcionalidades existentes em outros aplicativos

Facilitação da manutenção do código; sendo possível pois cada funcionalidade pode ser gerenciada por um time diferente e ter o seu próprio repositório com versionamento.

Base de código única para Android e iOS com atualizações rápidas de correções de defeitos de código direto para o dispositivo do usuário, resultados da implementação das tecnologias do React Native com a biblioteca de atualizações do Expo, agilizando o processo de desenvolvimento do início ao fim.

REFERÊNCIAS

APPLE INC.. **App Review**. 2020. Disponível em: https://developer.apple.com/app-store/review/. Acesso em: 04 nov. 2020.

DABIT, Nader. **React Native in Action**. Shelter Island: Manning Publications, 2019.

DARWIN, Ian F.. **Android Cookbook**. Sebastopol: O'reilly Media, 2012.

DIAGRAMS.NET. **Diagram Software and Flowchart Maker**. 2020. Disponível em: https://www.diagrams.net/ Acesso em: 31 mai. 2020.

EISENMAN, Bonnie. **Learning React Native**: building native mobile apps with javascript. 2. ed. Sebastopol: O'Reilly Media, 2017.

EXPO. **Updating your App Over-the-Air**. 2020. Disponível em: https://docs.expo.io/bare/updating-your-app/. Acesso em: 07 nov. 2020.

GAUCHAT, John D. **SwiftUI for Masterminds**: how to take advantage of swiftui to create insanely great apps for iphones, ipads, and macs. Publicado Independentemente, 2020.

GEERS, Michael. **Micro Frontends in Action**. Shelter Island: Manning Publications, 2020.

GEERS, Michael. **Micro Frontends**: extending the microservice idea to frontend development. extending the microservice idea to frontend development. 2020. Disponível em: https://micro-frontends.org/. Acesso em: 30 set. 2020.

GHOFRANI, Javad; LÜBKE, Daniel. **Challenges of Microservices Architecture: A Survey on the State of the Practice.** Hannover: CEUR, 2018.

GIL, A. C. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas, 2002.

GOOGLE. **Publish an app**. 2020. Disponível em: https://support.google.com/googleplay/android-developer/answer/6334282?hl=en. Acesso em: 04 nov. 2020.

GOOGLE. **Firebase**. 2020. Disponível em: https://firebase.google.com/?hl=pt-br/. Acesso em: 09 mai. 2020.

JACKSON, Cam. **Micro Frontends**. 2019. Disponível em: https://martinfowler.com/articles/micro-frontends.html. Acesso em: 30 set. 2020.

JUNG, C. F., **Metodologia Científica: Ênfase em Pesquisa Tecnológica.** 2003. Disponível em:<http://professor.pucgoias.edu.br/SiteDocente/admin/arquivosUpload/4490/material/Metodologia_Cientifica_4_Edicao_P_B.pdf />. Acesso em: 09 mai. 2020.

MICROSOFT. **Visual Studio Code**: code editing. redefined. 2020. Disponível em: https://code.visualstudio.com/. Acesso em: 19 abr. 2020.

MOZZILA. **JavaScript**. 2019. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 23 ago. 2020.

NEWMAN, Sam. **Monolith to Microservices: evolutionary patterns to transform your monolith.** Evolutionary Patterns to Transform Your Monolith. Sebastopol: O'Reilly Media, 2019.

NEWMAN, Sam. **Building Microservices**. Sebastopol: O'Reilly Media, 2015.

PAYNE, Rap. **Beginning App Development with Flutter**: create cross-platform mobile apps. Dallas: Apress, 2019.

PONCE, Francisco; MÁRQUEZ, Gastón; ASTUDILLO, Hernán. **Migrating from monolithic architecture to microservices: A Rapid Review.** Concepcion: IEEE, 2019.

RICHARDSON, Chris. **Microservices Patterns: with examples in java.** WITH EXAMPLES IN JAVA. New York: Manning, 2019.